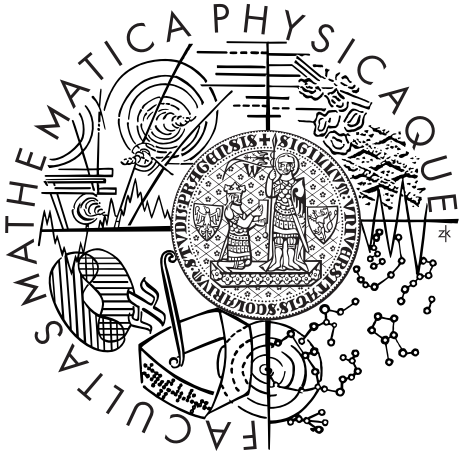
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Petr Nižnanský

Testování náhodnosti a použití statistických testů v kryptografii

Katedra algebry

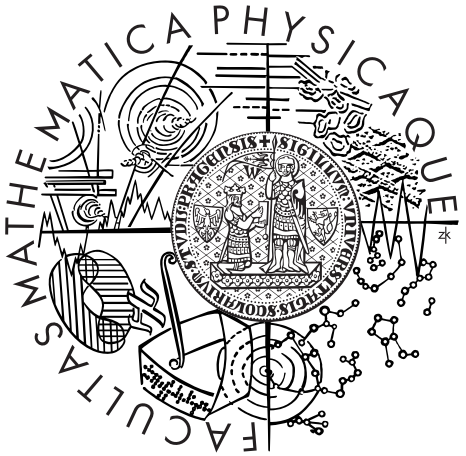
Vedoucí diplomové práce:	Mgr. Pavel Růžička, Ph.D.
Studijní program:	Matematika
Studijní obor:	Matematické metody informační bezpečnosti

Praha 2013

Charles University in Prague

Faculty of Mathematics and Physics

MASTER THESIS



Petr Nižnanský

Probability testing and application of statistical tests in cryptography

Department of Algebra

Supervisor of the master thesis:	Mgr. Pavel Růžička, Ph.D.
Study programme:	Mathematics
Specialization:	Mathematical Methods of Information Security

Prague 2013

I would like to thank Mgr. Pavel Růžička, Ph.D. for his tolerance and patience. My gratitude also belongs to all my friends, colleagues and teachers.

"A real friend is one who walks in when the rest of the world walks out."

"A scientist's aim in a discussion with his colleagues is not to persuade, but to clarify."

"A teacher is one who makes himself progressively unnecessary."

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University in Prague has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In Prague, July 2013 Petr Nižnanský

Název práce: Testování náhodnosti a použití statistických testů v kryptografii

Autor: Petr Nižnanský

Katedra: Katedra Algebry

Vedoucí diplomové práce: Mgr. Pavel Růžička, Ph.D.

Abstrakt: Pseudonáhodné generátory jsou jedním z nejdůležitějších témat kryptologie. Ke každé šifře musí být klíč generován náhodně, jinak můžeme ohrozit bezpečnost celé šifry. Dalším důvodem důležitosti pseudonáhodných generátorů je jejich úzký vztah k proudovým šifrám. V předložené práci nejdříve zformulujeme základní poznatky z matematické statistiky potřebné k testování náhodnosti. Popíšeme 8 klasických statistických testů. Představíme nový způsob předpovídání dalších bitů a ukážeme variaty dříve popsaných testů. S touto baterií testů otestujeme kadidáty na SHA-3, kteří se dostali do druhého kola. Dále je zde studován pojem senzitivity testů a jsou navrženy nové transformace.

Klíčová slova: náhodnost, statistické testování, SHA-3, next bit testy

Title: Probability testing and application of statistical tests in cryptography

Author: Petr Niznansky

Department: Department of Algebra

Supervisor: Mgr. Pavel Růžička, Ph.D.

Abstract: Pseudorandom generators belong to the primary focus of cryptology. The key to every cipher has to be generated at random, otherwise the security of the whole cipher is threatened. Another point of importance is the pseudorandom generators' close relationship to the stream ciphers. In this work, we first introduce statistical theory related to randomness testing. Then, we describe 8 classical statistical tests. We introduce a concept of next bit testing and derive variants of previous tests. Moreover, with this new battery of tests we examine the randomness of SHA-3 second round candidates and present the results. Also a sensitivity of tests is investigated and several useful transformations are shown.

Keywords: randomness, statistical testing, SHA-3, Next Bit Tests

Contents

Introduction **2**

1 Statistical approach **3**
1.1 Definitions . . . 3
1.2 Distributions . . . 5
1.2.1 The Central Limit Theorem . . . 6
1.3 Hypothesis testing . . . 7
1.4 Chi-squared test . . . 9

2 Statistical tests **11**
2.1 Frequency Test . . . 11
2.2 Frequency Test within a Block . . . 11
2.3 Runs Test . . . 12
2.4 Test for the Longest Run of Ones in a Block . . . 13
2.5 Binary Rank Matrix Test . . . 14
2.6 Random Walk Excursion Test . . . 14
2.7 Prime Number Test . . . 17
2.8 Irreducible Polynomial Test . . . 17

3 Sensitivity of Tests **19**

4 Next Bit Tests **23**
4.1 Next Bit Frequency Test . . . 25
4.2 Next Bit Frequency Test within a Block . . . 26
4.3 Next Bit Runs Test . . . 27
4.4 Next Bit Test for the Longest Run of Ones in a Block . . . 27
4.5 Next Bit Binary Rank Matrix Test . . . 29
4.6 Next Bit Random Walk Excursion Test . . . 30
4.7 Next Bit Prime Number Test . . . 31
4.8 Next Bit Irreducible Polynomial Test . . . 31
4.9 Next Bit Universal Test . . . 32

5 Results for SHA-3 second round candidates **33**

Conclusion **36**

Bibliography **37**

Appendix **38**

Introduction

Pseudorandom generators belong to the most important tools in cryptology. Beside their importance of their own, they are also important because of their close relationship to other areas of cryptology. One of the main parts of symmetric cryptography are the stream ciphers. Every stream cipher can be considered as a pseudorandom generator and vice versa. Moreover, differentiation between the stream cipher and the block cipher is not strict because a block cipher in the OFB mode can be considered as a stream cipher. Hence, these parts of cryptology are closely connected.

Another important part related to pseudorandom generators are the statistical tests which measure how much of an output of a pseudorandom generator is really random. Note that passing some of the statistical tests may not be sufficient proof of a good pseudorandom generator. But it is a tool for the cryptographers to gain at least some information about the generator/cipher. For many years, the Diehard [Mar95] test battery was considered the best among the statistical tests. In 2000, NIST completed its statistical test battery called the Statistical Test Suite, which gradually took over the prestige of the Diehard. The Statistical Test Suite was used, among others, to test the candidates for AES. Even though the tests were designed by such an authority as NIST, mistakes were found in Discrete Fourier Transform Test and Lempel-Ziv Test (see [KUH04]). Finally, we have to mention that new and particularly useful tests were proposed in [DEKS10] and applied to block ciphers: MARS, RC6, Rijndael (now AES), Serpent, and Twofish.

This thesis is organized as follows: In the first chapter, we describe the statistics that will be used in the thesis. The next chapter describes 8 statistical tests used for testing the randomness of sequences. In the third chapter the concept of sensitivity tests is investigated and several useful transformation are shown. In chapter 9 new tests are proposed and a new way of creating statistical tests is presented. The testing strategy as well as the results for SHA-3 second round candidates are presented in the last chapter. The complete results of testing are in the Appendix.

Chapter 1

Statistical approach

First, we have to summarize the probability and statistical theory that we will use in the thesis. This task will be done in this chapter.

1.1 Definitions

Definition 1.1. Let Ω be a set. A *σ-algebra* $\mathcal{A}$ on Ω is a nonempty set of subsets of Ω such that

1. $A \in \mathcal{A} \implies A^c \in \mathcal{A}$,
2. $A_n \in \mathcal{A}, n = 1, 2, \ldots \implies \cup_{n=1}^{\infty} A_n \in \mathcal{A}$.

Definition 1.2. The pair $(\Omega, \mathcal{A})$, where Ω is a set and $\mathcal{A}$ is σ-algebra, is called a *measurable space*. Given $(\Omega, \mathcal{A})$, each $A \in \mathcal{A}$ is called a *measurable set*.

Definition 1.3. A *measure* μ on $(\Omega, \mathcal{A})$ is a function $\mu : \mathcal{A} \to [0, \infty]$, with $\mu(\emptyset) = 0$, such that, for any sequence $(A_n : n \in \mathbb{N})$ of disjoint elements of $\mathcal{A}$,

$$\mu\left(\bigcup_n A_n\right) = \sum_n \mu(A_n).$$

Definition 1.4. The triple $(\Omega, \mathcal{A}, \mu)$, where $(\Omega, \mathcal{A})$ is measurable space and μ is measure on $\mathcal{A}$, is called a *measure space*.

Definition 1.5. If $\mu(\Omega) = 1$ then μ is a *probability measure* and $(\Omega, \mathcal{A}, \mu)$ is a *probability space*. We will use $\Pr[\cdot]$ instead of $\mu(\cdot)$.

Definition 1.6. Let $(\Omega, \mathcal{A})$ and $(Y, \mathcal{E})$ be measurable spaces. A function $f : \Omega \to Y$ is *measurable* if $f^{-1}(A) \in \mathcal{A}$ whenever $A \in \mathcal{E}$. Here $f^{-1}(A)$ is the inverse image of A by f

$$f^{-1}(A) = \{\, x \in \Omega : f(x) \in A \,\}.$$

Definition 1.7. Let $(\Omega, \mathcal{A}, \Pr)$ be a probability space and let $(Y, \mathcal{E})$ be a measurable space. A measurable function $X : \Omega \to Y$ is called *random variable in Y*.

Note 1.8. Y will be usually $\mathbb{R}$ (or more often in this thesis $\{\, 0, 1 \,\}$).

Definition 1.9. Let $(\Omega, \mathcal{A}, \Pr)$ be a probability space and let $(Y^n, \mathcal{E}^n)$ be a measurable space. A measurable function $\mathbf{X} : \Omega \to Y^n$ is called *random vector in* Y.

Note 1.10. The usual convention is consider $\mathbf{X}$ as a vector of random variables $\mathbf{X} = (X_1, X_2, \ldots, X_n)$.

Definition 1.11. A *discrete random variable* on $(\Omega, \mathcal{A}, \Pr)$ is a measurable function $X : \Omega \to Y$ such that the image of X is a countable subset of Y.

Definition 1.12. The *distribution function* (or *cumulative distribution function*) of a random variable $X : \Omega \to \mathbb{R}$, denoted by $F_X(x)$, is defined by

$$F_X(x) = \Pr[\{\, \omega : X(\omega) \leq x \,\}], \text{ for all } x.$$

Instead of $\Pr[\{\, \omega : X(\omega) \leq x \,\}]$ we will use $\Pr[X \leq x]$.

Definition 1.13. A *continuous random variable* on $(\Omega, \mathcal{A}, \Pr)$ is a measurable function $X : \Omega \to Y$ such that there exists a function f, called the *density function*, such that for all t we have

$$F(t) = \int_{-\infty}^{t} f(x)dx,$$

where F is distribution function of X.

Definition 1.14. Let X be a random variable defined on a probability space $(\Omega, \mathcal{A}, \Pr)$, then the *expected value* (or *mean*) of X, denoted by $\mathrm{E}\, X$, is defined as Lebesgue integral

$$\mathrm{E}\, X = \int_{\Omega} X d\Pr.$$

If integral does not converge then X does not have (finite) expected value.

Note 1.15. In case of a discrete random variable X an expected value of X can be expressed as $\mathrm{E}\, X = \int_{\Omega} X d\Pr = \sum_{x \in \mathbb{R}} x \cdot \Pr[X = x]$.

Definition 1.16. Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a random vector, then the *expected value* of $\mathbf{X}$, denoted by $\mathrm{E}\, \mathbf{X}$, is defined as

$$\mathrm{E}\, \mathbf{X} = (\mathrm{E}\, X_1, \mathrm{E}\, X_2, \ldots, \mathrm{E}\, X_n).$$

Definition 1.17. Let X be a random variable defined on a probability space $(\Omega, \mathcal{A}, \Pr)$ and has the expected value $\mathrm{E}\, X$, then the *variance* of X, denoted by $\mathrm{Var}(X)$, is defined as

$$\mathrm{Var}(X) = \mathrm{E}(X - \mathrm{E}\, X)^2.$$

Note 1.18. By expanding the variance, we can write: $\mathrm{Var}(X) = \mathrm{E}\, X^2 - 2(\mathrm{E}\, X)^2 + (\mathrm{E}\, X)^2 = \mathrm{E}\, X^2 - (\mathrm{E}\, X)^2$.

Note 1.19. If the random variable X has the mean μ and the variance σ^2, then the random variable $Y = (X - \mu)/\sigma$ has mean 0 and variance 1.

Definition 1.20. Let X and Y be a random variables defined on a probability space $(\Omega, \mathcal{A}, \Pr)$ and $\mathrm{E}\, X^2 < \infty, \mathrm{E}\, Y^2 < \infty$. The *covariance* $\mathrm{Cov}(X, Y)$ of X and Y is defined as

$$\mathrm{Cov}(X, Y) = \mathrm{E}(X - \mathrm{E}\, X)(Y - \mathrm{E}\, Y).$$

Definition 1.21. Let $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ be a random vector. The *covariance matrix* $\mathrm{Cov}(\mathbf{X})$ is defined as

$$\mathrm{Cov}(\mathbf{X}) = \begin{pmatrix} \mathrm{Var}(X_1) & \mathrm{Cov}(X_1, X_2) & \cdots & \mathrm{Cov}(X_1, X_n) \\ \mathrm{Cov}(X_2, X_1) & \mathrm{Var}(X_2) & \cdots & \mathrm{Cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \mathrm{Cov}(X_n, X_1) & \mathrm{Cov}(X_n, X_2) & \cdots & \mathrm{Var}(X_n) \end{pmatrix}.$$

1.2 Distributions

Definition 1.22 (Bernoulli distribution). We say that the random variable X has the *bernoulli distribution* with parameter $p \in [0, 1]$ if X has two values $0, 1$ and if

$$\Pr[X = 1] = p, \Pr[X = 0] = 1 - p.$$

Although the Bernoulli distribution is a simple distribution we will use it the most. All pseudo-random generators are operating with bits that are finally converted to a value we want to use, such as a random element from a finite set.

Definition 1.23 (Binomial distribution). We say that the random variable X has the *binomial distribution* ($\mathrm{Bi}(p,n)$) with parameters $p \in [0, 1]$ and $n \in \mathbb{N}$ if the possible values of X are $0, 1, 2, \ldots, n$ and

$$\Pr[X = k] = \binom{k}{n} p^k (1 - p)^{n-k}.$$

The Binomial distribution can be seen as n Bernoulli random variables summed together.

Definition 1.24 (Multinomial distribution). We say that the random vector $\mathbf{X} = (X_1, X_2, \ldots, X_k)$ has the *multinomial distribution* ($\mathrm{Mult}(n, \mathbf{p})$) with parameters $\mathbf{p} = (p_1, p_2, \ldots, p_k) \in [0, 1]^k$ and $n \in \mathbb{N}$ if every X_i has possible values $0, 1, \ldots, n$ and

$$\Pr[X_1 = x_1, X_2 = x_2, \ldots, X_k = x_k] = \frac{n!}{x_1! x_2! \ldots x_k!} p_1^{x_1} p_2^{x_2} \ldots p_k^{x_k},$$

where $n = x_1 + x_2 + \cdots + x_k$ and $1 = p_1 + p_2 + \cdots + p_k$.

Example 1.25. Assume we have a fair dice and we throw it 10 times. Here are the results: 2,4,5,1,3,2,1,4,5,6. Then $\mathbf{X} = (X_1, X_2, \ldots, X_6) = (2, 2, 1, 2, 2, 1)$, where X_i denotes the number of occurrences of i in a dice, has $\mathrm{Mult}(10, \mathbf{p})$, where $\mathbf{p} = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$.

Definition 1.26 (**Normal distribution).** A (continuous) random variable X has the *normal distribution* ($\mathrm{N}(\mu,\sigma^2)$) with the mean μ and the variance σ^2 if its probability density function is

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}.$$

If X is N(0,1), then X has a *standard normal distribution.* A standard normal distribution function is denoted by $\Phi(x)$.

Note 1.27. If $\mathbf{X} = (X_1, X_2, \ldots, X_n)$ is a random vector, where X_i is independent and has a normal distribution. Let $\mathbf{X}$ has mean $\boldsymbol{\mu}$ and covariance matrix Σ, then we will say $\mathbf{X}$ is $\mathrm{N}(\boldsymbol{\mu}, \Sigma)$.

Definition 1.28 (**$\boldsymbol{\chi^2}$-distribution).** Let $m \geq 1$ be an integer. A (continuous) random variable X has the *χ^2-distribution with m degrees of freedom* (denoted by χ^2_m) if its probability density function is defined by

$$f(x) = \begin{cases} \frac{1}{\Gamma(m/2)2^{m/2}} x^{(m/2)-1} e^{-x/2}, & 0 \leq x < \infty, \\ 0, & x < 0, \end{cases}$$

where Γ is the gamma function[1]. The mean and the variance of this distribution are $\mu = m$ and $\sigma^2 = 2m$.

Note 1.29. The χ^2-distribution arise when we sum squares of independent random variables with a standard normal distribution. More precisely: let $Y_1, \ldots, Y_m$ be independent random variables with distribution N(0,1). We define new random variable X as $X = \sum_{j=1}^{m} Y_i^2$ then X has χ^2-distribution with m degrees of freedom.

1.2.1 The Central Limit Theorem

Definition 1.30. A sequence $X_1, X_2, \ldots$ of random variables *converge in distribution* to a random variable X if

$$\lim_{n\to\infty} F_n(x) = F(x),$$

for every x at which F is continuous. Here F_n and F are the cumulative distribution functions of random variables X_n and X, respectively. Notation: $X_i \xrightarrow{d} X$.

Theorem 1.31 (**Central Limit Theorem).** *Let $X_1, X_2, \ldots$ be independent, identically distributed random variables with mean μ and finite nonzero variance σ^2, then*

$$Z = \frac{1}{\sqrt{n}} \sum_{i=1}^{n} \left(\frac{X_i - \mu}{\sigma} \right) \xrightarrow{d} \mathrm{N}(0,1).$$

[1]The gamma function is $\Gamma(t) = \int_0^\infty x^{t-1} e^{-x} dx$, for $t > 0$.

1.3 Hypothesis testing

By hypothesis testing we understand a process of deciding about the truth or falsity of hypotheses based on experimental evidence. If we want to decide with the knowledge of only some data (random sample), our decision may not be perfectly true but is true with a possible error. A null hypothesis (H_0) means that the tested sequence (or mapping) is random and the alternative hypothesis (H_a) means that the sequence (or mapping) is not random. During the hypothesis testing we either reject or not reject the null hypothesis. We have to point out that not rejecting is not the same as accepting. The true meaning of not rejecting the null hypothesis is that we don't have an evidence (based on a random sample) of non-randomness. With that in mind, the usual process of using hypothesis testing is taking as a null hypothesis the claim which we want to find an evidence against. This is the reason why we always take as a H_0 that the sequence (or mapping) is random.

There are four possible outcomes of our decision:

True situation	Conclusion	
	Not reject H_0	Reject H_0
Data/mapping is random (H_0 is true)	No error	Type I error
Data/mapping is not random (H_a is true)	Type II error	No error

The probability of a Type I error is called the *level of significance* of the test and is usually denoted as α.

The test itself is based on two components:

1. Random variable $T_n = T_n(X_1, \ldots, X_n)$, which is the function of a random sample (called the *test statistic*).

2. Set $\mathcal{C} \subset \mathbb{R}$, which is called the critical region.

The rule how to decide about a hypothesis is as follows:

- if $T_n \in \mathcal{C}$, then we reject a null hypothesis.
- if $T_n \notin \mathcal{C}$, then we can't reject a null hypothesis.

We create the test statistic depending on what event we want to observe. For example if our bit stream is $(X_1, X_2, \ldots, X_n)$, our test statistic could be $T_n = \sum_{i=1}^{n} X_i$. This test statistic counts the number of ones. If we set $\alpha = 0.05$ we can choose any $K \subseteq \{\, 0, 1, 2, \ldots, n \,\}$ such that $\sum_{i \in K} \Pr[X = i] = 0.05$. There are several examples of critical regions in Figure 1.1.

But almost always is a critical region taken with condition that probability of statistic T_n (under null hypothesis) is the smallest. In our example the (both sided) critical region is in Figure 1.2.

The two most common values for α is $\alpha = 0.01$ and $\alpha = 0.05$. The explanation I found is as follows. If you put your α to be (for example) 0.023 then your hypothesis would be suspect of being rejected on level 0.022 and this is the reason why you have chosen $\alpha = 0.023$.

The solution to this problem is *p-value*, which is the estimated probability of rejecting the null hypothesis, when that hypothesis is true. Rather than selecting the critical region ahead of time, the p-value of a test can be reported and the reader then makes a decision.

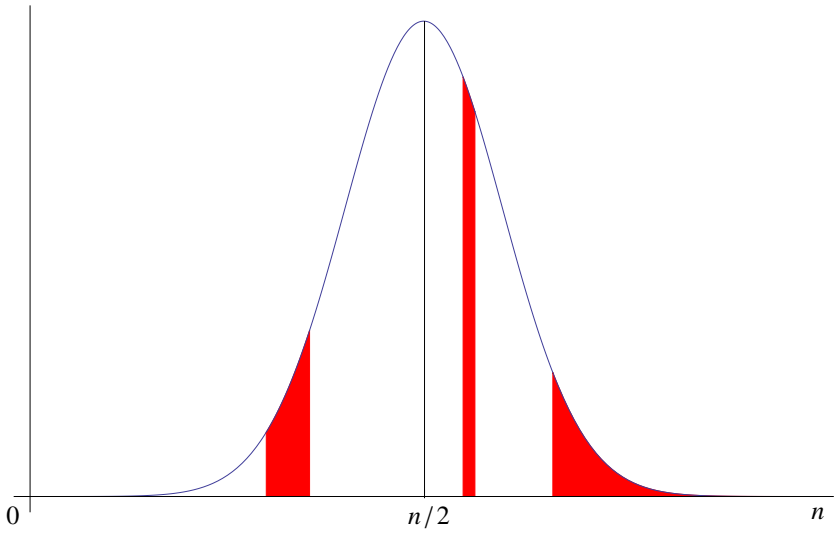


Figure 1.1: Example of several critical regions.

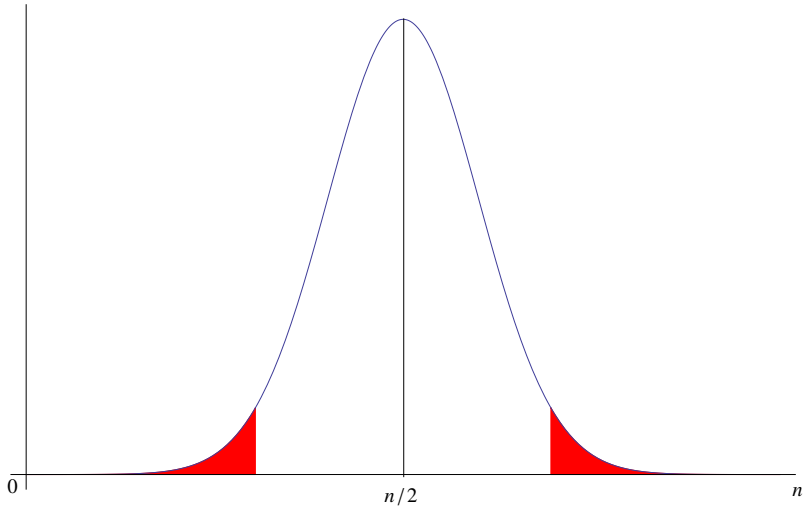


Figure 1.2: Classical critical region on both tails.

Example 1.32. Assume we have a sequence of ten bits $(0,1,0,1,1,0,1,1,1,0) = (X_1, X_2, \ldots, X_{10})$ and our test statistic is $T_{10} = \sum_{i=1}^{10} X_i$. Under null hypothesis distribution of T_{10} is binomial distribution with $p = 1/2$ and $T_{10} = 6$. Our p-value is

$$\begin{aligned} p\text{-value} &= (\Pr[T_{10} = 0] + \Pr[T_{10} = 10]) + (\Pr[T_{10} = 1] + \Pr[T_{10} = 9]) \\ &\quad + (\Pr[T_{10} = 2] + \Pr[T_{10} = 8]) + (\Pr[T_{10} = 3] + \Pr[T_{10} = 7]) \\ &\quad + (\Pr[T_{10} = 4] + \Pr[T_{10} = 6]) \\ &= 1 - \Pr[T_{10} = 5] = 1 - \binom{10}{5} 2^{-10} = 0.7539 \end{aligned}$$

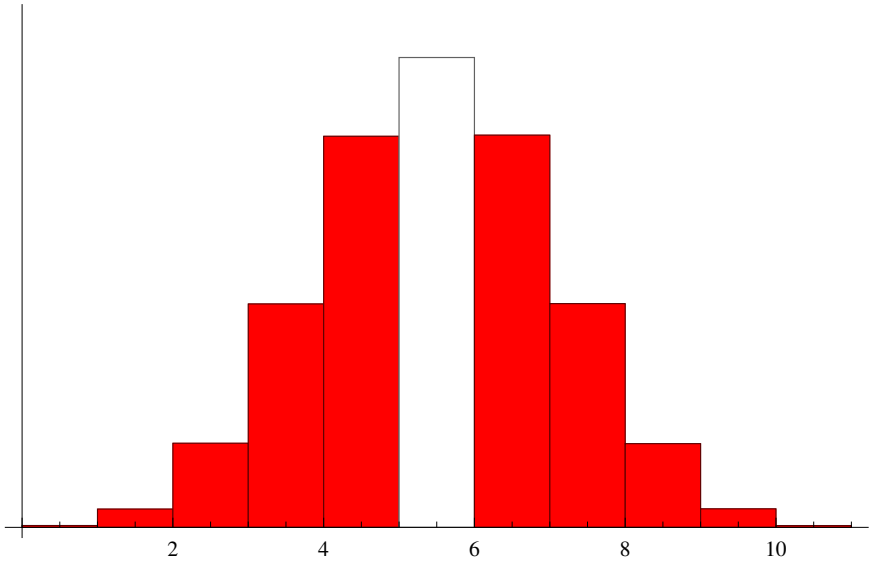


Figure 1.3: Critical region from example 1.32.

p-value is 0.7539 and this means we can't reject hypothesis about randomness of the data on any reasonable level.

1.4 Chi-squared test

Chi-squared test (or χ^2 test) is a test for deciding whether the empirical frequencies $X_1, \ldots, X_k$ of events $A_1, \ldots, A_k$ have approximately mean values $np_1^0, \ldots, np_k^0$, where probabilities $p_1^0, \ldots, p_k^0$ are computed according some probabilistic model. A null hypothesis is that the probability of events $A_1, \ldots, A_k$ are $p_1^0, \ldots, p_k^0$ and test statistic is:

$$X^2 = \sum_{i=1}^{k} \frac{(X_i - np_i^0)^2}{np_i^0}. \tag{1.1}$$

X^2 has approximately χ^2-distribution with $k-1$ degrees of freedom. We reject null hypothesis at α significance level in case $X^2 \geq \chi^2_{k-1}(\alpha)$, which can be interpreted as our test statistic not having χ^2-distribution and the probability of frequencies differing from $p_1^0, \ldots, p_k^0$.

Note 1.33. We consider only a one-sided test because a deviation on the left side can be interpreted as a sign that our data is fitting our model too well, which is actually desired.

Note that χ^2 test is asymptotic, hence n (the size of random sample) has to be large enough. The usual condition in literature is $np_i^0 > 5$ for all $i = 1, \ldots, m$. It is more a "rule of thumb" than anything else. Another condition is Yarnold's criterion on data. Yarnold's criterion states that data should satisfy

$$np_i^0 > 5q/k \text{ for all } i = 1, 2, \ldots, k \text{ where } k \geq 3$$

and q is number of events where $np_i^0 < 5$. The bounds on errors are estimated in [Yar07] as:

$$|\text{error}| \leq \frac{0.031}{\min(np_i^0)} \text{ when } \alpha = 0.01$$

$$|\text{error}| \leq \frac{0.047}{\min(np_i^0)} \text{ when } \alpha = 0.05.$$

Note that these bounds are results of numerical investigation of accuracy of approximations and are not theoretical bounds. The conclusion is that we have to be careful about deciding about our hypothesis when p-value is close to the level of significance of a test.

Example 1.34. Suppose we have a six-sided dice and we want to test if the dice is unbiased and random (each face has equal probability). To test this hypothesis, we roll the dice 300 times and observe the frequency of occurrence of each of the faces. We expect that the number on each face will occur 50 times. However, suppose we observe frequencies of occurrence as follows:

Face value	1	2	3	4	5	6
Occurrence	44	64	57	38	55	42

Probability of each face is 1/6 and we roll the dice for 300 times, hence $\mathbf{X} = (X_1, X_2, \ldots, X_6) = (44, 64, 57, 38, 55, 42)$ has $\text{Mult}(300, \mathbf{p})$, where $\mathbf{p} = (1/6, 1/6, 1/6, 1/6, 1/6, 1/6)$. Now, we can test our hypothesis and compute p-value using the multinomial distribution or we can use χ^2-distribution. We use χ^2 test and the test statistic is:

$$\begin{aligned} X^2 &= \sum_{i=1}^{k} \frac{(X_i - np_i^0)^2}{np_i^0} = \sum_{i=1}^{6} \frac{(X_i - 50)^2}{50} \\ &= \frac{(44-50)^2}{50} + \frac{(64-50)^2}{50} + \frac{(57-50)^2}{50} \\ &+ \frac{(38-50)^2}{50} + \frac{(55-50)^2}{50} + \frac{(42-50)^2}{50} \\ &= 514/50 = 10.28. \end{aligned}$$

The p-value is $\Pr[\chi_5^2 \geq 10.28] = 0.06768$ and the number of degrees of freedom is five. We can reject our hypothesis on 0.06768 significance level, which is greater than 0.05 (usual value of significance level).

Chapter 2

Statistical tests

In this chapter we describe several statistical tests with examples. In all tests, we will concentrate on testing of short sequences (several hundreds bits in general and 512 for concrete examples). The number 512 naturally arises when considering modern block ciphers and hash functions which has output 128, 256 or 512 bits and we can expect that in future 512 will dominate.

Test statistic will be denoted as X and bit sequence will be $(X_1, X_2, \ldots, X_n)$, where $X_i \in \mathbb{Z}_2$ for all i.

2.1 Frequency Test

Frequency Test is a basic test for testing of randomness and should be included in every battery of tests. Test compares the Hamming weight of the sequence with the expected weight $(n/2)$ of random sequence. The test statistic is

$$X = \sum_{i=1}^{n} X_i$$

and since every X_i has Bernoulli distribution, the X has Binomial distribution, therefore

$$\Pr[X = k] = \frac{\binom{n}{k}}{2^n}.$$

As is stated by the NIST [BRS+10], this test should be used in the first place. If a sequence does not pass this test, it will probably fail in many others.

2.2 Frequency Test within a Block

Frequency Test within a Block divides n-bit sequence into m-bit blocks and determine whether the frequency of ones in an m-bit block is approximately $m/2$, as would be expected in a random sequence. In our case $n = 512$, we take $m = 8$ and determine probabilities of four cases for blocks of size 8, see Table 2.1.

In test we will count number of blocks having weight 3, 4, 5 or different and finally χ^2 test is applied.

More precisely let n be the length of sequence and m be the length of block (sequence is divided into $k = \lfloor \frac{n}{m} \rfloor$ blocks and the remaining bits are discarded), $\pi_1 =$ number of blocks with weight 0,1,2,6,7 or 8, $\pi_2 =$ number of blocks with weight 3,

Hamming weight of 8 bit sequence	Probability
0, 1, 2, 6, 7, 8	$p_1 = 0.2890625$
3	$p_2 = 0.21875$
4	$p_3 = 0.2734375$
5	$p_4 = 0.21875$

Table 2.1: Probabilities of weights of 8-bit sequence.

π_3 = number of blocks with weight 4, π_4 = number of blocks with weight 5. The test statistic is

$$X^2 = \sum_{i=1}^{4} \frac{(\pi_i - kp_i)^2}{kp_i}$$

and has χ^2 distribution with 3 degrees of freedom.

Note that by the Frequency Test within a Block a slightly different test is usually meant (see [BRS+10], 2-2). This is the modifications which, I believe, would also deserve the same name.

2.3 Runs Test

A run is an uninterrupted maximal sequence of identical bits. In the Run Test the number of runs in the sequence is compared with the expected number of runs in a random sequence.

Example 2.1. A sequence 0010111001 has 6 runs: 00, 1, 0, 111, 00, 1.

The purpose of the Runs test is to observe whether ones and zeros are not changing too fast (like sequence 0101010101) or too slow (like 0000011111). Here, we present a slightly different modification of what is usually considered as the Runs test. The usual Runs test computes the distribution of runs for random sequence with weight w, see [SDEK10] or for a long sequences [BRS+10] 2-3. Here we derive the probability distribution for a sequence which has the length n and k runs, with no requirement for the weight of the sequence.

Denote a and b a different bits and the test statistic (number of runs) as X. For $k = 1$ we have two sequences: $aa \dots a$ and $bb \dots b$, hence $\Pr[X = 1] = \frac{2}{2^n}$.

For $k = 2$ the first and last bit is determined: $acc \dots cb$ where c it either a or b, hence we have $n-1$ possibilities where we switch from a to b, so $\Pr[X = 2] = 2\frac{n-1}{2^n}$.

Now for an arbitrary k we can imagine our sequence with partition after every bit: $a|c|c| \dots |b$. Again the first and the last bit are determined and there are $\binom{n-1}{k-1}$ possibilities how to choose the partition where we switch from ones to zeros or from zeros to ones, hence

$$\Pr[X = k] = 2\frac{\binom{n-1}{k-1}}{2^n}, \text{for } k = 1, 2, \dots, n.$$

2.4 Test for the Longest Run of Ones in a Block

The Test for the Longest Run of Ones in a Block divides n-bit sequence into blocks with the length of m, determines the longest run of ones in each block and compares it with expected values for a random sequence with χ^2 test. Here we chose $m = 8$ and determined the number of sequence having the longest run $0, 1, 2, \ldots, 8$. The results are in Table 2.2.

Longest run	Number of sequences
0	1
1	54
2	94
3	59
4	28
5	12
6	5
7	2
8	1

Table 2.2: Number of sequences with the longest run.

We must conclude the classes in [BRS+10] 3-4 are well chosen and we use them, see Table 2.3.

Longest run	Probability
0 or 1	55/256
2	94/256
3	59/256
4 or more	48/256

Table 2.3: Classes from [BRS+10] 3-4.

Let n be the length of the sequence and m be the length of block (sequence is divided into $k = \lfloor \frac{n}{m} \rfloor$ blocks and the remaining bits are discarded), π_1 = number of blocks with longest run of length 0 or 1, π_2 = number of blocks with longest run of length 2, π_3 = number of blocks with longest run of length 3, π_4 = number of blocks with longest run of length 4 or more. We denote $p_1 = \Pr[X \leq 1] = \frac{55}{256}$, $p_2 = \Pr[X = 2] = \frac{94}{256}$, $p_3 = \Pr[X = 3] = \frac{59}{256}$, $p_4 = \Pr[X \geq 4] = \frac{48}{256}$.

The test statistic is

$$X^2 = \sum_{i=1}^{4} \frac{(\pi_i - kp_i)^2}{kp_i},$$

which has χ^2 distribution with 3 degrees of freedom.

2.5 Binary Rank Matrix Test

The Binary Rank Matrix Test was first proposed by George Marsaglia [Mar95] in 1995. Binary matrices are formed from a sequence and their ranks are computed. The test measures whether the frequencies of ranks meet the expectations about a random sequence.

Suppose we have a binary matrix $m \times m$ and let X be the rank of a matrix. We will divide it into three cases: $\Pr[X = m]$, $\Pr[X = m-1]$ and $\Pr[X < m-1]$. In the case $\Pr[X = m]$ all rows are independent, then there are $2^m - 1$ choices for the first row, $2^m - 2$ for the second row, $2^m - 4$ for third row, $\ldots$, $2^m - 2^{i-1}$ for i^{th} row, $\ldots$, $2^m - 2^{m-1}$ for the last row, hence

$$\Pr[X = m] = \frac{\prod_{i=1}^{m}(2^m - 2^{i-1})}{2^{m^2}}$$

Now in the case $\Pr[X = m-1]$, first $m-1$ rows are independent so they are chosen in the same way as the first case and the last row has to be chosen that is linearly dependent with $m-1$ previous. If the i^{th} row is linearly dependent, there are 2^{i-1} choices, therefore there are $1 + 2 + 2^2 + \cdots + 2^{m-1} = 2^m - 1$ choices for the linear dependent row, hence

$$\Pr[X = m-1] = \frac{\prod_{i=1}^{m-1}(2^m - 2^{i-1})}{2^{m^2}}(2^m - 1).$$

For a sequence of length 512 we choose $m = 4$ and exact probabilities are given in Table 2.4. After all ranks are determined the χ^2 test is used. Hence the test statistic is

$$X^2 = \sum_{i=1}^{3} \frac{(\pi_i - kp_i)^2}{kp_i},$$

where π_1 is a number of matrices with full rank, π_2 is a number of matrices with rank $m-1$ and π_3 are the rest of matrices.

Rank	$< m-1$	$m-1$	m
Probability	$p_1 = 0.3076171875$	$p_2 = 0.576782226562$	$p_3 = 0.115600585938$

Table 2.4: Probabilities in the Binary Matrix Test for $m = 4$.

2.6 Random Walk Excursion Test

The Random Walk Excursion Test for short sequences was proposed in [FS06] but same test for long sequences was in [BRS+10] (all is meant in cryptographic context).

The Random Walk Excursion Test is test related to a random walk. A sequence of zeros and ones is transformed to a random walk by starting at point $[x_0, y_0] = [0, 0]$ and for every one in the sequence, we move one step up and one step ahead ($[x_{i+1}, y_{i+1}] = [i+1, y_i + 1]$) and for every zero in the sequence, we move one step down and one step ahead ($[x_{i+1}, y_{i+1}] = [i+1, y_i - 1]$).

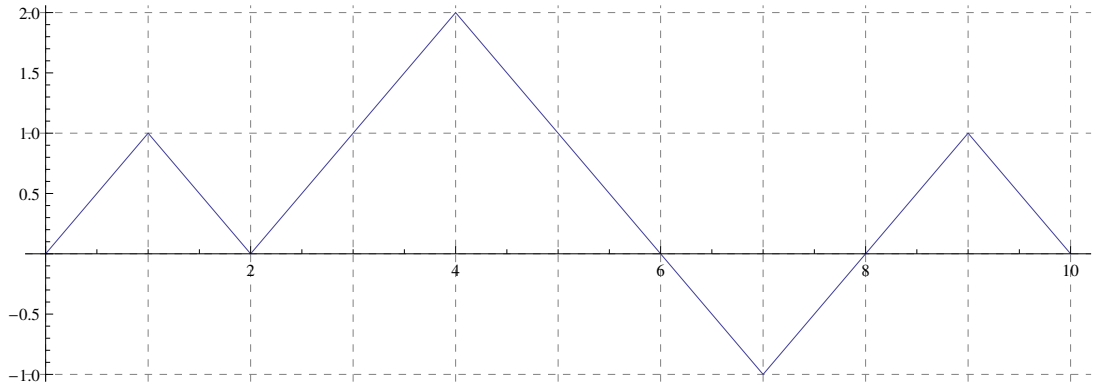


Figure 2.1: Random walk from Example 2.2.

Example 2.2. A sequence: 1011000110 is then transformed into the following random walk in Figure 2.1.

We created a sequence of pairs $[x_i, y_i]$ and the excursion of length k is a subsequence which started with $y_i = 0$, ended with $y_{i+k} = 0$ and for all $j \in \{1, 2, \ldots, k-1\} : y_{i+j} \neq 0$, hence a sequence from Example 2.2 has 4 excursions. The Random Walk Excursion Test measures whether a sequence produces a number of excursions that is expected from a random sequence.

In derivation of probability of a sequence of length n having k excursions, we first have to derive the probability of having excursion of length l. It is clear that l is even so $l = 2j$. We first ask how many paths there are from point $S = [x_0, y_0] = [0, 0]$ to point $F = [x_l, y_l] = [l, 0]$ without having any $y_i < 0$ for $i \in \{1, 2, \ldots, l-1\}$. The number of all paths are $\binom{2j}{j}$ because it is the number of ways how to choose j ones in sequence of length $2j$. Now every wrong path has i where $y_i < 0$ for the first time. From this point on, we draw a different random walk, every time when we should move up, we move down and vice versa, see Figure 2.2.

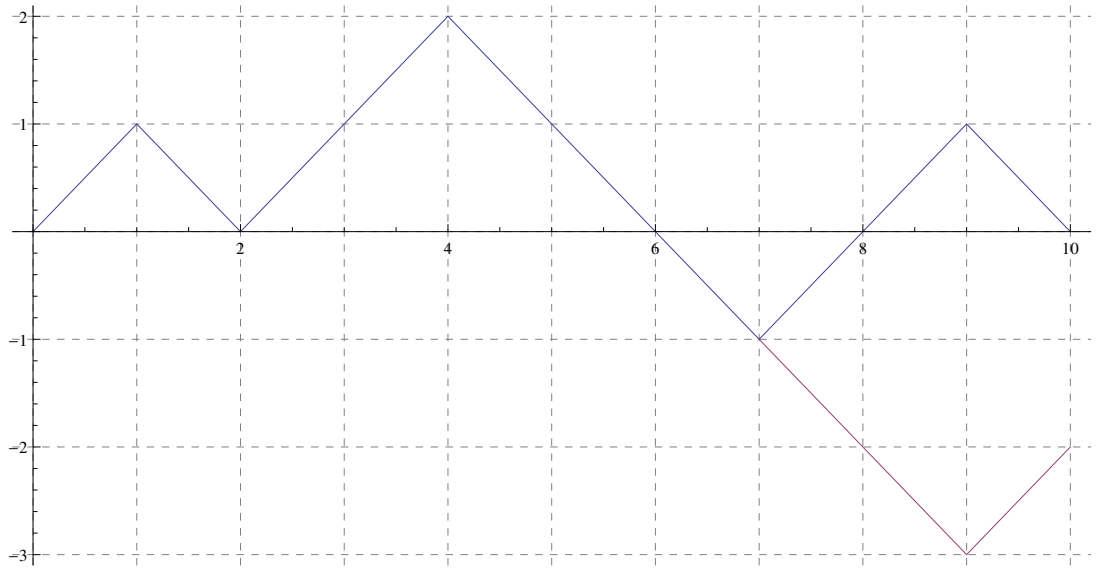


Figure 2.2: Modified random walk from Example 2.2.

It should be clear that for every wrong path we always end in the point $[x_l, y_l] = [l, -2]$. Conversely, every path from $[0,0]$ to $[l,-2]$ can be transformed to a wrong path, hence the number of wrong paths is equal to the number of paths from $[0,0]$ to $[l,-2]$, which is $\binom{2j}{j-1}$. So the number of paths from S to F without having any $y_i < 0$ for $i \in \{1, 2, \ldots, l-1\}$ is

$$C_j = \binom{2j}{j} - \binom{2j}{j+1} = \frac{(2j)!}{j!j!} - \frac{(2j)!}{(j+1)!(j-1)!} = \frac{(2j)!}{(j+1)!j!} = \frac{1}{j+1}\binom{2j}{j},$$

which is the Catalan number and relates many other combinatorial problems. Now to our original problem, which is the probability of excursion of length $2j$. It is

$$\Pr[\text{excursion of lenght } 2j] = P_{2j} = \frac{C_{j-1}}{2^{2j-1}},$$

because the first and the last step is determined (we move up or down) and from that point, there are C_{j-1} random walks without having $y_i = 0$. We denote $R_{2j} = 1 - \sum_{i=1}^{j} P_{2i}$, which is the probability of not having excursion in a random walk of length $2j$.

Now, the probability of having k excursions in a sequence of length n is

$$\Pr[X = k] = \sum_{\substack{2j_1+2j_2+\cdots+2j_k+2i=n \\ j_m>0; 1\le m\le k}} P_{2j_1} P_{2j_2} \ldots P_{2j_k} R_{2i}. \tag{2.1}$$

Note that these probabilities can be difficult to compute for larger n. In the Appendix are tables with probabilities for $n = 8, 16$ and 32.

Therefore in case $n = 512$, it would be very difficult to compute probabilities for such a long sequence. I decided to split a sequence into 16 bit blocks and count the excursions in these blocks. In the end the χ^2 test is used. The test statistic is

$$X^2 = \sum_{i=1}^{5} \frac{(\pi_i - kp_i)^2}{kp_i},$$

where p_i is from Table 2.5 and $\pi_1, \pi_2, \ldots, \pi_5$ are numbers of blocks with 0,1,2,3 and 4-8 excursions.

Excursions	Probability
0	$p_1 = 0.196380615234$
1	$p_2 = 0.196380615234$
2	$p_3 = 0.183288574219$
3	$p_4 = 0.157104492188$
4-8	$p_5 = 0.266845703$

Table 2.5: Probabilities in the Random Walk Excursion Test for 16 bit block.

2.7 Prime Number Test

The Prime Number Test is a proposal of a new test, which is related to integers. It means that our bit sequence is transformed into integers a these integers are then tested. This test measures if a sequence produces prime numbers with frequency that is expected from a random sequence.

Let n be the length of sequence and m is the length of block (sequence is divided into $k = \lfloor \frac{n}{m} \rfloor$ blocks and the remaining bits are discarded). Each block is converted to number between 0 and $2^m - 1$ and is determined whether is prime or not. We will create a new sequence from old one as follows: if the number is prime we will append 1 and 0 otherwise.

Example 2.3. Let $M = 0111011001001$ be the tested sequence, hence $n = 13$. We choose $m = 3$ and convert M: $011_2 = 3$, $101_2 = 5$, $100_2 = 4$, $100_2 = 4$ and the last 1 is discarded. Then we convert 3,5,4,4 into 1100.

Let $p = \frac{\text{number of primes between 0 and } 2^m-1}{2^m}$, then we have just created a sequence with the Bernoulli distribution with parameter p, hence sum of these bits has the Binomial distribution Bi(p,k).

For large values of m one can use the Prime Number Theorem which states that the number of primes less than or equal to $2^m - 1$ can be approximated by $\frac{2^m-1}{\log(2^m-1)} \approx \frac{2^m-1}{m}$. But there are two practical issues: 1) We have to estimate the error in the Prime Number Theorem and 2) determining primality of large numbers can be done quickly only with probabilistic algorithms. With that in mind, we recommend to use this test only with a small values of m, where the primality of numbers less than 2^m can be done by the trivial division or with a table of prime numbers less than 2^m.

2.8 Irreducible Polynomial Test

The Irreducible Polynomial Test is a proposal of a new test, which is related to polynomials over a finite field. The bit sequence is transformed into polynomials and these polynomials are then tested. This test measures if a sequence produces irreducible polynomials over a finite field with a frequency which is expected from a random sequence.

Let $\mathbb{F}_q$ be a finite field, n is the length of sequence and $m = q^l$ is the length of block (sequence is divided into $k = \lfloor \frac{n}{m} \rfloor$ blocks and the remaining bits are discarded). Each block is converted to a polynomial of degree less than l and is determined whether is irreducible or not. We will create a new sequence from old one as follows: if the polynomial is irreducible, we will append 1 and 0 otherwise.

Example 2.4. Let $M = 011101010101$ be the tested sequence. We choose a finite field $\mathbb{F}_2$, $m = 3$ and convert M: $011 = x + 1$, $101 = x^2 + 1$, $010 = x$, $101 = x^2 + 1$. Then we convert $x + 1, x^2 + 1, x, x^2 + 1$ into 1010.

Let $p = \frac{\text{number of irreducible polynomials of degree less than } l}{2^m}$, then we just created a sequence with the Bernoulli distribution with parameter p, hence the sum of these bits has the Binomial distribution Bi(p,k).

In contrast to the Prime Number Test, here we can enumerate the number of irreducible polynomials precisely. Let $M_{q,l}$ be the number of all irreducible polynomials of degree l over finite field $\mathbb{F}_q$, then from the Möbius inversion formula, we have

$$M_{q,l} = \frac{1}{l} \sum_{k|q} \mu(q/k) q^k,$$

where $\mu : \mathbb{N} \to \{ -1, 0, 1 \}$ is the Möbius function defined

$$\mu(n) = \begin{cases} 1 & \text{if } n = 1 \\ (-1)^k & \text{if } n \text{ is product of } k \text{ disting prime numbers} \\ 0 & \text{if } p^2 | n \text{ for some prime } p. \end{cases}$$

Hence the number of irreducible polynomials of degree less than l is $\sum_{i=1}^{l-1} M_{q,i}$.

Note 2.5. Obviously, this test and the Prime Number Test are special cases of a more universal test. In the more universal variant, we specify a set of m-bit long blocks/sequences P and for every occurrence of m-bit block in the original sequence, we append 1 and 0 otherwise. In the Prime Number Test, we choose P as a binary notation of prime numbers and in the Irreducible Polynomial Test binary notation of irreducible polynomials over finite field.

Chapter 3

Sensitivity of Tests

The purpose of this chapter is to investigate the transformations that could be used before testing, to modify the original sequence to a new one and then apply tests. If we set α (the type I error) to fix the value, we can consider a statistical test T as a deterministic Turing machine $T : \{0,1\}^n \to \{\text{accept}, \text{reject}\}$, that will decide about randomness of sequence of a length n. The test T divides a set $\{0,1\}^n$ into two different subsets:

$$S_a = \{ s \in \{0,1\}^n \,|T(s) = \text{accept} \}$$

$$S_r = \{ s \in \{0,1\}^n \,|T(s) = \text{reject} \}.$$

Trivially $S_a \cup S_r = \{0,1\}^n$ and note that $\alpha = |S_r|/2^n$. In [TDB08] a new concept of *sensitivity* is introduced.

Definition 3.1 ([TDB08])**.** Consider a randomness test T and a one-to-one transformation $\sigma : \{0,1\}^n \to \{0,1\}^n$, T is said to be *invariant* under σ if for any $s \in \{0,1\}^n : T(s) = T(\sigma(s))$.

The authors suggested that if $T(.)$ and $T(\sigma(.))$ are statistically independent, then $T(\sigma(.))$ can be added to the test suite as a new test.

The authors have investigated the following transformations:

- *Complementation*: bitwise XOR with 1, that is $\sigma_c(X_1, X_2, \ldots, X_n) = (X_1 \oplus 1, X_2 \oplus 1, \ldots, X_n \oplus 1)$.
- *l-Rotation*: circular shift, that is $\sigma_{l-rot}(X_1, X_2, \ldots, X_n) = (X_{l+1}, \ldots, X_n, X_1, \ldots, X_l)$.
- i^{th} *Bit flip*: flipping i^{th} bit of sequence, that is $\sigma_{f_i}(X_1, X_2, \ldots, X_n) = (X_1, \ldots, X_i \oplus 1, \ldots, X_n)$.
- *Reversing*: considering the sequence backwards, that is $\sigma_{rvs}(X_1, X_2, \ldots, X_n) = (X_n, X_{n-1}, \ldots, X_1)$.
- l^{th} *Derivative*: summation of the sequence and its l-bit rotation, that is $\sigma_{d_l}(X_1, X_2, \ldots, X_n) = (X_1 \oplus X_{l+1}, X_2 \oplus X_{l+2}, \ldots, X_n \oplus X_l)$. But this transformation is not one-to-one so a little modification is done: $\sigma_{d_l}(X_1, X_2, \ldots, X_n) = (X_1 \oplus X_{l+1}, X_2 \oplus X_{l+2}, \ldots, X_{n-1} \oplus X_{l-1}, X_n)$.

The authors divided these transformations for each test into three categories:

- Test T is invariant under σ.
- Transformation has small effect on results.
- $T(.)$ and $T(\sigma(.))$ are statistically independent.

Here a different view is presented. For every test T and every transformation σ a table of $|S_r \cap S_{r,\sigma}|/|S_r|$ is given, where $S_r = \{ s \in \{ 0,1 \}^n \, | T(s) = \text{reject} \}$ and $S_{r,\sigma} = \{ s \in \{ 0,1 \}^n \, | T(\sigma(s)) = \text{reject} \}$.

Furthermore another way of creation of transformations is given. We can consider a sequence of length n divided into m-bit blocks and then apply above-mentioned transformations. With that in mind, a new transformations are considered:

- *m-l-Rotation*: circular shift in m bit blocks (suppose m divides n), that is $\sigma_{m-l-rvs}(X_1, X_2, \ldots, X_m, X_{m+1}, \ldots, X_n) = (X_{l+1}, X_{l+2}, \ldots, X_1, X_m, \ldots, X_l, X_{m+l+1} \ldots, X_{m+l}, \ldots, X_{n-m+l})$.
- *m-Reversing*: considering every m bit block of the sequence backwards (suppose m divides n), that is $\sigma_{m-rvs}(X_1, X_2, \ldots, X_m, X_{m+1}, \ldots, X_n) = (X_m, X_{m-1}, \ldots, X_1, X_{2m}, \ldots, X_{m+1}, \ldots, X_n, \ldots, X_{n-m+1})$.
- *m-l-Derivation*: on every m bit block is l-Derivation applied, that is $\sigma_{m-l-der}(X_1, X_2, \ldots, X_m, X_{m+1}, \ldots, X_n) = (X_1 \oplus X_{l+1}, \ldots, X_{m-1} \oplus X_{l-1}, X_m, X_{m+1} \oplus X_{m+l+1}, \ldots, X_n)$.

Moreover a new transformation *avalanche XOR* is considered:

- *avalanche XOR*: in the i^{th} place is xor of all previous bits with the i^{th} one, that is $\sigma_{xor}(X_1, X_2, \ldots, X_n) = (X_1, X_1 \oplus X_2, \ldots, \oplus_{i=1}^{n} X_i)$.

And its m bit block variant:

- *m-avalanche XOR*: same as avalanche XOR but in m bit blocks, that is $\sigma_{xor}(X_1, X_2, \ldots, X_m, X_{m+1}, \ldots, X_n) = (X_1, X_1 \oplus X_2, \ldots, \oplus_{i=1}^{m} X_i, X_{m+1}, X_{m+1} \oplus X_{m+2}, \ldots, \oplus_{i=m+1}^{2m} X_i, \ldots, \oplus_{i=n-m+1}^{n} X_i)$.

Testing was done for all sequences of length $n = 20$ and α is set close to 0.01. While we are in a discrete case, we can't set α exactly to 0.01. The exact value of α is given in Table 3.1.

Tests	Frequency	Runs	Longest Run	RW Excursion
α	0.0118	0.0118	0.0127	0.0225

Table 3.1: Type I error used in tests.

Tests	Frequency	Runs	Longest Run	RW Excursion
Complementation	1	1	0.0021	1
i^{th} bit flip	0.3744	0.3744	0.5924	0.1076

Table 3.2: Sensitivity of randomness test for Complementation i^{th} bit flip for length of sequence $n = 20$.

Tests	Frequency	Runs	Longest Run	RW Excursion
Rotation	1	0.8683	0.6396	0.3388
10-Rotation	1	0.4754	0.3415	0.2601
5-Rotation	1	0.2080	0.4033	0.1826
4-Rotation	1	0.2229	0.4830	0.4017
2-Rotation	1	0.2802	0.5587	1

Table 3.3: Sensitivity of randomness test for Rotation for length of sequence $n = 20$.

Tests	Frequency	Runs	Longest Run	RW Excursion
Derivation	0.1077	0.0859	0.0117	0.0217
10-Derivation	0.1128	0.0362	0.0096	0.01704
5-Derivation	0.0944	0.0181	0.0035	0.0173
4-Derivation	0.1112	0.0367	0.0061	0.0153
2-Derivation	0.1780	0.1549	0.0029	0.0124

Table 3.4: Sensitivity of randomness test for Derivation for length of sequence $n = 20$.

Tests	Frequency	Runs	Longest Run	RW Excursion
Reversing	1	1	1	0.7174
10-Reversing	1	0.8683	0.4617	0.5870
5-Reversing	1	0.4526	0.2593	0.1456
4-Reversing	1	0.3310	0.4457	0.5435
2-Reversing	1	0.2802	0.5587	1

Table 3.5: Sensitivity of randomness test for Reversing for length of sequence $n = 20$.

Tests	Frequency	Runs	Longest Run	RW Excursion
XOR	0.1041	0.0434	0.0027	0.0187
10-XOR	0.1041	0.0386	0.0029	0.0175
5-XOR	0.0944	0.0286	0.0028	0.0315
4-XOR	0.0989	0.0350	0.0028	0.0183
2-XOR	0.1780	0.1549	0.0029	0.0124

Table 3.6: Sensitivity of randomness test for Reversing for length of sequence $n = 20$.

In Tables (3.2,3.3,3.4,3.5,3.6) are presented the results. For every transformation (that was describe before) and for four tests (Frequency Test, Runs Test, The Longest Run of Ones Test and Random Walk Excursion Test) a certain number is given. The number is $|S_r \cap S_{r,\sigma}|/|S_r|$ and represents the measure of indepen-

dence the test on the transformation σ. It means we test all 2^{20} sequences and for every $s \in \{0,1\}^{20}$ we observed if $T(s) = T(\sigma(s)) =$ reject or if $T(s) =$ reject and $T(\sigma(s)) =$ accept.

Number 1 means that T is invariant under σ, while all sequences rejected by $T(\sigma(.))$ was also rejected by T. Interesting are the numbers close to α because they indicate that $T(.)$ and $T(\sigma(.))$ are in relation which is close to independence. As it can be seen from the tables, some of the transformations do not have any effect on the test results.

Complementing significantly affects only the longest run of ones test, i^{th} bit flipping does not have a strong effect. Rotation seems to be more effective, if it is done by 5 bit blocks as well as Reversing. Frequency test is of course invariant under both of them. Moreover, ordinary Reverse does not affect runs test and longest run of ones and 2-1-Rotation is identical to 2-1-Reversing and both do not affect random walk excursion while they just reverse random walk by axis x. The most effective seems to be the derivative transformation (as stated by the authors of [TDB08]) and new transformation avalanche XOR. Both are effective in block variant to the same extent as the in ordinary variant. Hence the use of these two transformations (or their variants) and adding $T(\sigma(.))$ to test suite can be considered. Note that also their independence from other tests in the test suite is desirable.

Note 3.2. For i^{th} Bit flip, l-Rotation and l^{th} Derivative we present an average value. And also for m-l-Derivation and m-l-Rotation an average value (for m) is presented.

Note 3.3. Frequency and Runs mean Frequency Test and Runs Test as were described in the chapter 2. Longest Run means the Longest Run of Ones Test, therefore the same test as was described but not in block and same for RW Excursion (the Random Walk Excursion Test). We use only these four tests out of the eight tests that are described in chapter 2 because only these four tests can give a meaningful results for such short sequences.

Chapter 4

Next Bit Tests

The whole idea of the Next Bit Tests comes from the complexity theory. The definition of the Pseudorandom generator in the complexity theory is:

Definition 4.1. A deterministic polynomial-time Turing machine $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ is called *pseudorandom generator* if there exists a *stretch function* $\ell : \mathbb{N} \to \mathbb{N}$ (satisfying $\ell(k) > k$ for all k), such that for any probabilistic polynomial-time Turing machine D, for any positive polynomial p, and for all sufficiently large k's

$$|\Pr[D(G(U_k)) = 1] - \Pr[D(U_{\ell(k)}) = 1]| < \frac{1}{p(k)},$$

where U_n denotes the uniform distribution over $\{0,1\}^n$ and the probability is taken over U_k (resp. $U_{\ell(k)}$) as well as over the internal coin tosses of D.

In a very close relationship lies the definition of being next bit unpredictable.

Definition 4.2. A deterministic polynomial-time Turing machine $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}$ is called *next bit unpredictable* if for any probabilistic polynomial-time Turing machine D, for any positive polynomial p, and for all sufficiently large k's

$$|\Pr[D(G(U_k)_{\{1,2,\ldots,I-1\}}, I) = G(U_k)_I] - 1/2| < \frac{1}{p(k)},$$

where $G(x)_{\{1,2,\ldots,n\}}$ denotes the first n bits of $G(x)$ and the probability is taken over U_k, $I \in \{1,2,\ldots,\ell(k)\}$ and over the internal coin tosses of D.

The relationship is shown by the Yao's theorem.

Theorem 4.3 (Yao 1982)**.** *A deterministic polynomial-time Turing machine* $G : \{0,1\}^k \to \{0,1\}^{\ell(k)}, \ell(k) > k$ *is a pseudorandom generator iff* G *is next bit unpredictable.*

Suppose we have some function f that determines the next bit. What is the probability distribution of success in case of sequence of independent variables with the Bernoulli distribution with $p = 1/2$? Well, for a random sequence the success should be one half on every bit for every function f because if the bits of sequence $X_1, X_2, \ldots, X_n$ are truly independent from each other, the previous bits will not help us in a prediction of the next bits. The measurement of success of the function f can be done in the following way. We can imagine it as the

Frequency Test applied on a bit-wise xor both, of the old sequence, and of a new one, created by the function f. Every zero means success, and every one a failure. The proportion should be around one half. Every deviation means we guess too well or too badly. If we guess too badly, we can change our strategy into the opposite.

Example 4.4. Let $M = 1011001111001$ be the tested sequence. We choose f as the xor of all bits in the input and 0 otherwise. Therefore $f() = 0$, $f(1) = 1$, $f(10) = 1$, $f(101) = 0$, ..., $f(101100111100) = 1$. Denote this new sequence as $N = 0110111010111$, hence $M \oplus N = 1101110101110$ and this sequence is then tested with the Frequency Test.

Of course, the key property of the following procedure depends on the design of the function f that will guess next bits of a sequence. Construction of f can be done in various ways. I have chosen the way that is intuitive and seems to be logical. Imagine we have a test T and the test T measures some property of a sequence. For example let T be the Frequency Test. Suppose that a generator wants to fool our test in the easiest way. For example it produces a sequence where the first half consists only of ones and the second of zeros. This sequence passes the test T but it is obvious how to predict the next bits. So our function f will guess the most expected behaviour connected to property the of T. In this case, we will guess one, if we have seen more zeros than ones and vice versa.

Now, if we test a sequence always with a test T and then with his next bit variant, then we will reject the sequences that behave too poorly or too well with respect to the testing property of T. This is quite an innovative perspective in testing randomness because all other tests concentrate mostly on bad randomness properties and don't consider that too good behaviour could be also suspicious.

Here we describe the next bit variants of every test in chapter 2.

4.1 Next Bit Frequency Test

The Frequency Test measures the proportion of ones and zeros in sequence. Hence whenever we have more ones than zeros we will guess zero and vice versa.

Algorithm 1 Next Bit Frequency Test

Require: sequence seq of length n
Ensure: new sequence new created by Next Bit Frequency guessing

```
sum ← 0
new[0] ← random bit
for i = 0, 1, 2 ... n − 2 do
    sum ← sum + 2 ∗ seq[i] − 1
    if sum > 0 then
        new[i + 1] ← 0
    else if sum < 0 then
        new[i + 1] ← 1
    else
        new[i + 1] ← random bit
    end if
end for
```

4.2 Next Bit Frequency Test within a Block

The Frequency Test within a Block measures the proportion of ones and zeros in every block of sequence. Let m be the length of block, we created expected values of number of blocks with weight $w = 0, 1, \ldots, m$, which is $\left(k\binom{m}{0}/2^m, k\binom{m}{1}/2^m, \ldots, k\binom{m}{m}/2^m\right)$, where k is number of blocks. Then we guess what is the most probable bit according this weight vector and what we have already seen in block. After every block we upgrade expected values of weight of blocks.

Algorithm 2 Next Bit Frequency Test within a Block

Require: sequence seq of length n, length of block m
Ensure: new sequence new created by Next Bit Frequency within a Block guessing

```
k ← ⌊n/m⌋
expected ← (k·C(m,0)/2^m, k·C(m,1)/2^m, ..., k·C(m,m)/2^m)
for i = 0, 1, ..., k do
    weight ← 0 {weight is number of already seen ones in block}
    for j = 0, 1, ..., m − 1 do
        vector ← (expected[weight], expected[weight + 1], ..., expected[m])
        guess ← 0
        for index = 0, 1, ..., length(vector) − 1 do
            guess ← guess + vector[index] · (index/(m − j) − 1/2)
        end for
        if guess > 0 then
            new[i ∗ m + j] ← 1
        else if guess < 0 then
            new[i ∗ m + j] ← 0
        else
            new[i ∗ m + j] ← random bit
        end if
        weight ← weight + seq[i ∗ m + j]
    end for
    expected[weight] ← expected[weight] − 1
end for
```

In the algorithm the number $index$ means average number of ones in the rest of block and $index/(m - j)$ is this number normed to one bit.

4.3 Next Bit Runs Test

The Runs Test measures the number of runs (uninterrupted maximal sequence of identical bits) in a sequence as it is expected from a random sequence. First, we have to compute an expected number of runs in a sequence of length n. The first bit have one run and then with every other bit we have $1/2$ the probability of having next run, hence expected value is $1 + (n-1)/2 = (n+1)/2$. We will count the number of the observed runs and compare it with the expected number and guess the next bit accordingly.

Algorithm 3 Next Bit Runs Test

Require: sequence seq of length n
Ensure: new sequence new created by Next Bit Runs guessing
  $expected \leftarrow (n+1)/2$
  $new[0] \leftarrow$ random bit
  $runs \leftarrow 0$ {$runs$ is number of runs}
  for $i = 0, 1, \ldots n-2$ **do**
    update $runs$
    if $(expected - runs)/(n-i-1) > 1/2$ **then**
      $new[i+1] \leftarrow seq[i] \oplus 1$
    else if $(expected - runs)/(n-i-1) < 1/2$ **then**
      $new[i+1] \leftarrow seq[i]$
    else
      $new[i+1] \leftarrow$ random bit
    end if
  end for

4.4 Next Bit Test for the Longest Run of Ones in a Block

The Test for the Longest Run of Ones in a Block measures whether the proportion of the longest run of ones in blocks is as it would be expected from a random sequence. Let $PLRO_i$ denotes the probability of having the longest run of ones of length i in a block of length m, hence $PLRO_i =$ (number of blocks of length m with the longest run of ones of length i)$/2^m$. For every $j \in \{0, 1, \ldots, n-1\}$ we compute the most likely bit.

Algorithm 4 Next Bit Test for the Longest Run of Ones in a Block

Require: sequence *seq* of length n, length of block m
Ensure: new sequence *new* created by Next Bit Longest Run of Ones in a Block guessing

```
k ← ⌊n/m⌋
expected ← (k * PLRO_0, k * PLRO_1, ..., k * PLRO_m)
for i = 0, 1, ..., k do
    longest ← 0 {longest is the longest run of ones already seen in block}
    for j = 0, 1, ..., m − 1 do
        vector ← (expected[longest], expected[longest + 1], ..., expected[m])
        guess ← 0
        for index = 0, 1, ..., length(vector) − 1 do
            guess ← vector[index]·(predict(X_{im+j}, index, (X_{im}, X_{im+1}, ..., X_{im+j−1}))
        end for
        if guess > 0 then
            new[i * m + j] ← 1
        else if guess < 0 then
            new[i * m + j] ← 0
        else
            new[i * m + j] ← random bit
        end if
        update longest
    end for
    expected[longest] ← expected[longest] − 1
end for
```

To understand the algorithm we have to explain what function *predict* does. Recall X_{i-1} is the i^{th} bit of given sequence and by L we denote the length of the longest run of ones in a block.

$$\begin{aligned} &predict(X_j, index, (X_0, X_1, \ldots, X_{j-1}) = \\ &\Pr[X_j = 1 | X_0, X_1, \ldots, X_{j-1}, L = index] \\ &- \Pr[X_j = 0 | X_0, X_1, \ldots, X_{j-1}, L = index]. \end{aligned}$$

The probability $\Pr[X_j = i | X_0, X_1, \ldots, X_{j-1}, L = index]$ means the probability that $(j+1)^{th}$ bit will be i given the longest run of ones in a block is *index* and first j bits are $(X_0, X_1, \ldots, X_{j-1})$. Hence if $predict > 0$ then it is more likely that the next bit will be 1 and 0 otherwise. In the algorithm, we are taking into account the probabilities for every possibility on the longest run of ones in a block.

4.5 Next Bit Binary Rank Matrix Test

The Next Bit Binary Rank Matrix Test measures whether ranks of binary matrices are as they should be from a random sequence. The algorithm works in a similar manner as the previous one. Only, the probability of the next bit is computed according to the probabilities expected from random matrices. The length of the block will be m^2, while we consider $m \times m$ matrices and let R_i be the probability of a matrix $m \times m$ having rank i.

Algorithm 5 Next Bit Binary Rank Matrix Test

Require: sequence *seq* of length n, length of block m^2
Ensure: new sequence *new* created by Next Bit Binary Rank Matrix guessing

```
k ← ⌊n/m⌋
expected ← (k * R_0, k * R_1, ..., k * R_m)
for i = 0, 1, ..., k do
  lower ← 0
  upper ← m {lower and upper are boundaries on rank of partially build matrix}
  for j = 0, 1, ..., m − 1 do
    vector ← (expected[lower], expected[lower + 1], ..., expected[upper])
    for l = 0, 1, ..., m − 1 do
      guess ← 0
      for index = lower, lower + 1, ..., upper do
        guess ← vector[index] · (predict(X_{i*m^2+j*m+l}, index, (X_{i*m^2}, X_{i*m^2+1}, ..., X_{i*m^2+j*m+l−1}))
      end for
      if guess > 0 then
        new[i * m^2 + j * m + l] ← 1
      else if guess < 0 then
        new[i * m^2 + j * m + l] ← 0
      else
        new[i * m^2 + j * m + l] ← random bit
      end if
    end for
    update lower and upper
  end for
  expected[lower] ← expected[lower] − 1 {here lower = upper = rank of matrix}
end for
```

Function *predict* like in previous algorithm is:

$$\begin{aligned} &predict(X_{j*m+l}, index, (X_0, X_1, \ldots, X_{j*m+l-1})) = \\ &\Pr[X_{j*m+l} = 1 | X_0, X_1, \ldots, X_{j*m+l-1}, R = index] \\ &- \Pr[X_{j*m+l} = 0 | X_0, X_1, \ldots, X_{j*m+l-1}, R = index], \end{aligned}$$

where $\Pr[X_{j*m+l} = i | X_0, X_1, \ldots, X_{j*m+l-1}, R = index]$ denotes the probability that $(j * m + l + 1)^{th}$ bit will be i given the rank of matrix is *index* and first $j * m + l$ bits are $(X_0, X_1, \ldots, X_{j*m+l-1})$.

4.6 Next Bit Random Walk Excursion Test

The Random Walk Excursion Test measures whether number of excursions are as it is expected from a random sequence. Here PE_i denotes a probability of a block of length m having i excursions. Hence, PE_i = (number of blocks of length m with i excursions)$/2^m$.

Algorithm 6 Next Bit Random Walk Excursion Test

Require: sequence *seq* of length n, length of block m
Ensure: new sequence *new* created by Next Bit Random Walk Excursion guessing

```
k ← ⌊n/m⌋
expected ← (k * PE_0, k * PE_1, ..., PE_{m/2})
for i = 0, 1, ..., k do
  excursion ← 0 {excursion is the number of already seen excursions in block}
  for j = 0, 1, ..., m − 1 do
    vector ← (expected[excursion], expected[excursion + 1],
    ..., expected[m/2])
    guess ← 0
    for index = 0, 1, ..., length(vector) − 1 do
      guess ← vector[index]·(predict(X_{im+j}, index, (X_{im}, X_{im+1}, ..., X_{im+j−1}))
    end for
    if guess > 0 then
      new[i * m + j] ← 1
    else if guess < 0 then
      new[i * m + j] ← 0
    else
      new[i * m + j] ← random bit
    end if
    update excursion
  end for
  expected[excursion] ← expected[excursion] − 1
end for
```

Here the function *predict* is:

$$\begin{aligned} &predict(X_j, index, (X_0, X_1, \ldots, X_{j-1}) = \\ &\Pr[X_j = 1 | X_0, X_1, \ldots, X_{j-1}, E = index] \\ &- \Pr[X_j = 0 | X_0, X_1, \ldots, X_{j-1}, E = index], \end{aligned}$$

where $\Pr[X_j = i | X_0, X_1, \ldots, X_{j-1}, E = index]$ is probability of $(j+1)^{th}$ bit is i given the first j bits are $(X_0, X_1, \ldots, X_{j-1})$ and the block has *index* excursions.

4.7 Next Bit Prime Number Test

The Prime Number Test observes whether the proportion and location of bits is as expected from a random sequence in connection with binary notation numbers. The function *predict* works in a similar way as the previous algorithms.

Algorithm 7 Next Bit Prime Number Test

Require: sequence seq of length n, length of block m
Ensure: new sequence new created by Next Bit Prime Number guessing

```
k ← ⌊n/m⌋
expected ← (k * (number of primes less than 2^m)/2^m,
k * (number of composite numbers less than 2^m)/2^m)
for i = 0, 1, ..., k do
  for j = 0, 1, ..., m − 1 do
    guess ← 0
    for index = 0, 1 do
      guess ← expected[index]·(predict(X_{im+j}, index, (X_{im}, X_{im+1}, ..., X_{im+j−1}))
    end for
    if guess > 0 then
      new[i * m + j] ← 1
    else if guess < 0 then
      new[i * m + j] ← 0
    else
      new[i * m + j] ← random bit
    end if
  end for
  if last block was prime then
    expected[0] ← expected[0] − 1
  else
    expected[1] ← expected[1] − 1
  end if
end for
```

4.8 Next Bit Irreducible Polynomial Test

The Irreducible Polynomial Test is similar to the Prime Number Test but instead of measuring occurrences of prime numbers it measures the occurrences of irreducible polynomials. The Next Bit Irreducible Polynomial Test is almost identical with the Next Bit Prime Number Test, therefore we will omit a detailed description and just state that the algorithm is the same as the Algorithm 7 except of the replacement primes for the irreducible polynomials and composite numbers for the reducible polynomials.

4.9 Next Bit Universal Test

The Next Bit Universal Test is a compromise between all Next Bit Tests. This test guesses the next bit according to other next bit tests and votes for the most likely variant among them. From all other next bit tests, we have a new sequence that was guessed and for all bits in this sequence, we know whether we put there a random bit or a certain bit according the property that we try to guess.

Example 4.5. Suppose we have 4 next bit tests. These are the 4 sequences guessed by these tests: $S_1 = [[0,1],[0,1],[1,0],[1,1]]$, $S_2 = [[0,1],[0,0],[1,0],[1,1]]$, $S_3 = [[1,0],[0,1],[1,1],[0,1]]$, $S_4 = [[1,0],[0,0],[1,0],[0,1]]$. The first bit means that the bit is random (let 1 mean it is) and the second bit is the bit of the guessed sequence. For every position, we choose the most probable bit from all sequences that has the position of a non-random bit (on the first position is 0). The first bit is 1 while S_1 and S_2 has on the first position 1 and S_3 and S_4 has on this position random bit. The second bit is random because S_1 and S_3 vote for 1 and S_2 and S_4 vote for 0. The third bit is random and the forth is 1.

Moreover, we can consider a more general variant of this test, where the votes of sequences are not equal but the power of votes is given by some weight vector $w = (w_1, w_2, \ldots, w_l)$, where l is the number of guessed sequences in the Next Bit Universal Test.

Algorithm 8 Next Bit Universal Test

Require: sequences $S_1, S_2, \ldots, S_l$ of length n, weight vector $w = (w_1, w_2, \ldots, w_l)$
Ensure: new sequence *new* created by Next Bit Universal guessing

```
for i = 0, 1, ..., n − 1 do
    guess ← 0
    for j = 1, ..., l do
        if i^th bit of S_j is non-random then
            guess ← w_j · (2S_j[i] − 1)
        end if
    end for
    if guess > 0 then
        new[i] ← 1
    else if guess < 0 then
        new[i] ← 0
    else
        new[i] ← random bit
    end if
end for
```

Chapter 5

Results for SHA-3 second round candidates

Here we discuss how to test hash functions and show some interesting results for SHA-3 second round candidates. All tests are used for 512 bit sequences produced by 512-bit versions of hash functions. First of all, there are several possibilities how to create data sets for testing. Most of them are described in [Sot99]. We took an inspiration from there in our testing of four data sets.

Integers: Integers from 0 to $i, i \in \mathbb{N}$ are hashed and these outputs are tested.

Integers-XOR: Integers from 0 to $i, i \in \mathbb{N}$ are hashed and the outputs are XORed with the input and these sequences are tested.

Chaining: The first sequence is hash of 512-bit block of all zeros. The second sequence is two times hashed of 512-bit block of all zeros and the i^{th} tested sequence is $H^i(\bar{0})$, where H is hash function and $\bar{0}$ is 512-bit block of all zeros.

Chaining-XOR: Similar to previous data sets the i^{th} tested sequence is $H^i(\bar{0}) \oplus H^{i-1}(\bar{0})$.

The XOR variants of data sets are similar to *plaintext-ciphertext correlation* in [Sot99] except the plaintext is not random.

We have tested 1,000,000 bits generated according these four scenarios (more concretely $512 \cdot 1953 = 999,936$ bits) and observed for which data set and test the p-value is less than or equal to 0.01 (complete results of testing are in the Appendix). We have used the same testing strategy as in [SDEK10], which is probably the most reasonable in our situation.

We prepared four data sets (Integers, Integers-XOR, Chaining, Chaining-XOR) and each has the length $512 \cdot 1953 = 999,936$ bits. We tested each 512 bit block length with 17 tests that were described in this thesis (8 in the chapter 2 and 9 tests were in the chapter 4). Hence we obtained 1953 p-values for every test and every data set. Now, the strategy of testing that is mostly used is as follows.

Imagine, for the simplicity, that we don't test just 512 bits but we test for example more than one million bits and we use an approximation so that we

can work with continuous random variables (strategy the NIST used). In this case, the p-value is in the interval $[0, 1]$ and we can work with this p-value as with uniform continuous variable. Hence, we know that $\Pr[p\text{-value} < 0.1] = 0.1$, $\Pr[0.1 \leq p\text{-value} < 0.2] = 0.1$, $\ldots$, $\Pr[0.9 \leq p\text{-value}] = 0.1$. We divide p-values into 10 equal subintervals of [0,1] and measure their proportion with χ^2 test.

Let m be the number of sequences tested (1953 in our case) and F_i be the number of p-values in subinterval i for $i = 1, 2, \ldots, 10$. Then the χ^2 value and the corresponding p-value are calculated as

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m \cdot p_i)^2}{m \cdot p_i} \qquad \text{and} \qquad p\text{-value} = \mathsf{igamc}\left(\frac{9}{2}, \frac{\chi^2}{2}\right)^{1}.$$

Low p-value indicates poor proportion of p-values and hence could be use as an evidence of non-randomness. The NIST stated: *"If p-value* ≥ 0.0001*, the test results are considered to be uniformly distributed"* [BRS+10].

In our case of short sequences we have to modify this strategy as is done in [SDEK10]. The p-value is not continuous but discrete, hence it is not correct to think $\Pr[p\text{-value} < 0.1] = 0.1$ but $\Pr[p\text{-value} < 0.1] = p_1$ and p_1 has to be computed.

More generally, for every test we compute bounds $b_0 = 0 < b_1 < b_2 < \cdots < b_9 < b_{10} = 1$ and $p_1, p_2, \ldots, p_{10}$ such that

$$\Pr[b_{i-1} \leq p\text{-value} < b_i] = p_i, \text{ for } i = 1, 2, \ldots, 10.$$

These bounds $b_0, b_1, \ldots, b_{10}$ are chosen such that every p_i is around 0.1. Then the χ^2 value is calculated as

$$\chi^2 = \sum_{i=1}^{10} \frac{(F_i - m \cdot p_i)^2}{m \cdot p_i},$$

where F_i is the number of p-values in subinterval $[b_{i-1}, b_i)$ for $i = 1, 2, \ldots, 10$ and m is in our case equal to 1953. Using the incomplete gamma function we compute p-value $= \mathsf{igamc}\left(\frac{9}{2}, \frac{\chi^2}{2}\right)$.

Note 5.1. Not for all tests it was easy to create 10 subintervals and simultaneously fulfill the Yarnold's criterion for χ^2 test (see chapter 1). Therefore, I decided to create 8 or 9 subintervals for several tests. But this is only a technicality and doesn't effect the results.

[1] igamc is Incomplete gamma function complemented defined as $\mathsf{igamc}(a,x) = 1 - \frac{\gamma(a/2,x/2)}{\Gamma(a/2)} = 1 - \frac{\int_0^x t^{a-1}e^{-t}dt}{\int_0^\infty t^{a-1}e^{-t}dt}$ and is used for computing p-values from χ^2-distribution.

In Table 5.1 are results of candidates testing with p-values less than or equal to 0.01.

Hash function	Statistical test	Data set	p-value
BLAKE	Next Bit Rank Test	Chaining	0.00982
BMW	Frequency Test	Chaining-XOR	0.00862
CubeHash	Irreducible Polynomial Test	Chaining	0.00684
CubeHash	Prime Number Test	Integers	0.00668
Grøstl	Next Bit Frequency Test	Integers	0.01000
Hamsi	Longest Run of Ones	Integers	0.00047
JH	Next Bit Irreducible Pol. Test	Chaining-XOR	0.00922
JH	Next Bit Irreducible Pol. Test	Chaining	0.00028
SIMD	Next Bit Runs Test	Chaining-XOR	0.00746

Table 5.1: Tests and p-values less than or equal to 0.01 for second round candidates.

As we can see from the results, all p-values are greater then 0.0001 which was used by NIST in a testing the randomness of candidates on AES. But they are lower than or equal to 0.01 which should be also taken into consideration in evaluation of candidates on SHA-3. It is interesting that some of the third round candidates (the third round candidates were: BLAKE, Grøstl, JH, Keccak and Skein) are in Table 5.1. These candidates were selected from the second round candidates and should be more suitable to be a new hash standard. Of course, randomness is not the only criterion for selection on a new hash function. NIST stated that the new hash standard should be fast (in software and also in hardware) and should be at least secure as SHA-2. Also, there was a tendency to use other construction for the hash function than Merkle–Damgård that is used in SHA-2.

On October 2, 2012, NIST announced Keccak as the winner of the SHA-3 Cryptographic Hash Algorithm Competition. As we can see from Table 5.1 we have no evidence for non-randomness of Keccak. So we have no arguments against Keccak as the new hash standard in the field of randomness.

Note 5.2. In the Appendix, there are complete results for every data set, every test and every SHA-3 second round candidate. The values in tables are p-values from test that was described in this chapter.

Conclusion

In this thesis, we have been dealing with statistical testing of cryptographic primitives which is an important part of the cryptanalysis. Statistical tests play a significant role in evaluation of ciphers and hash functions security although they do not consider the inner structure of primitives. Failing in the statistical tests could be a sign of a poor inner structure or of dependencies inside the cipher, which should be motivation for a harder cryptanalysis and modification of the cipher.

The contribution of this thesis is the following:

- We proposed a new class of statistical tests, based on the idea of next bit guessing. We showed the next bit variance of all tests in chapter 2. A future research topic could be the transformation of more tests into their next-bit variant and the examination of their properties and their relationship to other tests.
- We introduced the Universal Next Bit Test, which is a kind of compromise between various Next Bit Tests.
- Considering the conclusions of the paper [TDB08], we focused on the sensitivity of tests. We analysed the transformations mentioned in the paper and showed that their block variants are more usable. Moreover, the new transformation, avalanche XOR, was proposed as an effective transformation.
- We proposed two tests (Prime Number Test, Irreducible polynomial Test).
- We used our tests (standard tests and the new next-bit variance of them) on the second round candidates on SHA-3 and commented on the results.

Bibliography

[BRS+10] Lawrence E. Bassham, III, Andrew L. Rukhin, Juan Soto, James R. Nechvatal, Miles E. Smid, Elaine B. Barker, Stefan D. Leigh, Mark Levenson, Mark Vangel, David L. Banks, Nathanael Alan Heckert, James F. Dray, and San Vo. Sp 800-22 rev. 1a. a statistical test suite for random and pseudorandom number generators for cryptographic applications. Technical report, Gaithersburg, MD, United States, 2010.

[DEKS10] Ali Doganaksoy, Baris Ege, Onur Koçak, and Fatih Sulak. Cryptographic randomness testing of block ciphers and hash functions. *IACR Cryptology ePrint Archive*, 2010:564, 2010. `http://eprint.iacr.org/`.

[FS06] Çagdas Çalik Meltem Sönmez Turan Fatih Sulak, Ali Doğanaksoy. New randomness tests using random walk. 2006. `http://www.metu.edu.tr/~ccalik/Randomwalk.pdf`.

[KUH04] Song-Ju Kim, Ken Umeno, and Akio Hasegawa. Corrections of the nist statistical test suite for randomness. Cryptology ePrint Archive, Report 2004/018, 2004. `http://eprint.iacr.org/`.

[Mar95] George Marsaglia. Diehard battery of tests of randomness. `http://i.cs.hku.hk/~diehard/cdrom/`, 1995.

[SDEK10] Fatih Sulak, Ali Doganaksoy, Baris Ege, and Onur Koçak. Evaluation of randomness test results for short sequences. pages 309–319, 2010.

[Sot99] Juan Soto. Randomness testing of the aes candidate algorithms. 1999. `http://csrc.nist.gov/publications/nistir/ir6390.pdf`.

[TDB08] Meltem Sönmez Turan, Ali Doganaksoy, and Serdar Boztas. On independence and sensitivity of statistical randomness tests. In Solomon W. Golomb, Matthew G. Parker, Alexander Pott, and Arne Winterhof, editors, *Sequences and Their Applications - SETA 2008, 5th International Conference, Lexington, KY, USA, September 14-18, 2008, Proceedings*, volume 5203 of *Lecture Notes in Computer Science*, pages 18–29. Springer, 2008.

[Yar07] J.K. Yarnold. The minimum expectation in χ^2 goodness of fit tests and the accurancy of approximations fot the null distribution. *Journal of the American Statistical Association*, 65:864–886, 2007.

Appendix

Probabilities in the Random Walk Excursion Test

Excursions	Probability
0	0.2734375
1	0.2734375
2	0.234375
3	0.15625
4	0.0625

Table 2: Probabilities in the Random Walk Excursion Test for $n = 8$ bit block.

Excursions	Probability
0	0.196380615234
1	0.196380615234
2	0.183288574219
3	0.157104492188
4	0.120849609375
5	0.08056640625
6	0.0439453125
7	0.017578125
8	0.00390625

Table 3: Probabilities in the Random Walk Excursion Test for $n = 16$ bit block.

Excursions	Probability
0	0.139949934091
1	0.139949934091
2	0.135435420088
3	0.126406392083
4	0.113329868764
5	0.0971398875117
6	0.079151019454
7	0.06088539958
8	0.0438374876976
9	0.0292249917984
10	0.0177891254425
11	0.00970315933228
12	0.00462055206299
13	0.0018482208252
14	0.000583648681641
15	0.000129699707031
16	0.0000152587890625

Table 4: Probabilities in the Random Walk Excursion Test for $n = 32$ bit block.

Results of testing SHA-3 candidates

Data set: Integers

Test/Hash	Frequency	Block Frequency	Run	Longest Run
BLAKE	0.86818	0.09247	0.38322	0.33947
BMW	0.77838	0.97675	0.86551	0.86092
CubeHash	0.91461	0.87642	0.45955	0.61986
ECHO	0.48749	0.95085	0.56375	0.92982
Fugue	0.28614	0.56459	0.68639	0.78020
Grøstl	0.81059	0.41951	0.12233	0.21004
Hamsi	0.48323	0.79096	0.91303	0.00047
JH	0.15052	0.60625	0.08571	0.21558
Keccak	0.54157	0.39668	0.58472	0.88671
Luffa	0.84413	0.99722	0.48638	0.86057
Shabal	0.27070	0.87913	0.93433	0.82181
SHAvite-3	0.99338	0.63616	0.59482	0.30574
SIMD	0.48957	0.46782	0.66738	0.69608
Skein	0.87967	0.71758	0.54604	0.17708

Table 5: P-values from the first half of normal tests, data set: Integers

Test/Hash	Rank	Excursion	Prime	Irreducible
BLAKE	0.80638	0.87370	0.33119	0.95849
BMW	0.40700	0.38024	0.23345	0.91801
CubeHash	0.47564	0.35642	0.00668	0.12585
ECHO	0.43389	0.64145	0.52259	0.11161
Fugue	0.48716	0.67574	0.89239	0.28736
Grøstl	0.27715	0.74182	0.36403	0.50442
Hamsi	0.99512	0.61680	0.04891	0.58282
JH	0.23577	0.02933	0.19354	0.80216
Keccak	0.74490	0.86886	0.27951	0.78228
Luffa	0.56528	0.05851	0.05451	0.04648
Shabal	0.62030	0.60609	0.38683	0.63121
SHAvite-3	0.33324	0.55115	0.73404	0.80907
SIMD	0.24183	0.72301	0.32600	0.81211
Skein	0.27423	0.04922	0.70264	0.29751

Table 6: P-values from the second half of normal tests, data set: Integers

Test/Hash	Next Freq.	Next Block Freq.	Next Runs	Next Longest Run
BLAKE	0.53685	0.53071	0.35200	0.59027
BMW	0.52613	0.75038	0.42039	0.15074
CubeHash	0.04220	0.64535	0.80434	0.68629
ECHO	0.73506	0.78601	0.43240	0.83500
Fugue	0.94885	0.66683	0.54142	0.36224
Grøstl	0.01000	0.30642	0.04703	0.37676
Hamsi	0.50588	0.16261	0.30313	0.38052
JH	0.65607	0.65899	0.24734	0.97848
Keccak	0.66817	0.54586	0.73097	0.86700
Luffa	0.98967	0.28506	0.36140	0.79745
Shabal	0.62298	0.51587	0.76578	0.67791
SHAvite-3	0.30330	0.88139	0.21109	0.64484
SIMD	0.28043	0.55403	0.69436	0.21311
Skein	0.08750	0.59054	0.04146	0.48565

Table 7: P-values from the first half of next bit tests, data set: Integers

Test/Hash	Next Rank	Next Excursion	Next Prime	Next Ir.	Next Uni.
BLAKE	0.99479	0.69908	0.06942	0.81031	0.17306
BMW	0.54724	0.33944	0.34371	0.13784	0.18483
CubeHash	0.12630	0.32618	0.72195	0.81999	0.63780
ECHO	0.32035	0.98573	0.03343	0.75874	0.24671
Fugue	0.79586	0.89550	0.33612	0.09304	0.17223
Grøstl	0.77273	0.26635	0.66822	0.81341	0.75669
Hamsi	0.21972	0.27708	0.58849	0.41042	0.74563
JH	0.73435	0.17201	0.60713	0.87542	0.59818
Keccak	0.66364	0.03160	0.03021	0.65274	0.57927
Luffa	0.59212	0.52665	0.96903	0.16582	0.55944
Shabal	0.77642	0.68631	0.98507	0.77393	0.36883
SHAvite-3	0.11486	0.36425	0.90851	0.32801	0.83210
SIMD	0.22218	0.64796	0.07314	0.94978	0.36719
Skein	0.55058	0.63417	0.18723	0.42103	0.22183

Table 8: P-values from the second half of next bit tests, data set: Integers

Results of testing SHA-3 candidates
Data set: Integers-XOR

Test/Hash	Frequency	Block Frequency	Runs	Longest Run
BLAKE	0.38435	0.55527	0.54272	0.42010
BMW	0.44633	0.72275	0.01456	0.42751
CubeHash	0.79198	0.97156	0.46249	0.25746
ECHO	0.50981	0.76643	0.57249	0.17457
Fugue	0.85699	0.93009	0.91978	0.84697
Grøstl	0.02098	0.87456	0.09290	0.67366
Hamsi	0.62912	0.63721	0.48515	0.34116
JH	0.42748	0.97666	0.89838	0.97794
Keccak	0.14617	0.23279	0.29258	0.29826
Luffa	0.96107	0.43850	0.65458	0.39186
Shabal	0.38421	0.91236	0.52086	0.23398
SHAvite-3	0.91038	0.97332	0.49624	0.57883
SIMD	0.95612	0.64530	0.49542	0.98740
Skein	0.19621	0.33991	0.56030	0.08009

Table 9: P-values from the first half of normal tests, data set: Integers-XOR

Test/Hash	Rank	Excursion	Prime	Irreducible
BLAKE	0.83967	0.90938	0.73186	0.52997
BMW	0.56438	0.83786	0.76871	0.73633
CubeHash	0.16531	0.64090	0.52773	0.13591
ECHO	0.55892	0.21251	0.28376	0.70027
Fugue	0.57059	0.53200	0.25201	0.34132
Grøstl	0.59698	0.73281	0.92652	0.10683
Hamsi	0.51045	0.71222	0.72975	0.85822
JH	0.90235	0.89618	0.13992	0.61457
Keccak	0.89782	0.18122	0.16284	0.11903
Luffa	0.47145	0.17714	0.75627	0.07958
Shabal	0.62305	0.03710	0.87909	0.26234
SHAvite-3	0.04579	0.24926	0.81876	0.82803
SIMD	0.01049	0.03119	0.60728	0.18051
Skein	0.20983	0.22184	0.94658	0.86001

Table 10: P-values from the second half of normal tests, data set: Integers-XOR

Test/Hash	Next Freq.	Next Block Freq.	Next Runs	Next Longest Run
BLAKE	0.30823	0.53951	0.52028	0.66094
BMW	0.46357	0.31497	0.10817	0.96554
CubeHash	0.84658	0.14474	0.21613	0.22979
ECHO	0.60213	0.21076	0.51236	0.99312
Fugue	0.50530	0.13007	0.11258	0.66793
Grøstl	0.88679	0.35073	0.62945	0.14891
Hamsi	0.64820	0.44501	0.01718	0.17820
JH	0.23365	0.70713	0.89124	0.08122
Keccak	0.98307	0.12477	0.77045	0.40273
Luffa	0.73683	0.50919	0.66942	0.88341
Shabal	0.34259	0.99426	0.48890	0.83213
SHAvite-3	0.13301	0.81666	0.41861	0.37905
SIMD	0.21343	0.02090	0.37060	0.79878
Skein	0.66969	0.37829	0.98768	0.17937

Table 11: P-values from the first half of next bit tests, data set: Integers-XOR

Test/Hash	Next Rank	Next Excursion	Next Prime	Next Ir.	Next Uni.
BLAKE	0.01530	0.39539	0.54441	0.04868	0.14789
BMW	0.04788	0.37234	0.91555	0.36512	0.87757
CubeHash	0.86470	0.55212	0.73036	0.26885	0.69840
ECHO	0.52263	0.84570	0.30680	0.10459	0.84682
Fugue	0.82353	0.76234	0.02833	0.44359	0.76482
Grøstl	0.73353	0.30234	0.86097	0.42030	0.42940
Hamsi	0.43445	0.28130	0.33758	0.29394	0.44128
JH	0.90073	0.71361	0.23675	0.19031	0.77348
Keccak	0.10106	0.69958	0.62989	0.60544	0.15467
Luffa	0.78309	0.47881	0.01866	0.82443	0.47990
Shabal	0.02862	0.66755	0.06425	0.31908	0.77720
SHAvite-3	0.51430	0.92906	0.71273	0.81551	0.96899
SIMD	0.35582	0.75106	0.43788	0.09537	0.92510
Skein	0.71259	0.54961	0.08474	0.08981	0.95433

Table 12: P-values from the second half of next bit tests, data set: Integers-XOR

Results of testing SHA-3 candidates

Data set: Chaining

Test/Hash	Frequency	Block Frequency	Runs	Longest Run
BLAKE	0.12577	0.31957	0.15940	0.37699
BMW	0.03397	0.88533	0.38711	0.44117
CubeHash	0.85581	0.95639	0.31506	0.83504
ECHO	0.68028	0.91140	0.75933	0.52450
Fugue	0.15241	0.98838	0.96363	0.36735
Grøstl	0.73506	0.76885	0.11383	0.19453
Hamsi	0.61644	0.93969	0.79415	0.82051
JH	0.87083	0.97897	0.94510	0.22787
Keccak	0.04336	0.79406	0.07994	0.42717
Luffa	0.73150	0.96234	0.45954	0.48680
Shabal	0.82181	0.98257	0.85183	0.51035
SHAvite-3	0.47141	0.92895	0.40095	0.86890
SIMD	0.16779	0.15148	0.69431	0.90654
Skein	0.82175	0.84141	0.43193	0.29473

Table 13: P-values from the first half of normal tests, data set: Chaining

Test/Hash	Rank	Excursion	Prime	Irreducible
BLAKE	0.41638	0.54433	0.45604	0.03467
BMW	0.58885	0.15565	0.47807	0.64671
CubeHash	0.90138	0.29666	0.65624	0.00684
ECHO	0.16307	0.84412	0.63679	0.05548
Fugue	0.92675	0.57843	0.91344	0.52646
Grøstl	0.24653	0.51590	0.63893	0.01597
Hamsi	0.74755	0.92746	0.80031	0.52575
JH	0.14739	0.51748	0.72086	0.79376
Keccak	0.29992	0.10764	0.03106	0.48370
Luffa	0.34850	0.65233	0.34171	0.88950
Shabal	0.49074	0.54166	0.17312	0.03876
SHAvite-3	0.97858	0.99089	0.90381	0.24729
SIMD	0.84683	0.69941	0.22658	0.23848
Skein	0.89610	0.72971	0.59539	0.27941

Table 14: P-values from the second half of normal tests, data set: Chaining

Test/Hash	Next Freq.	Next Block Freq.	Next Runs	Next Longest Run
BLAKE	0.15448	0.75907	0.10509	0.90515
BMW	0.16504	0.28226	0.24895	0.04678
CubeHash	0.65753	0.56708	0.04574	0.71149
ECHO	0.57008	0.17640	0.07152	0.60798
Fugue	0.15247	0.89189	0.84224	0.31444
Grøstl	0.97774	0.42945	0.27821	0.14020
Hamsi	0.98362	0.17237	0.77952	0.96220
JH	0.75385	0.03613	0.18964	0.88726
Keccak	0.88685	0.95349	0.54384	0.58847
Luffa	0.14301	0.96057	0.49453	0.60495
Shabal	0.40493	0.67662	0.92011	0.74646
SHAvite-3	0.63038	0.06872	0.26970	0.32110
SIMD	0.50351	0.41262	0.08264	0.97308
Skein	0.30626	0.64008	0.51801	0.19727

Table 15: P-values from the first half of next bit tests, data set: Chaining

Test/Hash	Next Rank	Next Excursion	Next Prime	Next Ir.	Next Uni.
BLAKE	0.00982	0.24207	0.82538	0.20473	0.25266
BMW	0.43863	0.68670	0.91163	0.59578	0.18085
CubeHash	0.41519	0.27754	0.84978	0.84352	0.92713
ECHO	0.80832	0.14074	0.85491	0.44654	0.51537
Fugue	0.09181	0.81325	0.35591	0.86787	0.21553
Grøstl	0.13179	0.35510	0.70440	0.97134	0.43385
Hamsi	0.49213	0.84648	0.88161	0.12953	0.84271
JH	0.18382	0.67147	0.49228	0.00028	0.59195
Keccak	0.74564	0.94913	0.61154	0.42426	0.66812
Luffa	0.78655	0.93590	0.18187	0.31795	0.12105
Shabal	0.52090	0.35407	0.54014	0.68375	0.05662
SHAvite-3	0.17244	0.80227	0.64612	0.04197	0.56228
SIMD	0.61237	0.20812	0.64178	0.51568	0.88870
Skein	0.37811	0.18514	0.06795	0.28397	0.30973

Table 16: P-values from the second half of next bit tests, data set: Chaining

Results of testing SHA-3 candidates
Data set: Chaining-XOR

Test/Hash	Frequency	Block Frequency	Runs	Longest Run
BLAKE	0.38255	0.47610	0.90288	0.71217
BMW	0.00862	0.58020	0.71254	0.61375
CubeHash	0.76259	0.97847	0.56905	0.69076
ECHO	0.73520	0.87963	0.98895	0.26691
Fugue	0.78893	0.74415	0.43505	0.57602
Grøstl	0.95835	0.97155	0.74385	0.39402
Hamsi	0.45343	0.91517	0.78817	0.78851
JH	0.84693	0.69399	0.88873	0.41993
Keccak	0.05271	0.96813	0.07616	0.22139
Luffa	0.37160	0.65585	0.06006	0.23832
Shabal	0.76024	0.94530	0.48405	0.73771
SHAvite-3	0.53788	0.70874	0.73616	0.89031
SIMD	0.39170	0.32031	0.04741	0.81985
Skein	0.57450	0.96609	0.15921	0.18459

Table 17: P-values from the first half of normal tests, data set: Chaining-XOR

Test/Hash	Rank	Excursion	Prime	Irreducible
BLAKE	0.59863	0.49544	0.50464	0.12297
BMW	0.14470	0.85198	0.50086	0.47402
CubeHash	0.99865	0.48037	0.91796	0.21467
ECHO	0.32557	0.44058	0.19461	0.10561
Fugue	0.88962	0.28634	0.55655	0.79322
Grøstl	0.46535	0.15856	0.36385	0.38809
Hamsi	0.91327	0.91208	0.08885	0.62991
JH	0.14207	0.23040	0.24557	0.33916
Keccak	0.34529	0.84384	0.05702	0.33372
Luffa	0.74367	0.53078	0.60274	0.74316
Shabal	0.09109	0.97211	0.29707	0.45284
SHAvite-3	0.80827	0.27113	0.52957	0.25862
SIMD	0.98048	0.67485	0.82959	0.47102
Skein	0.48029	0.38452	0.88540	0.05309

Table 18: P-values from the second half of normal tests, data set: Chaining-XOR

Test/Hash	Next Freq.	Next Block Freq.	Next Runs	Next Longest Run
BLAKE	0.24491	0.36114	0.57696	0.41962
BMW	0.90374	0.36600	0.89468	0.04692
CubeHash	0.61506	0.49776	0.36243	0.99380
ECHO	0.43094	0.16038	0.93734	0.44257
Fugue	0.13735	0.90804	0.86698	0.02076
Grøstl	0.59628	0.23498	0.98764	0.30259
Hamsi	0.40250	0.35945	0.26388	0.77907
JH	0.48009	0.96744	0.36974	0.58761
Keccak	0.67676	0.45944	0.79499	0.48706
Luffa	0.26566	0.78745	0.03222	0.17150
Shabal	0.79273	0.47492	0.36635	0.87024
SHAvite-3	0.81221	0.06471	0.94457	0.46922
SIMD	0.10203	0.31285	0.00746	0.28006
Skein	0.68489	0.67778	0.07184	0.80851

Table 19: P-values from the first half of next bit tests, data set: Chaining-XOR

Test/Hash	Next Rank	Next Excursion	Next Prime	Next Ir.	Next Uni.
BLAKE	0.36483	0.81876	0.92408	0.86583	0.85925
BMW	0.21747	0.77158	0.51704	0.67519	0.33680
CubeHash	0.23450	0.51157	0.28913	0.64768	0.96098
ECHO	0.96193	0.01909	0.90110	0.45085	0.65417
Fugue	0.26534	0.68994	0.18345	0.75177	0.98015
Grøstl	0.57753	0.40507	0.78701	0.36768	0.07679
Hamsi	0.83986	0.38932	0.89338	0.57817	0.56829
JH	0.61365	0.20346	0.18494	0.00922	0.75504
Keccak	0.76227	0.56874	0.60305	0.06575	0.73514
Luffa	0.74364	0.11393	0.67127	0.40630	0.68593
Shabal	0.24446	0.89921	0.38692	0.09796	0.90467
SHAvite-3	0.17220	0.18329	0.66005	0.86148	0.66685
SIMD	0.55398	0.70329	0.79870	0.73679	0.21898
Skein	0.88940	0.63234	0.11498	0.38589	0.69646

Table 20: P-values from the second half of next bit tests, data set: Chaining-XOR