Analysis of Randomness of GAEN keys

April Sheeran

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Computer Science

Supervisor: Stephen Farrell

April 2024

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

April Sheeran

April 15, 2024

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

April Sheeran

April 15, 2024

Analysis of Randomness of GAEN keys

April Sheeran, Master of Computer Science

University of Dublin, Trinity College, 2024

Supervisor: Stephen Farrell

Digital contact tracing applications, such as those using the Google/Apple Exposure Notification (GAEN) system, have highlighted the critical role of cryptographic keys, particularly Temporary Exposure Keys (TEKs), in protecting user privacy while enabling contact tracing. This dissertation analyses the randomness of GAEN keys, specifically TEKs, to verify the privacy and security claims of the GAEN applications. This study employs a battery of statistical tests called Dieharder and a number of other statistical tests and visualisations to evaluate the randomness of GAEN keys generated by mobile devices across the world. This study contributes to the broader discourse on cryptographic key randomness and privacy-preserving technologies in the context of digital contact tracing.

Acknowledgments

I would like to acknowledge my dissertation supervisor Stephen Farrell, for his invaluable guidance, support, and encouragement throughout the entire research process. His expertise, insightful feedback, and assistance have been instrumental in the completion of this dissertation. Special thanks to my family and friends for their encouragement, support and understanding throughout this past year.

April Sheeran

University of Dublin, Trinity College

April 2024

Contents

List of Tables

List of Figures

Chapter 1

Introduction

This section explains the motivation behind this dissertation and provides a table of terms and definitions used throughout this paper.

1.1 Motivation

The motivation for this dissertation on the analysis of randomness of Google/Apple Exposure Notification (GAEN) keys spreads across multiple areas including cryptography and digital privacy. The primary motive stems from the fundamental importance of randomness in cryptography as it ensures the unpredictability and security of cryptographic keys and encryption protocols. Non randomness in encryption can result in vulnerabilities to the confidentiality and integrity of sensitive data. By assessing the randomness of the GAEN keys, further confidence can be gained in the security and privacy of the GAEN apps.

Exposure notification and contact tracing systems like GAEN have been used across the world as apps in an effort to combat the spread of Covid-19. GAEN uses Bluetooth technology to exchange cryptographic keys between devices in the proximity of the user and these keys allow users to be alerted if they were potentially exposed to the virus.

These apps claim to guarantee to provide user privacy and anonymity. The randomness of the keys being used in these systems is essential to ensure confidentiality of the users.

The GAEN apps provide a unique situation where, due to the design of the GAEN system, a substantial volume of keys generated by devices such as smartphones are readily available. Unlike most cryptographic systems where the keys are not published to the public, the keys generated in the GAEN apps are publicly accessible. These keys have been generated on a wide variety of devices across the world and across different manufacturers and operating systems. Notably, these devices use the same pseudo random number generators (PRNGs) to generate the GAEN keys for other cryptographic applications, for example AES (Advanced Encryption Standard) and HMAC (Hash-based Message Authentication Code). The quality of the randomness produced by PRNGs is integral to the security of the encryption algorithms that use them and a lack of randomness can lead to predictability which can compromise security. Therefore, the GAEN keys present a rare opportunity to not only analyse and evaluate a large dataset of keys for randomness, but also to give insight into the PRNGs being used in all these devices.

In essence, this dissertation is motivated by an opportunity to explore the security and privacy implications of exposure notification systems by testing the randomness of GAEN keys. This paper aims to contribute to enhancing the trustworthiness of these apps and validating the privacy claims of the system by ensuring that the keys produced by GAEN are random.

1.2 Terms and Definitions

Term	**Definition**
TEK	Temporary Exposure Key

Term	**Definition**
GAEN	Google/Apple Exposure Notification, a system for contact tracing
Randomness	"Randomness is the absence of pattern or predictability in a sequence of events or outcomes." (Grinstead and Snell (1997)
RNG/PRNG	(Pseudo) Random Number Generator
Binary Sequence	A sequence of ones and zeros
P-value	A probability between 0 and 1, measures the strength of the evidence against the null hypothesis (Dahiru (2008))
Degrees of Freedom	Indicates the number of independent values that can vary in a statistical analysis without breaking any constraints. (Frost (2024))
Level of Significance (alpha, a)	A measure of the strength of the evidence required to reject the null hypothesis and conclude that the result is statistically significant (Frost (2024)).
Entropy	A measure of unpredictable randomness (Zolfaghari et al. (2022))
Kolmogorov-Smirnov (KS) Test	A statistical test that may be used to determine if a set of data comes from a particular probability distribution (NIST (2012))
Word	A predefined substring consisting of a fixed pattern/template (e.g., 010, 0110) (NIST (2012)).
Alphabet	the set of symbols or values that can occur within the data being tested, 0 and 1 for binary

Term	**Definition**
Rank (of a matrix)	The dimension of the vector space generated by the matrix's columns (Axler (2015)).
Poisson Distribution	A discrete probability distribution that models probabilities for counts of events that happen in a specified observation space (Frost (2024)).
Covariance matrix	A square matrix of the covariance between each pair of elements of a given random vector.
Normal Distribution	A continuous probability distribution that is symmetrical around its mean, values are clustered around the central peak of the bell-curve (Frost (2024).
Exponential Distribution	A right-skewed continuous probability distribution with smaller values occurring more frequently than higher values (Frost (2024)).
Euclid's Method	An algorithm to computed the Greatest Common Divisors of two integers.
Binomial Distribution	A discrete probability distribution that finds the likelihood that an event will occur a specific number of times in a set number of chances (Frost (2024)).

Table 1.1: Terms and Definitions.

Chapter 2

State of the Art

2.1 Background

This section will explain what Google/Apple Exposure Notification (GAEN) is, explore the current research done into the privacy and security concerns of GAEN apps and define randomness for this project's context.

2.1.1 What is GAEN

Contact Tracing Apps

Google and Apple developed the Google/Apple Exposure Notification (GAEN) system to facilitate contact tracing in response to the Covid-19 Pandemic. (Mention conventional contact tracing). Nations across the world used this technology to create contract tracing apps, for example Covid Tracker in Ireland and SwissCovid in Switzerland (Leith and Farrell (2021)).

The way the GAEN contact tracing works, if a user enables it, is as follows (Google (2020):

- Every 10-20 minutes the user's device will generate a random 128-bit key, referred

to as a Temporary Exposure Key (TEK).

- The user's device will broadcast these keys using Bluetooth Low Energy (BLE).
- The user's device will listen and store the TEKs being broadcasted from other devices within a certain radius. These TEKs are stored locally on the device.
- If a user tests positive for Covid, they can log this into the app. The app will send the user's recent TEKs (around the 14 days) to a central server managed by the local health authority.
- Every approx. 2 hours, the user's device will download the TEKs from the central server.
- The app compares these downloaded TEKs to the TEKs stored locally on the device.
- If there is a match, this means that the user has potentially been exposed to Covid and the app will notify them.

Studies on GAEN-based Apps

There have been numerous studies done on contact tracing apps that use GAEN technology. Google and Apple acknowledge that keeping users' information private and secure is essential to the success of the contact tracing app and claim to have created their system with this central to the design (Google (2020)). Health data is very personal, sensitive data which is essential to keep private and preserve the anonymity of the user. This section explores the current research into the privacy, security and effectiveness of GAEN apps and highlights the gap in research that this dissertation seeks to fill, an analysis of the keys produced by GAEN which are crucial to its functionality.

The major privacy concerns of GAEN apps according to (Nguyen et al. (2022)) are the identification of users, tracking users or extracting the social graph of users. Contact tracing should aim to identify encounters rather than actual users, by doing so they

should not leak any information that could be used to identify the user. Similarly, the data collected should not be able to be used to create the social graph of the user, i.e. the social connections and relationships of a user. Having this information could potentially result in the user being identified.

Security is also a requirement for a contact tracing app according to (Nguyen et al. (2022)). The system should be resilient to large scale data pollution attacks. These could be fake exposure claims, where users may falsely claim they have been exposed in order to get out of work or another obligation or in an attempt to damage the reputation and credibility of the contact tracing apps. Fake exposure injection, as a relay attack, may send users false notifications of potential exposures. The attacker could do this by capturing the TEKs of some user and broadcasting them in another location, leading to people being falsely notified of exposure. This could result in panic among users and the population, putting further strain on the healthcare system by creating a demand for unnecessary tests. It could also damage the trust in the contact tracing apps as their accuracy would be no longer trusted.

(Nguyen et al. (2022)) assess the GAEN apps on a variety of requirements. In terms of effectiveness, GAEN has been found to be imprecise at determining the distances between user devices (do i need to reference an internal reference?). Its use of BLE means scanning of the user's surroundings for other devices can only happen with frequent pauses to save battery life of the device. Many factors like positioning of the device's antenna, obstacles in the way and orientation of the device affect the computation of the distance between devices and the errors are significant. GAEN fails to account for 'superspreaders' of the virus, an individual who is very contagious and infects a number of other people. GAEN also does not have any mechanism for dealing with asymptomatic individuals, people that are infected with the virus and are contagious but do not show symptoms. Unknowingly, these people spread the virus. These individuals are unlikely to get tested and therefore

won't log their infection in the app, meaning those that come in contact with them will not be notified of a potential exposure. This significantly impacts the effectiveness of the contact tracing apps.

An investigation into the data shared by Europe's contact tracing apps that use GAEN (Leith and Farrell (2021)) discovered that a significant amount of data was being sent to Google servers. The android implementations of the GAEN systems use Google Play Services to facilitate GAEN-based contact tracing. The user must enable Google Play Services. It was found that Google Play Services connects to Google servers approximately every 20 minutes, sending requests that include the handset IP address, location data and persistent identifiers to link requests coming from the same device. The data sent to Google in other types of requests also include phone IMEI, device hardware serial number, SIM serial number and IMSI, phone number, WiFi MAC address, user email and Android ID. While sharing data to backend servers is not in itself an intrusion of privacy, the ability to link this data to a real-world user is problematic. Given that the user's IP address is being sent to Google very frequently, this could be used as location tracking. It is possible to de-anonymise this location data and potentially identify the user. Given that the user must enable Google Play Services, and therefore this data sharing, to do contact tracing, this does raise a concern to the privacy of the user.

(Avitabile et al. (2023)) examines several potential threats and privacy concerns of GAEN apps. They introduce a 'paparazzi attack' which involves using passive Bluetooth devices to capture the keys being broadcasted from a targeted user. If this user tests positive, the attacker can match their locally stored keys to those made publically available on the central server and learn that the user is positive for covid. This is a form of deanonymization. Similarly, an attacker could exploit the movements of a targeted user to gain money by linking the locations of the passive devices to the keys and sell this data to interested parties.

The effectiveness of these GAEN contact tracing apps is undetermined (Leith and Farrell (2021)) (Nguyen et al. (2022))

Numerous alternative to the GAEN system have been proposed such as TraceCorona by (Nguyen et al. (2022)) in an attempt to remedy the above privacy and security concerns.

2.1.2 What is Randomness

In order to test for randomness/non randomness we must first define what randomness is. "Randomness is the absence of pattern or predictability in a sequence of events or outcomes." (Grinstead and Snell (1997).

A random bit sequence could be explained as the result of flipping an unbiased coin, with two sides 1 and 0, which has an equal chance of 50 percent of landing on side 1 or side 0. Each flip of the coin does not affect any future coin flips which means the flips are independent of each other. This unbiased coin can therefore be considered a perfect random bit stream generator as the appearances of 1s and 0s will be randomly and uniformly distributed. All elements in the sequence are independent of each other and future elements in the sequence cannot be predicted using previous elements (Dang (2012)). This simple example gives us an understanding of what it means for a set of keys to be random.

The keys must exhibit certain properties in order to be accepted as random:

- Independent meaning no previously generated keys affect a new key (Cortez et al. (2020)).
- Equally likely meaning that the probability of a 0 or 1 appearing at any point in the key is equal to 1/2 (Cortez et al. (2020)).

- Scalable meaning that if the key is random, then any extracted sub sequence is also random (Cortez et al. (2020)).

Any indication of a dependency or bias within the data would indicate non randomness.

2.2 Literature Review

This section explores how randomness testing is performed. The literature in this area is examined and examples of the impact of randomness failures are given.

2.2.1 How to Test for Randomness

It is important to note that you cannot say for certain whether something is random or not, you can only find evidence against non randomness. It is not possible to give theoretical proof of randomness of a sequence (Turan et al. (2008)). Various statistical tests can be performed on the data in an attempt to compare and evaluate the data against a truly random sequence since the outcome when a statistical test is applied to a truly random sequence is known. (Bassham et al. (2010)).

A challenge when testing for randomness is that there is no agreed upon complete set of statistical tests to deem a sequence random (Bassham et al. (2010)). There is an infinite number of tests that you could run in order to find the presence or absence of a pattern or bias within the data. The existence of a pattern or bias within the data would indicate that it is non random.

Hypothesis Testing

Statistical testing is used to test against a defined null hypothesis (h0). The null hypothesis in this case is that the keys being tested are random. The alternative hypothesis (h1) is that the keys are not random. The challenge here is to determine which of these

hypotheses can be accepted (Luengo and Villalba (2021)) . For each statistical test run on the data, the result accepts or rejects the null hypothesis.

The following table shows the possible results on a hypothesis test (Bassham et al. (2010)):

True Situation	Conclusion: Accept H0	Conclusion: Accept H1 (Reject H0)
H0 is True, data is random	No Error	Type 1 Error
H1 is True, data is not random	Type 2 Error	No Error

Table 2.1: Hypothesis Test Results

The above situations are somewhat unknown but some control can be gained by knowing the probability of each of the error situations. The probability of Error Type 1 is defined as α, the level of significance (Luengo and Villalba (2021)). This value is typically 0.01, 0.05 or 0.10. The probability of Error Type 2 is defined as β, referred to as contrast power and is usually used as 1-β. (Luengo and Villalba (2021)). If the data is truly random, rejecting the null hypothesis, meaning determining that the data is non random, will occur a small percentage of the time. For example if α is 0.01, it would be expected that 1 sequence in 100 sequences is rejected (Bassham et al. (2010)).

In practice, p-values are used to reject or accept the null hypothesis. A p-value can be defined as A probability between 0 and 1, measures the strength of the evidence against the null hypothesis (Dahiru (2008)). In the context of this project, a p-value equal to 1 indicates that the data is perfectly random while a p-value equal to 0 indicates that the data is completely non random. If the p-value is greater than or equal to α, the null hypothesis is accepted and the data appears to be random. If the p-value is less than α, the null hypothesis is rejected and the data is deemed non random.

2.2.2 Studies on Randomness Testing

The following sections explores the applications of randomness testing and its use in cryptography.

Given that encryption is essential for maintaining data security in cloud computing, (Mohamed et al. (2012)) performed randomness testing on eight modern encryption techniques, including AES, MARS and DES. They tested on two different platforms, desktop computer and Amazon EC2 Micro Instance. They evaluated the encryption techniques implemented as Pseudo Random Number Generators (PRNGs). They used the NIST Test Suite to perform the randomness testing. With a significance level of 0.01, any p-value less than 0.01 meant that sequence was rejected. They found no strong evidence of any statistical non randomness across the 8 encryption algorithms however some differences were found between them on the two different platforms.

Statistical analysis has been run on an enhanced SDEx encryption method based on the SHA-512 hash function (Hłobaż (2020)). Using various tests like frequency, cumulative sums and runs, with a significance level of 0.01, it was found that this encryption algorithm was sufficiently random and passed the tests. They concluded that this SDEx method based on the SHA-512 hash function was quicker and equally or more secure than AES with a 256-bit key. They hope to use this method to secure end-to-end encryption for data transfer.

Similarly, statistical tests for randomness have been run on new algorithms, like a proposed stream cipher cryptographic algorithm based on the popular Vernam Cipher (Brosas et al. (2020)). The algorithm had a success rate of 99.5% across the statistical tests performed on it, which included frequency and longest runs of one's tests. Due to this success, the paper deemed the proposed algorithm effective in producing a random ciphertext sequence and detailed further work of implementing it to help secure medical records.

SHA256 (a hashing algorithm) is vulnerable to length extension attacks which involve misusing particular hashes as authentication codes and using them to include extra information. (Cortez et al. (2020)) introduces a new and improved padding scheme and hashing process for SHA256 to deal with this issue. To verify that the solution is cryptographically secure, statistical tests for randomness are performed on the output of the Message Digest. Tests such as monobit frequency, frequency within a block and runs were carried out on the data. The results validate that the number of ones and zeros are randomly distributed in the final hash value.

Statistical tests for randomness were also used to identify encrypted and unencrypted bit sequences (Wu et al. (2015)). Unencrypted bit sequences are less random than encrypted ones. From the SP800-22 rev1a standard, five tests were selected and a significance level of 0.01 was chosen. If the sequence passes more than 3 of the tests, it was concluded that that sequence was encrypted. Otherwise the sequence was concluded as unencrypted. The results of the experiment were that 89% of the time, unencrypted sequences were identified correctly and 99% of the time encrypted sequences were identified correctly.

2.2.3 Examples of Randomness Failures

Randomness failures pose a serious threat to cryptographic security (Schuldt and Shinagawa (2017)). The consequences can be severe and there are many examples of real-world incidents.

There are many examples of pseudo random number generators (PRNGs) failing and being guessable. Notably, the Debian Linux vulnerability in 2008 that left cryptographic keys to be guessable. It was caused by the code used to gather entropy, which provided the

seed the PRNG used to create private keys, were removed. This resulted in only 32,768 possible keys meaning the connections made with these keys were insecure (Schneier (2008)

In 2015, Juniper Networks announced that there were multiple security vulnerabilities due to unauthorised code in their operating system, for their NetScreen VPN routers, called ScreenOS (Checkoway et al. (2016)). These vulnerabilities were due to Juniper's use of Dual EC (Elliptical Curve) as a PRNG. Dual EC had a weakness that was exploited in the Juniper incident. It was possible for an attacker, who knew the discrete logarithm of an input parameter Q with respect to a generator point, to see a number of consecutive bytes from the output and hence calculate the internal state of the generator. This allowed the attacker to predict all the future output of the generator. They were able to exploit this lack of randomness and passively decrypt VPN traffic.

Bitcoin thefts in 2013 were due to a compromised PRNG used in Android wallets (bitcoin.org (2013). Applications on Android using Java Cryptography Architecture (JCA) for key generation, signing and generating random numbers were not receiving cryptographically strong values because of an improper initialization of the underlying PRNG SecureRandom on Android devices (Klyubin (2013). The predictability of the values being generated by SecureRandom was exploited and attackers were able to guess the private keys used in Bitcoin Wallets and steal the Bitcoins the wallet contained. Again, attackers were able to exploit the lack of randomness.

Chapter 3

Design

This section explains how the test suite used was chosen, the tests contained in that test suite and the other tests and dataset used to analyse the TEKs.

3.1 Challenges

3.1.1 Review of Test Suites

There are a variety of statistical test suites that have been developed to test for randomness. Also referred to as batteries, these are a collection of statistical tests. In this section I will review these test suites in order to choose the appropriate one for this project's purposes. The notable test suites are Dieharder and NIST STS, being the most widely used and therefore most tested.

Dieharder (Brown (2003)) is a random number test suite which was designed to test RNGs used in a variety of applications such as cryptography and computer simulation. It consists of 26 tests, extending the original Diehard battery which was created by George Marsaglia in 1995 (Luengo and Villalba (2021)). This testing suite is widely used (Luengo (2022)) (Hurley-Smith and Hernandez-Castro (2020)) (Lu et al. (2023)) and is well tested. It is designed to be able to change the parameters in order to make failure

unambiguous. It incorporates many of the tests in NIST STS. Because Dieharder is open source and encourages users to give feedback, contributions are continuously being made to improve and bug fix the tests. This results in a more reliable and stronger test suite than one that is closed or lacks a good feedback system like NIST STS (Brown (2003).

NIST STS stands for the National Institute of Standards and Technology (NIST) Statistical Test Suite (NIST (2012)). It is used to test RNGs used in applications such as cryptography, modelling and simulation. It contains 15 tests and is widely used (Mohamed et al. (2012)) (Hłobaż (2020)). These are standardised with its test parameters fixed, reducing its flexibility.

TestU01 (L'Ecuyer and Simard (2007)) is a test suite that was created by L'Ecuyer and Simard and implemented in C. It is an extensive battery that incorporates some of the tests in Dieharder and NIST STS (Luengo and Villalba (2021)). It consists of six test batteries each focusing on testing a different aspect of randomness. It only accepts 32-bit input and interprets it as values with the range of 0 and 1 (L'Ecuyer and Simard (2007). This can lead to inaccuracies in the most-significant bits.

Ent (Walker (2002)) is a lesser used test suite created by Walker (Luengo and Villalba (2021)). Its purpose is testing for simulation and cryptographic applications. It has two modes, binary and byte, with different statistics being calculated depending on the mode. Although it is fast and simple, the Ent battery has some issues with dependencies between tests, for example the Entropy test and the Chi-squared test (Luengo and Villalba (2021)).

3.2 Methodology

3.2.1 Data Preparation

The data used in this project was sourced from the Testing Apps for COVID-19 Tracing (TACT) project by Farrell and Leith (Leith and Farrell (2023). The TACT project was a study on whether the BLE used in GAEN-based contact tracing applications was effective at identifying users who were in proximity for long enough to be deemed as exposed to covid, if one of the users was later positive for the virus. The project ran from April 2020 until September 2023 and a number of reports were written on the findings, these include (Leith and Farrell (2020a)), (Leith and Farrell (2020b)) and (Leith and Farrell (2021))

While the project was ongoing, the TEKs being published in 33 regions, including Ireland, Germany and Brazil, were downloaded hourly. This resulted in a huge amount of data and allowed for insight into the functioning of these apps. The TEKs downloaded were in a large number of zip files.

In order to get the keys in the correct format to test, first the keys were extracted from the zipped files. Following this, duplicate keys were identified and removed, resulting in a file composed of only the unique keys. This significantly reduced the data size, from an initial 56GB of all the keys in the zip files, down to 4GB, removing a substantial amount of duplicate keys. The final dataset consists of a total of 137 million unique TEKs in ascii format, allowing for efficient analysis of the keys. For Dieharder, the TEKs were shuffled to make sure they were not sorted and converted in raw binary.

3.2.2 Chosen Test Suite

Dieharder was selected as the test suite for this project as it is widely used and well tested. It also accepts files of numbers as input, which is useful as our data is a file of TEKs. NIST STS was the other potential candidate however it only accepts streams of

data being produced by an RNG, which is not applicable in this project. The other test suites, TestU01 and Ent are much lesser used and less reputable than NIST STS and Dieharder and thus were not chosen. Therefore, Dieharder was the appropriate choice to test the TEKs, given its input type, wide range of tests and good reputation.

Description of Dieharder Tests

Below are a description of all the Dieharder tests, . ((Brown (2003)), (Luengo and Villalba (2021)))

- "Birthdays" test: this test selects m birthdays of a year of n days. It creates a list of the intervals between two birthdays (time between two consecutive events). It expects that the repeated intervals are distributed in a Poisson distribution, if the data is random.

- Overlapping 5-Permutations Test (OPERM5): this test studies a sequence of one million 32-bit random integers. There are 120 possible permutations of each set of 5 consecutive integers. The number of appearances of each permutation is recorded. Each permutation is expected to appear with equal probability across the sequence.

- 32x32 Binary Rank Test: 32x32 matrices are randomly formed from the data. The rank of the matrix is determined. The rank can be a value from 0-32, with infrequent ranks below 29 being pooled with those of rank 29. A chi-squared test is performed on the counts of matrices for ranks 32, 31, 30 and $\leq$ 29. This test is performed on 400,000 matrices each time.

- 6x8 Binary Rank Test: six random 32-bit integers are taken from the data. A specified byte is chosen and the six bytes, one from each 32-bit integer, create a 6x8 matrix. The rank of the matrix is determined.The rank can be a value from 0-6, with infrequent ranks below 4 being pooled with those of rank 4. A chi-squared test is performed on the counts of matrices for ranks 6, 5 and $\leq$ 4. This test is performed on 100,000 matrices each time.

- Bitstream Test: in this test, the data is viewed as a stream of bits, a = {ai} and an alphabet with two letters 0 and 1. The stream is considered as successive 20 letter words that overlap. For example, the first word would be bits a1-a20 and the second word would be bits a2-a21. The test counts the number of missing 20-letter words in a string of 2^{21} overlapping 20-letter words. Given that there are 2^{20} possible overlapping 20-letter words, the number of missing words j in a 2^{21+19} letter (bit) string is expected to be (nearly) normally distributed with a mean of 141,909 and sigma of 428. This test is repeated 20 times each time.

- Overlapping Pairs Sparse Occupance (OPSO): This test considers 2 letter words from an alphabet containing 1024 letters. Each letter is found by a specified 10 bits from a 32-bit integer in the data. 2^{21} overlapping 2-letter words from 2^{21+1} "keystrokes" are generated by the test and it counts the number of 2-letter words that do not appear in the data. These counts are expected to be (nearly) a normal distribution with mean of 141,900 and sigma 290. Therefore the number of missing words minus the mean divided by sigma should be a standard normal variable. This test extracts 32 bits at a time from the data file and uses a specific 10 bits, the file is then restarted and the next 10 bits are taken and so on.

- Overlapping Quadruples Sparse Occupancy (OQSO): This test is similar to OPSO but takes 4-letter words from an alphabet of 32 letters. Each letter is determined by a specific string of 5 bits from the data, which is assumed to contains 32-bit random numbers. The test calculates the average number of missing words in a sequence of 2^{21} four-letter words, which is equivalent to 2^{21+3} "keystrokes". The average number of missing words is 141909, with a standard deviation (sigma) of 295. It compares the results with this expected distribution.

- DNA Test: this test considers an alphabet made up of four letters: C, G, A, and T. These letters are determined by two designated bits in the sequence being tested. The test looks at words that are 10 letters long, like OPSO and OQSO tests, which

means there are 2^{20} possible words. For a string of 2^{21} overlapping 10-letter words (which equals 2^{21+9} "keystrokes"), the average number of missing words is 141909. The standard deviation, sigma, is 339. It compares the results with this expected distribution.

- Count the 1s (stream): This test considers the data being tested as a stream of bytes, with four bytes making up each 32-bit integer. Each byte can contain anywhere from 0 to 8 occurrences of the number 1, with different probabilities for each count. This stream of bytes is considered as a series of overlapping 5-letter words. Each "letter" in these words is determined by the number of 1s in a byte: 0, 1, or 2 gives A, 3 gives B, 4 gives C, 5 gives D, and 6, 7, or 8 gives E. There are 5^5 possible 5-letter words, the count of how often each word appears in a string of 256,000 overlapping 5-letter words is calculated. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chi-squared test. The test returns two p-values for 5-letter and 4-letter cell counts.

- Count the 1s Test (byte): The test considers the data being tested as a series of 32-bit integers. From each integer, specific byte is selected, the leftmost byte (bits 1 to 8). This byte can have anywhere from 0 to 8 instances of the number 1, with probabilities of 1, 8, 28, 56, 70, 56, 28, 8, and 1 out of 256. These specified bytes are taken from consecutive integers and turned into a string of overlapping 5-letter words. In these words, each "letter" is determined by how many times the number 1 appears in that byte: 0, 1, or 2 gives A, 3 gives B, 4 gives C, 5 gives D, and 6, 7, or 8 gives E. There are 5^5 possible 5-letter words, and from a set of 256,000 overlapping 5-letter words, how often each word appears is counted. The quadratic form in the weak inverse of the covariance matrix of the cell counts provides a chi-squared test. The test returns two p-values for 5-letter and 4-letter cell counts.

- Parking Lot Test: This test examines how attempts to randomly park a square car of length 1 on a 100x100 parking lot without crashing are distributed. The number of

attempts (n) is plotted against the number of attempts that didn't "crash" because the car squares overlapped (k). This is compared to what would be expected from a perfectly random set of parking coordinates. The results are compared to when n=12,000, where k should average 3523 with a standard deviation of 21.9. This average is very close to being normally distributed. The formula (k - 3523) / 21.9 is used to get a standard normal variable. Converting this to a uniform p-value, it is used as input for a KS test with a default of 100 samples.

- Minimum Distance (2d Circle) Test: In this test 8,000 points are randomly chosen from within a square of side 10,000. The minimum distance (d) between (n^{r^2} - n)/2 points is computed. The minimum distance is squared to get d^2. If the data is random then d^2 should be (very close to) exponentially distributed with a mean of 0.995. Thus 1-exp(-d^2/.995) should be distributed according to a U(0,1) random variable and a KS test is preformed on the resulting 100 values serves to test for uniformity for random points in the square.

- 3d Sphere (Minimum Distance) Test: In this test 4,000 points are randomly chosen from within a cube of edge 1,000. At each point, a sphere large enough to reach the next closest point is centered. The volume of the smallest sphere is approximately exponentially distributed with mean 120pi/3. Thus the radius cubed is exponentially distributed with mean 30. 4000 spheres are generated 20 times by the test. Each minimum radius cubed corresponds to a uniform variable obtained by applying the function 1 - exp(-r^3/30) to the radius, and a KS test is done on the 20 p-values.

- Squeeze Test: This test floats random integers from the data to get uniformly distributed values on the interval (1,0). An integer k-2^{31} is multiplied by these values until it is reduced to 1. The test calculates t, the number of iterations that were required to reduce k to 1, where the reduction is k=ceiling(k*U) with U being the floating integers from the data being tested. t is computed 100,000 times with the number of times t is less than 7 and greater than 47 expected to be exponential.

- Sums Test (Broken): The test is noted in the documentation as unreliable and incorrect so this test will not be considered while testing the TEKs.

- Runs Test: This test counts the number of runs (increasing and decreasing) in the data. The covariance matrices for runs-up and runs-down are well-known, allowing for chi-square tests on quadratic forms involving the weak inverses of these covariance matrices. Runs are counted for sequences of length 10,000 and is done 10 times and then repeated.

- Craps Test: This test plays 200,000 games of 'craps' which is a dice game. The number of wins and the number of throws necessary to win are recorded. The number of wins is expected to be very close to a normal distribution with a mean of 200000p and a variance of 200000p(1-p), where p=244/495. The number of throws to end a game vary from 1 to infinity, but counts that are greater than 21 are grouped with 21. A chi-squared test is carried out on the number of throws. Each 32-bit integer from the data by normalising it to a an interval [0,1), then multiplying it by 6 and adding 1 to the integer part of the result.

- Greatest Common Divisor Marsaglia and Tsang Test: This test randomly chooses two 32-bit positive integers, u and v, from the data. Euclid's Method is applied to these values and two statistics are calculated: their greatest common divisor (w) and the number of steps (k) of Euclid's Method it took to find it. This results in two frequency tables, the number of appearances of each value for k and the number of occurrences of each GCD w. Chi-squared tests are done on each of these distributions and produces two p-values. 100 p-values are calculated and then a KS test is performed on them. 10^7 tsamples of u and v are generated.

- Monobit Test (STS): This test counts the 1 bits in the a long string of random integers chosen from the data. It compares this count to the expected number and returns a p-value.

- Runs Test (STS): This test counts the total number of 0 runs and the total number of 1 runs across a sample of bits selected from the data. If the data is random, then these counts should be uniformly distributed.

- Serial Test (STS): This test calculates the frequencies of overlapping n-tuples of bits throughout the data. The probabilities of the 2^n n-bit overlapping patterns is expected to be equal.

- Bit Distribution Test: This test calculates the frequencies of all non-overlapping n-tuples of bits in the data and compares this to a binomial distributions using chi-squared test.

- Generalised Minimum Distance Test: This test generalises the 2d circle and 3d sphere minimum distance tests and calculates the minimum distance between pairs of points in n dimensions, where n=2,3,4,5. It examines their distribution and compares it to the expected distribution.

- Permutations Test: This test counts the order permutations of n random numbers selected from the data. There are n! permutations, of which are expected to be equally likely. The n samples are independent and a chi-squared test on the counts with n! - 1 degrees of freedom is carried out.

- Lagged Sums Test: In this test pairs of values are selected from the data with a specific lag (distance) between them and these pairs are summed. The values are converted into p-values and a KS test is applied.

- Kolmongorov-Smirnov Test Test: This test generates a vector of uniform deviates from the data and then applies a KS test to it. The test is run multiple times and a final p-value is calculated.

- Byte Distribution Test: This test extracts n independent bytes from each of k consecutive words, incrementing indexed counters in all of the n tables resulting in a total of 256*n counters. A chi-squared fitting test is used to calculate the p-value.

- Discrete Cosine Transform (DCT) (Frequency Analysis) Test: This test performs a DCT on independent blocks of n-tuple words from the data. The absolute value of each transform is checked for uniformity and independence using a chi-squared test.

- Fill Tree Test: This test inserts words from the data into small binary trees. If a word is unable to be inserted into the tree, the current count of words in the tree and the would-be position of the word is recorded. The test produces two p-values from a chi-squared test against the expected values and a chi-squared test for uniformity of where the insertions are failing.

- Fill Tree 2 Test: This test is the same as the Fill Tree test above but uses bits instead of words.

- Monobit 2 Test: This test performs the Monobit Test described above on blocks of a specified length from the data.

P-values

The null hypothesis (that the data is random) is usually rejected if the p-value is less than the level of significance, which is usually 0.05. However, this is an issue for testing for randomness as a perfect RNG is expected return a p-value less than 0.05 5% of the time (Brown (2003). Since Dieharder runs each test multiple times, 100 p-values are returned by each test and it is expected that good random data will fail a test 5 times. This gives rise to uncertainty and does not provide conclusive results.

To remedy this and give accurate results, Dieharder treats the p-values as test statistics. The p-values returned by the tests should be uniformly distributed in the range of 0-1. The set of p-values is converted into a singular p-value, used to determine the result of the test, by using a Kolmogorv-Smirnov (KS) test to compare the set of p-values against the expected uniform distribution (Brown (2003)). This final p-value is used to determine if the test fails or passes.

3.2.3 Other Tests

Other tests to supplement the Dieharder test suite were implemented, these include: chi-squared test, spectral test, lag plot and plot of counts.

Lag Plot

A lag plot is a graphical test for randomness, it displays any patterns or relationships in the data. Random data should not have any identifiable structure in the lag plot, a structure in the lag plot indicates that the data is not random (NIST/SEMATECH (2012). A lag plot involves plotting a set of the data against another set of the data that occurs later. For example, given a data set Y1, Y2 ..., Yn, Y2 and Y7 have lag 5 since 7 - 2 = 5. Lag plots can be generated for any arbitrary lag (NIST/SEMATECH (2012). A plot of lag 1 would be a plot of Yi against Yi-1.

Chi-Squared Test

The chi-squared test for randomness iterates through the list of TEKs, counting the occurrences of '0' and '1' in each key. The observed frequencies are calculated based on these counts. Assuming equal probability for '0' and '1', the expected frequencies are calculated. The chi-squared statistic is computed and the corresponding p-value for the observed and expected frequencies. The chi-squared statistic measures the difference between the observed frequencies and the expected frequencies under the null hypothesis of independence. The p-value indicates the probability of obtaining a chi-squared statistic as extreme as the observed one, assuming that the null hypothesis is true. This test helps to quantify the degree of randomness in the TEK dataset, providing valuable insights into its distribution and potential biases.

Spectral Test

This test looks at the peak heights in the discrete Fast Fourier Transform. The test detects periodic features (repetitive patterns that are near each other) in the sequence that would indicate non randomness. Bassham et al. (2010). The binary strings are converted into a sequence of values where '0' is replaced by -1 and '1' is replaced by 1. It computes the discrete Fourier transform of the sequence and calculates the modulus of the first half of the Fourier transform. It computes a threshold value (tau) based on the length of the sequence (n) and the significance level (5 percent). It counts the actual number of peaks that exceed the threshold (tau). It calculates a statistic (d) based on the counts of peaks. It computes the p-value using the complementary error function (spc.erfc). If the p-value is less than 0.05 (indicating statistical significance at the 5 percent level), that sequence is rejected and the count of significant results is incremented. It returns the amount of significant results (the amount of rejected keys) out of all the tested sequences.

Plot of Counts

This is a visualisation used to provide an insight into the distribution and randomness of the data. The number of ones and zeros in each bit position are counted and the results are plotted. A distribution where each bit position shows roughly equal counts of 1s and 0s indicates randomness, while deviations from this could suggest non randomness. Plotting these counts helps with the identification of any biases or correlations that might exist within the data, aiding in the assessment of its randomness.

Hilbert Curve

The Hilbert curve can be used to visually represent randomness or non randomness in data. The Hilbert curve is a space-filling curve that maps the data while preserving locality and gives a meaningful presentation of the data (Estevez-Rams et al. (2015)as it allows for the identification of patterns.

If the data is randomly distributed, its Hilbert curve will display a relatively uniform distribution throughout its space-filling path. This means that nearby points in the original dataset will likely remain close to each other in the curve, creating a visually uniform pattern. If the data contains some non randomness, its Hilbert curve will reflect this by displaying areas of higher density or clustering along portions of the curve. By visualizing data using the Hilbert curve, patterns in the data can be identified , providing insights into the randomness or non randomness of the dataset.

3.2.4 Random Dataset

A dataset containing random numbers was sourced from Random.org (Haahr (1998). This website generates true random numbers from atmospheric noise. These numbers are better than pseudo random numbers as they do not require a seed and are truly random. This service was created in 1998 by Dr Mads Haahr in Trinity College Dublin and is now run by Randomness and Integrity Services Ltd.

Previous testing has been done to ensure the randomness of Random.orgs numbers, including previous dissertations in 2005 and 2001 (Haahr (1998). Dieharder was also used to validate the numbers being generated.

The same randomness testing is applied to this random dataset as the TEKs. This allows for a baseline to compare the results to as well as validating the tests. The dataset contains 169 million 128-bit numbers, very similar to the set of TEKs.

Chapter 4

Evaluation

This section analyses the results from Dieharder and the other tests on the TEKs and the random keys.

4.1 Dieharder Results

Dieharder is designed to push the tests to unambiguous failure (Brown (2003). It contains a number of flag options to alter the parameters of the tests and change their acceptance criteria. The command used to run the Dieharder test suite for this project was:

```
Dieharder -a -k 2 -Y 1 -f <filename>
```

The -a flag runs all the tests in the Dieharder test suite, as described in section 3.2.2. The -k flag is to decide the KS settings. The -Y flag is the Xtrategy flag which is used to control the 'test to failure' modes (Brown (2003). This flag is set to 1 to use the 'resolve ambiguity' mode. Dieharder can return 'weak' as a test result which can be difficult to interpret. Even perfect random numbers will return some 'weak' results at some point because the p-values are uniformly distributed and will have a result in the tails of the distribution from time to time. Even if a test returns more than one weak result, this is not conclusive evidence that the data is non random. The 'resolve ambiguity' mode resolves this issue by adding p-samples (in blocks of 100) until the test results in a definitive

pass, weak or it proceeds to failure.

An extract of the Dieharder results for the TEKs can be seen in table 4.1 and the results for the random keys can be seen in table 4.2. The full outputs are in the Appendix, in table 5.1 and 5.2 respectively.

The TEKs passed all of the Dieharder tests, which indicates that the TEKs are good random numbers and do not show any obvious pattern that would suggest non randomness.

Interestingly, the random keys passed the majority of the tests, but had nine weak results and one failure, out of the 125 results returned by Dieharder. The weak results are not a cause for concern as there is some statistical fluctuation when it comes to RNGs and the numbers they produce. However, the failure of the Greatest Common Divisor Marsaglia and Tsang test is unexpected and would be worth further exploration by Random.org (the source of the random keys dataset) to find the cause.

Test Name	n	N	d	p-value	Result
diehard_birthdays	0	100	100	0.98740819	PASSED
diehard_operm5	0	1000000	100	0.95452336	PASSED
diehard_rank_32x32	0	40000	100	0.59831249	PASSED
diehard_rank_6x8	0	100000	100	0.54719972	PASSED
diehard_bitstream	0	2097152	100	0.38293106	PASSED
diehard_opso	0	2097152	100	0.35772252	PASSED
diehard_oqso	0	2097152	100	0.69390124	PASSED
diehard_dna	0	2097152	100	0.42865960	PASSED

Table 4.1: Extract from Dieharder Results of TEKS.

Test Name	n	N	d	p-value	Result
diehard_birthdays	0	100	100	0.76394892	PASSED
diehard_operm5	0	1000000	100	0.18935905	PASSED
diehard_rank_32x32	0	40000	100	0.21580801	PASSED
diehard_rank_6x8	0	100000	100	0.02770416	PASSED
diehard_bitstream	0	2097152	100	0.75370418	PASSED
diehard_opso	0	2097152	100	0.32644238	PASSED
diehard_oqso	0	2097152	100	0.89648137	PASSED
diehard_dna	0	2097152	100	0.59449638	PASSED

Table 4.2: Extract from Dieharder Results of Random keys.

4.2 Other Test Results

4.2.1 Plot of Counts

Figure 4.1 shows the result of the plot of counts on the set of TEKs. Across all 128 bit positions, the percentage of ones and zeros are equal at 50%. The straight line down the middle of the chart clearly shows that there is an equal number of ones and zeros in the set of TEKs. This result would indicate randomness as one of the characteristics of a sequence of bits being random is an equal probability of a one appearing or a zero appearing.

Figure 4.2 shows the result of the plot of counts on the set of random keys. Like the TEKs, the percentage of ones and zeros across all 128 bit positions are equal at 50%. This is the result that was expected from the set of random numbers. The graphs from both the TEKs and the random keys are identical and there is no evidence from this test of non randomness within the TEKs.

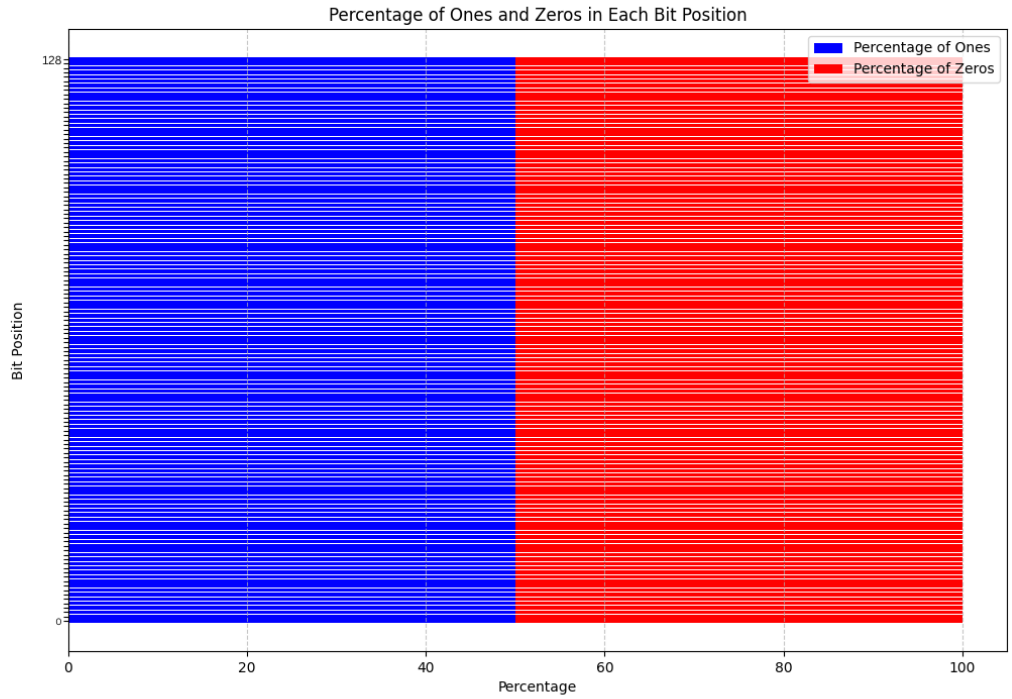

Figure 4.1: Plot of Counts for TEKs

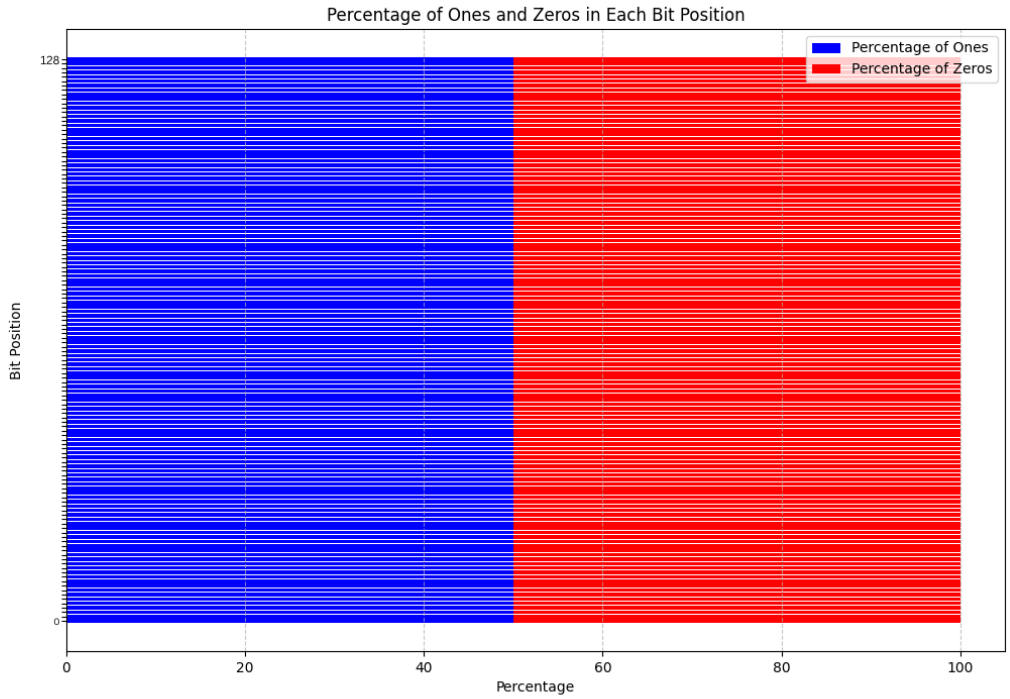

Figure 4.2: Plot of Counts for random keys

4.2.2 Hilbert Curve

The Hilbert Curve of the TEKs can be seen in Figure 4.3 and the Hilbert Curve of the random keys can be seen in Figure 4.4. The Hilbert curve produced from both datasets are very similar and both show no evidence of non randomness. A uniform pattern in the graph indicates that the data is randomly distributed. There is no clustering or high density areas within either graph that would indicate non randomness.The curves are this particular shape because of the approach the Hilbert curve takes when visiting each of the keys to plot them. It fills the space systematically.

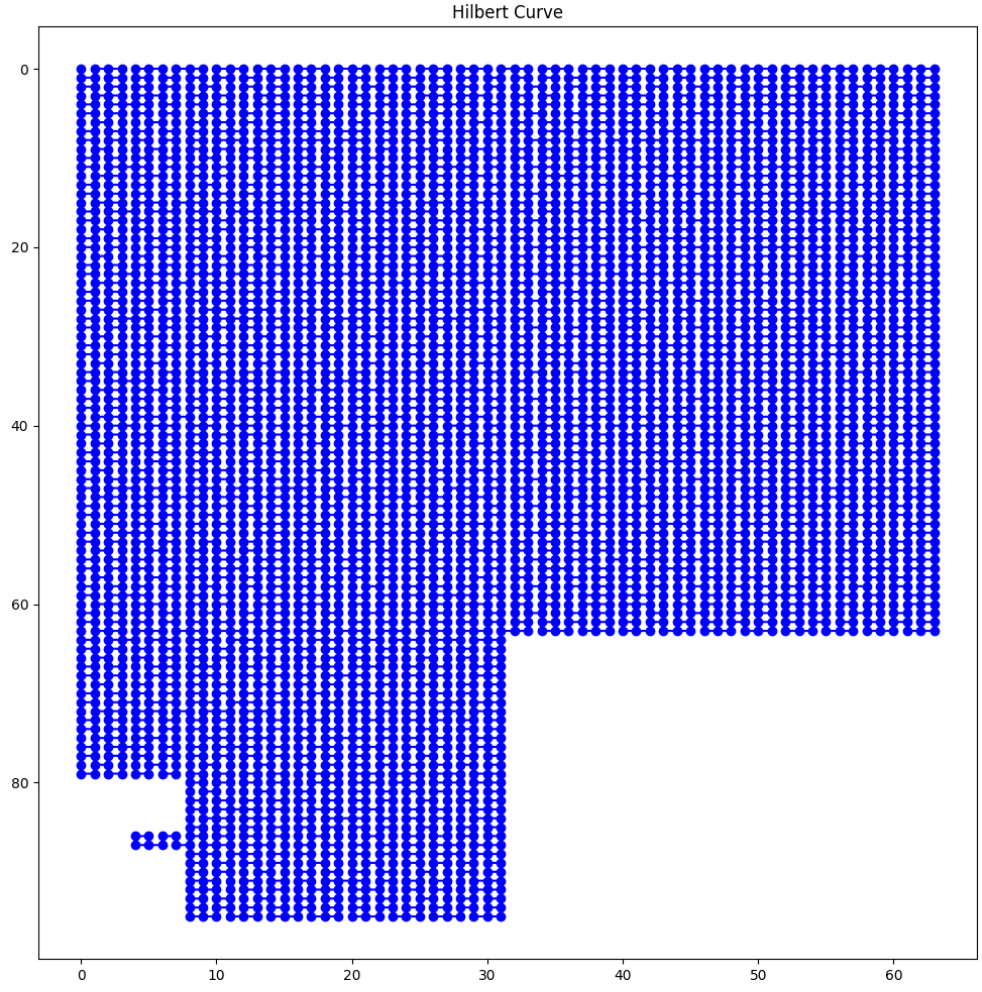

Figure 4.3: Hilbert Curve of TEKs

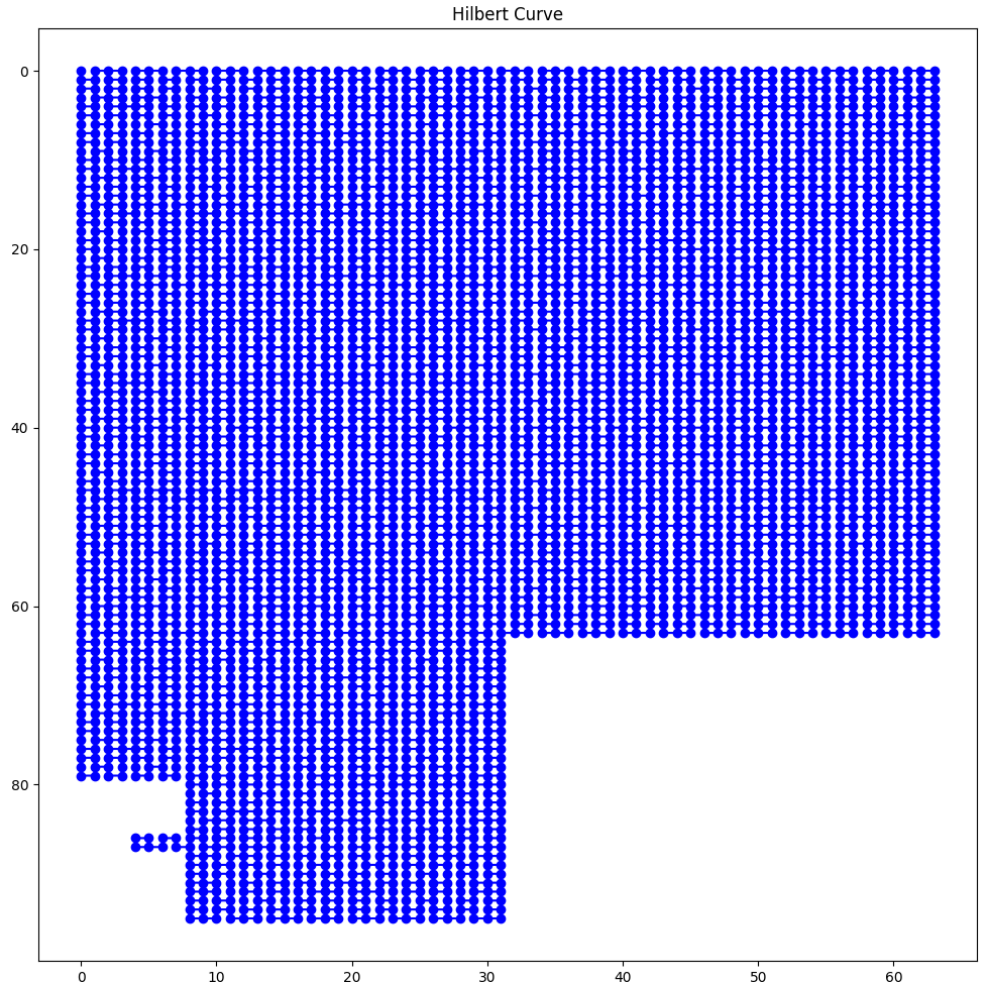

Figure 4.4: Hilbert Curve of Random keys

4.2.3 Lag Plot

The lag plot for the TEKs are show in Figure 4.5 and the random keys in Figure 4.6. The lag plot of both of the datasets are extremely similar, showing the same shape and pattern. When examining the lag plots of the two datasets side by side, the patterns in the plots are nearly identical. There is a discernible pattern in the plots and the points are not as randomly scattered across the plot as expected. The similarity in the lag plots

of the two datasets suggests that they share similar correlation structures or randomness properties.

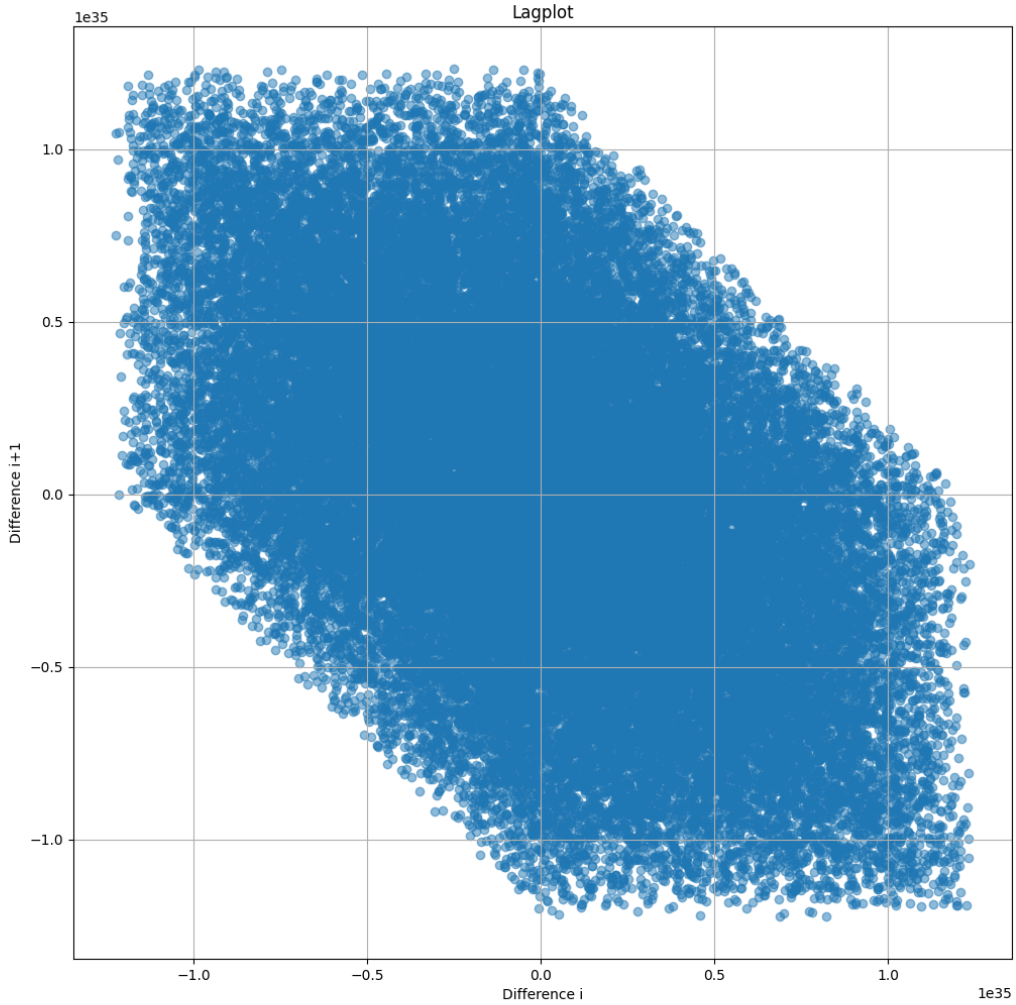

Figure 4.5: Lag Plot of TEKs

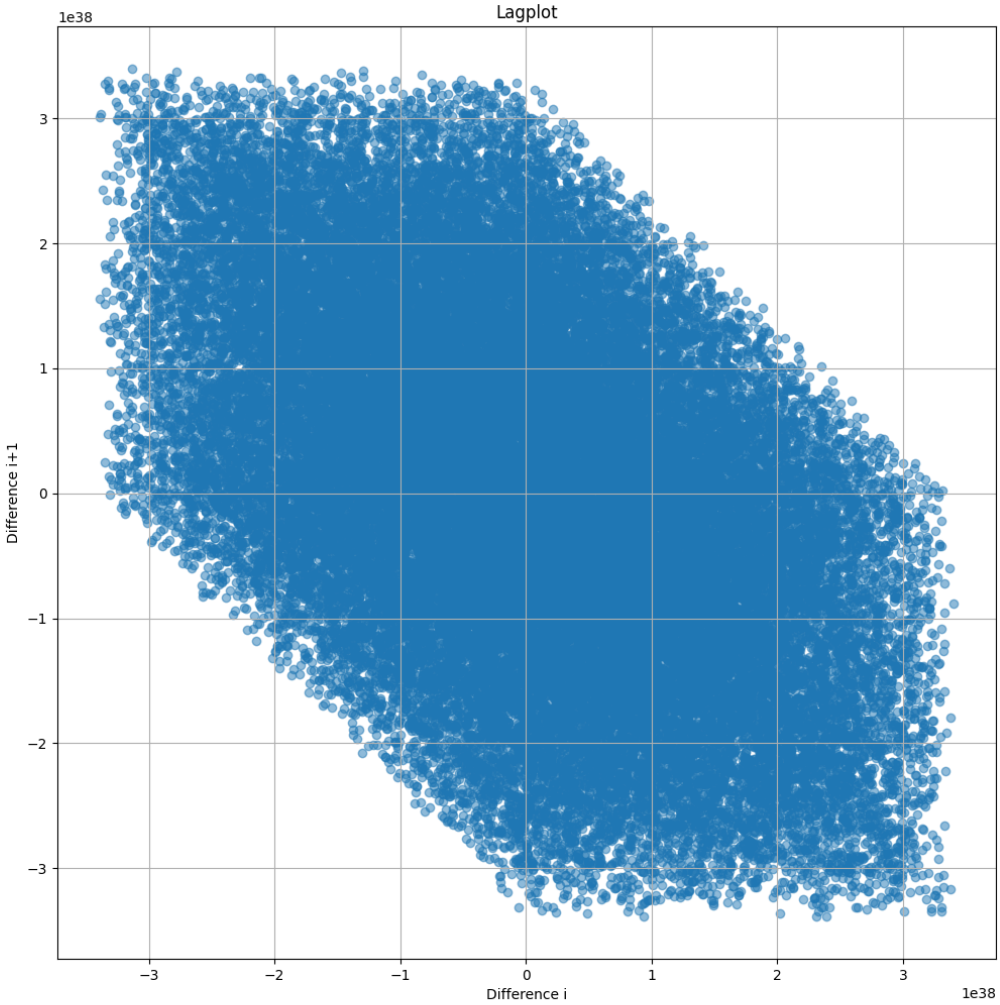

Figure 4.6: Lag Plot of Random keys

4.2.4 Chi-squared Test

Dataset	**Chi-squared Statistic**	**P-value**
TEKs	0.4601405325882503	0.49755830241590926
Random keys	0.6229208548153906	0.42996394455596043

Table 4.3: Chi-squared Test Result

The results of the chi-squared test on the TEKs and the random keys are shown in 4.3.

For the TEKs, the chi-squared statistic is calculated to be 0.4601, with a corresponding p-value of approximately 0.4976. The expected frequencies for this dataset are [8,722,282,911.0, 8,722,282,911.0]. Similarly, for the random keys dataset, the chi-squared statistic is 0.6229, with a corresponding p-value of approximately 0.4300. The expected frequencies for this dataset are [10,765,893,632.0, 10,765,893,632.0].

For both datasets, the p-values are much larger than the significance level of 0.05 (5%). This indicates that there is not sufficient evidence to reject the null hypothesis of independence between the observed frequencies and the expected frequencies. There is no significant difference between the observed and expected frequencies in either the TEKs or the random keys.

The chi-squared statistics are also low, supporting the conclusion that the observed frequencies closely match the expected frequencies. These results suggest that the distribution of frequencies within both datasets follows the expected patterns and does not give evidence of non randomness.

4.2.5 Spectral Test

Dataset	**no. rejected**	**total**	**Percentage %**
TEKs	6080630	137358786	4.426822758902368
Random keys	7501794	169541632	4.424750376355938

Table 4.4: Spectral Test Result

The results of the spectral test on the TEKs and the random keys, shown in 4.4 reveal that a similar percentage of keys were rejected by the test for both datasets.

For the TEKs, out of a total of 137,358,786 samples, 6,080,630 samples were rejected. This corresponds to a rejection rate of 4.43%. Similarly, for the set of random numbers, out of a total of 169,541,632 samples, 7,501,794 samples were rejected, corresponding to a rejection rate of 4.42%.

Both datasets have a rejection rate of approximately 4.4%, which is below the level of significance of 5%. This indicates that the observed patterns or spectral characteristics in the datasets are not statistically significant at the 5% level. Therefore, while some keys were rejected by the test, the overall percentage of rejections is within an acceptable range and there is no significant indication of non randomness within the TEKs.

4.3 Conclusions

In conclusion, this dissertation thoroughly examined the randomness properties of keys produced by GAEN (TEKs) and found strong evidence supporting their randomness. Through statistical analysis and testing, the distribution, correlations, and patterns within the TEKs dataset was examined to detect any deviations from randomness.

This analysis produced no evidence to indicate non randomness within the TEKs. Across various statistical tests and methodologies, the TEKs consistently showed characteristics of randomness. By being random, TEKs enhance the confidentiality and security of sensitive data exchanged in a GAEN contact tracing system. Furthermore, the absence of non random patterns in TEKs improves the confidence in the reliability and effectiveness of cryptographic mechanisms related to these keys.

In conclusion, this study findings confirm the randomness of TEKs, validating their role in the security and privacy of in GAEN contact tracing applications.

Chapter 5

Appendix

5.0.1 TEKs Results

Table for teks

Test Name	n	N	d	p-value	Result
diehard_birthdays	0	100	100	0.98740819	PASSED
diehard_operm5	0	1000000	100	0.95452336	PASSED
diehard_rank_32x32	0	40000	100	0.59831249	PASSED
diehard_rank_6x8	0	100000	100	0.54719972	PASSED
diehard_bitstream	0	2097152	100	0.38293106	PASSED
diehard_opso	0	2097152	100	0.35772252	PASSED
diehard_oqso	0	2097152	100	0.69390124	PASSED
diehard_dna	0	2097152	100	0.42865960	PASSED
diehard_count_1s_str	0	256000	100	0.29872959	PASSED
diehard_count_1s_byt	0	256000	100	0.08174702	PASSED
diehard_parking_lot	0	12000	100	0.91403506	PASSED
diehard_2dsphere	2	8000	100	0.23586557	PASSED
diehard_3dsphere	3	4000	100	0.63468855	PASSED
diehard_squeeze	0	100000	100	0.28961226	PASSED

Test Name	n	N	d	p-value	Result
diehard_sums	0	100	100	0.00750044	PASSED
diehard_runs	0	100000	100	0.88654068	PASSED
diehard_runs	0	100000	100	0.23670757	PASSED
diehard_craps	0	200000	100	0.89199102	PASSED
diehard_craps	0	200000	100	0.68235234	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.03007484	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.32230404	PASSED
sts_monobit	1	100000	100	0.15180696	PASSED
sts_runs	2	100000	100	0.50053774	PASSED
sts_serial	1	100000	100	0.05380206	PASSED
sts_serial	2	100000	100	0.84490409	PASSED
sts_serial	3	100000	100	0.39453536	PASSED
sts_serial	3	100000	100	0.12475855	PASSED
sts_serial	4	100000	100	0.01782397	PASSED
sts_serial	4	100000	100	0.62109194	PASSED
sts_serial	5	100000	100	0.46912814	PASSED
sts_serial	5	100000	100	0.77530843	PASSED
sts_serial	6	100000	100	0.29587704	PASSED
sts_serial	6	100000	100	0.56715645	PASSED
sts_serial	7	100000	100	0.84905787	PASSED
sts_serial	7	100000	100	0.40116976	PASSED
sts_serial	8	100000	100	0.93664924	PASSED
sts_serial	8	100000	100	0.10486711	PASSED
sts_serial	9	100000	100	0.16044494	PASSED
sts_serial	9	100000	100	0.05301976	PASSED
sts_serial	10	100000	100	0.92312992	PASSED

Test Name	n	N	d	p-value	Result
sts_serial	10	100000	100	0.35673434	PASSED
sts_serial	11	100000	100	0.98426112	PASSED
sts_serial	11	100000	100	0.80067302	PASSED
sts_serial	12	100000	100	0.99373847	PASSED
sts_serial	12	100000	100	0.88803021	PASSED
sts_serial	13	100000	100	0.66824428	PASSED
sts_serial	13	100000	100	0.50746190	PASSED
sts_serial	14	100000	100	0.98238726	PASSED
sts_serial	14	100000	100	0.26538675	PASSED
sts_serial	15	100000	100	0.39728743	PASSED
sts_serial	15	100000	100	0.97041422	PASSED
sts_serial	16	100000	100	0.25630622	PASSED
sts_serial	16	100000	100	0.05364901	PASSED
rgb_bitdist	1	100000	100	0.44512215	PASSED
rgb_bitdist	2	100000	100	0.23502241	PASSED
rgb_bitdist	3	100000	100	0.32191988	PASSED
rgb_bitdist	4	100000	100	0.22761597	PASSED
rgb_bitdist	5	100000	100	0.30123524	PASSED
rgb_bitdist	6	100000	100	0.83677612	PASSED
rgb_bitdist	7	100000	100	0.68327297	PASSED
rgb_bitdist	8	100000	100	0.91556750	PASSED
rgb_bitdist	9	100000	100	0.98795982	PASSED
rgb_bitdist	10	100000	100	0.19379581	PASSED
rgb_bitdist	11	100000	100	0.53441056	PASSED
rgb_bitdist	12	100000	100	0.64656408	PASSED
rgb_minimum_distance	2	10000	1000	0.95411695	PASSED

Test Name	n	N	d	p-value	Result
rgb_minimum_distance	3	10000	1000	0.54118074	PASSED
rgb_minimum_distance	4	10000	1000	0.25746852	PASSED
rgb_minimum_distance	5	10000	1000	0.52797286	PASSED
rgb_permutations	2	100000	100	0.68240647	PASSED
rgb_permutations	3	100000	100	0.71482801	PASSED
rgb_permutations	4	100000	100	0.95248263	PASSED
rgb_permutations	5	100000	100	0.76625613	PASSED
rgb_lagged_sum	0	1000000	100	0.86229846	PASSED
rgb_lagged_sum	1	1000000	100	0.85446006	PASSED
rgb_lagged_sum	2	1000000	100	0.72559541	PASSED
rgb_lagged_sum	3	1000000	100	0.44885925	PASSED
rgb_lagged_sum	4	1000000	100	0.96344825	PASSED
rgb_lagged_sum	5	1000000	100	0.25164161	PASSED
rgb_lagged_sum	6	1000000	100	0.28999845	PASSED
rgb_lagged_sum	7	1000000	100	0.25609882	PASSED
rgb_lagged_sum	8	1000000	100	0.29886696	PASSED
rgb_lagged_sum	9	1000000	100	0.63708913	PASSED
rgb_lagged_sum	10	1000000	100	0.11998965	PASSED
rgb_lagged_sum	11	1000000	100	0.02340253	PASSED
rgb_lagged_sum	12	1000000	100	0.92961604	PASSED
rgb_lagged_sum	13	1000000	100	0.53433063	PASSED
rgb_lagged_sum	14	1000000	100	0.63635183	PASSED
rgb_lagged_sum	15	1000000	100	0.12766584	PASSED
rgb_lagged_sum	16	1000000	100	0.35077156	PASSED
rgb_lagged_sum	17	1000000	100	0.75718004	PASSED
rgb_lagged_sum	18	1000000	100	0.35537855	PASSED

Test Name	n	N	d	p-value	Result
rgb_lagged_sum	19	1000000	100	0.73789139	PASSED
rgb_lagged_sum	20	1000000	100	0.14002184	PASSED
rgb_lagged_sum	21	1000000	100	0.05885242	PASSED
rgb_lagged_sum	22	1000000	100	0.34112889	PASSED
rgb_lagged_sum	23	1000000	100	0.92939060	PASSED
rgb_lagged_sum	24	1000000	100	0.41104728	PASSED
rgb_lagged_sum	25	1000000	100	0.68323308	PASSED
rgb_lagged_sum	26	1000000	100	0.07730139	PASSED
rgb_lagged_sum	27	1000000	100	0.07269523	PASSED
rgb_lagged_sum	28	1000000	100	0.23320287	PASSED
rgb_lagged_sum	29	1000000	100	0.36469854	PASSED
rgb_lagged_sum	30	1000000	100	0.21883650	PASSED
rgb_lagged_sum	31	1000000	100	0.33511740	PASSED
rgb_lagged_sum	32	1000000	100	0.05785507	PASSED
rgb_kstest_test	0	10000	1000	0.65613857	PASSED
dab_bytedistrib	0	51200000	1	0.60605570	PASSED
dab_dct	256	50000	1	0.49995832	PASSED
dab_filltree	32	15000000	1	0.75555153	PASSED
dab_filltree	32	15000000	1	0.40968822	PASSED
dab_filltree2	0	5000000	1	0.10438267	PASSED
dab_filltree2	1	5000000	1	0.80708126	PASSED
dab_monobit2	12	65000000	1	0.74779198	PASSED

Table 5.1: Dieharder Results of TEKS.

5.0.2 Random keys Results

Table for random value

Test Name	n	N	d	p-value	Result
diehard_birthdays	0	100	100	0.76394892	PASSED
diehard_operm5	0	1000000	100	0.18935905	PASSED
diehard_rank_32x32	0	40000	100	0.21580801	PASSED
diehard_rank_6x8	0	100000	100	0.02770416	PASSED
diehard_bitstream	0	2097152	100	0.75370418	PASSED
diehard_opso	0	2097152	100	0.32644238	PASSED
diehard_oqso	0	2097152	100	0.89648137	PASSED
diehard_dna	0	2097152	100	0.59449638	PASSED
diehard_count_1s_str	0	256000	100	0.37707042	PASSED
diehard_count_1s_byt	0	256000	100	0.38613176	PASSED
diehard_parking_lot	0	12000	100	0.98428899	PASSED
diehard_2dsphere	2	8000	100	0.57449647	PASSED
diehard_3dsphere	3	4000	100	0.47967697	PASSED
diehard_squeeze	0	100000	100	0.02908390	PASSED
diehard_sums	0	100	100	0.01302183	PASSED
diehard_runs	0	100000	100	0.21268023	PASSED
diehard_runs	0	100000	100	0.48718248	PASSED
diehard_craps	0	200000	100	0.43589180	PASSED
diehard_craps	0	200000	100	0.48427891	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.39544378	PASSED
marsaglia_tsang_gcd	0	10000000	100	0.00073901	WEAK
marsaglia_tsang_gcd	0	10000000	200	0.20103622	PASSED
marsaglia_tsang_gcd	0	10000000	200	0.00000116	WEAK
marsaglia_tsang_gcd	0	10000000	300	0.15296148	PASSED

Test Name	n	N	d	p-value	Result
marsaglia_tsang_gcd	0	10000000	300	0.00000000	FAILED
sts_monobit	1	100000	100	0.46954912	PASSED
sts_runs	2	100000	100	0.99997882	WEAK
sts_runs	2	100000	200	0.65262205	PASSED
sts_serial	1	100000	100	0.75039800	PASSED
sts_serial	2	100000	100	0.46782328	PASSED
sts_serial	3	100000	100	0.96422500	PASSED
sts_serial	3	100000	100	0.38336381	PASSED
sts_serial	4	100000	100	0.78436629	PASSED
sts_serial	4	100000	100	0.32790423	PASSED
sts_serial	5	100000	100	0.93561180	PASSED
sts_serial	5	100000	100	0.78894832	PASSED
sts_serial	6	100000	100	0.30162394	PASSED
sts_serial	6	100000	100	0.45041009	PASSED
sts_serial	7	100000	100	0.60514729	PASSED
sts_serial	7	100000	100	0.52470608	PASSED
sts_serial	8	100000	100	0.38468429	PASSED
sts_serial	8	100000	100	0.97346520	PASSED
sts_serial	9	100000	100	0.67677659	PASSED
sts_serial	9	100000	100	0.21257824	PASSED
sts_serial	10	100000	100	0.06065198	PASSED
sts_serial	10	100000	100	0.19676120	PASSED
sts_serial	11	100000	100	0.25074198	PASSED
sts_serial	11	100000	100	0.44324414	PASSED
sts_serial	12	100000	100	0.06981655	PASSED
sts_serial	12	100000	100	0.26688079	PASSED

Test Name	n	N	d	p-value	Result
sts_serial	13	100000	100	0.50138141	PASSED
sts_serial	13	100000	100	0.51148391	PASSED
sts_serial	14	100000	100	0.38406027	PASSED
sts_serial	14	100000	100	0.32968726	PASSED
sts_serial	15	100000	100	0.20468696	PASSED
sts_serial	15	100000	100	0.21397465	PASSED
sts_serial	16	100000	100	0.16604967	PASSED
sts_serial	16	100000	100	0.60748515	PASSED
rgb_bitdist	1	100000	100	0.36452077	PASSED
rgb_bitdist	2	100000	100	0.94306987	PASSED
rgb_bitdist	3	100000	100	0.80290189	PASSED
rgb_bitdist	4	100000	100	0.29185388	PASSED
rgb_bitdist	5	100000	100	0.69257767	PASSED
rgb_bitdist	6	100000	100	0.95139967	PASSED
rgb_bitdist	7	100000	100	0.57919220	PASSED
rgb_bitdist	8	100000	100	0.45468153	PASSED
rgb_bitdist	9	100000	100	0.14850293	PASSED
rgb_bitdist	10	100000	100	0.54937469	PASSED
rgb_bitdist	11	100000	100	0.28545144	PASSED
rgb_bitdist	12	100000	100	0.99636354	WEAK
rgb_bitdist	12	100000	200	0.29765398	PASSED
rgb_minimum_distance	2	10000	1000	0.99960225	WEAK
rgb_minimum_distance	2	10000	1100	0.99544792	WEAK
rgb_minimum_distance	2	10000	1200	0.99960423	WEAK
rgb_minimum_distance	2	10000	1300	0.76490064	PASSED
rgb_minimum_distance	3	10000	1000	0.88490597	PASSED

Test Name	n	N	d	p-value	Result
rgb_minimum_distance	4	10000	1000	0.56548230	PASSED
rgb_minimum_distance	5	10000	1000	0.84433241	PASSED
rgb_permutations	2	100000	100	0.81514673	PASSED
rgb_permutations	3	100000	100	0.86382468	PASSED
rgb_permutations	4	100000	100	0.99152004	PASSED
rgb_permutations	4	100000	100	0.99152004	PASSED
rgb_permutations	5	100000	100	0.31316379	PASSED
rgb_lagged_sum	0	1000000	100	0.26035279	PASSED
rgb_lagged_sum	1	1000000	100	0.99920642	WEAK
rgb_lagged_sum	1	1000000	200	0.29519499	PASSED
rgb_lagged_sum	2	1000000	100	0.28259248	PASSED
rgb_lagged_sum	3	1000000	100	0.99684866	WEAK
rgb_lagged_sum	3	1000000	200	0.75546305	PASSED
rgb_lagged_sum	4	1000000	100	0.97658438	PASSED
rgb_lagged_sum	5	1000000	100	0.44382799	PASSED
rgb_lagged_sum	6	1000000	100	0.97652946	PASSED
rgb_lagged_sum	7	1000000	100	0.87715734	PASSED
rgb_lagged_sum	8	1000000	100	0.42657999	PASSED
rgb_lagged_sum	9	1000000	100	0.32151735	PASSED
rgb_lagged_sum	10	1000000	100	0.48476972	PASSED
rgb_lagged_sum	11	1000000	100	0.94092042	PASSED
rgb_lagged_sum	12	1000000	100	0.72099398	PASSED
rgb_lagged_sum	13	1000000	100	0.81002140	PASSED
rgb_lagged_sum	14	1000000	100	0.15611012	PASSED
rgb_lagged_sum	15	1000000	100	0.92648578	PASSED
rgb_lagged_sum	16	1000000	100	0.78312121	PASSED

Test Name	n	N	d	p-value	Result
rgb_lagged_sum	17	1000000	100	0.48839571	PASSED
rgb_lagged_sum	18	1000000	100	0.52741490	PASSED
rgb_lagged_sum	19	1000000	100	0.57382712	PASSED
rgb_lagged_sum	20	1000000	100	0.73919893	PASSED
rgb_lagged_sum	21	1000000	100	0.72509541	PASSED
rgb_lagged_sum	22	1000000	100	0.41689814	PASSED
rgb_lagged_sum	23	1000000	100	0.60260730	PASSED
rgb_lagged_sum	24	1000000	100	0.90147593	PASSED
rgb_lagged_sum	25	1000000	100	0.40242266	PASSED
rgb_lagged_sum	26	1000000	100	0.73316626	PASSED
rgb_lagged_sum	27	1000000	100	0.97377935	PASSED
rgb_lagged_sum	28	1000000	100	0.40299842	PASSED
rgb_lagged_sum	29	1000000	100	0.55415146	PASSED
rgb_lagged_sum	30	1000000	100	0.91395113	PASSED
rgb_lagged_sum	31	1000000	100	0.59378537	PASSED
rgb_lagged_sum	32	1000000	100	0.57760364	PASSED
rgb_kstest_test	0	10000	1000	0.12119363	PASSED
dab_bytedistrib	0	51200000	1	0.79112485	PASSED
dab_dct	256	50000	1	0.55110228	PASSED
dab_filltree	32	15000000	1	0.55562237	PASSED
dab_filltree	32	15000000	1	0.93018984	PASSED
dab_filltree2	0	5000000	1	0.48358687	PASSED
dab_filltree2	1	5000000	1	0.66316672	PASSED
dab_monobit2	12	65000000	1	0.85585632	PASSED

Table 5.2: Dieharder Results of Random keys.

Github link to the repository for this project: https://github.com/AprilS21/Dissertation

Bibliography

Avitabile, G., Botta, V., Iovino, V., and Visconti, I. (2023). Privacy and integrity threats in contact tracing systems and their mitigations. IEEE Internet Computing, 27(2):13–19.

Axler, S. (2015). Linear Algebra Done Right. Undergraduate Texts in Mathematics. Springer, 3rd edition.

Bassham, L. E., Rukhin, A. L., Soto, J., Nechvatal, J. R., Smid, M. E., Barker, E. B., Leigh, S. D., Levenson, M., Vangel, M., Banks, D. L., Heckert, N. A., Dray, J. F., and Vo, S. (2010). A statistical test suite for random and pseudorandom number generators for cryptographic applications:.

bitcoin.org (2013). Critical vulnerability in android wallets. Web Page. A critical weakness in Android's secure random number generator has been discovered, affecting all Android wallets. Updates and key rotation necessary.

Brosas, D. G., Sison, A. M., Hernandez, A. A., and Medina, R. P. (2020). Analysis of the randomness performance of the proposed stream cipher based cryptographic algorithm. In 2020 11th IEEE Control and System Graduate Research Colloquium (ICSGRC), pages 76–81.

Brown, R. G. (2003). Dieharder: A random number test suite. `http://webhome.phy.duke.edu/~rgb/General/dieharder.php`.

Checkoway, S., Maskiewicz, J., Garman, C., Fried, J., Cohney, S., Green, M., Heninger,

N., Weinmann, R.-P., Rescorla, E., and Shacham, H. (2016). A systematic analysis of the juniper dual ec incident. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, CCS '16, page 468–479, New York, NY, USA. Association for Computing Machinery.

Cortez, D. M. A., Sison, A. M., and Medina, R. P. (2020). Cryptographic randomness test of the modified hashing function of sha256 to address length extension attack. In Proceedings of the 2020 8th International Conference on Communications and Broadband Networking, ICCBN '20, page 24–28, New York, NY, USA. Association for Computing Machinery.

Dahiru, T. (2008). P-value, a true test of statistical significance? a cautionary note. Annals of Ibadan postgraduate medicine, 6(1):21–26.

Dang, Q. (2012). Recommendations for applications using approved hash algorithms.

Estevez-Rams, E., Perez-Davidenko, C., Aragón Fernández, B., and Lora-Serrano, R. (2015). Visualizing long vectors of measurements by use of the hilbert curve. University of Havana.

Frost, J. (2024). Statistics by jim. `https://statisticsbyjim.com/`. Accessed: 2023-04-11.

Google, A. (2020).

Grinstead, C. M. and Snell, J. L. (1997). Introduction to Probability. American Mathematical Society.

Haahr, M. (1998). Random.org - true random number service. `https://www.random.org`. Accessed: 2023-04-11.

Hurley-Smith, D. and Hernandez-Castro, J. (2020). Quantum leap and crash: Searching and finding bias in quantum random number generators. ACM Trans. Priv. Secur., 23(3).

Hłobaż, A. (2020). Statistical analysis of enhanced sdex encryption method based on sha-512 hash function. In 2020 29th International Conference on Computer Communications and Networks (ICCCN), pages 1–6.

Klyubin, A. (2013). Android security: Prng fixes. Blog post discussing the compromise of a bitcoin transaction due to improper initialization of the PRNG on Android devices and the subsequent fixes.

L'Ecuyer, P. and Simard, R. (2007). Testu01: A c library for empirical testing of random number generators. `http://simul.iro.umontreal.ca/testu01/tu01.html`.

Leith, D. and Farrell, S. (2023). Testing apps for covid-19 tracing (tact). `https://down.dsg.cs.tcd.ie/tact/`. Accessed: 2023-04-11.

Leith, D. J. and Farrell, S. (2020a). Coronavirus contact tracing: Evaluating the potential of using bluetooth received signal strength for proximity detection.

Leith, D. J. and Farrell, S. (2020b). Coronavirus contact tracing: Evaluating the potential of using bluetooth received signal strength for proximity detection.

Leith, D. J. and Farrell, S. (2021). Contact tracing app privacy: What data is shared by europe's gaen contact tracing apps. In IEEE INFOCOM 2021 - IEEE Conference on Computer Communications, pages 1–10.

Lu, Y., Yang, Y., Hu, R., Liang, H., Yi, M., Zhengfeng, H., Ma, Y., Chen, T., and Yao, L. (2023). High-efficiency trng design based on multi-bit dual-ring oscillator. ACM Trans. Reconfigurable Technol. Syst., 16(4).

Luengo, E. A. (2022). Gamma pseudo random number generators. ACM Comput. Surv., 55(4).

Luengo, E. A. and Villalba, L. J. G. (2021). Recommendations on statistical randomness test batteries for cryptographic purposes. ACM Comput. Surv., 54(4).

Mohamed, E. M., El-Etriby, S., and Abdul-kader, H. S. (2012). Randomness testing of modern encryption techniques in cloud environment. In 2012 8th International Conference on Informatics and Systems (INFOS), pages CC–1–CC–6.

Nguyen, T. D., Miettinen, M., Dmitrienko, A., Sadeghi, A.-R., and Visconti, I. (2022). Digital contact tracing solutions: Promises, pitfalls and challenges. IEEE Transactions on Emerging Topics in Computing, pages 1–12.

NIST (2012). Critical values and p values.

NIST/SEMATECH (2012). e-handbook of statistical methods.

Schneier, B. (2008). Random number bug in debian linux. Schneier on Security. Available online; accessed 14 April 2024.

Schuldt, J. C. and Shinagawa, K. (2017). On the robustness of rsa-oaep encryption and rsa-pss signatures against (malicious) randomness failures. In Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security, ASIA CCS '17, page 241–252, New York, NY, USA. Association for Computing Machinery.

Turan, M., Doğanaksoy, A., and Boztas, S. (2008). On independence and sensitivity of statistical randomness tests. volume 5203, pages 18–29.

Walker, J. (2002). Ent: A pseudorandom number sequence test program. `https://www.fourmilab.ch/random/`.

Wu, Y., Wang, T., and Li, J. (2015). Effectiveness analysis of encrypted and unencrypted bit sequence identification based on randomness test. In 2015 Fifth International Conference on Instrumentation and Measurement, Computer, Communication and Control (IMCCC), pages 1588–1591.

Zolfaghari, B., Bibak, K., and Koshiba, T. (2022). The odyssey of entropy: Cryptography. Entropy, 24(2):266.