FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Randomness Testing of the Elephant Stream Cipher
Student:	Bc. Jiří Hájek
Supervisor:	Mgr. Olha Jurečková
Study program:	Informatics
Branch / specialization:	Computer Security
Department:	Department of Information Security
Validity:	until the end of summer semester 2023/2024

Instructions

The National Institute of Standards and Technology (NIST) initiated the Lightweight Cryptography (LWC) project to standardize lightweight cryptography algorithms for resource-constrained devices. In 2021, after two rounds, NIST announced ten finalists, including the Elephant stream cipher, which this work focuses on. The task of the thesis is to explain the structure of this cipher and test the randomness of its keystream using known tests.

Instructions:
1. Describe the Elephant cipher from the LWC project and at least five tests from the NIST Statistical Test Suite.
2. Describe and implement the Monomial for testing the randomness of binary sequences.
3. Apply the above tests to the Elephant cipher and compare the results.
4. In addition, try applying the Cube testers on the Elephant cipher.

Electronically approved by prof. Ing. Róbert Lórencz, CSc. on 26 January 2023 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Randomness Testing of the Elephant Stream Cipher

Jiří Hájek

Department of Information Security
Supervisor: Mgr. Olha Jurečková

May 4, 2023

Acknowledgements

I would like to thank my supervisor, Mgr. Olha Jurečková, for the feedback she has provided during the whole process of writing this thesis. I would also like to thank my friends and family for their support and encouragement.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 4, 2023 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Hájek, Jiří. *Randomness Testing of the Elephant Stream Cipher*. Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2023.

Abstract

This thesis presents an analysis of the randomness of the Dumbo instance of the Elephant cipher. The author used various statistical tests, including tests from NIST's Statistical Test Suite, the d-monomial test and cube testers.

The results showed that Dumbo passed the majority of the tests from NIST's STS with only a few failures that were not considered significant. The d-monomial test also showed no bias in any particular keystream bit. The most significant finding of this work was the discovery that Dumbo had difficulty passing the balance test from the family of cube testers as some keystream bits only passed every other test case and many cubes resulted in a pass rate of around 20 %.

The thesis concludes that Dumbo demonstrates good randomness properties overall but highlights the need for further investigation of this cipher to identify any potential major issues.

Keywords random bit generator, statistical testing, randomness, NIST STS, d-monomial test, cube testers, Dumbo, Elephant, Lightweight Cryptography project

Abstrakt

Tato práce předkládá analýzu náhodnosti instance Dumbo šifry Elephant. Autor využil různé statistické testy, včetně testů ze sady statistických testů od NIST, *d*-monomiálního testu a cube testerů.

Výsledky ukázaly, že Dumbo prošla většinou testů ze sady statistických testů od NIST s pouhými několika selháními, která nebyla považována za významná. Podobně ani *d*-monomiální test neukázal žádné zaujetí v jakémkoliv z bitů keystreamu. Nejvýznamnějším nálezem této práce bylo zjištění, že Dumbo má problém s tzv. balance testem z rodiny cube testerů, kdy některé bity keystreamu uspěly pouze v každém druhém testovacím případě a mnoho voleb tzv. cubes vykázalo úspěšnost kolem pouhých 20 %.

Práce je završena konstatováním, že Dumbo celkově prokazuje dobré vlastnosti týkající se náhodnosti, ale zdůrazňuje potřebu dalšího výzkumu pro identifikaci potenciálních závažných problémů této šifry.

Klíčová slova generátor náhodných bitů, statistické testy, náhodnost, NIST STS, d-monomiální test, cube testery, Dumbo, Elephant, Lightweight Cryptography project

Contents

List of Figures

List of Tables

Introduction

We have seen rapid technology advancements in the last few decades and recently also a boom of Internet of Things (IoT) devices. Many people are adopting smart home appliances, such as smart refrigerators, security cameras or alarms.

These devices, especially in the case of a smaller form factor, do not pack much computing power compared to traditional desktop computers or laptops. Aside from the size limitations, there is also a desire to keep the costs down for the consumers, which also puts a limit on the amount of computing power contained in the device. Last but not least, if the device is wireless, i.e., powered by a battery, it is in the interest of the manufacturer to keep the power consumption as low as possible to make the device last longer on a single charge, putting yet another constraint on computing power.

To ensure secure and efficient data transmission for these devices, there is a need for cryptographic algorithms that take into account these constraints. In 2019, the National Institute of Standards and Technology (NIST) has initiated a process to standardize lightweight cryptographic algorithms that are suitable for use in these constrained environments. A couple of years went by and in March 2021, after two rounds of evaluations, NIST announced ten finalists of the initial 56 candidates.

This thesis focuses on one of these finalists — the Elephant cipher, namely the Elephant-*Spongent*-π[160] (Dumbo) instance, which is the primary member of the submission.

The main goal of this thesis is to assess whether the keystream of the cipher has good randomness properties, which is something we would want for the sake of security. If the keystream was not random, various attacks could take advantage of the biases in keystream bits.

In Chapter 1, we provide the necessary mathematical background, present the Lightweight Cryptography (LWC) project and describe the inner workings of the Elephant cipher.

In Chapter 2, we define what a Random Bit Generator (RBG) is, why we need to test RBGs and how the Elephant cipher fits into this concept.

In Chapter 3, we go over some of the basics of statistics and hypothesis testing, which are employed in the next chapter.

In Chapter 4, we discuss known tests of randomness for RBGs and dive into the details of their implementation. We focus on tests from the NIST Statistical Test Suite (STS), monomial tests and cube testers.

Finally, in Chapter 5, we apply the described tests to the Elephant cipher and discuss the results.

Chapter 1

Elephant cipher

In this chapter, we will start off by introducing fundamental definitions and theorems necessary to understand the concepts behind the Elephant cipher. Then, we will present the LWC project and discuss the motivation behind it. We will dive into the inner workings of the Elephant-Spongent-π[160] (Dumbo) instance of the Elephant cipher, where we will describe how encryption is done as well as some inner structures used by the Dumbo instance of the Elephant cipher. Finally, we will conclude the chapter by describing a modification of the encryption algorithm to extract the keystream.

1.1 Preliminaries

1.1.1 Notation

In this section, we will describe the notation used throughout the rest of this chapter. We denote by $x \ll i$ (resp., $x \gg i$) a shift to the left (resp., right) over i positions and likewise by $x \lll i$ (resp., $x \ggg i$), we denote a rotation of x to the left (resp., right) over i positions. For $X \in \{0,1\}^*$, we define

$$X_1 \dots X_\ell \xleftarrow{n} X$$

to be a function that partitions X into $\ell = \lceil |X|/n \rceil$ blocks of n bits, where the last block is padded with zeros. We denote by $\lfloor x \rfloor_i$ the i left-most bits of x.

1.1.2 Stream ciphers

Elephant is a symmetric stream cipher. In this section, we will define these terms and briefly go over the basics of stream ciphers.

The following definitions in this section come from [1]. Let us first define what a cipher is. This definition applies to both symmetric and asymmetric ciphers.

Definition 1 (Cipher) *Cipher is a quintuple (M, C, K, E, D), where M is the space of all plaintexts, C is the space of all ciphertexts and K is the space of all keys. E and D are functions, which for each unique key $k \in K$ specify the encryption transformation $E_k : M \to C$ and the decryption transformation $D_k : C \to M$. Every combination of $(m, k) \in M \times K$ satisfies $D_k(E_k(m)) = m$.*

From the perspective of the nature of the encryption and decryption transformations, ciphers can be divided into two groups — symmetric ciphers and asymmetric ciphers.

Definition 2 (Symmetric cipher) *A symmetric cipher is a cipher with a construction such that it is possible to determine a decryption transformation D_k from the knowledge of the encryption transformation E_k and vice versa for all $k \in K$.*

Definition 3 (Asymmetric cipher) *An asymmetric cipher is a cipher, for which it is impossible for almost all keys $k \in K$ to determine a decryption transformation D_k from the knowledge of the encryption transformation E_k. In practice, the key k is a secret used to set up two parameters (e, d), which are called the public key and the private key respectively. These are then used to derive the final encryption and decryption transformations.*

Next, we divide ciphers into another two groups based on how they use the key to derive ciphertext corresponding to the given plaintext — stream ciphers and block ciphers. For the sake of simplicity, we will limit ourselves to symmetric stream ciphers as we will not work with asymmetric nor block ciphers in this thesis.

Definition 4 (Symmetric stream cipher) *Let A be an alphabet of q symbols, $M = C$ a set of all strings over A and K set of all keys. Symmetric stream cipher consists of transformation G, function E and function D. For every key $k \in K$, the generator G generates a keystream sequence $h(1), h(2), \ldots$, where $h(i)$ represents arbitrary substitutions $E_{h(1)}, E_{h(2)}, \ldots$ over the alphabet A. Functions E and D define the encryption transformation E_k and the decryption transformation D_k based on given key $k \in K$. Ciphertext for given plaintext $m = m(1), m(2), \ldots$ is calculated as $c(1) = E_{h(1)}(m(1)), c(2) = E_{h(2)}(m(2)), \ldots, c(|m|) = E_{h(|m|)}(m(|m|))$, where $|m|$ is the length of the plaintext. Similarly, plaintext for given ciphertext $c = c(1), c(2), \ldots$ is calculated as $m(1) = D_{h(1)}(c(1)), m(2) = D_{h(2)}(c(2)), \ldots, m(|c|) = D_{h(|c|)}(c(|c|))$, where $|c|$ is the length of the ciphertext. Note that $D_{h(i)} = E^{-1}_{h(i)}$. See Figure 1.1 for a depiction of the encryption and decryption transformations.*

In order to allow for encryption of multiple messages with the same key without compromising on security, a stream cipher usually uses an Initialization Vector (IV), also referred to as a nonce. The IV is usually a random

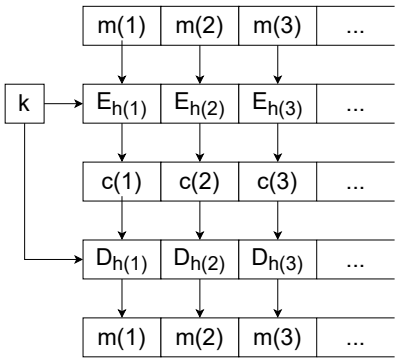

Figure 1.1: Depiction of encryption transformation E_k and decryption transformation D_k of a symmetric stream cipher. Adapted from [1].

sequence of bits or a number, which is, together with the key, used to generate the keystream. This results in generating a different ciphertext for the same plaintexts and keys (assuming the IVs differ) and therefore masks the fact that two ciphertexts that used the same key correspond to the same plaintext.

1.1.3 Linear feedback shift register

Linear Feedback Shift Registers (LFSRs) are used in many stream ciphers as keystream generators [2]. The authors of [2] list several reasons for this:

1. LFSRs are well suited for hardware implementation,
2. they can produce sequences of large period,
3. they can produce sequences with good statistical properties,
4. they can be easily analyzed using algebraic techniques due to the nature of their structure.

An LFSR is a shift register which consists of L stages $0, \ldots, L-1$, each capable of storing some information and a clock controlling data exchange [3]. Although the Elephant cipher uses an LFSR that stores an 8-bit word in each stage [4], we will explain the principles of LFSRs using stages with one-bit storage for the sake of simplicity. At each clock cycle, the following operations are performed [2].

1. The content of the initial stage 0 is output and forms part of the output sequence,
2. the content of stage i is moved to stage $i-1$ for each $i \in \{1, \ldots, L-1\}$,

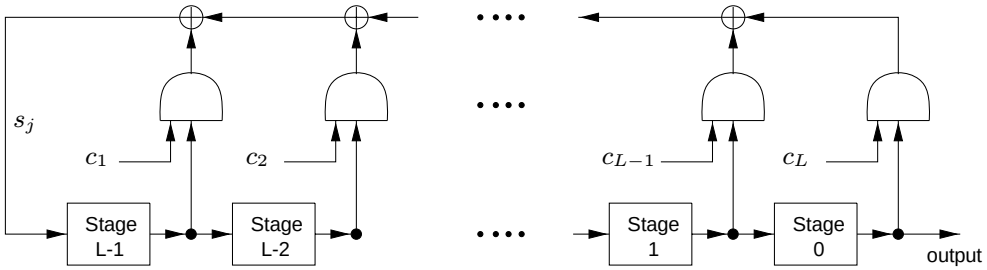

Figure 1.2: A linear feedback shift register (LFSR) of length L. Adapted from [2].

3. the new content of stage $L-1$ is the feedback denoted by s_j, which is calculated by mixing together contents of a fixed subset of stages $0, \ldots, L-1$ using the XOR operation.

The following definitions and theorems in this section are adapted from [2].

Definition 5 *The LFSR in Figure 1.2 is defined by a couple $\langle L, C(D)\rangle$, where L is the length and $C(D) = 1 + c_1 D + c_2 D^2 + \cdots + c_L D^L \in \mathbb{Z}_2[D]$ is the connection polynomial. The LFSR is said to be non-singular if the degree of $C(D)$ is L (i.e., $c_L = 1$). The initial content $[s_{L-1}, \ldots, s_0]$, where for each $i \in \{0, \ldots, L-1\}$, s_i is the initial content of stage i, is called the initial state of the LFSR.*

Theorem 1 *If the initial state of the LFSR in Figure 1.2 is $[s_{L-1}, \ldots, s_0]$, then the output sequence $s = s_0, s_1, s_2, \ldots$ is uniquely determined by the following recursion:*

$$s_j = c_1 s_{j-1} \oplus c_2 s_{j-2} \oplus \cdots \oplus c_L s_{j-L} \text{ for } j \geq L.$$

Theorem 2 *Every output sequence (i.e., for all initial states) of an LFSR $\langle L, C(D)\rangle$ is periodic if and only if the connection polynomial $C(D)$ has degree L (i.e., the LFSR is non-singular).*

The LFSR used in Dumbo is defined as a $\mathbb{Z}_2$-linear map by the following formula [4]:

$$(x_{19}, \ldots, x_0) \mapsto (x_0 \lll 3 \oplus x_3 \ll 7 \oplus x_{13} \ggg 7, x_{19}, \ldots, x_1),$$

where x_i is an 8-bit word (i.e., $x_i \in \{0,1\}^8$).

Example 1 *Let us consider the LFSR used in Dumbo with decimal representation of its stages. Let*

$$a_0 = (19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1, 0)$$

be the initial state. Let us walk through the first three clock cycles. In the first cycle, 19 will be replaced by

$$0 \lll 3 \oplus 3 \ll 7 \oplus 13 \ggg 7 = 0 \oplus 129 \oplus 26 = 155,$$

resulting in

$$a_1 = (155, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2, 1).$$

In the second cycle, 155 will be replaced by

$$1 \lll 3 \oplus 4 \ll 7 \oplus 14 \ggg 7 = 8 \oplus 2 \oplus 28 = 22,$$

resulting in

$$a_2 = (22, 155, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3, 2).$$

Finally, in the third cycle, 22 will be replaced by

$$2 \lll 3 \oplus 5 \ll 7 \oplus 15 \ggg 7 = 16 \oplus 130 \oplus 30 = 140,$$

resulting in

$$a_3 = (140, 22, 155, 19, 18, 17, 16, 15, 14, 13, 12, 11, 10, 9, 8, 7, 6, 5, 4, 3).$$

1.1.4 Spongent permutation

Spongent is a family of lightweight hash functions with various hash sizes based on a sponge construction with a `Present`-type permutation [5]. The sponge construction is an iterated design that takes an input of variable length and produces an output of arbitrary length based on a permutation π_b operating on a state of fixed number of bits b [5]. The size of the internal state b is called width and satisfies $b = r + c \geq n$, where r is the rate (the number of bits input or output per one permutation call) and c is the capacity (internal state bits not used for input or output) [5]. As described in [5], the sponge construction proceeds in the following three phases. See Figure 1.3 for a depiction of this process.

Initialization phase The message is padded by a single 1 bit followed by 0 bits up to a multiple of r bits (e.g., if $r = 4$, then a one-bit message "0" could be transformed to "0100") and then cut into blocks of r bits.

Absorbing phase The r-bit input message blocks are mixed into the first r bits of the state using the XOR operation, interleaved with applications of the permutation π_b.

Squeezing phase The first r bits of the state are returned as output, interleaved with applications of the permutation π_b until n bits are returned.

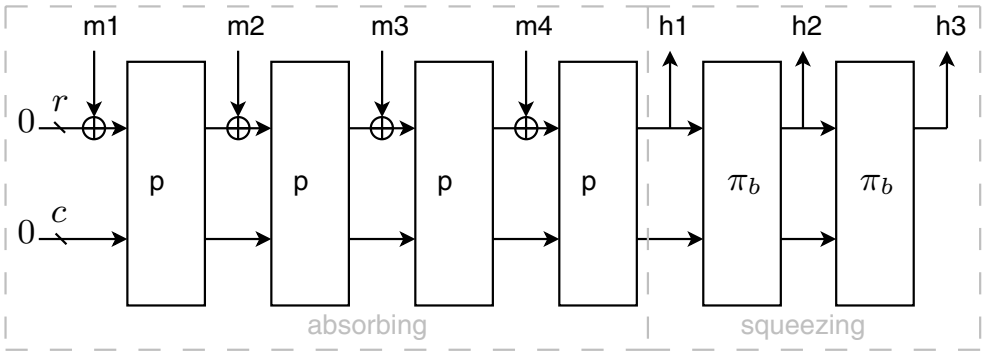

Figure 1.3: Sponge construction based on a b-bit permutation π_b with capacity c bits and rate r bits. m_i are r-bit message blocks and h_i are parts of the hash value. Adapted from [5].

Algorithm 1 160-bit Spongent permutation. Adapted from [4].

1: **for** $i = 1, \ldots, 80$ **do**
2: $\quad X \leftarrow X \oplus 0^{153} || \texttt{lCounter}_{160}(i) \oplus \texttt{rev}(0^{153} || \texttt{lCounter}_{160}(i))$
3: $\quad X \leftarrow \texttt{sBoxLayer}_{160}(X)$
4: $\quad X \leftarrow \texttt{pLayer}_{160}(X)$

In the context of the Elephant cipher, we denote by *Spongent*-π[160]: $\{0,1\}^{160} \rightarrow \{0,1\}^{160}$ the 80-round *Spongent* permutation from [5]. It takes a 160-bit input X [4] and operates as described in Algorithm 1.

Let us now describe the functions used in Algorithm 1 as defined in [4].

`rev` This function reverses the order of the bits of its input.

$\texttt{lCounter}_{160}$ This function represents a 7-bit LFSR [1] defined by the polynomial $p(x) = x^7 + x^6 + 1$ and initial state "1110101". It outputs the state of the LFSR at the given clock cycle. The first output is the initial state.

$\texttt{sBoxLayer}_{160}$ This function consists of an S-box $S : \{0,1\}^4 \rightarrow \{0,1\}^4$ applied 40 times in parallel defined as

X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
$S(X)$	E	D	B	0	2	1	4	F	7	A	8	5	9	C	3	6

[1] The authors of [5] used a slightly different LFSR construction which shifts the stages to the left, maps the feedback coefficients $c_1, \ldots, c_L$ in reverse order and puts the feedback in stage 0.

pLayer$_{160}$ This function moves the j-th bit of its input to bit position $P_{160}(j)$, where

$$P_{160}(j) = \begin{cases} 40 \cdot j \mod 159 & \text{if } j \in \{0, \ldots, 158\}, \\ 159 & \text{if } j = 159. \end{cases}$$

Example 2 *Let us walk through the first iteration of the 160-bit Spongent permutation (Algorithm 1). We will start with initial state $X = 0^{160}$ for the sake of simplicity. Since the first output of the* **lCounter**$_{160}$ *function is the initial state,* **lCounter**$_{160}(1) = 1110101$. *This results in the following computation on line two of the algorithm:*

$$\begin{aligned} X &= 0^{160} \oplus 0^{153} || 1110101 \oplus \textit{rev}(0^{153} || 1110101) \\ &= 0^{160} \oplus 0^{153} || 1110101 \oplus 1010111 || 0^{153} \\ &= 10101110 || 0^{144} || 01110101. \end{aligned}$$

After line two, the state is $X =$ `AE 00 00 ... 00 00 75`.

On line three, we apply the **sBoxLayer**$_{160}$ *function to each of the 40 bytes in the state separately. The first two bytes* `AE` *are mapped to* `83` *because* $S($`A`$) =$ `8` *and* $S($`E`$) =$ `3`*, zero bytes are mapped to* $S($`0`$) =$ `E` *and the last two bytes* `75` *are mapped to* `F1` *because* $S($`7`$) =$ `F` *and* $S($`5`$) =$ `1`*. This results in state* $X =$ `83 EE EE ... EE EE F1`.

On line four, we apply the **pLayer**$_{160}$ *function, which mixes the bits of the state. The first bit* $X[0] = 1$ *stays at position* $P_{160}(0) = 40 \cdot 0 \mod 159 = 0$*, the second bit* $X[1] = 0$ *moves to position* $P_{160}(1) = 40 \cdot 1 \mod 159 = 40$*, the fifth bit* $X[4] = 1$ *moves to position* $P_{160}(4) = 40 \cdot 4 \mod 159 = 1$ *and so on. After applying* **pLayer**$_{160}$*, the first iteration is finished and the state is*

$X =$ `BF FF FF FF FE 3F FF FF FF FE 7F FF FF FF FE 40 00 00 00 03`.

The remaining 79 iterations would be computed in a similar manner. The next state of the LFSR would be **lCounter**$_{160}(2) = 1101010$ *(the XOR of the first two bits is appended to the right and the left-most bit is then discarded).*

1.2 Lightweight Cryptography project

A couple of years ago, NIST has initiated a process to standardize lightweight cryptographic algorithms suitable for use in constrained environments, for which the regular NIST cryptographic standards were not adequate due to poor performance [6]. With the recent growth of IoT devices like home assistants or security cameras, which people are still adopting today, there was a need for a cryptography standard that would be suitable for these devices in terms of performance while still providing good security.

In August 2018, NIST published a call for algorithms to be considered for lightweight cryptographic standards and received 57 submissions [6]. NIST

then performed three rounds of reviews and evaluations and selected ten finalists, one of which is the Elephant cipher this thesis focuses on [6, 7]. It should be noted that NIST has recently announced the selection of the Ascon family for lightweight cryptography standardization [6], meaning the cipher this thesis focuses on did not end up being the lightweight cryptography standard.

1.3 Elephant cipher

Elephant is an authenticated encryption scheme, whose mode is a nonce-based encrypt-then-MAC construction, where encryption is performed using counter mode and message authentication using a variant of the protected counter sum MAC function [4]. Both of these modes internally use a cryptographic permutation masked using LFSRs [4]. The Elephant encryption algorithm generates a keystream, which is then mixed with the given plaintext using the XOR operation [4], making it a stream cipher. There are three different instances of the Elephant cipher [4]. As listed in [4], these are

`Dumbo:` `Elephant-Spongent-`$\pi[160]$ This instance achieves 112-bit security as long as the online complexity does not exceed around 2^{46} blocks.

`Jumbo:` `Elepehant-Spongent-`$\pi[176]$ The Jumbo instance is similar to the Dumbo instance, but achieves 127-bit security under the same conditions for online complexity.

`Delirium:` `Elepehant-Keccak-`$f[200]$ This instance uses the `Keccak` permutation instead of `Spongent` and is more aimed towards use in software. It also achieves 127-bit security, but has a higher bound for online complexity at around 2^{70} blocks.

In the rest of this thesis, we will only focus on the `Elephant-Spongent-`$\pi[160]$ (Dumbo) instance of the Elephant cipher, which is the primary member of the LWC project submission and the most optimized instance for hardware [4].

1.3.1 Keystream extraction

In this section, we will finally dive into the Elephant cipher. First, we will introduce the regular encryption algorithm as defined in [4] and then modify it so that it only extracts the keystream instead of returning the ciphertext.

Let $k, m, n, t \in \mathbb{N}$, where $k, m, t \leq n$. In the case of the Dumbo instance, $n = 160$. Let $P : \{0,1\}^{160} \to \{0,1\}^{160}$ be a 160-bit permutation as specified in Section 1.1.4 and $\varphi_1 : \{0,1\}^n \to \{0,1\}^n$ be an LFSR. Define $\varphi_2 = \varphi_1 \oplus \texttt{id}$, where `id` is the identity function. Define the function $\texttt{mask} : \{0,1\}^k \times \mathbb{N}^2 \to \{0,1\}^n$ as

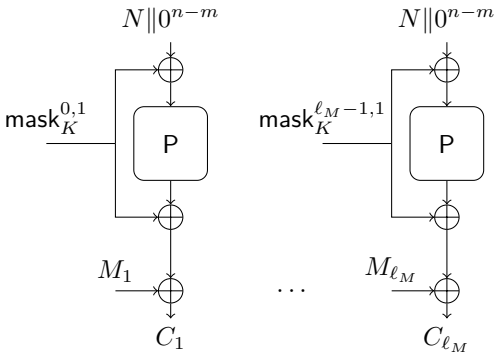

Figure 1.4: Depiction of the Elephant encryption algorithm. The message is padded as $M_1 \dots M_{\ell_M} \xleftarrow{n} M$ and ciphertext equals $C = \lfloor C_1 \dots C_{\ell_M} \rfloor_{|M|}$. Adapted from [4].

$$\mathtt{mask}_K^{a,b} = \mathtt{mask}(K, a, b) = \varphi_2^b \circ \varphi_1^a \circ P(K\|0^{n-k})$$

The input of the regular Elephant encryption algorithm consists of a key $K \in \{0,1\}^k$, a nonce $N \in \{0,1\}^m$, associated data $A \in \{0,1\}^*$ and a message $M \in \{0,1\}^*$. The algorithm returns corresponding ciphertext $C \in \{0,1\}^{|M|}$ and a tag $T \in \{0,1\}^t$. See Algorithm 2 and Figure 1.4 for more details.

Algorithm 2 Elephant encryption. Adapted from [4].

$M_1 \dots M_{\ell_M} \xleftarrow{n} M$
for $i = 1, \dots, \ell_M$ **do**
    $C_i \leftarrow M_i \oplus P(N\|0^{n-m} \oplus \mathtt{mask}_K^{i-1,1}) \oplus \mathtt{mask}_K^{i-1,1}$
$C \leftarrow \lfloor C_1 \dots C_{\ell_M} \rfloor_{|M|}$
$A_1 \dots A_{\ell_A} \xleftarrow{n} N\|A\|1$
$C_1 \dots C_{\ell_C} \xleftarrow{n} C\|1$
$T \leftarrow A_1$
for $i = 2, \dots, \ell_A$ **do**
    $T \leftarrow T \oplus P(A_i \oplus \mathtt{mask}_K^{i-1,0}) \oplus \mathtt{mask}_K^{i-1,0}$
for $i = 1, \dots, \ell_C$ **do**
    $T \leftarrow T \oplus P(C_i \oplus \mathtt{mask}_K^{i-1,2}) \oplus \mathtt{mask}_K^{i-1,2}$
$T \leftarrow P(T \oplus \mathtt{mask}_K^{0,0}) \oplus \mathtt{mask}_K^{0,0}$
return $(C, \lfloor T \rfloor_t)$

To extract the keystream, we simply remove the XOR operation, which mixes it with the message. We can also take out the tag computation, since the tag does not serve us any purpose. The input of the keystream extraction algorithm consists of a key $K \in \{0,1\}^k$, a nonce $N \in \{0,1\}^m$ and a message

$M \in \{0,1\}^*$. At the end, the corresponding keystream $S \in \{0,1\}^{|M|}$ is returned. See Algorithm 3 for the modified pseudocode.

Algorithm 3 Elephant keystream extraction

1: $M_1 \dots M_{\ell_M} \xleftarrow{n} M$
2: **for** $i = 1, \dots, \ell_M$ **do**
3: $\quad S_i \leftarrow P(N \| 0^{n-m} \oplus \mathtt{mask}_K^{i-1,1}) \oplus \mathtt{mask}_K^{i-1,1}$
4: $S \leftarrow \lfloor S_1 \dots S_{\ell_M} \rfloor_{|M|}$
5: **return** S

Chapter 2

Testing of random bit generators

In this chapter, we will start off with motivation behind testing of RBGs. Then, we will elaborate on how this topic is connected to stream ciphers and finally discuss why lack of randomness in stream ciphers is an issue.

2.1 Random bit generators

The security of many cryptographic systems depends on being able to generate unpredictable sequences of bits [2] — e.g., the keystream in stream ciphers or secret parameters in various encryption schemes. In all cases, the generated sequences must be of sufficient size and random in the sense that the probability of any particular value being selected must be sufficiently small to prevent an adversary from gaining an advantage though the optimization of a search strategy based on such probability [2].

Let us now define what a random bit generator is. The following definitions in this section are adapted from [2].

Definition 6 (Random bit generator) *Random Bit Generator (RBG) is a device or algorithm that outputs a sequence of statistically independent and unbiased binary digits.*

Definition 7 (Pseudorandom bit generator) *Pseudorandom Bit Generator (PRBG) is a deterministic algorithm, which, given a truly random binary sequence of length k, outputs a binary sequence of a much greater length l which "appears" to be random. The input of the PRBG is called the seed, while the output of the PRBG is called a pseudorandom bit sequence.*

2.2 Motivation

The output of a PRBG is not truly random [2]. The idea behind PRBGs is to take a small truly random sequence and expand it to a sequence of a much greater length in a way such that an adversary cannot efficiently distinguish between output sequences of the PRBG and truly random sequences [2].

A True Random Bit Generator (TRBG) requires a naturally occuring source of randomness [2]. Designing a device or a piece of software to utilize this randomness in order to produce a bit sequence that is free of biases and correlations is a difficult task [2]. On top of that, for most cryptographic applications, the generator must not be subject to observation or manipulation by an adversary [2].

RBGs based on natural sources of randomness are subject to influence by external factors as well as malfunction [2]. Therefore, it is important that such devices be tested periodically, e.g., by using statistical tests, some of which we will introduce in Chapter 4. Although PRBGs are not subject to the same external influences as TRBGs, they should also undergo some testing to ensure the underlying algorithms really generate unpredictable sequences, which is exactly what we will be doing later on in this thesis.

2.3 Randomness in ciphers

As we have hinted earlier in this chapter, stream ciphers depend on the ability to generate unpredictable sequences — the keystream. If the keystream was not random and contained some form of bias, an adversary would be able to guess some of its bits and have an easier time attacking the cipher, e.g., recovering plaintext from given ciphertext without the knowledge of the key that was used. This applies to Elephant as well, since, as we have mentioned before, it too is a stream cipher.

2.4 Issues with lack of randomness

Take the RC4 stream cipher as an example. The authors of [8] showed how an attacker can exploit biases in the keystream to distinguish RC4 keystreams of 2^{26} bytes from random sequences with success rate of more than 66 %. They also discovered a family of patterns in the keystream, which can be used to predict bits and words, e.g., after 2^{45} output words, a single bit can be predicted with probability of 85 % and after 2^{50} output words, a single byte can be predicted with probability of 82 %, contradicting the unpredictability property.

Consider the DES encryption algorithm as another example. DES has a key space of size 2^{56} [2]. If the secret key was selected using a TRBG, an adversary would on average have to try 2^{55} possible keys before guessing the

correct one [2]. On the other hand, if the key was selected by first choosing a 16-bit random secret that would then be expanded into a 56-bit key using a publicly known (or predictable in some way) function f, the adversary would on average only have to try 2^{15} keys obtained by running each one of the 2^{15} possible values through the function f or some prediction algorithm [2].

Chapter 3

Basics of hypothesis testing

In this chapter, we will introduce the mathematical background behind testing of hypotheses, which will be applied in the next chapter.

3.1 Preliminaries

Let us begin by explaining what a statistical hypothesis and statistical hypothesis test is. The explanations in the rest of this chapter are adapted from [9].

A statistical hypothesis is a statement about the properties of the probability distribution observed from a random variable. Hypotheses that concern the parameter values of probability distributions of random variables are called parametric hypotheses. Conversely, hypotheses that do not make assumptions about the distribution of a random variable are called nonparametric hypotheses. Hypotheses that are formulated such that the probability distribution of a random variable is uniquely determined are called simple hypotheses. Opposite of that are composite hypotheses, which do not specify the probability distribution of a random variable uniquely.

Statistical hypothesis testing is a procedure that allows us to determine whether the experimentally observed data satisfy an assumption we made before conducting the test. In hypothesis testing, we always compare two hypotheses. The hypothesis being tested is called the null hypothesis H_0 and we contrast it with an alternative hypothesis H_A.

Let $X = (X_1, \ldots, X_n)$ be a random sample from distribution $\mathcal{R}(\theta)$, where θ is a (possibly multidimensional) parameter. Let $h(\theta)$ be a parametric function and $k \in \mathbb{R}$ a constant. Furthermore, let the null hypothesis be

$$H_0 : h(\theta) = k.$$

The alternative hypothesis may be defined in the following three ways.

1. Right-tailed alternative hypothesis: $H_A : h(\theta) > k$.

Reality	**Test outcome**	
	H_0 not rejected	H_0 rejected
H_0 is true	correction decision	type I error
H_0 is false	type II error	correct decision

Table 3.1: Possible outcomes of a statistical hypothesis test in the context of type I and type II errors. Adapted from [9].

2. Left-tailed alternative hypothesis: $H_A : h(\theta) < k$.

3. Two-tailed alternative hypothesis: $H_A : h(\theta) \neq k$.

During the testing procedure, we decide whether or not to reject the given hypothesis H_0. Note that errors can be made during the decision process. There are two types of errors — type I error and type II error. Type I error occurs when the null hypothesis H_0 is true and we reject it. Type II error occurs when the null hypothesis H_0 is not true and we fail to reject it. There are four possible outcomes of a statistical hypothesis test, which are shown in Table 3.1.

In the case of a simple hypothesis, the probability of type I error is called the level of significance and is usually denoted by α. In the case of a composite hypothesis, α is the smallest upper bound of the probability of type I error. Suppose $X = (X_1, \ldots, X_n)$ is a random sample from a certain distribution and there is a null and alternative hypothesis laid out. The decision whether to reject the hypothesis H_0 or not is based on building a certain statistic $T = T(X_1, \ldots, X_n)$, which we call the test statistic. The test statistic T is a random variable that is a function of the observations $X_1, \ldots, X_n$. The decision is made by choosing a suitable set $W \subset \mathbb{R}$, called the critical region. In case that $T \in W$, we reject the null hypothesis H_0. Otherwise, we do not reject it. We choose the size of the critical region so that we reject the null hypothesis with probability α at most. Usually, we choose $\alpha = 0.05$ or $\alpha = 0.01$.

There is also a procedure, which we employ during testing of hypotheses to determine the smallest level of significance at which we would still reject the null hypothesis H_0. We call this level of significance the p-value. It expresses the smallest upper bound of the probability that we would obtain precisely our realization of the test statistic T or even more contradictory realizations of the tested hypothesis assuming the null hypothesis H_0 is true.

Let $X = (X_1, \ldots, X_n)$ be a random sample from a certain distribution, $T = T(X_1, \ldots, X_n)$ be a test statistic and t_0 be its value. Then, we can calculate the p-value using one of the three possible formulas, depending on the shape of the alternative hypothesis.

1. If we have a right-tailed alternative hypothesis, we compute the p-value

Condition	**Test outcome**
p-value $\leq \alpha$	Reject H_0
p-value $> \alpha$	Do not reject H_0

Table 3.2: Possible outcomes of a statistical hypothesis test based on p-value and level of significance α. Adapted from [9].

using the following formula:

$$\text{p-value} = \Pr(T \geq t_0).$$

We use this definition of p-value in cases where the observed data suggests that the test statistic could take on *larger* values than those corresponding to the distribution of the test statistic under the assumption that the null hypothesis H_0 is true.

2. If we have a left-tailed alternative hypothesis, we compute the p-value using the following formula:

$$\text{p-value} = \Pr(T \leq t_0).$$

We use this definition of p-value in cases where the observed data suggests that the test statistic could take on *smaller* values than those corresponding to the distribution of the test statistic under the assumption that the null hypothesis H_0 is true.

3. If we have a two-tailed alternative hypothesis, we calculate the p-value using the following formula:

$$\text{p-value} = 2 \cdot \min\{\Pr(T \leq t_0), \Pr(T \geq t_0)\}.$$

We use this definition of p-value in cases where the observed data suggests that the test statistic could take on *either larger or smaller* values than the values corresponding to the distribution of the test statistic under the assumption that the null hypothesis H_0 is true. In this case, we can use the above method of calculating the p-value only when the density of the distribution corresponding to the null hypothesis is symmetric.

The outcome of hypothesis testing depends on the chosen level of significance α. Given the calculated p-value, the decision regarding the rejection of the null hypothesis shall be made based on Table 3.2.

3.2 χ^2 goodness of fit test

Let us now describe one of statistical hypothesis tests that is commonly used when testing RBGs — the χ^2 goodness of fit test.

This test assesses the fit of the observed frequencies $X_1, \ldots, X_n$ of events $A_1, \ldots, A_n$ with corresponding expected frequencies $mp_1, \ldots, mp_n$, where the probabilities $p_1, \ldots, p_n$ are based on an assumption of a certain probability model and m is the size of the sample, i.e., the sum of observed frequencies $X_1 + \cdots + X_n$.

The null hypothesis states that the probabilities of events $A_1, \ldots, A_n$ are equal to $p_1, \ldots, p_n$ respectively. The test statistic takes the form:

$$X^2 = \sum_{i=1}^{n} \frac{(X_i - mp_i)^2}{mp_i}.$$

The random variable X^2 is approximately χ^2-distributed with $n-1$ degrees of freedom. The null hypothesis is rejected at the significance level α if $X^2 > \chi^2_{1-\alpha;n-1}$, where $\chi^2_{1-\alpha;n-1}$ is the $(1-\alpha)$-quantile of the χ^2 distribution with $n-1$ degrees of freedom. If this is the case, it means that the probabilities of the events are different from the probabilities $p_1, \ldots, p_n$. It should be noted that the χ^2 goodness of fit test is asymptotic and can only be used for a sufficiently large sample size m. It is often recommended in various literature that $mp_i \geq 5$ for every $i \in \{1, \ldots, n\}$.

Chapter 4

Known RBG tests

In this chapter, we will describe some known tests that assess the randomness of a given bit sequence. These tests will later be applied to the Dumbo instance of the Elephant cipher to test the randomness of its keystream. We will start with tests from the NIST STS, which are very well known and widely utilized. Then, we will follow up with monomial tests and finally conclude the chapter with cube testers.

4.1 NIST STS

The STS from NIST is a statistical test suite for random and pseudorandom bit generators for cryptographic applications consisting of 15 tests that were developed to test the randomness of arbitrarily long binary sequences produced by either hardware or software based cryptographic random or pseudorandom bit generators [10]. These tests focus on a variety of different types of non-randomness that could exist in a sequence and may be useful as a first step in determining whether or not a generator is suitable for a particular cryptographic application [10]. However, no set of statistical tests is able to prove that a given generator is truly random [10]. In other words, statistical testing cannot serve as a substitute for cryptanalysis [10]. In the rest of this section, we will explain 10 out of 15 tests from this suite. All of the descriptions in the rest of this section apart from examples are adapted from [10]. Note that the recommendations for input size will sometimes not be satisfied in our examples for the sake of simplicity. We will assume level of significance $\alpha = 0.01$ for all tests.

4.1.1 Notation

We denote the input bit sequence by $\epsilon_1, \ldots, \epsilon_n$ and its length by n. By `erfc`, we denote the ANSI C complementary error function contained in the *math.h*

header file and its corresponding mathematical library, defined as

$$\texttt{erfc}(z) = \frac{2}{\sqrt{\pi}} \int_z^\infty e^{-u^2} du.$$

By `igamc`, we denote the incomplete gamma function based on an approximation formula. Depending on the values of its parameters a and x, the incomplete gamma function may be approximated using either a continued fraction development or a series development.

Gamma function $\Gamma(z) = \int_0^\infty t^{z-1} e^{-t} dt$.

Incomplete gamma function $P(a,x) \equiv \frac{\gamma(a,x)}{\Gamma(a)} \equiv \frac{1}{\Gamma(a)} \int_0^x e^{-t} t^{a-1} dt$, where $P(a,0) = 0$ and $P(a,\infty) = 1$.

Incomplete gamma function $Q(a,x) \equiv 1 - P(a,x) \equiv \frac{\Gamma(a,x)}{\Gamma(a)}$ $\equiv \frac{1}{\Gamma(a)} \int_x^\infty e^{-t} t^{a-1} dt$, where $Q(a,0) = 1$ and $Q(a,\infty) = 0$.

By Φ, we denote the Standard Normal Cumulative Distribution Function (SNCDF) defined as

$$\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^z e^{-u^2/2} du.$$

4.1.2 Frequency (Monobit) test

4.1.2.1 Test purpose

This test focuses on the proportion of zeros and ones for the entire sequence. The purpose of this test is to determine whether the number of ones and zeros in a sequence is approximately the same as what would be expected for a truly random sequence. The test assesses how close the fraction of ones is to $\frac{1}{2}$, meaning the number of ones and zeros in the sequence should be about the same. All subsequent tests depend on the passing of this test.

4.1.2.2 Test description

1. Convert zeros and ones of the input sequence to values of -1 and $+1$ respectively and add them together to produce $S_n = X_1 + X_2 + \cdots + X_n$, where $X_i = 2\epsilon_i - 1$ for $i \in \{1, \ldots, n\}$.

2. Compute the test statistic $s_{obs} = \frac{|S_n|}{\sqrt{n}}$.

3. Compute p-value = $\texttt{erfc}(\frac{s_{obs}}{\sqrt{2}})$, where `erfc` is the complementary error function as defined in Section 4.1.1.

4.1.2.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.2.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 100 bits, i.e., $n \geq 100$.

4.1.2.5 Example

- Let the input sequence be

  $$\epsilon = 1101110000011100110101100111100011110000000101111010$$
  $$10101000000101010110111110000011010001100100010111000$$

  of length $n = 104$.

- Summing the transformed sequence $\{+1, -1\}^{104}$ gives us $S_n = -6$.

- We compute the test statistic

  $$s_{obs} = \frac{|S_n|}{\sqrt{n}} = \frac{-6}{\sqrt{104}} \approx 0.5883$$

  and the p-value

  $$\text{p-value} = \texttt{erfc}(\frac{s_{obs}}{\sqrt{2}}) = \texttt{erfc}(\frac{0.5883}{\sqrt{2}}) \approx 0.5563.$$

- Since the computed p-value ≈ 0.5563 is greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.3 Frequency test within a block

4.1.3.1 Test purpose

This test focuses on the proportion of zeros and ones within M-bit blocks. The purpose of this test is to determine whether the frequency of ones and zeros in an M-bit block is about the same as would be expected under the assumption of randomness. For block size $M = 1$, this test degenerates to the Frequency (Monobit) test described in Section 4.1.2.

4.1.3.2 Test description

1. Partition the input sequence into $N = \lfloor \frac{n}{M} \rfloor$ non-overlapping blocks and discard any unused bits.
2. Determine the proportion of ones π_i in each M-bit block using the formula $\pi_i = \frac{\sum_{j=1}^{M} \epsilon_{(i-1)M+j}}{M}$ for $i \in \{1, \ldots, N\}$.
3. Compute the test statistic $\chi^2(obs) = 4M \sum_{i=1}^{N} (\pi_i - \frac{1}{2})^2$.
4. Compute p-value = $\texttt{igamc}(N/2, \chi^2(obs)/2)$, where `igamc` is the incomplete gamma function for $Q(a, x)$ as defined in Section 4.1.1.

4.1.3.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.3.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 100 bits, i.e., $n \geq 100$. Note that $n \geq MN$. The block size M should be selected such that $M \geq 20$, $M > 0.01n$ and $N < 100$.

4.1.3.5 Example

- Let the input sequence be

  ϵ =1101110000011100110101100111100011110000000101111010
  10101000000101010110111110000011010001100100010111000

  of length $n = 104$ and block size $M = 20$.

- We partition the input sequence into $n = \lfloor \frac{n}{M} \rfloor = \lfloor \frac{104}{20} \rfloor = 5$ non-overlapping blocks. The last four unused bits are discarded.

- We determine the proportion of ones π_i in each M-bit block:

$$\pi_1 = \frac{\sum_{j=1}^{M} \epsilon_j}{M} = \frac{11}{20} = 0.55,$$
$$\pi_2 = \frac{\sum_{j=1}^{M} \epsilon_{M+j}}{M} = \frac{10}{20} = 0.5,$$
$$\pi_3 = \frac{\sum_{j=1}^{M} \epsilon_{2M+j}}{M} = \frac{9}{20} = 0.45,$$
$$\pi_4 = \frac{\sum_{j=1}^{M} \epsilon_{3M+j}}{M} = \frac{10}{20} = 0.5,$$
$$\pi_4 = \frac{\sum_{j=1}^{M} \epsilon_{4M+j}}{M} = \frac{8}{20} = 0.4.$$

- We compute the test statistic

$$\begin{aligned}\chi^2(obs) &= 4M \sum_{i=1}^{N} (\pi_i - \frac{1}{2})^2 \\ &= 4 \cdot 20 \cdot ((0.55 - 0.5)^2 + (0.5 - 0.5)^2 \\ &+ (0.45 - 0.5)^2 + (0.5 - 0.5)^2 + (0.4 - 0.5)^2) \\ &= 1.2\end{aligned}$$

  and the p-value

$$\text{p-value} = \texttt{igamc}(N/2, \chi^2(obs)/2) = \texttt{igamc}(5/2, 1.2/2) \approx 0.9449.$$

- Since the computed p-value ≈ 0.9449 is greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.4 Runs test

4.1.4.1 Test purpose

This test focuses on the total number of runs in the sequence, where a run is an uninterrupted sequence of identical bits. A run of length k consists of exactly k identical bits and is bounded before and after with a bit of the opposite value. The purpose of this test is to determine whether the number of runs of ones and zeros of various lengths is as expected for a random sequence. In particular, this test determines whether the oscillation between such zeros and ones is too fast or too slow.

4.1.4.2 Test description

1. Compute the pre-test proportion π of ones in the input sequence: $\pi = \frac{\sum_j \epsilon_j}{n}$.
2. Compute the test statistic $V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1$, where $r(k) = 0$ if $\epsilon_k = \epsilon_{k+1}$ and 1 otherwise.
3. Compute p-value $= \texttt{erfc}(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)})$, where `erfc` is the complementary error function as defined in Section 4.1.1.

4.1.4.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.4.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 100 bits, i.e., $n \geq 100$.

4.1.4.5 Example

- Let the input sequence be

$$\epsilon = 1101110000011100110101100111100011110000000101111010$$
$$10101000000101010110111110000011010001100100010111000$$

  of length $n = 104$.

- We compute the pre-test proportion of ones in the input sequence

$$\pi = \frac{\sum_j \epsilon_j}{n} = \frac{49}{104} \approx 0.4712.$$

- We compute the test statistic

$$V_n(obs) = \sum_{k=1}^{n-1} r(k) + 1 = 50$$

  and the p-value

$$\begin{aligned} \text{p-value} &= \texttt{erfc}\left(\frac{|V_n(obs) - 2n\pi(1-\pi)|}{2\sqrt{2n}\pi(1-\pi)}\right) \\ &= \texttt{erfc}\left(\frac{|50 - 2 \cdot 104 \cdot 0.4712(1-0.4712)|}{2\sqrt{2 \cdot 104} \cdot 0.4712(1-0.4712)}\right) \\ &\approx 0.7192. \end{aligned}$$

- Since the computed p-value ≈ 0.7192 is greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.5 Binary matrix rank test

4.1.5.1 Test purpose

This test focuses on the rank of disjoint sub-matrices of the entire sequence. The purpose of this test is to check for linear dependence among fixed-length substrings of the original sequence. Let $M = 32$ be the number of rows in each matrix and $Q = 32$ the number of columns in each matrix.

4.1.5.2 Test description

1. Sequentially divide the sequence into $(M \cdot Q)$-bit disjoint blocks; there will be $N = \lfloor \frac{n}{MQ} \rfloor$ such blocks. Discarded bits will be reported as not being used in the computation within each block. Collect the $M \cdot Q$ bit segments into M by Q matrices. Each row of the matrix is filled with successive Q-bit blocks of the original sequence ϵ.

2. Determine the binary rank R_ℓ of each matrix, where $\ell \in \{1, \ldots, N\}$. To determine the rank, apply elementary row operations where the addition operator is taken to be the XOR operation. The matrices are reduced to upper triangular form using forward row operations and the operation is repeated in reverse in order using backward row operations in order to arrive at a matrix in triangular form. The rank is then taken to be the number of nonzero rows in the resulting Gaussian-reduced matrix.

3. Let F_M be the number of matrices with $R_\ell = M$ (full rank), F_{M-1} the number of matrices with $R_\ell = M - 1$ (full rank $- 1$) and $N - F_M - F_{M-1}$ the number of remaining matrices.

4. Compute the test statistic $\chi^2(obs) = \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N} + \frac{(N - F_M - F_{M-1} - 0.1336N)^2}{0.1336N}$.

5. Compute p-value $= e^{-\chi^2(obs)/2}$.

4.1.5.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.5.4 Input size recommendation

The minimum number of bits to be tested must be such that $n \geq 38MQ$, i.e., at least 38 matrices are created. For $M = Q = 32$, each sequence to be tested should consist of a minimum of $38,912$ bits.

4.1.5.5 Example

- Let the input sequence be

  $$\epsilon = 1101110000011100110101100111100011110000000101111010$$
  $$10101000000101010110111110000011010001100100010111000$$

  of length $n = 104$ and $M = Q = 5$.

- We divide the sequence into $M \cdot Q = 25$-bit disjoint blocks; there are $N = \lfloor \frac{n}{MQ} \rfloor = \lfloor \frac{104}{25} \rfloor = 4$ such blocks. The last four unused bits are discarded.

- We collect the 25-bit segments into four 5×5 matrices

$$\mathcal{M}_1 = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \end{pmatrix}, \mathcal{M}_2 = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 \end{pmatrix},$$
$$\mathcal{M}_3 = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 \end{pmatrix}, \mathcal{M}_4 = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{pmatrix}.$$

- We determine the rank R_ℓ of each matrix:

$$R_1 = 5,\ R_2 = 4,\ R_3 = 3,\ R_4 = 4.$$

- We count the number of matrices with $R_\ell = M$ (full rank) $F_M = 1$, the number of matrices with $R_\ell = M - 1$ (full rank $-$ 1) $F_{M-1} = 2$ and the number of remaining matrices $N - F_M - F_{M-1} = 4 - 2 - 1 = 1$.

- We compute the test statistic

$$\begin{aligned} \chi^2(obs) &= \frac{(F_M - 0.2888N)^2}{0.2888N} + \frac{(F_{M-1} - 0.5776N)^2}{0.5776N} \\ &+ \frac{(N - F_M - F_{M-1} - 0.1336N)^2}{0.1336N} \\ &= \frac{(1 - 0.2888 \cdot 4)^2}{0.2888 \cdot 4} + \frac{(2 - 0.5776 \cdot 4)^2}{0.5776 \cdot 4} \\ &+ \frac{(1 - 0.1336 \cdot 4)^2}{0.1336 \cdot 4} \\ &\approx 0.5771 \end{aligned}$$

  and the p-value

$$\text{p-value} = e^{-\chi^2(obs)/2} = e^{-0.5771/2} \approx 0.7494.$$

- Since the computed p-value ≈ 0.7494 is greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.6 Discrete Fourier transform (Spectral) test

4.1.6.1 Test purpose

This test focuses on the peak heights in the Discrete Fourier Transform of the sequence. The purpose of this test is to detect periodic features (i.e., repetitive

patterns that are near each other) in the tested sequence that would indicate a deviation from the assumption of randomness. The intention is to detect whether the number of peaks exceeding the 95 % threshold is significantly different than 5 %.

4.1.6.2 Test description

1. The zeros and ones of the input sequence are converted to values of −1 and +1 respectively to create the sequence $X = x_1, x_2, \ldots, x_n$, where $x_i = 2\epsilon_i - 1$ for $i \in \{1, \ldots, n\}$.

2. Apply the Discrete Fourier Transform (DFT) on X and store the result in S. A sequence of complex variables is produced which represents periodic components of the sequence of bits at different frequencies.

3. Calculate $M = \texttt{modulus}(S') \equiv |S'|$, where S' is the substring consisting of the first $\frac{n}{2}$ elements in S and the `modulus` function produces a sequence of peak heights.

4. Compute the 95 % peak height threshold value $T = \sqrt{(\log \frac{1}{0.05})n}$. Under the assumption of randomness, 95 % of the values obtained from the test should not exceed T.

5. Compute $N_0 = \frac{0.95n}{2}$. N_0 is the expected theoretical (95 %) number of peaks that are less than T under the assumption of randomness.

6. Compute N_1 — the actual observed number of peaks in M that are less than T.

7. Compute $d = \frac{(N_1 - N_0)}{\sqrt{0.95 \cdot 0.05 \cdot n/4}}$.

8. Compute p-value = $\texttt{erfc}(\frac{|d|}{\sqrt{2}})$, where `erfc` is the complementary error function as defined in Section 4.1.1.

4.1.6.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.6.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 1000 bits, i.e., $n \geq 1000$.

4.1.6.5 Example

- Let the input sequence be

$$\epsilon = 11011100000111001101011001111000111100000010111101010100000010101011011111000001101000110010001011000$$

of length $n = 104$.

- We produce a transformed sequence X by converting the zeros and ones of ϵ to values of -1 and $+1$ respectively and apply DFT on X to produce S.

- We store the first $\frac{n}{2}$ elements of S in S' and calculate the sequence of peak heights $M = \texttt{modulus}(S')$.

- We compute

$$T = \sqrt{(\log \frac{1}{0.05})n} = \sqrt{(\log \frac{1}{0.05}) \cdot 104} \approx 17.65,$$
$$N_0 = \frac{0.95n}{2} = \frac{0.95 \cdot 104}{2} = 49.4.$$

- We count the number of peaks in M that are less than $T \approx 17.65$ and obtain $N_1 = 49$.

- We compute the test statistic

$$d = \frac{(N_1 - N_0)}{\sqrt{0.95 \cdot 0.05 \cdot n/4}} = \frac{(49 - 49.4)}{\sqrt{0.95 \cdot 0.05 \cdot 104/4}} \approx -0.3599.$$

- We compute the p-value

$$\text{p-value} = \texttt{erfc}(\frac{|d|}{\sqrt{2}}) = \texttt{erfc}(\frac{|-0.3599|}{\sqrt{2}}) \approx 0.7189.$$

- Since the computed p-value ≈ 0.7189 is greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.7 Maurer's "universal statistical" test

4.1.7.1 Test purpose

This test focuses on the number of bits between matching patterns — a measure that is related to the length of a compressed sequence. The purpose of this test is to detect whether or not the sequence can be significantly compressed without loss of information. A significantly compressible sequence is considered to be non-random.

4.1.7.2 Test description

1. Partition the n-bit input sequence into two segments: an initialization segment consisting of Q L-bit non-overlapping blocks and a test segment consisting of K L-bit non-overlapping blocks. Bits remaining at the end of the sequence that do not form a complete L-bit block are discarded. The first Q blocks are used to initialize the test. The remaining $K = \lfloor \frac{n}{L} \rfloor - Q$ blocks are the test blocks. See Figure 4.1 for a depiction of this step.

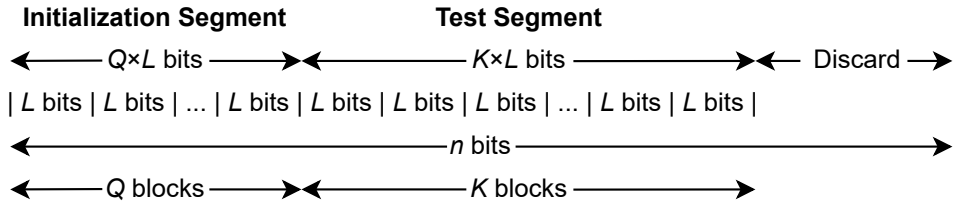

Figure 4.1: Depiction of the first step of Maurer's "universal statistical" test. Adapted from [10].

2. Using the initialization segment, a table is created for each possible L-bit value (i.e., the L-bit value is used as an index into the table). The block number of the last occurrence of each L-bit block is noted in the table as $T_j = i$ for $i \in \{1, \ldots, Q\}$, where j is the decimal representation of the contents of the ith L-bit block.

3. Examine each of the K blocks in the test segment and determine the number of blocks since the last occurrence of the same L-bit block (i.e., $i - T_j$). Replace the value in the table with the location of the current block (i.e., $T_j = i$). Add the calculated distance between reoccurrences of the same L-bit block to an accumulating $\log_2$ sum of all the differences detected in the K blocks (i.e., $sum = sum + \log_2(i - T_j)$).

4. Compute the test statistic $f_n = \frac{1}{K} \sum_{i=Q+1}^{Q+K} \log_2(i - T_j)$, where T_j is the table entry corresponding to the decimal representation of the contents of the ith L-bit block.

5. Compute p-value = $\texttt{erfc}(|\frac{f_n - \texttt{expectedValue}(L)}{\sqrt{2}\sigma}|)$, where `erfc` is the complementary error function as defined in Section 4.1.1. Values for the function $\texttt{expectedValue}(L)$ are taken from Table 4.1 of precomputed values. Under the assumption of randomness, $\texttt{expectedValue}(L)$, i.e., the sample mean, is the theoretical expected value of the computed

statistic for the given L-bit length. The theoretical standard deviation is given by $\sigma = c\sqrt{\frac{\texttt{variance}(L)}{K}}$, where $c = 0.7 - \frac{0.8}{L} + (4 + \frac{32}{L})\frac{K^{-3/L}}{15}$.

L	expectedValue	variance
6	5.2177052	2.954
7	6.1962507	3.125
8	7.1836656	3.238
9	8.1764248	3.311
10	9.1723243	3.356
11	10.170032	3.384
12	11.168765	3.401
13	12.168070	3.410
14	13.167693	3.416
15	14.167488	3.419
16	15.167379	3.421

Table 4.1: Table of precomputed values for `expectedValue`(L). Adapted from [2].

4.1.7.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.7.4 Input size recommendation

This test requires a long sequence of bits ($n \geq (Q + K)L$) which are divided into two segments of L-bit blocks, where L should be chosen so that $6 \leq L \leq 16$. The first segment consists of Q initialization blocks, where Q should be chosen so that $Q = 10 \cdot 2^L$. The second segment consists of K test blocks, where $K = \lceil \frac{n}{L} \rceil - Q \approx 1000 \cdot 2^L$. The values of L, Q and n should be chosen as indicated in Table 4.2.

4.1.7.5 Example

- Let the input sequence be $\epsilon = \{1111000000\}^{38784}$ of length $n = 387840$, block size $L = 6$, length of the initialization segment $Q = 640$ and length of the test segment $K = 64000$.

n	L	$Q = 10 \cdot 2^L$
≥ 387,840	6	640
≥ 904,960	7	1,280
≥ 2,068,480	8	2,560
≥ 4,654,080	9	5,120
≥ 10,342,400	10	10,240
≥ 22,753,280	11	20,480
≥ 49,643,520	12	40,960
≥ 107,560,960	13	81,920
≥ 231,669,760	14	163,840
≥ 496,435,200	15	327,680
≥ 1,059,061,760	16	655,360

Table 4.2: Input size recommendation for Maurer's "universal statistical" test. Adapted from [10].

Block	Last occurrence
`000000`	640
`000011`	637
`001111`	639
`110000`	638
`111100`	636

Table 4.3: Last occurrences of 6-bit blocks in the initialization segment of our example for the Maurer's "universal statistical" test. Only non-zero values are shown.

- Using the initialization segment, we create a table mapping the block number of the last occurrence of each possible 6-bit block. The table is shown in Table 4.3.

- We examine each of the $K = 64000$ blocks in the test segment and compute a $\log_2$ sum of distances between reoccurrences of the same 6-bit blocks. We obtain $sum \approx 148603.4$.

- We compute the test statistic

$$f_n = \frac{1}{K}\sum_{i=Q+1}^{Q+K}\log_2(i - T_j) = \frac{1}{64000}\sum_{i=640+1}^{640+64000}\log_2(i - T_j) \approx 2.3219.$$

- We compute the p-value

$$\text{p-value} = \texttt{erfc}(|\frac{f_n - \texttt{expectedValue}(L)}{\sqrt{2}\sigma}|) = \texttt{erfc}(|\frac{2.3219 - 5.2177}{\sqrt{2}\cdot 0.0039}|) \approx 0.$$

- Since the computed p-value ≈ 0 is less than the level of significance $\alpha = 0.01$, we conclude that the sequence is non-random.

4.1.8 Linear complexity test

4.1.8.1 Test purpose

This test focuses on the length of an LFSR. The purpose of this test is to determine whether or not the input sequence is complex enough to be considered random. Random sequences are characterized by longer LFSRs. An LFSR that is too short implies non-randomness.

4.1.8.2 Test description

1. Partition the n-bit input sequence into N independent blocks of M bits, where $n = MN$.
2. Using the Berlekamp-Massey algorithm from [2], determine the linear complexity L_i of each of the N blocks ($i \in \{1, \ldots, N\}$). L_i is the length of the shortest LFSR sequence that generates all bits in the block i. Within any L_i-bit sequence, some combination of the bits, when added together modulo 2, produces the next bit in the sequence (bit $L_i + 1$).
3. Under the assumption of randomness, calculate the theoretical mean

$$\mu = \frac{M}{2} + \frac{9 + (-1)^{M+1}}{36} - \frac{\frac{M}{3} + \frac{2}{9}}{2^M}.$$

4. For each substring, calculate $T_i = (-1)^M(L_i - \mu) + \frac{2}{9}$.
5. Let $v_0 = v_1 = \cdots = v_6 = 0$ and record the T_i values in $v_0, \ldots, v_6$ as follows:
   - If $T_i \leq -2.5$, increment v_0 by one.
   - If $-2.5 < T_i \leq -1.5$, increment v_1 by one.
   - If $-1.5 < T_i \leq -0.5$, increment v_2 by one.
   - If $-0.5 < T_i \leq 0.5$, increment v_3 by one.

- If $0.5 < T_i \leq 1.5$, increment v_4 by one.
- If $1.5 < T_i \leq 2.5$, increment v_5 by one.
- If $T_i > 2.5$, increment v_6 by one.

6. Compute the test statistic $\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i}$, where $\pi_0 = 0.010417$, $\pi_1 = 0.03125$, $\pi_2 = 0.125$, $\pi_3 = 0.5$, $\pi_4 = 0.25$, $\pi_5 = 0.0625$ and $\pi_6 = 0.020833$ are precomputed probabilities [2].

7. Compute p-value = $\texttt{igamc}(\frac{K}{2}, \frac{\chi^2(obs)}{2})$, where `igamc` is the incomplete gamma function for $Q(a, x)$ as defined in Section 4.1.1.

4.1.8.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.8.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits, i.e., $n \geq 10^6$. The value of M must be in the range $500 \leq M \leq 5000$ and N must satisfy $N \geq 200$ for the χ^2 result to be valid.

4.1.8.5 Example

- Let the input sequence be $\epsilon = \{1010011010\}^{100000}$ of length $n = 10^6$ and $M = 500$. The input sequence is partitioned into $N = 2000$ blocks.
- We determine the linear complexity L_i of each of the N blocks using the Berlekamp-Massey algorithm.
- We calculate the theoretical mean

$$\begin{aligned}\mu &= \frac{M}{2} + \frac{9 + (-1)^{M+1}}{36} - \frac{\frac{M}{3} + \frac{2}{9}}{2^M} \\ &= \frac{500}{2} + \frac{9 + (-1)^{500+1}}{36} - \frac{\frac{500}{3} + \frac{2}{9}}{2^{500}} \\ &\approx 250.28.\end{aligned}$$

- For each substring, we calculate

$$T_i = (-1)^M (L_i - \mu) + \frac{2}{9} = (-1)^{500}(L_i - 250.28) + \frac{2}{9}.$$

- We collect $v_0, \ldots, v_6$ by processing values of T_i and obtain

$$v_0 = 2000,\ v_1 = v_2 = \cdots = v_6 = 0.$$

[2] Refer to [10] for more details about how these probabilities were computed.

- We compute the test statistic

$$\chi^2(obs) = \sum_{i=0}^{K} \frac{(v_i - N\pi_i)^2}{N\pi_i} = \sum_{i=0}^{6} \frac{(v_i - 2000 \cdot \pi_i)^2}{2000 \cdot \pi_i} \approx 189022.07.$$

- We compute the p-value

$$\text{p-value} = \texttt{igamc}(\frac{K}{2}, \frac{\chi^2(obs)}{2}) = \texttt{igamc}(\frac{6}{2}, \frac{189022.07}{2}) \approx 0.$$

- Since the computed p-value ≈ 0 is less than the level of significance $\alpha = 0.01$, we conclude that the sequence is non-random.

4.1.9 Serial test

4.1.9.1 Test purpose

This test focuses on the frequency of all possible overlapping m-bit patterns across the entire sequence. The purpose of this test is to determine whether the number of occurrences of the 2^m m-bit overlapping patterns is approximately the same as what would be expected for a random sequence. Random sequences have uniformity, i.e., every m-bit pattern has the same probability of appearing as every other m-bit pattern. Note that for $m = 1$, the Serial test is equivalent to the Frequency (Monobit) test described in Section 4.1.2.

4.1.9.2 Test description

1. Extend the input sequence by appending the first $m-1$ bits to the end of it to form an augmented sequence ϵ'.

2. Determine the frequency of all possible overlapping m-bit blocks, all possible overlapping $(m-1)$-bit blocks and all possible overlapping $(m-2)$-bit blocks. Let $v_{i_1 \ldots i_m}$ denote the frequency of the m-bit pattern $i_1 \ldots i_m$, $v_{i_1 \ldots i_{m-1}}$ denote the frequency of the $(m-1)$-bit pattern $i_1 \ldots i_{m-1}$ and $v_{i_1 \ldots i_{m-2}}$ denote the frequency of the $(m-2)$-bit pattern $i_1 \ldots i_{m-2}$.

3. Compute

   - $\psi_m^2 = \frac{2^m}{n} \sum_{i_1 \ldots i_m} (v_{i_1 \ldots i_m} - \frac{n}{2^m})^2 = \frac{2^m}{n} \sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2 - n,$
   - $\psi_{m-1}^2 = \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} (v_{i_1 \ldots i_{m-1}} - \frac{n}{2^{m-1}})^2$
     $= \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} v_{i_1 \ldots i_{m-1}}^2 - n,$
   - $\psi_{m-2}^2 = \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} (v_{i_1 \ldots i_{m-2}} - \frac{n}{2^{m-2}})^2$
     $= \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} v_{i_1 \ldots i_{m-2}}^2 - n.$

4. Compute

- $\nabla\psi_m^2 = \psi_m^2 - \psi_{m-1}^2$,
- $\nabla^2\psi_m^2 = \psi_m^2 - 2\psi_{m-1}^2 + \psi_{m-2}^2$.

5. Compute

   - p-value1 = $\texttt{igamc}(2^{m-2}, \nabla\psi_m^2)$,
   - p-value2 = $\texttt{igamc}(2^{m-3}, \nabla^2\psi_m^2)$,

   where `igamc` is the incomplete gamma function for $Q(a, x)$ as defined in Section 4.1.1.

4.1.9.3 Decision rule

If the p-values obtained in step five of Section 4.1.9.2 are both larger or equal to 0.01, conclude that the sequence is random. Otherwise, conclude that the sequence is non-random.

4.1.9.4 Input size recommendation

It is recommended to choose m and n such that $m < \lfloor \log_2 n \rfloor - 2$.

4.1.9.5 Example

- Let the input sequence be

  ϵ =1101110000011100110101100111100011110000000101111010
  10101000000101010110111110000011010001100100010110000

  of length $n = 104$ and block size $m = 4$. The input sequence is extended by appending the first $m - 1 = 3$ bits (110) to the end of it to form an augmented sequence ϵ'.

- We determine the frequency of all possible overlapping 4-bit blocks, all possible overlapping 3-bit blocks and all possible overlapping 2-bit

blocks:

$v_{0000} = 11$, $v_{000} = 19$, $v_{00} = 30$,
$v_{0001} = 8$, $v_{001} = 11$, $v_{01} = 25$,
$v_{0010} = 4$, $v_{010} = 12$, $v_{10} = 25$,
$v_{0011} = 7$, $v_{011} = 13$, $v_{11} = 24$,
$v_{0100} = 3$, $v_{100} = 11$,
$v_{0101} = 9$, $v_{101} = 14$,
$v_{0110} = 7$, $v_{110} = 13$,
$v_{0111} = 6$, $v_{111} = 11$,
$v_{1000} = 8$,
$v_{1001} = 3$,
$v_{1010} = 8$,
$v_{1011} = 6$,
$v_{1100} = 8$,
$v_{1101} = 5$,
$v_{1110} = 6$,
$v_{1111} = 5$.

- We compute

$$\begin{aligned}\psi_m^2 &= \frac{2^m}{n} \sum_{i_1 \ldots i_m} \left(v_{i_1 \ldots i_m} - \frac{n}{2^m}\right)^2 \\ &= \frac{2^m}{n} \sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2 - n \\ &= \frac{2^4}{104} \sum_{i_1 \ldots i_m} v_{i_1 \ldots i_m}^2 - 104 \\ &\approx 11.08,\end{aligned}$$

$$\begin{aligned}\psi_{m-1}^2 &= \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} \left(v_{i_1 \ldots i_{m-1}} - \frac{n}{2^{m-1}}\right)^2 \\ &= \frac{2^{m-1}}{n} \sum_{i_1 \ldots i_{m-1}} v_{i_1 \ldots i_{m-1}}^2 - n \\ &= \frac{2^3}{104} \sum_{i_1 \ldots i_{m-1}} v_{i_1 \ldots i_{m-1}}^2 - 104 \\ &\approx 3.85,\end{aligned}$$

$$\begin{aligned}\psi^2_{m-2} &= \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} \left(v_{i_1 \ldots i_{m-2}} - \frac{n}{2^{m-2}}\right)^2 \\ &= \frac{2^{m-2}}{n} \sum_{i_1 \ldots i_{m-2}} v^2_{i_1 \ldots i_{m-2}} - n \\ &= \frac{2^2}{104} \sum_{i_1 \ldots i_{m-2}} v^2_{i_1 \ldots i_{m-2}} - 104 \\ &\approx 0.85.\end{aligned}$$

- We compute the test statistics

$$\begin{aligned}\nabla\psi^2_m &= \psi^2_m - \psi^2_{m-1} = 11.08 - 3.85 \approx 7.23, \\ \nabla^2\psi^2_m &= \psi^2_m - 2\psi^2_{m-1} + \psi^2_{m-2} = 11.08 - 2 \cdot 3.85 + 0.85 \approx 4.23.\end{aligned}$$

- We compute the p-values

$$\begin{aligned}\text{p-value1} &= \texttt{igamc}(2^{m-2}, \nabla\psi^2_m) = \texttt{igamc}(2^2, 7.23) \approx 0.5120, \\ \text{p-value2} &= \texttt{igamc}(2^{m-3}, \nabla^2\psi^2_m) = \texttt{igamc}(2^1, 4.23) \approx 0.3757.\end{aligned}$$

- Since the computed p-values 0.5120 and 0.3757 are both greater than the level of significance $\alpha = 0.01$, we conclude that the sequence is random.

4.1.10 Cumulative sums (Cumsum) test

4.1.10.1 Test purpose

This test focuses on the maximal excursion from zero of the random walk defined by the cumulative sum of adjusted (-1, +1) digits in the sequence. The purpose of the test is to determine whether the cumulative sum of partial sequences occurring in the tested sequence is too large or too small relative to the expected behavior of that cumulative sum for random sequences. This cumulative sum may be considered as a random walk. For a random sequence, the excursions of the random walk should be near zero. For certain types of non-random sequences, the excursions of this random walk from zero will be large.

4.1.10.2 Test description

1. Convert the zeros and ones of the input sequence to values X_i of -1 and $+1$ using the formula $X_i = 2\epsilon_i - 1$ to form a normalized sequence.

2. Compute partial sums S_i of successively larger subsequences, each starting with X_1 (if $mode = 0$) or X_n (if $mode = 1$). In other words,

$S_k = S_{k-1} + X_k$ for $mode = 0$ and $S_k = S_{k-1} + X_{n-k+1}$ for $mode = 1$. See Table 4.4 for an example of this computation.

mode = 0 (forward)	**mode = 1 (backward)**
$S_1 = X_1$	$S_1 = X_n$
$S_2 = X_1 + X_2$	$S_2 = X_n + X_{n-1}$
$S_3 = X_1 + X_2 + X_3$	$S_3 = X_n + X_{n-1} + X_{n-2}$
.	.
.	.
$S_k = X_1 + X_2 + \cdots + X_k$	$S_k = X_n + X_{n-1} + \cdots + X_{n-k+1}$
.	.
.	.
$S_n = X_1 + X_2 + \cdots + X_k + \cdots + X_n$	$S_n = X_n + X_{n-1} + \cdots + X_{k-1} + \cdots + X_1$

Table 4.4: Example of the computation of partial sums S_i in forward and backward mode. Adapted from [10].

3. Compute the test statistic $z = \max_{1 \le k \le n} |S_k|$, where $\max_{1 \le k \le n} |S_k|$ is the largest of the absolute values of the partial sums S_k.

4. Compute p-value $= 1 - \sum_{k=(\frac{-n}{z}+1)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+1)z}{\sqrt{n}}) - \Phi(\frac{(4k-1)z}{\sqrt{n}})]$ $+ \sum_{k=(\frac{-n}{z}-3)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+3)z}{\sqrt{n}}) - \Phi(\frac{(4k+1)z}{\sqrt{n}})]$, where Φ is the standard normal cumulative distribution function defined in Section 4.1.1.

4.1.10.3 Decision rule

If the computed p-value is less than 0.01, conclude that the sequence is non-random. Otherwise, conclude that the sequence is random.

4.1.10.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 100 bits, i.e., $n \ge 100$.

4.1.10.5 Example

- Let the input sequence be

  ϵ =1101110000011100110101100111100011110000000101111010
  10101000000101010110111110000011010001100100011011000

  of length $n = 104$.

- We transform the input sequence to values of −1 and +1 using the formula $X_i = 2\epsilon_i - 1$ to form a normalized sequence.

- We compute partial sums S_i of successively larger subsequences, each starting with X_1 (forward / mode 0) or X_n (backward / mode 1).

- We compute the test statistic

$$z = \max_{1\le k\le n} |S_k| = \max_{1\le k\le n} |S_k| = 6 \text{ (forward / mode 0)},$$
$$z = \max_{1\le k\le n} |S_k| = \max_{1\le k\le n} |S_k| = 12 \text{ (backward / mode 1)}.$$

- We compute the p-value

$$\begin{aligned}
\text{p-value} &= 1 - \sum_{k=(\frac{-n}{z}+1)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+1)z}{\sqrt{n}}) - \Phi(\frac{(4k-1)z}{\sqrt{n}})] \\
&\quad + \sum_{k=(\frac{-n}{z}-3)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+3)z}{\sqrt{n}}) - \Phi(\frac{(4k+1)z}{\sqrt{n}})] \\
&= 1 - \sum_{k=(\frac{-104}{6}+1)/4}^{(\frac{104}{6}-1)/4} [\Phi(\frac{(4k+1)\cdot 6}{\sqrt{104}}) - \Phi(\frac{(4k-1)\cdot 6}{\sqrt{104}})] \\
&\quad + \sum_{k=(\frac{-104}{6}-3)/4}^{(\frac{104}{6}-1)/4} [\Phi(\frac{(4k+3)\cdot 6}{\sqrt{104}}) - \Phi(\frac{(4k+1)\cdot 6}{\sqrt{104}})] \\
&\approx 0.9639 \text{(forward / mode 0)},
\end{aligned}$$

$$\begin{aligned}
\text{p-value} &= 1 - \sum_{k=(\frac{-n}{z}+1)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+1)z}{\sqrt{n}}) - \Phi(\frac{(4k-1)z}{\sqrt{n}})] \\
&\quad + \sum_{k=(\frac{-n}{z}-3)/4}^{(\frac{n}{z}-1)/4} [\Phi(\frac{(4k+3)z}{\sqrt{n}}) - \Phi(\frac{(4k+1)z}{\sqrt{n}})] \\
&= 1 - \sum_{k=(\frac{-104}{12}+1)/4}^{(\frac{104}{12}-1)/4} [\Phi(\frac{(4k+1)\cdot 12}{\sqrt{104}}) - \Phi(\frac{(4k-1)\cdot 12}{\sqrt{104}})] \\
&\quad + \sum_{k=(\frac{-104}{12}-3)/4}^{(\frac{104}{12}-1)/4} [\Phi(\frac{(4k+3)\cdot 12}{\sqrt{104}}) - \Phi(\frac{(4k+1)\cdot 12}{\sqrt{104}})] \\
&\approx 0.4778 \text{(backward / mode 1)}.
\end{aligned}$$

- Since the computed p-values 0.9639 (forward / mode 0) and 0.4778 (backward / mode 1) are both greater than the level of significance $\alpha = 0.01$, we would conclude in both cases (modes) that the sequence is random.

4.1.11 Random excursions test

4.1.11.1 Test purpose

This test focuses on the number of cycles having exactly K visits in a cumulative sum random walk. The cumulative sum random walk is derived from partial sums after the input sequence of zeros and ones is transformed to the appropriate sequence of -1 and +1 values. A cycle of a random walk consists of a sequence of steps of unit length taken at random that begin at and return to the origin. The purpose of this test is to determine if the number of visits to a particular state within a cycle deviates from what one would expect for a random sequence. This test is actually a series of eight tests (and thus results in eight conclusions) — one test for each of the states: -4, -3, -2, -1 and +1, +2, +3, +4.

4.1.11.2 Test description

1. Form a normalized sequence X, where the zeros and ones of the input sequence are transformed into values of -1 and +1 using the formula $X_i = 2\epsilon_i - 1$.

2. Compute the partial sums S_i of successively larger subsequences, each starting with X_1. Form a set $S = \{S_i \mid i \in \{1, \ldots, n\}\}$.

   $S_1 = X_1$

   $S_2 = X_1 + X_2$

   $S_3 = X_1 + X_2 + X_3$

   .

   .

   $S_k = X_1 + X_2 + X_3 + \cdots + X_k$

   .

   .

   $S_n = X_1 + X_2 + X_3 + \cdots + X_k + \cdots + X_n$

3. Form a new sequence S' by attaching zeros before and after the set S. That is, $S' = 0, S_1, S_2, \ldots, S_n, 0$.

4. Let J be the total number of zero crossings in S', where a zero crossing is a value of zero in S' that occurs after the starting zero. J is also the number of cycles in S', where a cycle in S' is a subsequence of S' consisting of an occurrence of zero, followed by non-zero values and ending with another zero. The ending zero in one cycle may be the beginning zero in another cycle. If $J < 500$, discontinue the test [3].

5. For each cycle and for each non-zero state value x having values $-4 \leq x \leq -1$ and $1 \leq x \leq 4$, compute the frequency of each x within each cycle.

6. For each of the eight states of x, compute $\upsilon_k(x)$, the total number of cycles in which state x occurs exactly k times among all cycles, for $k \in \{0, 1, \ldots, 5\}$. Store all frequencies ≥ 5 in $\upsilon_5(x)$. Note that $\sum_{k=0}^{5} \upsilon_k(x) = J$.

7. For each of the eight states of x, compute the test statistic $\chi^2(obs) = \sum_{k=0}^{5} \frac{(\upsilon_k(x) - J\pi_k(x))^2}{J\pi_k(x)}$, where $\pi_k(x)$ is the probability that the state x occurs k times in a random distribution. See Table 4.5 for precomputed π_k values [4].

	$\pi_0(x)$	$\pi_1(x)$	$\pi_2(x)$	$\pi_3(x)$	$\pi_4(x)$	$\pi_5(x)$
x = 1	0.5000	0.2500	0.1250	0.0625	0.0312	0.0312
x = 2	0.7500	0.0625	0.0469	0.0352	0.0264	0.0791
x = 3	0.8333	0.0278	0.0231	0.0193	0.0161	0.0804
x = 4	0.8750	0.0156	0.0137	0.0120	0.0105	0.0733
x = 5	0.9000	0.0100	0.0090	0.0081	0.0073	0.0656
x = 6	0.9167	0.0069	0.0064	0.0058	0.0053	0.0588
x = 7	0.9286	0.0051	0.0047	0.0044	0.0041	0.0531

Table 4.5: Precomputed π_k values. Adapted from [10].

8. For each state of x, compute p-value = `igamc`$(\frac{5}{2}, \frac{\chi^2(obs)}{2})$, where `igamc` is the incomplete gamma function for $Q(a, x)$ as defined in Section 4.1.1. Eight p-values will be produced.

[3] This is because the empirical rule for χ^2 computations would not be satisfied. Refer to [10] for details.

[4] Refer to [10] for details about the calculation method of $\pi_k(x)$.

4.1.11.3 Decision rule

If the p-values obtained in step eight of Section 4.1.11.2 are all larger or equal to 0.01, then conclude that the sequence is random. Otherwise, conclude that the sequence is non-random.

4.1.11.4 Input size recommendation

It is recommended that each sequence to be tested consist of a minimum of 1,000,000 bits, i.e., $n \geq 10^6$.

4.1.11.5 Example

- Let the input sequence be $\epsilon = \{1010011010\}^{100000}$ of length $n = 10^6$.
- We transform the input sequence to values of -1 and $+1$ using the formula $X_i = 2\epsilon_i - 1$.
- We compute partial sums S_i of successively larger subsequences, each starting with X_1.
- We form a new sequence $S' = 0, S_1, S_2, \ldots, S_n, 0$.
- We determine the number of zero crossings in S' and obtain $J = 500000$.
- We compute the frequency of each $-4 \leq x \leq -1$ and $1 \leq x \leq 4$ within each cycle.
- We compute $v_k(x)$ for each $-4 \leq x \leq -1$, $1 \leq x \leq 4$ and $k \in \{0, 1, \ldots, 5\}$:

$$
\begin{array}{llll}
v_0(-4) = 500000, & v_0(-3) = 500000, & v_0(-2) = 500000, & v_0(-1) = 400000, \\
v_0(1) = 100000, & v_0(2) = 500000, & v_0(3) = 500000, & v_0(4) = 500000, \\
v_1(-4) = 0, & v_1(-3) = 0, & v_1(-2) = 0, & v_1(-1) = 100000, \\
v_1(1) = 400000, & v_1(2) = 0, & v_1(3) = 0, & v_1(4) = 0, \\
v_2(-4) = 0, & v_2(-3) = 0, & v_2(-2) = 0, & v_2(-1) = 0, \\
v_2(1) = 0, & v_2(2) = 0, & v_2(3) = 0, & v_2(4) = 0, \\
v_3(-4) = 0, & v_3(-3) = 0, & v_3(-2) = 0, & v_3(-1) = 0, \\
v_3(1) = 0, & v_3(2) = 0, & v_3(3) = 0, & v_3(4) = 0, \\
v_4(-4) = 0, & v_4(-3) = 0, & v_4(-2) = 0, & v_4(-1) = 0, \\
v_4(1) = 0, & v_4(2) = 0, & v_4(3) = 0, & v_4(4) = 0, \\
v_5(-4) = 0, & v_5(-3) = 0, & v_5(-2) = 0, & v_5(-1) = 0, \\
v_5(1) = 0, & v_5(2) = 0, & v_5(3) = 0, & v_5(4) = 0.
\end{array}
$$

- We compute the test statistic and p-value for each $-4 \leq x \leq -1$ and $1 \leq x \leq 4$. We obtain the results shown in Table 4.6.

State	χ^2(obs)	p-value
-4	71428.57	0
-3	100000.00	0
-2	166666.67	0
-1	220000.00	0
1	820000.00	0
2	166666.67	0
3	100000.00	0
4	71428.57	0

Table 4.6: Results for our example of the random excursions test

- Since all of the computed p-values are zero, which is less than the level of significance $\alpha = 0.01$, we conclude that the sequence is non-random.

4.2 Monomial tests

In this section, we will explore monomial tests described in [11]. The original concept was introduced by E. Filiol in 2002 as "Möbius tests" [12]. Let us start off with mathematical preliminaries adapted from [11] that are necessary to understand the tests.

4.2.1 Preliminaries

A boolean function f of n variables is simply a mapping $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2$. There are exactly 2^{2^n} distinct boolean functions of n variables, each uniquely defined by its truth table.

Definition 8 (Algebraic normal form of a boolean function)
A function $\hat{f} : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2$ *satisfying*

$$\hat{f}(x) = \sum_{a \in \mathbb{Z}_2^n} f(a) \prod_{i=1}^{n} x_i^{a_i}$$

is an algebraic normal form representation of a boolean function $f : \mathbb{Z}_2^n \mapsto \mathbb{Z}_2$.

Let us now briefly go over some of the important properties of the Algebraic Normal Form (ANF).

(P1) A unique $\hat{f}$ exists for all boolean functions f.

(P2) The ANF transform is its own inverse, i.e., an involution. Therefore, if $g = \hat{f}$, then $\hat{g} = f$.

(P3) We define a *partial order* for vectors x as $x \preceq y$ if $x_i \leq y_i$ for all i. Using this partial order, Definition 8 can be written as $\hat{f} = \sum_{a \preceq x} f(a)$.

(P4) The *Hamming distance* $\mathtt{d}(x, y)$ between x and y is the number of positions where $x_i \neq y_i$.

(P5) A norm $\mathtt{wt}(x) = \mathtt{d}(0, x)$ called the *Hamming weight* is equivalent to the number of positions in x where $x_i = 1$.

(P6) The *algebraic degree* $\mathtt{deg}(f)$ is the maximum Hamming weight x that satisfies $\hat{f}(x) = 1$. This is equivalent to the length of the longest monomial (most variables) in the polynomial representation of f.

We will also introduce an algorithm for computing the ANF of a boolean function. Let $z : \mathbb{Z}_2^n \mapsto \mathbb{Z}$ be the standard mapping from binary vectors to integers; $z(x) = \sum_{i=1} n 2^{i-1} x_i$. Let v be a binary-valued vector of length 2^n that contains the truth table of a boolean function f; $v_{z(x)+1} = f(x)$ for all x. The method for computing the ANF is captured in Algorithm 4. It uses two auxiliary vectors t and u of length 2^{n-1} and stores the result in v.

Algorithm 4 ANF computation. Adapted from [11].

```
1: for j = 1, ..., n do
2:     for i = 1, ..., 2^{n-1} do
3:         t_i ← v_{2i-1}
4:         u_i ← v_{2i-1} ⊕ v_{2i}
5:     v ← t||u
```

4.2.2 *d*-monomial test

Contrary to the NIST STS which takes one long keystream sequence and applies various statistical tests, the *d*-monomial test is based on generating a lot of short keystream sequences from different IV values and looking at the statistical properties of a small portion of the output, e.g., only the first bit [13].

The *d*-monomial test examines whether or not an ANF expression of a boolean function has the expected number of *d*-degree monomials [11]. For $d = 0$, the test is called the *Affine test* and for $d > 0$ the *d-monomial test* [11]. The explanations in the rest of this section are adapted from [11].

The *d*-monomial test involves counting the number of ones $\hat{f}(x) = 1$ of an ANF-transformed function f at positions x with Hamming weight $\mathtt{wt}(x) = d$. A χ^2 statistical test is then applied to this count to see if the count is

exceptionally high or low. Note that for a randomly chosen n-bit boolean function f, $\Pr[\hat{f}(x) = 1] = \frac{1}{2}$ for all x.

Consider an n-bit function f. Our null hypothesis is that the expected bit count $\sum_{\mathtt{wt}(x)=d} \hat{f}(x)$ is equal to $\frac{1}{2}\binom{n}{d}$ and binomially distributed. The alternative hypothesis is that there is a bias in this sum.

Suppose that we sample $\hat{f}$ at N distinct points with $\mathtt{wt}(x) = d$ and M of those points satisfy $\hat{f}(x) = 1$. We apply Pearson's classic χ^2 goodness of fit test and calculate the test statistic as

$$\chi^2 = \frac{1}{N}(2M - N)^2.$$

Since "0" and "1" cases in bit count are mutually exclusive, there is one degree of freedom in the test. See Algorithm 5 for a description of the entire test process. The algorithm takes as input the position of keystream bit to analyze p, key to be used for the encryption k, target monomial weight d and level of confidence α. The bit length of the keystream is denoted by n. The algorithm returns *cipher* when the keystream bit is biased and *random* when it is random.

Algorithm 5 d-monomial test. Adapted from [13].

for $iv = 1, \ldots, 2^n - 1$ **do**
    $s \leftarrow$ keystream derived from key k and nonce iv
    $v[iv] \leftarrow s[p]$
$v \leftarrow$ ANF of vector v
for $i = 1, \ldots, 2^n - 1$ **do** ▷ Iterate over monomials
    if $v[i] = 1$ **then**
        $w \leftarrow \mathtt{wt}(i)$
        $distr[w] \leftarrow distr[w] + 1$
$\chi^2 \leftarrow \frac{(distr[d] - \frac{1}{2}\binom{n}{d})^2}{\frac{1}{2}\binom{n}{d}}$
if $\chi^2 > \chi^2(1 - \alpha, 1)$ **then**
    return *cipher*
else
    return *random*

4.2.2.1 Example

- Consider a stream cipher with 3-bit IV and that we want to analyze the first keystream bit for biases. Let f be a boolean function derived from 2^3 keystream computations of all possible IVs and $d = 2$. The truth table of f is shown in Table 4.7.

x	f(x)
000	0
001	1
010	0
011	1
100	1
101	0
110	1
111	0

Table 4.7: Truth table for function f in our d-monomial test example

- We compute the ANF of f:

$$\hat{f}(x_1, x_2, x_3) = x_1 \oplus x_3.$$

- We sample $\hat{f}$ at all three points with $\mathtt{wt}(x) = 2$ and count the number of ones. We obtain $distr[2] = 2$.

- We compute the test statistic

$$\chi^2 = \frac{(distr[d] - \frac{1}{2}\binom{n}{d})^2}{\frac{1}{2}\binom{n}{d}} = \frac{(2 - \frac{1}{2}\binom{3}{2})^2}{\frac{1}{2}\binom{3}{2}} \approx 0.1667.$$

- We compare the test statistic with the critical value $\chi^2(1-\alpha, 1) = \chi^2(0.99, 1) \approx 6.635$. Since $\chi^2 \approx 0.1667 < \chi^2(0.99, 1) \approx 6.635$, the test passed, i.e., the 2-monomial test does not indicate any biases in the first bit of the keystream.

4.3 Cube testers

In this section, we will describe what cube testers are and how they can be used to determine whether a given bit sequence is random. We begin by presenting the preliminaries from [14] necessary to understand the concepts of cube testers.

4.3.1 Preliminaries

Let $\mathcal{F}_n$ be the set of all functions mapping $\{0,1\}^n$ to $\{0,1\}$, where $n > 0$. Let $f \in \mathcal{F}_n$ be a boolean function. Notice that for any function $f \in \mathcal{F}_n$, the XOR

sum of all entries in the truth table

$$\sum_{x \in \{0,1\}^n} f(x)$$

is equal to the coefficient of the highest degree monomial $x_1 \dots x_n$ in the ANF [5] of f. For example, let $n = 4$ and f be a boolean function defined as

$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4 \oplus x_4.$$

By summing f over all 16 distinct inputs, we get one, i.e., the coefficient of the monomial $x_1x_2x_3x_4$. In the case of cube testers, we sum over a subset of the inputs. For example, summing over the four possible values of (x_1, x_2) gives

$$f(0, 0, x_3, x_4) \oplus f(0, 1, x_3, x_4) \oplus f(1, 0, x_3, x_4) \oplus f(1, 1, x_3, x_4) = 1 \oplus x_3 \oplus x_3x_4,$$

where $1 \oplus x_3 \oplus x_3x_4$ is the polynomial that multiplies x_1x_2 in f:

$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2(1 \oplus x_3 \oplus x_3x_4) \oplus x_4.$$

In general, given a set of indices $I \subsetneq \{1, \dots, n\}$, any function in $\mathcal{F}_n$ can be represented algebraically in the form

$$f(x_1, \dots, x_n) = t_I \cdot p(\cdots) \oplus q(x_1, \dots, x_n),$$

where t_I is the monomial containing all x_i for which $i \in I$ and p is a polynomial that has no variable in common with t_I. Also, no monomial in the polynomial q contains t_I. In other words, we factored f by the monomial t_I. As for the terminology, p is called the *superpoly* of I in f. A *cube* t_I is called a *maxterm* if and only if its superpoly p has degree 1, i.e., it is linear but not a constant. The polynomial f is called the *master polynomial*.

For a given n, a random function is a random element of $\mathcal{F}_n$ ($|\mathcal{F}_n| = 2^{2^n}$). A *distinguisher* for a family $\mathcal{F} \subsetneq \mathcal{F}_n$ is a procedure that, given a function f randomly sampled from $\mathcal{F}^\star \in \{\mathcal{F}, \mathcal{F}_n\}$, efficiently determines which one of these two families was chosen as $\mathcal{F}^\star$. A family $\mathcal{F}$ is pseudorandom if and only if there exists no efficient distinguisher for it.

To distinguish $\mathcal{F} \subsetneq \mathcal{F}_n$ from $\mathcal{F}_n$, cube testers partition the set of public variables $\{x_1, \dots, x_n\}$ into two complementary subsets:

- Cube Variables (CVs),
- Superpoly Variables (SVs).

[5] See Definition 8 for the definition of ANF.

Let us reuse the example from above. Given

$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4 \oplus x_4.$$

we considered the cube x_1x_2 and computed its superpoly $1 \oplus x_3 \oplus x_3x_4$. Here, the CVs are $\{x_1, x_2\}$ and the SVs are $\{x_3, x_4\}$. Therefore, by setting a value to x_3 and x_4, e.g., $x_3 = 1$ and $x_4 = 0$, we can compute $1 \oplus x_3 \oplus x_3x_4 = 1 \oplus 1 \oplus 1 \cdot 0 = 0$ by summing $f(x_1, x_2, x_3, x_4)$ for all possible choices of (x_1, x_2). Note that it is not required for an SV to actually appear in the superpoly of the cube. For example, if $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3$, then the superpoly for CVs $\{x_1, x_2\}$ is $1 \oplus x_3$, but both x_3 and x_4 are SVs. For the sake of efficiency, not all inputs have to necessarily be considered as variables. For example, if f maps 1024 bits to 256 bits, we may choose 20 CVs and 10 SVs and set a fixed value to the other inputs.

A *family tester* for a family of functions $\mathcal{F}$ takes as input a function f of the same domain $\mathcal{D}$ and tests if f is close to $\mathcal{F}$, with respect to a bound ϵ on the distance

$$\delta(f, \mathcal{F}) = \min_{g \in \mathcal{F}} \frac{|\{x \in \mathcal{D}, f(x) \neq g(x)\}|}{|\mathcal{D}|}.$$

The tester accepts if $\delta(f, \mathcal{F}) = 0$, rejects with high probability if f and $\mathcal{F}$ are not ϵ-close and behaves arbitrarily otherwise. Such test captures the notion of property testing when a property is defined by belonging to a family of functions $\mathcal{P}$. A *property tester* is thus a family tester for property $\mathcal{P}$.

Suppose we wish to distinguish a family $\mathcal{F} \subsetneq \mathcal{F}_n$ from $\mathcal{F}_n$, i.e., given a random $f \in \mathcal{F}^*$, to determine whether $\mathcal{F}^*$ is $\mathcal{F}$ or $\mathcal{F}_n$. If $\mathcal{F}$ is efficiently testable, we can use a family tester for $\mathcal{F}$ on f to distinguish it from a random function.

4.3.2 Testing with cube testers

In this section, we will finally dive into how cube testers are used to assess the randomness of a given sequence of bits. The concepts described in the rest of this section apart from examples are adapted from [14].

Cube testers distinguish a family of functions from random functions by testing properties of their superpolys for a specific choice of CVs and SVs. They combine an efficient property tester on the superpoly, which is viewed either as a polynomial or as a mapping, with a statistical decision rule. As soon as the superpoly has some "unexpected" property, it is identified as non-random.

4.3.2.1 Balance test

A random function is expected to contain the same amount of zeros and ones in its truth table. Superpolys that have a strongly unbalanced truth table can thus be distinguished from random polynomials by testing whether they

evaluate to one or zero. We can do this either deterministically by evaluating the superpoly for each possible input or probabilistically over a random subset of SVs. Let $x_1, \ldots, x_C$ be CVs, $x_{C+1}, \ldots, x_n$ be SVs and $\mathcal{D}$ be some decision rule. Algorithm 6 describes the deterministic balance test.

Algorithm 6 Deterministic balance test. Adapted from [14].

```
c ← 0
for all values of (x_{C+1}, ..., x_n) do
    compute p(x_{C+1}, ..., x_n) = Σ_{(x_1,...,x_C)} f(x_1, ..., x_n)
    c ← c + p(x_{C+1}, ..., x_n)
return D(c) ∈ {0, 1}
```

4.3.2.2 Balance test example

- Let
$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4 \oplus x_4$$
be the ANF of a boolean function derived from keystream computations for various IVs and a particular keystream bit. Let x_1, x_2 be CVs and x_3, x_4 be SVs.

- The superpoly of f for cube x_1x_2 is $p(x_3, x_4) = 1 \oplus x_3 \oplus x_3x_4$.

- We count the number of ones in superpoly evaluations for all possible values of x_3, x_4:
$$\begin{aligned} p(0,0) &= 1 \oplus 0 \oplus 0 \cdot 0 = 1, \\ p(0,1) &= 1 \oplus 0 \oplus 0 \cdot 1 = 1, \\ p(1,0) &= 1 \oplus 1 \oplus 1 \cdot 0 = 0, \\ p(1,1) &= 1 \oplus 1 \oplus 1 \cdot 1 = 1, \end{aligned}$$
and obtain $c = 3$.

- Using some decision rule, e.g., the χ^2 goodness of fit test, we conclude whether the superpoly is balanced.

- We compute the test statistic
$$\chi^2 = \frac{\left(c - \frac{2^n}{2}\right)^2}{\frac{2^n}{2}} = \frac{\left(3 - \frac{2^4}{2}\right)^2}{\frac{2^4}{2}} = 3.125.$$

- We compare the test statistic with the critical value $\chi^2(1-\alpha, 1) = \chi^2(0.99, 1) \approx 6.635$. Since $\chi^2 = 3.125 < \chi^2(0.99, 1) \approx 6.635$, the test passed.

4.3.2.3 Presence of linear variables test

The ANF of a random function contains a given variable in at least one monomial of degree at least 2 with probability close to 1. Therefore, we can test whether a given superpoly variable appears only linearly in the superpoly. See Algorithm 7 for a description of this process for variable x_1. The test succeeds when the algorithm returns "non-linear" and fails when it returns "linear". It answers correctly with a probability of about $1 - 2^{-N}$. If a stream cipher is shown to have an IV bit linear with respect to a set of CVs in the IV, independently of the choice of the key, then it directly gives a distinguisher.

Algorithm 7 Presence of linear variables test. Adapted from [14].

```
for i ← 1 to N do
    pick random (x_2, ..., x_S)
    if p(0, x_2, ..., x_S) = p(1, x_2, ..., x_S) then
        return non-linear
return linear
```

4.3.2.4 Presence of linear variables test example

- Let
$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4 \oplus x_4$$
be the ANF of a boolean function derived from keystream computations for various IVs and a particular keystream bit. Let x_1, x_2 be CVs and x_3, x_4 be SVs.
- The superpoly of f for cube x_1x_2 is $p(x_3, x_4) = 1 \oplus x_3 \oplus x_3x_4$.
- Suppose we want to test the SV x_4.
- We can see that x_4 is contained in monomial x_3x_4 of degree 2. Therefore, it is not linear and we would expect the test to pass.
- Since there are only two possible inputs for x_3, we perform two iterations of the test:
$$\begin{aligned} p(0,0) &= 1 \oplus 0 \oplus 0 \cdot 0 = 1, \\ p(0,1) &= 1 \oplus 0 \oplus 0 \cdot 1 = 1, \\ p(1,0) &= 1 \oplus 1 \oplus 1 \cdot 0 = 0, \\ p(1,1) &= 1 \oplus 1 \oplus 1 \cdot 1 = 1. \end{aligned}$$
- In the first iteration, we find that $p(0,0) = p(0,1) = 1$. Therefore, we right away conclude that the variable x_4 is not linear in the superpoly, i.e., the test passed.

4.3.2.5 Presence of neutral variables test

Similarly to the previous test, we can test whether an SV is neutral in the superpoly, i.e., whether it appears in at least one monomial. See Algorithm 8 for a description of this process for variable x_1. The test succeeds when the algorithm returns "non-neutral" and fails when it returns "neutral". It answers correctly with a probability of about $1 - 2^{-N}$.

Algorithm 8 Presence of neutral variables test. Adapted from [14].

```
for i ← 1 to N do
    pick random (x_2, ..., x_S)
    if p(0, x_2, ..., x_S) ≠ p(1, x_2, ..., x_S) then
        return non-neutral
return neutral
```

4.3.2.6 Presence of neutral variables test example

- Let
$$f(x_1, x_2, x_3, x_4) = x_1 \oplus x_1x_2 \oplus x_1x_2x_3 \oplus x_1x_2x_3x_4 \oplus x_4$$
be the ANF of a boolean function derived from keystream computations for various IVs and a particular keystream bit. Let x_1, x_2 be CVs and x_3, x_4 be SVs.
- The superpoly of f for cube x_1x_2 is $p(x_3, x_4) = 1 \oplus x_3 \oplus x_3x_4$.
- Suppose we want to test the SV x_4.
- We can see that x_4 is contained in monomial x_3x_4. Therefore, it is not neutral and we would expect the test to pass.
- Since there are only two possible inputs for x_3, we perform two iterations of the test:
$$\begin{aligned} p(0,0) &= 1 \oplus 0 \oplus 0 \cdot 0 = 1, \\ p(0,1) &= 1 \oplus 0 \oplus 0 \cdot 1 = 1, \\ p(1,0) &= 1 \oplus 1 \oplus 1 \cdot 0 = 0, \\ p(1,1) &= 1 \oplus 1 \oplus 1 \cdot 1 = 1. \end{aligned}$$
- In the first iteration, we find that $p(0,0) = p(0,1) = 1$, so we continue the test.
- In the second iteration, we find that $p(1,0) = 0 \neq 1 = p(1,1)$. Therefore, we conclude that the variable x_4 is not neutral in the superpoly, i.e., the test passed.

Chapter 5

Results

In this chapter, we will discuss the results we obtained by applying previously described tests to the Dumbo instance of the Elephant cipher.

5.1 NIST STS

Let us start off with the NIST STS. Using the `crypto.randomBytes` function in Node.js version 19.0.0, we generated three random keys and nonces. With these parameters, we then generated corresponding keystreams ks_1, ks_2 and ks_3 of 10,485,760 bits (10 MB) each using Algorithm 3. These keystreams were then fed to the NIST STS [6] version 3.2.6 [7] to assess their randomness by utilizing the tests we described in Section 4.1. Each pair of parameters (key, nonce) was divided into ten iterations of 1,048,576 bits and the level of significance α was left at the default value of 0.01. The tests were run using the following command.

```
sts -t 1,2,3,4,6,7,10,12,14,15 -F a -S 1048576
  -i 10 /path/to/data.bin
```

The results were mostly positive. Keystreams ks_1 and ks_2 passed all of the tests, while ks_3 failed the Runs test and one of the eight Random excursions tests. Complete results are shown in Appendix A. Upon further inspection, we discovered that two out of ten iterations of ks_3 failed the Runs test with p-values 0.001105 and 0.004236. The random excursions tests performed four iterations each [8]. One of these tests failed in one iteration with state $x = 3$ as it resulted in a p-value of 0.00929. Detailed results for ks_3 are shown in Table 5.1. Failed tests are highlighted with an asterisk next to their name.

[6] `https://github.com/arcetri/sts`

[7] The repository was cloned at commit `8359589e2446f84a4f803ec4753a4b6b5b460e4c`.

[8] This is because six of the iterations were discarded due to insufficient amount of cycles (< 500).

$\mathbf{C_1}$	$\mathbf{C_2}$	$\mathbf{C_3}$	$\mathbf{C_4}$	$\mathbf{C_5}$	$\mathbf{C_6}$	$\mathbf{C_7}$	$\mathbf{C_8}$	$\mathbf{C_9}$	$\mathbf{C_{10}}$	**P**	**Prop.**	**Test**
2	3	2	1	0	0	0	2	0	0	0.21	10/10	Freq.
1	1	0	1	1	1	1	1	2	1	0.99	10/10	BF
2	1	2	2	2	0	0	0	1	0	0.53	10/10	CS
2	0	4	1	1	1	0	0	0	1	0.12	10/10	CS
2	0	1	1	0	0	2	2	0	2	0.53	8/10	*Runs
1	0	1	2	0	0	1	3	0	2	0.35	10/10	Rank
0	2	2	1	1	1	0	1	1	1	0.91	10/10	DFT
1	0	0	3	1	1	1	2	1	0	0.53	10/10	Univ.
0	0	0	0	0	0	2	1	0	1		4/4	RE
0	0	0	1	1	0	0	1	0	1		4/4	RE
1	2	0	0	0	0	0	0	1	0		4/4	RE
0	0	0	1	1	1	1	0	0	0		4/4	RE
0	1	0	1	0	0	0	1	0	1		4/4	RE
0	1	1	0	1	1	0	0	0	0		4/4	RE
1	1	0	0	0	0	0	2	0	0		3/4	*RE
0	1	0	1	0	0	1	0	1	0		4/4	RE
1	2	0	0	2	2	0	2	0	1	0.53	9/10	Ser.
1	2	1	0	1	1	1	1	0	2	0.91	10/10	Ser.
1	1	2	0	1	1	1	1	1	1	0.99	10/10	LC

Table 5.1: Results of tests from the NIST STS. Columns $C_1, \ldots, C_{10}$ indicate how many p-values fell into intervals $[0, 0.1), [0.1, 0.2), \ldots, [0.9, 1)$, P represents the resulting p-value of uniformity testing of p-values computed for the given test and *Prop.* says what proportion of sequences (iterations) passed the given test.

The failed Random excursions test iteration does not immediately raise a red flag because its p-value is very close to the level of significance $\alpha = 0.01$. On the other hand, failed iterations of the Runs test may be an issue since they resulted in very low p-values. This could indicate that keystreams produced by Dumbo may be biased such that some runs appear more often than others. However, since this only appeared in one of the three pairs of (key, nonce) input parameters, we do not consider this to be a major issue. Perhaps this bias could be linked to some subset of the input resulting in some class of "weak" keys or nonces, though this is outside the scope of this thesis.

5.2 d-monomial test

For the d-monomial test, we used the same three keys we generated earlier for tests from NIST STS. As for the nonce, we wish to find a subset of input bits that is likely to receive less mixing during the nonce setup process than other bits [11]. This is likely to be either at the beginning or the end of the nonce bit-vector [11]. Therefore, we used the following methods of nonce iteration.

`begin0` Iterate the first n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 0.

`begin1` Iterate the first n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 1.

`end0` Iterate the last n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 0.

`end1` Iterate the last n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 1.

We collected the first 32 bytes of the keystream using Algorithm 3 for each of the three keys and each of the four nonce iteration methods with $n = 24$. We then built the corresponding truth tables mapping the n bits of nonce that were being iterated to the value of a particular keystream bit. We did this for the first 32 bits of the keystream. After that, we computed the ANF for each one of the truth tables and evaluated the d-monomial test described in Algorithm 5 for $d \in \{0, \ldots, 24\}$. The level of significance α was set to 0.01.

All in all, a total of $3 \cdot 4 \cdot 32 \cdot 25 = 9600$ d-monomial tests were conducted. The vast majority of the tests passed — only six tests failed and none of them shared the keystream bit position or the d parameter. Also, p-values of the failed tests were still relatively high, the lowest one being 0.0023. Let us label the used keys K_1, K_2 and K_3. The results of the failed d-monomial tests can be seen in Table 5.2.

Since only very few cases of the d-monomial test failed and their p-values were still relatively high, we do not think there is any significant bias in the

Key	Iteration method	Bit position	d	p-value
K_1	`end1`	20	6	0.0043
K_2	`begin0`	11	22	0.0083
K_2	`end0`	3	3	0.0023
K_2	`end0`	6	18	0.0073
K_3	`end0`	32	11	0.0071
K_3	`end1`	5	7	0.0064

Table 5.2: Results of the failed d-monomial tests

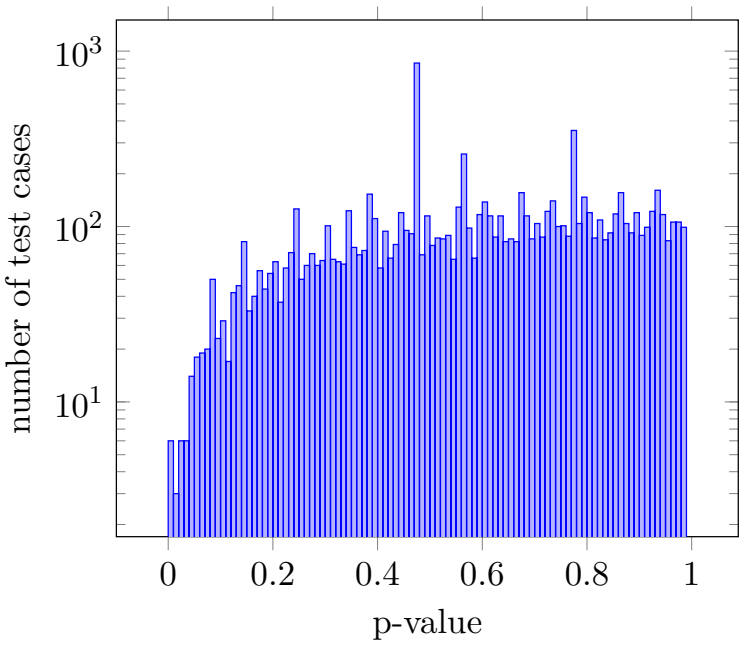

Figure 5.1: Histogram of p-values from all d-monomial test cases with interval size 0.01

keystream produced by Dumbo based on these results. Let us also examine the p-values of the remaining tests that passed. The histogram of these p-values is depicted in Figure 5.1. We can see a slight trend of increasing number of test cases with increasing p-value with noticably lower amount of test cases having p-value less than 0.2. There are a few intervals with significantly more test cases than others. For example, the $[0.47, 0.48)$ interval contains 855 test cases, though this does not raise any red flags.

5.3 Cube testers

Once again, we used the same three keys from previous tests. We also chose the same four nonce iteration methods used in the d-monomial test to build the truth tables.

`begin0` Iterate the first n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 0.

`begin1` Iterate the first n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 1.

`end0` Iterate the last n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 0.

`end1` Iterate the last n nonce bits from 0 to $2^n - 1$. Set the remaining bits to 1.

Due to a noticeable increase in computational complexity, we only iterated through the first $n = 8$ bits of the nonce. Once we collected the truth tables, we computed the ANF for each one of them. As for the choice of CVs and SVs, we chose to only use the nonce. We iterated through all possible choices of CVs (bits) in $n = 8$ bits of the nonce and labeled the remaining bits (within the $n = 8$ bits) that were not chosen as SVs. The cubes (i.e., choices of CVs in the $n = 8$ bits of nonce that were being iterated) were represented as n-character strings $\{1, -\}^n$ (e.g., `11-1--1-`). The same representation was used for SVs, where the characters `1` and `-` denoted whether the given variable was chosen as an SV or not respectively. For every iteration of CVs and SVs, we computed the corresponding superpoly and applied property testers described in Section 4.3.

We performed the *balance*, *presence of linear variables* and *presence of neutral variables* tests for each combination of key, nonce iteration method, keystream bit position (from 1 to 32) and cube. This resulted in 884,736 unique tests, which were then merged by cube and keystream bit position as an attempt to uncover "well-performing" cubes and biased keystream bits.

5.3.1 Balance test

We performed the balance test as described in Algorithm 6 by evaluating the superpoly for all possible choices of CVs in the nonce and counting how many times it evaluated to zero or one. We then arrived at a conclusion using the χ^2 goodness of fit test by computing the test statistic

$$\chi^2 = \frac{\left(c - \frac{2^n}{2}\right)^2}{\frac{2^n}{2}},$$

where c is the amount of times the ANF evaluated to one and n is the number of bits in the nonce that were being iterated, i.e., there were 2^n possible CV inputs. In the case of a random function, half of those inputs would evaluate to zero and the other half would evaluate to one. We then computed the p-value with one degree of freedom. The test passed if the p-value was greater or equal to the level of significance $\alpha = 0.01$ and failed otherwise.

Overall, 56,169 out of the 98,304 balance tests passed (57.14 % pass rate). The pass rates for different keystream bit positions and cubes are shown in Table 5.3 and Table 5.4 respectively. Pass rates grouped by keystream bit position ranged from 54.75 % to 60.29 %, while pass rates grouped by cube ranged from 0 % to 100 %. Note that pass rates of 0 % and 100 % were achieved using cubes `11111111` and `--------` respectively, though there were also cubes with one CV that achieved a pass rate of over 99 % and cubes with 2 CVs that achieved a pass rate of almost 95 %.

Since a random function is expected to contain as many zeros as ones in its truth table [14], we were able to find cubes for which the superpoly is distinguishable from a random polynomial as their pass rates are very low. This is a notable weakness of Dumbo.

5.3.2 Presence of linear variables test

We performed the presence of linear variables test as described in Algorithm 7 for all SVs. We randomly set the rest of the SVs to zero or one aside from the currently selected SV. We then checked whether the superpoly evaluation changed when the selected variable was flipped from zero to one. If the evaluation changed, we continued, otherwise we labeled the SV as non-linear. We repeated this process up to $N = 10$ times for each SV to achieve $1 - 2^{-10} \approx 99.9\,\%$ confidence in case the variable was labeled linear. The test passed if the variable was labeled non-linear within ten iterations and failed otherwise.

We did this for a total of 393,216 tests, 378,716 of which (96.31 %) passed. The pass rates for different keystream bit positions and cubes are shown in Table 5.5 and Table 5.6 respectively. Pass rates grouped by keystream bit position ranged from 94.87 % to 97.67 %, while pass rates grouped by cube ranged from 44.79 % to 99.96 %.

For this test, it is also interesting to look at results for combinations of cubes and keystream bit positions (12 tests in total for each combination). If we were to show that a nonce bit is linear with respect to a set of CVs in the nonce independently of the choice of the key, the test would directly give us a distinguisher [14]. However, we were not able to showcase this since the combination of CVs and keystream bit position with the lowest pass rate still had a pass rate of 8.33 % across all keys and nonce iteration methods. The top 25 combinations with the lowest pass rates are shown in Table 5.7.

Keystream bit position	Pass rate
12	54.75 %
31	54.92 %
11	55.57 %
7	55.76 %
4	55.99 %
28	55.99 %
22	56.02 %
32	56.05 %
5	56.18 %
26	56.38 %
14	56.54 %
17	56.61 %
20	56.61 %
15	56.67 %
8	56.74 %
2	56.77 %
1	56.84 %
19	57.03 %
29	57.06 %
16	57.26 %
9	57.29 %
13	57.49 %
18	57.58 %
21	57.81 %
6	58.17 %

Table 5.3: Balance test pass rates of the top 25 out of 32 analyzed keystream bits with the lowest pass rates, sorted by pass rate

Cube	Pass rate
11111111	0 %
-1--1111	20.05 %
--11-111	21.61 %
11--1-11	21.88 %
-1-111-1	22.92 %
1-11-1-1	22.92 %
1-1111--	22.92 %
1111---1	22.92 %
11-11--1	23.18 %
1-1-111-	23.44 %
11--11-1	23.44 %
11-1-1-1	23.44 %
111--1-1	23.44 %
--1-1111	23.70 %
1--111-1	23.70 %
111-11--	23.70 %
--111-11	23.96 %
-111--11	23.96 %
1-1-11-1	23.96 %
-1-11-11	24.22 %
1-1--111	24.22 %
1-11--11	24.22 %
11---111	24.22 %
11--111-	24.22 %
111-1--1	24.22 %

Table 5.4: Balance test pass rates of the top 25 out of 256 cubes with the lowest pass rates, sorted by pass rate

Keystream bit position	Pass rate
10	94.87 %
30	95.00 %
3	95.52 %
23	95.52 %
8	95.60 %
2	95.67 %
18	95.73 %
25	95.74 %
24	96.06 %
19	96.07 %
6	96.18 %
9	96.22 %
27	96.30 %
20	96.31 %
15	96.33 %
1	96.35 %
29	96.37 %
31	96.41 %
16	96.49 %
17	96.49 %
21	96.50 %
14	96.53 %
4	96.65 %
11	96.69 %
32	96.80 %

Table 5.5: Presence of linear variables test pass rates of the top 25 out of 32 analyzed keystream bits with the lowest pass rates, sorted by pass rate

Cube	Pass rate
-1111111	44.79 %
1-111111	44.79 %
11-11111	44.79 %
111-1111	44.79 %
1111-111	44.79 %
11111-11	44.79 %
111111-1	44.79 %
1111111-	44.79 %
1--11111	75.13 %
11-11-11	75.65 %
1-111-11	75.78 %
11--1111	76.43 %
-1-11111	76.69 %
1-1-1111	76.69 %
--111111	76.82 %
11-1-111	76.82 %
1-11-111	76.95 %
111-1-11	77.34 %
-1111-11	77.47 %
1111--11	77.47 %
11-1111-	77.60 %
1-11111-	77.73 %
11-111-1	77.99 %
-11-1111	78.13 %
1-1111-1	78.13 %

Table 5.6: Presence of linear variables test pass rates of the top 25 out of 256 cubes with the lowest pass rates, sorted by pass rate

Cube	Keystream bit position	Pass rate
-1111111	5	8.33 %
1-111111	5	8.33 %
11-11111	5	8.33 %
111-1111	5	8.33 %
1111-111	5	8.33 %
11111-11	5	8.33 %
111111-1	5	8.33 %
1111111-	5	8.33 %
-1111111	7	25 %
-1111111	12	25 %
-1111111	13	25 %
-1111111	26	25 %
1--11111	10	25 %
1-111111	7	25 %
1-111111	12	25 %
1-111111	13	25 %
1-111111	26	25 %
11-11111	7	25 %
11-11111	12	25 %
11-11111	13	25 %
11-11111	26	25 %
111-1111	7	25 %
111-1111	12	25 %
111-1111	13	25 %
111-1111	26	25 %

Table 5.7: Presence of linear variables test pass rates of the top 25 cube and keystream bit position combinations with the lowest pass rates, sorted by pass rate

5.3.3 Presence of neutral variables test

Similarly, we performed the presence of neutral variables test as described in Algorithm 8 for all SVs. We once again randomly set the rest of the SVs to zero or one aside from the currently selected SV and then checked whether the superpoly evaluation changed when the selected variable was flipped from zero to one. If the evaluation did not change, we continued, otherwise we labeled the SV as non-neutral. We repeated this process up to $N = 10$ times for each SV to achieve $1 - 2^{-10} \approx 99.9\,\%$ confidence in case the variable was labeled neutral. The test passed if the variable was labeled non-neutral within ten iterations and failed otherwise.

We did this for a total of 393,216 tests, 378,732 of which (96.32 %) passed. The pass rates for different keystream bit positions and cubes are shown in Table 5.8 and Table 5.9 respectively. Pass rates grouped by keystream bit position ranged from 93.82 % to 98.31 %, while pass rates grouped by cube ranged from 55.21 % to 99.96 %.

We also looked at results for combinations of cubes and keystream bit positions (12 tests in total for each combination) like we did in the previous test. We were once again unable to show that a particular nonce bit was neutral with respect to a set of CVs independently of the choice of the key or nonce iteration method. The combination of cube and keystream bit position with the lowest pass rate still passed 8.33 % of the tests. The top 25 combinations with the lowest pass rates are shown in Table 5.10.

Keystream bit position	Pass rate
30	93.82 %
8	94.63 %
23	94.78 %
10	95.38 %
2	95.52 %
19	95.52 %
18	95.64 %
9	95.74 %
3	95.98 %
29	96.07 %
25	96.15 %
7	96.17 %
31	96.19 %
1	96.21 %
15	96.23 %
20	96.24 %
14	96.34 %
4	96.42 %
24	96.44 %
32	96.48 %
11	96.69 %
6	96.73 %
28	96.77 %
17	96.79 %
16	96.79 %

Table 5.8: Presence of neutral variables test pass rates of the top 25 out of 32 analyzed keystream bits with the lowest pass rates, sorted by pass rate

Cube	Pass rate
-1111111	55.21 %
1-111111	55.21 %
11-11111	55.21 %
111-1111	55.21 %
1111-111	55.21 %
11111-11	55.21 %
111111-1	55.21 %
1111111-	55.21 %
111111--	74.61 %
-11111-1	75.26 %
1111-1-1	75.39 %
111-11-1	75.65 %
-111111-	75.78 %
1111-11-	75.78 %
111-111-	76.04 %
11111--1	76.17 %
-111-111	76.69 %
-11-1111	76.82 %
111--111	76.82 %
11111-1-	76.82 %
1-1111-1	76.95 %
11-111-1	77.21 %
1-11111-	77.47 %
11-1111-	77.47 %
-1111-11	77.60 %

Table 5.9: Presence of neutral variables test pass rates of the top 25 out of 256 cubes with the lowest pass rates, sorted by pass rate

Cube	Keystream bit position	Pass rate
-1111111	10	8.33 %
1-111111	10	8.33 %
11-11111	10	8.33 %
111-1111	10	8.33 %
1111-111	10	8.33 %
11111-11	10	8.33 %
111111-1	10	8.33 %
1111111-	10	8.33 %
-1111111	23	25 %
1-111111	23	25 %
11-11111	23	25 %
111-1111	23	25 %
1111-111	23	25 %
11111-11	23	25 %
111111-1	23	25 %
1111111-	23	25 %
-1111111	18	33.33 %
-1111111	25	33.33 %
-1111111	30	33.33 %
1-111111	18	33.33 %
1-111111	25	33.33 %
1-111111	30	33.33 %
11-11111	18	33.33 %
11-11111	25	33.33 %
11-11111	30	33.33 %

Table 5.10: Presence of neutral variables test pass rates of the top 25 cube and keystream bit position combinations with the lowest pass rates, sorted by pass rate

Conclusion

This thesis focused on testing the randomness of the Dumbo instance of the Elephant cipher. In the beginning, we explained the ideas behind stream ciphers, presented the Elephant cipher, specifically the Dumbo instance, and discussed the motivation behind the Lightweight Cryptography project, which the Elephant cipher competed in.

We then delved into random bit generators and why there is a need to test these generators. We also touched on why we need randomness in ciphers and the issues that may arise when there is lack of it. We followed up with basics of hypothesis testing, which we concluded with an introduction of the χ^2 goodness of fit test.

Next, we presented several categories of tests, which we later applied to Dumbo, namely the Statistical Test Suite from NIST, variations of monomial tests and cube testers. We chose to use ten tests from NIST's suite, the d-monomial test inspired by Filiol's Möbius tests and three superpoly property testers from the family of cube testers. For all of those tests, we provided the necessary mathematical background and algorithms that describe the tests.

After that, we presented the results of our tests. Dumbo passed the vast majority of tests from NIST's STS, failing only two out of ten iterations of the runs test and one iteration of the random excursions test. We did not consider this a major issue because the resulting p-values were still relatively high (in the case of the random excursions test) or because the test only failed for one out of three tested keystreams (in the case of the runs test). We also conducted 9,600 d-monomial tests, out of which only six failed and did not suggest bias in any particular keystream bit. We provided the histogram of p-values from all of the d-monomial tests and found nothing suspicious. Finally, we used property testers to test the superpoly for many choices of cube and superpoly variables. We conducted balance tests, presence of linear variables tests and presence of neutral variables tests. We found that some keystream bits only passed almost every other balance test and also that many cubes resulted in a pass rate of around 20 % in this test. In the author's opinion, this was the

most significant finding of this work. As for the linear and neutral variables presence tests, we were unable to show that a distinguisher independent of the used key exists, although the pass rates of those tests were very low for some combinations of (cube, keystream bit).

Bibliography

1. KLÍMA, Vlastimil. Základy moderní kryptologie – Symetrická kryptografie II. 2005. Version 1.3.
2. VAN OORSCHOT, Paul C; MENEZES, Alfred J; VANSTONE, Scott A. *Handbook of applied cryptography*. CRC press, 1996.
3. CUSICK, Thomas W.; STANICA, Pantelimon. Chapter 2 - Fourier Analysis of Boolean Functions. In: CUSICK, Thomas W.; STANICA, Pantelimon (eds.). *Cryptographic Boolean Functions and Applications (Second Edition)*. Second Edition. Academic Press, 2017, pp. 7–29. ISBN 978-0-12-811129-1. Available from DOI: https://doi.org/10.1016/B978-0-12-811129-1.00002-X.
4. MENNINK, B. et al. *Elephant v2 Specification* [online]. 2021. [visited on 2023-02-21]. Available from: https://csrc.nist.gov/CSRC/media/Projects/lightweight-cryptography/documents/finalist-round/updated-spec-doc/elephant-spec-final.pdf.
5. BOGDANOV, A.; KNEŽEVIĆ, M.; LEANDER, G.; TOZ, D.; VARICI, K.; VERBAUWHEDE, I. spongent: A Lightweight Hash Function. In: PRENEEL, B.; TAKAGI, T. (eds.). *Cryptographic Hardware and Embedded Systems – CHES 2011*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 312–325. ISBN 978-3-642-23951-9.
6. National Institute of Standards and Technology. *Lightweight Cryptography Overview* [online]. [N.d.]. [visited on 2023-02-21]. Available from: https://csrc.nist.gov/projects/lightweight-cryptography.
7. National Institute of Standards and Technology. *Lightweight Cryptography Finalists* [online]. [N.d.]. [visited on 2023-02-21]. Available from: https://csrc.nist.gov/Projects/lightweight-cryptography/finalists.

8. MANTIN, Itsik. Predicting and distinguishing attacks on RC4 keystream generator. In: *Advances in Cryptology–EUROCRYPT 2005: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005. Proceedings 24*. Springer, 2005, pp. 491–506.
9. JUREČKOVÁ, Olha. Testy generátorů pseudonáhodných čísel. 2015.
10. BASSHAM III, Lawrence E; RUKHIN, Andrew L; SOTO, Juan; NECHVATAL, James R; SMID, Miles E; BARKER, Elaine B; LEIGH, Stefan D; LEVENSON, Mark; VANGEL, Mark; BANKS, David L, et al. *Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications*. National Institute of Standards & Technology, 2010.
11. SAARINEN, Markku-Juhani O. Chosen-IV statistical attacks on eSTREAM stream ciphers. *Proc. Stream Ciphers Revisited SASC*. 2006.
12. FILIOL, Eric. A new statistical testing for symmetric ciphers and hash functions. In: *Information and Communications Security: 4th International Conference, ICICS 2002 Singapore, December 9–12, 2002 Proceedings 4*. Springer, 2002, pp. 342–353.
13. ENGLUND, H.; JOHANSSON, T.; SÖNMEZ TURAN, M. A framework for chosen IV statistical analysis of stream ciphers. In: *Progress in Cryptology–INDOCRYPT 2007: 8th International Conference on Cryptology in India, Chennai, India, December 9-13, 2007. Proceedings 8*. Springer, 2007, pp. 268–281.
14. AUMASSON, Jean-Philippe; DINUR, Itai; MEIER, Willi; SHAMIR, Adi. Cube testers and key recovery attacks on reduced-round MD6 and Trivium. In: *Fast Software Encryption: 16th International Workshop, FSE 2009 Leuven, Belgium, February 22-25, 2009 Revised Selected Papers*. Springer, 2009, pp. 1–22.

Appendix A

NIST STS results

A.1 ks_1

```
A total of 19 tests (some of the 15 tests actually consist
of multiple sub-tests) were conducted to evaluate the
randomness of 10 bitstreams of 1048576 bits from:

  data/ks1.bin

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The numerous empirical results of these tests were then
interpreted with an examination of the proportion of
sequences that pass a statistical test (proportion analysis)
and the distribution of p-values to check for uniformity
(uniformity analysis). The results were the following:

  19/19 tests passed successfully both the analyses.
  0/19 tests did not pass successfully both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Here are the results of the single tests:

 - The "Frequency" test passed both the analyses.

 - The "Block Frequency" test passed both the analyses.

 - The "Cumulative Sums" (forward) test passed both the
   analyses.
```

```
   The "Cumulative Sums" (backward) test passed both the
   analyses.

 - The "Runs" test passed both the analyses.

 - The "Binary Matrix Rank" test passed both the analyses.

 - The "Discrete Fourier Transform" test passed both the
   analyses.

 - The "Maurer's Universal Statistical" test passed both the
   analyses.

 - 8/8 of the "Random Excursions" tests passed both the
   analyses.

 - The "Serial" (first) test passed both the analyses.
   The "Serial" (second) test passed both the analyses.

 - The "Linear Complexity" test passed both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The missing tests (if any) were whether disabled manually by
the user or disabled at run time due to input size
requirements not satisfied by this run.
```

A.2 ks_2

```
A total of 19 tests (some of the 15 tests actually consist
of multiple sub-tests) were conducted to evaluate the
randomness of 10 bitstreams of 1048576 bits from:

  data/ks2.bin

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The numerous empirical results of these tests were then
interpreted with an examination of the proportion of
sequences that pass a statistical test (proportion analysis)
and the distribution of p-values to check for uniformity
(uniformity analysis). The results were the following:

  19/19 tests passed successfully both the analyses.
  0/19 tests did not pass successfully both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Here are the results of the single tests:

 - The "Frequency" test passed both the analyses.

 - The "Block Frequency" test passed both the analyses.

 - The "Cumulative Sums" (forward) test passed both the
   analyses.
   The "Cumulative Sums" (backward) test passed both the
   analyses.

 - The "Runs" test passed both the analyses.

 - The "Binary Matrix Rank" test passed both the analyses.

 - The "Discrete Fourier Transform" test passed both the
   analyses.

 - The "Maurer's Universal Statistical" test passed both the
   analyses.

 - 8/8 of the "Random Excursions" tests passed both the
   analyses.
```

```
 - The "Serial" (first) test passed both the analyses.
   The "Serial" (second) test passed both the analyses.

 - The "Linear Complexity" test passed both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The missing tests (if any) were whether disabled manually by
the user or disabled at run time due to input size
requirements not satisfied by this run.
```

A.3 ks_3

```
A total of 19 tests (some of the 15 tests actually consist
of multiple sub-tests) were conducted to evaluate the
randomness of 10 bitstreams of 1048576 bits from:

  data/ks3.bin

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The numerous empirical results of these tests were then
interpreted with an examination of the proportion of
sequences that pass a statistical test (proportion analysis)
and the distribution of p-values to check for uniformity
(uniformity analysis). The results were the following:

  17/19 tests passed successfully both the analyses.
  2/19 tests did not pass successfully both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

Here are the results of the single tests:

 - The "Frequency" test passed both the analyses.

 - The "Block Frequency" test passed both the analyses.

 - The "Cumulative Sums" (forward) test passed both the
   analyses.
   The "Cumulative Sums" (backward) test passed both the
   analyses.

 - The "Runs" test FAILED the proportion analysis.

 - The "Binary Matrix Rank" test passed both the analyses.

 - The "Discrete Fourier Transform" test passed both the
   analyses.

 - The "Maurer's Universal Statistical" test passed both the
   analyses.

 - 7/8 of the "Random Excursions" tests passed both the
   analyses.
```

```
   1/8 of the "Random Excursions" tests FAILED the
   proportion analysis.

 - The "Serial" (first) test passed both the analyses.
   The "Serial" (second) test passed both the analyses.

 - The "Linear Complexity" test passed both the analyses.

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

The missing tests (if any) were whether disabled manually by
the user or disabled at run time due to input size
requirements not satisfied by this run.
```

Appendix B

Acronyms

ANF Algebraic Normal Form. 45, 46, 48, 49, 52, 57, 59, 60

CV Cube Variable. 49–53, 59, 60, 66

DFT Discrete Fourier Transform. 29, 30

IoT Internet of Things. 1, 9

IV Initialization Vector. 4, 5, 46, 47, 51–53

LFSR Linear Feedback Shift Register. 5, 6, 8–10, 34

LWC Lightweight Cryptography. 1, 3, 10

NIST National Institute of Standards and Technology. 1, 2, 9, 10, 21, 46, 55

PRBG Pseudorandom Bit Generator. 13, 14

RBG Random Bit Generator. 2, 13, 14, 20

SNCDF Standard Normal Cumulative Distribution Function. 22

STS Statistical Test Suite. 2, 21, 46, 55

SV Superpoly Variable. 49–53, 59, 60, 66

TRBG True Random Bit Generator. 14

Appendix C

Contents of attached archive

- `implementation`
  - `data` ..... directory with testing data
  - `elephant160v2` ..... directory with Elephant160v2 implementation
  - `scripts` ..... directory with RBG tests, utilities and other scripts
  - `sts` ..... directory with NIST Statistical Test Suite implementation
- `text`
  - `thesis.pdf` ..... compiled text of the thesis in PDF
- `thesis` ..... directory with LaTeX sources of the thesis
- `README.md` ..... description of contents