Innovating Democracy

Scytl iVote 2.0 System

Response to “NSWEC-7 Final Report”

by David Hook and Carsten Schürmann, January 2019

Public release

June 2019

Scytl – Secure Electronic Voting

STRICTLY CONFIDENTIAL when contains "Draft" designation

Revision chart

Version	*Date*	*Primary Author(s)*	*Reviewed by*	*Description*
1.0	11/6/2019	Security and Product departments	Technical Writing team	For publication

Table of contents

1 Introduction

This document is intended to provide technical responses to points found in the review performed as part of the iVote Refresh project by the DEMTECH GROUP and released in January 2019 on the iVote System. The source code review that is responded to in this document is "NSWEC-7 Final Report" by David Hook and Carsten Schürmann, January 2019 (the "Final Report").

The responses provided in this material include a combination of development and security-related information, intended to shed some light on topics raised.

To assist readability these[1] are treated in the same order as the Final Report.

The feedback gathered is valuable and useful for the improvement of the Scytl software and can enrich areas such as the source code and technical documentation. Some of these findings have been considered in the version of the voting system used for the 2019 election and others are taken into account for future releases of the system.

Raised items will continue to be reviewed and included as part of patches produced from time to time as part of a continuous improvement program for Scytl software.

Summary of report:

It is the view of Scytl that whilst the Final Report did find areas of interest and discussion for both the reviewers and Scytl, nothing significant was found relating to the security and integrity of the system.

Scytl acknowledges that areas of the code are not easy to review and have provided challenges to reviewers and hopes that information shared in documentation has assisted reviewers to understand the design and implementation of the system.

Scytl has work to do regarding code upkeep, which we acknowledge, and continue to do as an ongoing activity. Other matters raised by the reviewers are described in the response herein.

It is important for readers to note that the Final Report was completed on an early release of the application, and not the final version used in production in the election.

[1] This document includes only responses for those points raised in the Final Report that require a clarification from a technical and/or security perspective.

2 Major Findings (Part 3 of the Final Report)

2.1 Client side

> The client side code is heavily dependent on the use of JavaScript. In some cases the files have been obfuscated, possibly in order to reduce their size. The use of obfuscation has made the review of the code harder but not impossible. There are tools for removing obfuscation which we have used and we are assuming as such tools are readily available it should clear the obfuscation would not affect the security of the system.

Scytl response:

- The client-side code is dependent on the use of JavaScript because it needs to be executed on browsers.
- The Javascript is not obfuscated, rather it is minified[2]. The aim is size reduction and does not serve a security purpose. Security is provided by means of other elements.

2.2 JavaScript, unused functionality

> The JavaScript appears to be full of unused functionality. We recommend getting some clarity on this and having any excess code removed. The issue with unused functionality is that as the JavaScript is readily downloading, an adversary is more likely to find exploits, with unused functionality as it is also often not tested or maintained sometimes providing an easier way in.

Scytl response:

- Code cleansing and/or refactoring are both activities included in the Software Development Life Cycle document which depicts all the development phases and areas of responsibility. In that document, refactoring is typified as part of the Development team regular tasks. This team uses Sonar to detect any unused code.
- Some of the libraries are used in multiple Scytl projects and so are included in the code base.
- In addition, the Security department runs its set of tests oriented to detect possible exploits and coordinates actions to solve them when required.

[2] https://en.wikipedia.org/wiki/Minification_(programming)

2.3 JavaScript, TODO markers

The JavaScript code contains 39 “TODO” markers. While this is not a sign an issue in itself, we would recommend that any existing TODO are checked to make sure they have been forgotten. Again as with unused functionality, incomplete code can also provide a lever for exploit by an adversary

Scytl response:

- The Development team has a method to track TODO’s included to the code. This process consists on adding a bullet point in a Technical Debt JIRA ticket for each of the new TODO that is added.
- At the same time, the team performs code cleansing and/or refactoring as part of software continuous improvement.

2.4 Misuse of the Voting App

An authenticated adversary can send invalid votes using the voting app by calling standard JavaScript functions with wrong keys, and then cry wolf and subsequently allege incompetence or a cyber attack. Both cases are possible, but any case will be a nightmare for NSWEC.

Scytl response:

It is potentially possible for authenticated adversaries to send invalid votes to the system, as long as the invalid vote is correctly signed, and a valid Schnorr proof is created. In other words, an adversary can encrypt a vote with invalid data, but cannot do it using invalid keys or generating incorrect proofs. Note that these invalid contents will be always detected if the vote is verified.

In this case an authenticated adversary can send one invalid vote using the voting app by calling standard JavaScript functions with wrong keys. This vote will be detected as invalid when decrypting the content of the Ballot Box, and so will not be taken into account in the result of the election. Please note that a verification of this vote through the Mobile Verification Application will result in an error, so the voter will be aware that his vote is invalid.

This is the equivalent of sending an invalid or null ballot in a traditional election. Considering the NSW election rules, this attacker will only be able to cast a single vote with a pair of credentials (iVote Number and PIN), thus it cannot be used to attack the system by adding multiple ballots to the Ballot Box.

2.5 Commenting

The Java/JavaScript code base does not exhibit a clear and consistent commenting policy. It would be so easy for the code to reference back to the design documentation and vice versa. This weakness makes manual code reviews unnecessarily cumbersome

Scytl response:

- Acknowledged. The commenting policy will be reviewed.

2.6 Entropy and randomness in the JavaScript client

It was not immediately clear what kind of entropy the JavaScript code uses to create randomness. The JavaScript code uses cryptolib-js-securerandom, which indicates that robust sources of randomness are included. We could not identify code that would conduct a entropy quality assessment. After further discussion and assistance from Scytl it appears the JavaScript RNG has been analyzed using both the Dieharder test suite and the NIST Statistical Test Suite. Both the test suites used are definitely appropriate for the task. We would strongly recommend the NSWEC acquire copies of these assessments as the quality of the entropy is a cornerstone of the security of the overall system, and it is highly likely that anyone querying the quality of the system will want such reports to be made available. In fact, it really would be a testimony to the quality system if such reports were available on short notice for later reviewers as well

Scytl response:

In order to provide some clarity about the entropy and randomness in the JavaScript client, an explanation about the used tools was provided to the reviewers about the how entropy is collected, randomness generated and how it was tested to ensure the statistical distribution of the process is random.

The JavaScript client implements a Pseudo Random Number Generator (PRNG) based on the Fortuna[3] proposal from Schenier to produce values with strong randomness. The client also implements several entropy collectors to feed the buffer of the Fortuna PRNG. The amount of entropy provided from each source and the proper implementation of the PRNG was considered in the implementation to ensure the quality of the randomness produced. For this purpose, in this phase, the PRNG output is tested against statistical analysis tools intended for such types of PRNGs.

As mentioned in the report, the tools used in this testing are the **Dieharder test suite** and the **NIST Statistical test suite**:

[3] Niels Ferguson and Bruce Schneier. Practical Cryptography. John Wiley & Sons, Inc., New York, NY, USA, 1edition, 2003

The **Dieharder test suite** (`http://www.phy.duke.edu/rgb/General/dieharder.php`) has been used to analyze the output of the PRNG. This tool is a well-known and highly reputable Random Number Generator testing suite, intended to test generators. It includes tests from the original Diehard Battery of Tests of Randomness, as well as tests from the Statistical Test Suite (STS) developed by the National Institute for Standards and Technology (NIST) and tests developed by the author of the Diehardder test suite.

- From the set of tests available in the Diehardder Suite, those which need an overwhelming quantity of data (more than 4GB, some tests even require 800GB) have been discarded, as well those that are not recommended by the author.
- Discarding those sorts of tests, does not affect the quality of the testing, as there were only a small number of tests of such kind and some of them would overlap others.
- Tests which supported different configurations have been performed using different input parameters.

In total:

- 78 Statistical tests have been performed, on the output of the PRNG.
- 400 instances of the PRNG have been created with 400 different seeds. For each one of the instances, 10.000 256-byte arrays of random values have been generated sequentially using the PRNG. Each of the 256-byte arrays of random values have been used to create 64 32-bit unsigned integers.
- Therefore, 640.000 32-bit unsigned integers have been generated for each seed, and a total of 256.000.000 random integers compose the output dataset to be tested against the Diehardder test suite.

Values generated with the PRNG initialized with different seeds have been used in order to test, not only that the values sequentially generated by a PRNG instance have the expected properties (corresponding to a sequence of random numbers), but also to test that different instances of the PRNG (initialized with different seeds) generate uncorrelated random numbers.

In order to have a reference of what should be expected from the tests on the implemented PRNG, two other PRNGs have also been evaluated in the same way[4]:

1) A flawed PRNG which generates sequences of correlated random numbers.
2) A PRNG which is considered as a “gold standard” by the author of Diehardder and that passes all statistical tests perfectly.

[4] As a proof of fact and as Annex to this material, find attached to this document a complementary list of security tests performed by Scytl: Annex1.dieharder_tests_prng_flawed, Annex2.dieharder_tests_prng_gold_standard and Annex3. dieharder_tests_prng_scytl – Refer to Section 4 of this document.

These two are tested against the PRNG that the user of the test suite wants to analyze, in this case Scytl´s. This is the practice recommended by the own author as a standard.

As expected, all the tests have been successfully passed by the implemented PRNG, as well as by the "gold standard PRNG". However, the flawed PRNG has failed most of them.

Each of the test output files has a table with multiple columns and multiple rows.

- Each row represents a different test. If two rows have the same name it means that different input parameters were used.
- The first columns correspond to different datasets and show the `p-value` of the tests on the datasets. The other columns are functions of the previous columns in order to analyze the results of the tests.

When executing a test on a file of random integers, the result is a number between 0 and 1, called the `p-value`. In order to define whether a test succeeds or fails, one has to set a significance level (usual values range from 0.1 to 0.001): if the `p-value` is smaller than the significance level consistently in many executions of the same test over different datasets, then the test fails.

This is why one of the columns of the files is the maximum `p-value`: if it is smaller than the significance level, then the test fails.

We have set a significance level of 0.005, but it is easy to see that, for both Scytl's and the "gold standard" PRNG, the `p-values` are much higher than 0.005, which means that they pass the tests with significant success.

On the other hand, as one would expect, the flawed PRNG fails many of the tests.

Another issue to be considered is that the `p-value` is a random variable and, as such, the same test over different datasets should produce noticeably different `p-values`. Therefore, in each file we add a column which lists the difference between the minimum and the maximum `p-values` for each test, and another column which lists the variance of the `p-values`. Then, we compare the difference between the maximum and minimum `p-value` and make the test fail if this difference is smaller than 0.1.

Both Scytl's and the "gold standard PRNG" succeed with this test as well, in particular such difference is much bigger than 0.1 for all the tests. On the other hand, the flawed PRNG has many similar `p-values`, and as a consequence it fails most of the tests.

Test results for the flawed, standard and Scytl's PRNG are provided as Annex of this document:

- `Annex1 dieharder_tests_prng_flawed.pdf`
- `Annex2 dieharder_tests_prng_gold_standard.pdf`
- `Annex3 dieharder_tests_prng_scytl.pdf`

Information collected by the entropy gathering system is accumulated and, when required, it is used to initialize the PRNG. The estimations about collected entropy have been done based on the following article https://crypto.stanford.edu/sjcl/acsac.pdf.

2.7 Coding style

In some parts of the code, a concept of Job is defined to exploit concurrency and to administer computations elegantly and effectively. Such programming patterns contribute to efficient execution of programming tasks, but they make it more difficult for the reviewer to interpret and evaluate the quality of the code.

Scytl response:

- Scytl acknowledges that the Job concept is used widely, and that the complexity of the task can bring some code ambiguity.
- Scytl will review the usage of the pattern by adding comments when and if code interpretation is difficult.

2.8 Potential overflow

In the code [SecondCommittmentGenerator.java, line 55-56], a product of two potentially large numbers is computed, which might lead to a potential overflow. If triggered, this problem would mostly likely crash the application during mixing.

Scytl response:

Regarding the potential overflow, we explained why this situation cannot be achieved and for a better understanding, a deeper explanation follows:

- Currently by default m = 1; n =`sizeEncryptedBallots` [BGParamsProvider.java]
- Considering that NSW population is under 8 million, which is less than 2^23, the overflow cannot be triggered unintentionally even if `sizeEncryptedBallots` is set to the number of participants.
- As for the intentional code `modi_cation`, we assume that any adversary who has the power to modify code and/or election settings would exploit more `e_cient` attach vectors.

```
// from MixingService.java
final int sizeEncryptedBallots = encryptedBallots.getBallots().size
    ();
BGParams bgParams = bGParamsProvider.getParamsForGiven(
    sizeEncryptedBallots);

----------------------
//from BGParamsProvider.java
public BGParams getParamsForGiven(final int partitionSize) {

// if partitionSize is not in the bGParams.json,
// use default BGParams params
...
}

//By default BGParams params are:

private static BGParams createDefaultBGParams(final int size) {

BGParams bgParams = new BGParams();
bgParams.setM(1);
bgParams.setN(size);
bgParams.setMu(2);
bgParams.setNumIterations(0);
return bgParams;
}
```

2.9 Secret key reuse

The code is very clear that the secret election key is reconstructed for the purpose of mixing and decrypting the ballots. What kind of mechanisms are in place to prevent secret key reuse? Once the secret key is constructed it may be stored and stolen. Also, since the secret election key is being constructed, once lost, it can be used to decrypt the original ballots breaking vote secrecy. This renders mixing a nice but rather ineffective mechanism to protect the secrecy of the vote. (see [4], Section 2.6.3.1)

Scytl response:

The main mechanism that is implemented to protect the key is the secret sharing scheme, where the key is broken into shares and then destroyed so that it does not exist in as a complete key until needed. Even though the election key is divided into shares and stored on separate smartcards, at the time of generation and reconstruction for its usage, this key is present in the server memory as a single key and only during the time that the key is used to decrypt and generate the decryption proof of the votes. After this, the key is eliminated and is to be reconstructed again in the case that it is needed. This means that, should the server where the key is used is compromised, the key could be stolen only at the decryption phase and until the decryption process finishes.

This is the reason why the steps of election key generation and ballot decryption are made in an air-gapped server avoiding the possibility of an online attack. In addition, these two steps are made during a ceremony supervised by auditors and observers to ensure the procedure is correctly performed. This is a combination of the form of the software and the process under which it must be used supporting the security.

2.10 Missing arguments from calls to hash-functions

While reviewing the generation of the parallel shuffle proof, we observed a deviation between documentation [4], page 15, and the implementation. The calls to hash functions in Prover.java, and Verifier.java appear to be be applied to fewer documents than required. (see [Verifier.java, line 141–143], [Prover.java, line 114–116], [prover.js, line 96], [verifier.js, line 84]).

Scytl response:

All hashes in proof generators and proof verifiers are computed according to the same rule (please find extracts below).

```
//Prover.java
Exponent hash =
generateHash(publicValues, preComputedValues.getPhiOutputs(),
data);
```

```
//Verifier.java
calculatedHash = calculateHash(publicValues, computedValues, data);
```

```
//prover.js
 var hash =
generateHash(group, publicValues, preComputation.phiOutputs, data);
```

```
//verifier.js
 var calculatedHash =
generateHash(group, publicValues, generatedValues, data);
```

3 Detailed Findings of Functional Matching and Verifiability Analysis (Part 6 of the Final Report)

These response to questions are generally based on the system architecture.

3.1 Election Key Generation

> The secret key is generated and then split into different shares (see also [3] Page 74). Eventually, the election private key is reconstructed. There are two problems with this approach. First, the system may be used with the same private election key for several elections. Second, the private election key is already known before the election and may be used by an adversary (who is assumed to be in possession of the key) to decrypt ballots as they arrive and it is reconstructed after the election which would an adversary allow to decrypt the unshuffled and uncleansed ballot box (assuming the adversary has access to the database).
>
> ….

Scytl response:

As explained in the section dedicated to the Secret Key reuse, all the actions dedicated to the election key generation and the ballot decryption, are performed in an air-gapped server to avoid malicious online attacks.

- Responding to comments related to the **Credential Manager code** and to clarify, Credentials are assigned at Institution level, hence the `credentialID` depends on the `institutionAlias`. With this relation it is easy to identify to which Institution the Credential belongs to. Similarly, the `credentialID` depends on the `apiKey` since the `apiKey` dictates the combination of Channel and mode.
- The construction of a **spare credential** is used in the computation of the credential ID. This is created by the Credential Generator and imported to the Credential Manager. The current data of a spare credential is transparent for the Credential Manager.
- Regarding the **hard-coded username password combination**, and to clarify, the mechanism based on Scytl JWT (JSON Web Token) is applied in order to secure endpoints in importing credentials to the Credential Manager. The `CustomUserDetailsService` is called after the JWT has been validated, at which point the user is considered authenticated. Credential Manager does not hold a list of usernames/password for its Back Office, so the login is delegated to the Voting Back Office, which is where the JWT is generated. If the Credential Manager receives a valid JWT, the user is considered authenticated.
- All these explanations will be included in detail to the relevant document to avoid further misunderstanding.

3.2 Voter Authentication

We reviewed the source code that implements voter authentication. Most notable, we could not verify the parameter to the PBKDF2 key derivation function. In order to be able to identify locations in the JavaScript code, we needed to deobfuscate and pretty print it. The line numbers refer to the line numbers in the deobfuscated file. The JavaScript code is also structured in blocks. For simplified access we refer here to to the block numbers as well.

….

Scytl response:

- The `pbkdf2` generation uses a 16 byte key, which is equivalent a 128 bit key.
- The authentication transform type used in `pnyx-govlab` component (e.g. Edmonton, MAEE) and its description, will be included in the relevant documentation to avoid further misunderstanding.
- The remaining `pnyx-govlab` modules have been decommissioned in this version and are planned to be removed on the following iteration, hence they are not included in any document.

3.3 Vote Casting Details

Section 2.4 of [4] and section 5.4 of [1] appear to describe different mechanisms for constructing a ballot. After consultation with Scytl it became clear that Section 2.4 of [4] was the mechanism being used in the iVote system. It would be helpful to later reviewers if the the description in [1] was either removed or brought into line with the one in [4].ll.

….

Scytl response:

- The pertinent documents will be updated following the suggestions provided regarding the Ballot construction, naming in files and documents, authentication methods, and so on.
- Regarding the functionality for "**castVote**", and as explained in previous sections, the voting server and the rest of the backend services are designed taking into consideration that the JavaScript voting client is not a trusted component and that an invalid vote can be received. The document reference will be reviewed.
- Regarding the **public election key encryption**, and to clarify, the crypto service depends on the certificates downloaded by the `getCertificates` endpoint. Then, the team assigns the

selected crypto module to the crypto service. This information in detail will be disclosed in the relevant document.

- As a quick explanation about the `sendVote` method, this passes the response to the Receipt Service that will validate the response. This information will be depicted in detail in the relevant documentation.
- The field "password" allows adding an extra challenge for voter authentication. This can be configured during the phase of credential generation, by adding some voter's personal information (e.g. date of birth, postal code, etc) as part of the string (user + pin + password) that will be derived in the client during voter authentication in order to get the credentialD and the PKCS12 password. When this extra configuration is not used then the password field is set to the hardcoded value of "password" for all the voters. This extra configuration is not used in NSW iVote. In this case the field is hidden from the UI.
- Regarding [ball.candidate.js, line 200], this module is planned to be reviewed in future revisions.
- As a response to the “Dead code” comments, it is relevant to remark that the `getEncryptedOption` is used by the Cipher Service through the encrypt, hence it relies on the module of the crypto used in the election.
- The initialization vector description will be updated and included in the relevant document to be aligned to implementation, also the format of the credential will be described.
- Regarding the “secret”, the `saveSecret` is set to false in order to delete it from the object where it is saved and it is not stored along the encrypted ballot. The object that has `saveSecret` set to true is used to create the secret and is not used directly to generate the encrypted ballot. The `secret`, previously generated, is stored in a constant used to create and verify the proofs. Then the gamma and phis from the first encrypted elements is taken to build another encrypted element object without any `saveSecret` set to true. From that, we start generating the encrypted ballot. We can confirm that the secret is not stored along the encrypted ballot.
- To the references made about “Imprecise Contract”, this is a Cryptolib related issue. The code implemented should validate the `preComputation` object passed. This validation would point to its interface without marking whether it is undefined or not. It could be defined although the object type is different from the standard one and could generate errors.
- In relation to the **hash computation**, it is important to explain that the hash generator computes hash over the set of public values as well as over additional data. Public values contain only the proof commitment, while additional field ‘data’ defines all other public parameters that can be used for hash computation [`hash-generator.js`]:

```
  /**
* Generates the hash.
*
* @function generate
* @memberof HashGenerator
* @param {ZpGroupElement[]}
*             publicValues The public values.
* @param {ZpGroupElement[]}
*             generatedValues The generated values.
* @param {Uint8Array|string}
*             data Auxiliary data.
* @returns {Uint8Array} The generated hash.
*/
this.generate = function(publicValues, generatedValues, data) {
digester_.update(elementsToString(publicValues));
digester_.update(elementsToString(generatedValues));
digester_.update(data);

return digester_.digest();
};
```

For example, “`data`” for the Schnorr proof is defined as follows [`schnorr-proof-handler.js`]:

```
  if (voterId && electionEventId) {
    data = VOTER_ID_PREFIX + voterId + ELECTION_EVENT_ID_PREFIX +
    electionEventId;
  }
```

Regarding the use of the method "`createEnvelope()`" with the encrypted vote, of the `SecureMessageHelper`, it is needed to clarify that this is not the one used for a Verifiable Mixing election type, the one used by iVote. Instead, the `SecureMessageHomoHelper` is loaded (see line 500 in method "`createVote()`" of file "`restCalls.js`") which has a specific "`createEnvelope()`" method.

The method observed by the reviewers is the one used for the non-verifiable mixing case, based on RSA encryption, and that is *not* used in iVote.

3.4 Cast-as- intended verification

Internet Voting Verification

Just as the voting app, the verification app consists of an embedded JavaScript. Interestingly enough, both JavaScript programs ivapi-0.8.0-min.js (voting app) are ivapi-0.7.0.js (verification app) are equal in substantial parts of the code, it almost feels like the voting app runs a new newer version of the JavaScript library than the verification app. This is very concerning as maintainers could easily get confused, and developers may already have done so. The JavaScript code is started from the Android App or the iPhone app, with “nsw-vote-verification” [MainActivity.java, line 12], which is also our entry point for the source code review.

…

Scytl response:

- Code analysis has been conducted on a beta version when the software versions were not aligned. The software versions are aligned in the final release.
- Regarding the field `isforVerification` in the login, this parameter indicates whether the login is performed to vote or to verify the vote. In the second scenario, there are certain operations that are not needed and are skipped for performance reasons. To achieve that, when the login is performed in the Voter Portal the parameter is not passed. When the login is performed in the Verification app the parameter is passed to true.
- The lines of the code that indicate that randomness is computed from the seeds that was obtained from the QR code, are in the method "`initDecrypter(randomness)`" of the "`VerifiableMixer.js`". There, a new PRNG is created with the seed passed by parameter.
- As detailed in the documentation provided, in order to decrypt the vote using the randomness derived from the seed in the QR code, it is necessary to exponentiate the public key to the negation of the randomness and then multiply it by the appropriate part of the encrypted message (the part that is the public key exponentiated to the randomness and multiplied by the message, a.k.a. `phi` value in the code).

  This code can be found in the method "`decrypt()`" of the file "`prng-decrypter.js`" of the module `ScytlElGamal`.
- The “isforVerification” parameter does not open iVote up for a timing attack. This just differentiates a call of login for voting or for validating the vote. It does not provide information to allow the login credentials to be inferred. The only information that could be inferred if the two variants of the call have a difference in time is if the call is authentication for voting or for validating a vote. It is equivalent to have two different authentication calls, one for each case.

3.5 Phone voting verification

The software provided to us did not contain the essential modules for the IVR system that would allowed us to review the phone voting verification. packages2/default/config.json

Typos Different default fields with a name ending in “length” are misspelled. This could be problematic if the names are spelled correctly when referred elsewhere.

Scytl response:

At the time of this security review, the IVR component was not complete nor operational.

3.6 Cleansing

The source code contains different modules that refer to cleansing. Some functionality can be found in the pnxy-govlab/mixing module, the other in the pnxy-govlab/cleansing. This is confusing. The code should perhaps be refactored

Scytl response:

- These modules are considered isolated services that can work independently. A study will be conducted to identify the need for refactoring in time as part of continuous improvements to the code.
- Regarding the cleansing implementation concern of not being software independent, taking as reference the definition[5] “*A voting system is software-independent if an undetected change or error in its software cannot cause an undetectable change or error in an election outcome*”, then cleansing is software independent as it is a deterministic process. We would like to suggest an ideal way to audit it, which is in line with its essence and characteristics. Because the cleansing is a deterministic process and should always output the same result given an input, the ideal way to audit the cleansing results is to implement a parallel cleansing application and confirm that the results are identical in both cases.
- To respond to the doubts raised about the specifications presented in the relevant document, when the votes are fetched, the last vote of each one of the voters is identified and is the one that will be counted during the tally process. The flag used to identify the last vote from a voter is called `lastVote`. Additionally, for voters whose votes are not to be counted can be deactivated, which will prevent their vote from being counted during the tally process. The certificate validation is using the `CAValidationService` which does check the certificate

[5] Rivest, Ronald. (2008). On the notion of 'software independence' in voting systems. Philosophical transactions. Series A, Mathematical, physical, and engineering sciences. 366. 3759-67. 10.1098/rsta.2008.0149.

chain. Along with this explanation, the team will review the relevant document to ensure the specifications provided are aligned to this description.

- In the Class conversion point, a code refactor will be considered to detect possible errors as the best solution to avoid failure in keys.
- Duplication of code and other elements such as the `sql.append` are subject to code refactor consideration. In the case of the `sql.append` it may be removed and the append itself would continue being chained.
- As a quick explanation about the Cleanser functionalities, when the votes are fetched from the database, there is a conditional statement that consider the last timestamp (which will remove any duplication). The case when `final_table.creation_date` = (select max (creation_date) from secure messages where `voter_id=final_table.voter_id` and `election_id=final_table.election_id)` then 1 else 0 end as is_`lastvote`. As the duplicated vote is not even loaded into the application (during the cleansing), there is no auditable vote for such scenario.
- To the question related to the homomorphic counting method being part of the code base, Scytl development is done towards a product that can support multiple and different security models.
- The section 2.6.1.3 of the document [4] Voting Protocol Description, underline that the validity of the certificate chain or the X509 certificate is indeed not checked here but the Channel and Mode information included within the certificate. The certificate validation uses the `CAValidationService` which does check the certificate chain.
- The signature of the vote is checked in the method "`validate()`" of the class `VoteValidation` in the package `com.scytl.mixing.validation`.

  In any case, the mentioned part of the document will be reviewed in detail to ensure there is not confusion in terms and processes.
- The Receipt verifier inherits its "`verify`" method from "`BaseVerifier`" which indeed does not add any new polymorphic behaviour. This will be planned to be refactored in the future and the "`BaseVerifier`" renamed.
- To ensure that the information related to the number of votes that were cast and the reference "`TOO_MANY_VOTES`" is included appropriately in both documentation and source code, the VoterValidation, line 53 and section 6 of the relevant document will be updated.

3.7 Mixing

> The mixing module implemented in the iVote-system appears to be new and under construction. Many error cases are not implemented completely, and judging from some of the comments in the code, it is not clear what the correct functionality is. In the form the code is now, it is possible for it to crash with an an “Unmatched exception” during mixing (as opposed to dying gracefully). The mathematical foundations of the mixer code are extremely challenging. The paper by Bayer and Groth present different versions of their mixing algorithm, and judging from our review, the one that is implemented is not the one that is described in [4].
>
> …

Scytl response:

- Given that the mixing is executed asynchronously if one error happens, the exception is caught then, the exception is logged, and the mixing status updated. Finally, the exception is thrown again so that spring gracefully kills the thread.
- Due to the complexity of the paper produced by *Bayer and Growth,* the information provided in the relevant documentation may not have been precise. However, their shuffle argument is the one followed in this system. A review of the documents has been conducted to improve the information provided in the production system.

Mixing/ mixing-core

- TODOs allow the team to identify areas of the platforms that can be improved in a future iteration and may be part of the refactoring work the team will perform, bearing in mind that the feature is already implemented and meeting the requirements.

Mixing/ commons

- Responding to the point raised regarding the mixing code lack of exception management, it is needed to explain that the *Gjosteen ElGamal* (decryption method) validates the input parameter, after that, the algorithm serializes only the data into some JAVA objects. In the *Gjosteen ElGamal* (encrypt method) an object is instantiated based on an array, so if the array length is 0, the object will reproduce the "right" answer regarding the current length 0; `GjosteenElGamalRandomness` -> as this class cannot be instantiated without having the "r" instance variable with a value, the other methods should work fine. The Exponent already validates the input parameters inside its own constructor, ensuring a right value for the "r". As we just need a correct exponent in this class, the methods will not have any further problem.

Mixing/ mixnet- shuffle

The code in `ElGamalShuffler.java` generates permutation, shuffles ballots and re-encrypts them, as presented in this snip:

```
LOG.info("Generating permutation...");
permutation = permutationGenerator.generate(numBallots);

LOG.info("Permuting ballots...");
final List<Ciphertext> permutatedListEncryptedBallots =
permutator.createPermutatedList(encryptedBallots.getBallots(),
    permutation);

shuffledEncryptedBallots = new ElGamalEncryptedBallots(
    permutatedListEncryptedBallots);
```

```
 LOG.info("Re-encrypting ballots...");
reEncryptedBallots =
reEncrypter.reEncrypt(shuffledEncryptedBallots,
    completedRandomExponents, preComputations);
```

In relation to the comment made regarding the security proof, it provides a perfect completeness of non-interactive SHVZK, which is a requirement for proving that the Mixnet is secure. For example, mix-nets based on a Randomized partial checking provide strong evidence of correctness, but not the complete correctness since the security is mainly proven heuristically.

Mixing/ proof generation

Resetting *m* is an explanation that will be included in the technical documentation.

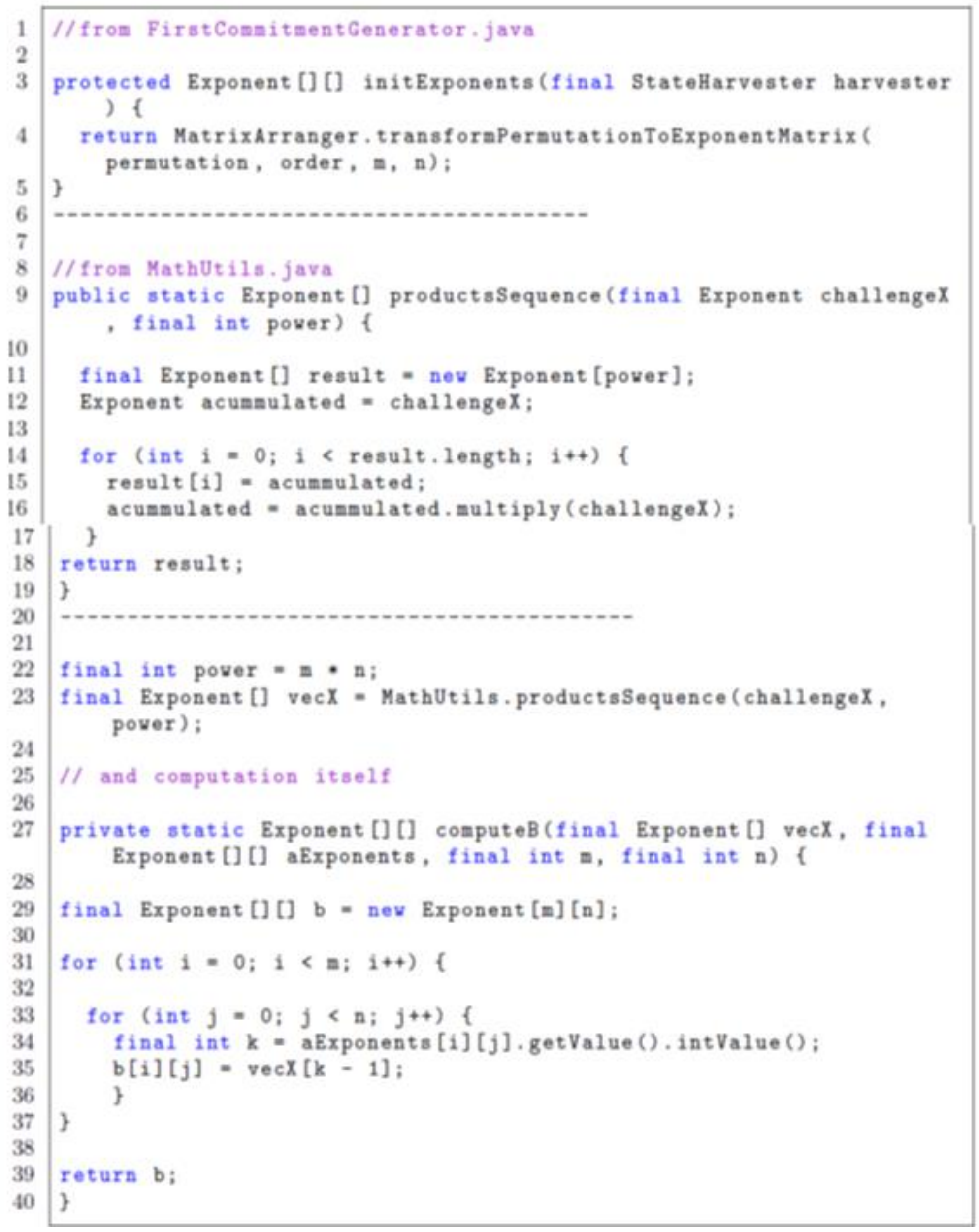
```
// Note: m must be set to 2 in this proof when the m of
// the Hadamard proof is 1, but only in this case
if (this.m == 1) {
this.m = 2;
}
```

By default, *m* = 1 and *n* = N, however Zero Argument computation requires the generation of an element *~a0* and then computing $\vec{a} = \sum_{i=0}^{m} x^i \vec{a}_i$.

To avoid problems with insides, it was adjusted from the beginning. The code follows the logic described in the Bayer and Growth paper (specifically Zero Argument 5.1).

```
final PrivateCommitment[] cZeroArgA = new PrivateCommitment[m];
System.arraycopy(ca, 1, cZeroArgA, 0, m - 1);
cZeroArgA[m - 1] = new PrivateCommitment(ExponentTools.
    getMinus1Vector(n, groupOrder),
new Exponent(groupOrder, BigInteger.ZERO), comParams,
    multiExponentiation);
```

In reference to the class “`Multiexponentiator`”, this is described in `scytl-math package`. According to the paper of reference, $\vec{b} = \{x^{\pi(i)}\}_{i=1}^{N}$, the code described below gives exactly the same output, but transformed into a matrix.



```
//from FirstCommitmentGenerator.java

protected Exponent[][] initExponents(final StateHarvester harvester
    ) {
  return MatrixArranger.transformPermutationToExponentMatrix(
    permutation, order, m, n);
}
----------------------------------------

//from MathUtils.java
public static Exponent[] productsSequence(final Exponent challengeX
    , final int power) {

  final Exponent[] result = new Exponent[power];
  Exponent acummulated = challengeX;

  for (int i = 0; i < result.length; i++) {
    result[i] = acummulated;
    acummulated = acummulated.multiply(challengeX);
  }
return result;
}
------------------------------------------

final int power = m * n;
final Exponent[] vecX = MathUtils.productsSequence(challengeX,
    power);

// and computation itself

private static Exponent[][] computeB(final Exponent[] vecX, final
    Exponent[][] aExponents, final int m, final int n) {

final Exponent[][] b = new Exponent[m][n];

for (int i = 0; i < m; i++) {

  for (int j = 0; j < n; j++) {
    final int k = aExponents[i][j].getValue().intValue();
    b[i][j] = vecX[k - 1];
    }
}

return b;
}
```

The document that describes the voting protocol (NSW Electoral Commission. iVote Voting System, Voting Protocol Description) will be updated to incorporate implementation details based on multi-dimensional matrices instead of one-dimensional vectors.

3.8 Decryption

> We started the review with AbstractBallotDecryptionService.java which contains the implementation of the decryption algorithm is implemented. The main part of "ProveDec" (see [4], page 15) is implemented in Prover.java. The implementation matches the decryption process as described in documentation.
>
> …

Scytl response:

- As a future improvement Scytl plans to work on a glossary to the align naming used in documentation and code.
- More details about validity checks will be included in the design documents. In this sense, the following explanation will be added:

The function `validateDecryptionProofGeneratorInput` verifies that all values provided as input (`publicKey, ciphertext,plaintext, privateKey`) are not null, lengths of public and private keys are equal, lengths of ciphertext and plaintext are `publicKey` + 1:

```
private void validateDecryptionProofGeneratorInput(
final ElGamalPublicKey publicKey, final Ciphertext ciphertext,
final List<ZpGroupElement> plaintext,
final ElGamalPrivateKey privateKey)
throws GeneralCryptoLibException {

Validate.notNull(publicKey, "ElGamal public key");
Validate.notNull(ciphertext, "Ciphertext");
Validate.notNullOrEmptyAndNoNulls(plaintext, "Plaintext");
Validate.notNull(privateKey, "ElGamal private key");
int publicKeyLength = publicKey.getKeys().size();
int privateKeyLength = privateKey.getKeys().size();
int ciphertextLength = ciphertext.size();
int plaintextLength = plaintext.size();
Validate.equals(publicKeyLength, privateKeyLength,
"ElGamal public key length", "ElGamal private key length");
Validate.equals(ciphertextLength, publicKeyLength + 1,
"Ciphertext length", "ElGamal public key length plus 1");
Validate.equals(plaintextLength, publicKeyLength,
"Plaintext length", "ElGamal public key length");
}
```

- The use of "c" in the code, instead of "h", is widely extended as it refers to "challenge".

3.9 Proof of Correct Decryption

> We are not totally clear on the decryption proof overall as the library code appears to be written to support several kinds, however it does appear that VerifiableMixer.js calls schnorr-proof-handler.js and does so with options.voterId and options.electionId set.
>
> …

Scytl response:

The Proof of Correct Encryption is computed during the ballot generation process and sent along with the encrypted ballot to the server. In such case, the Schnorr proof is needed to convince the verifier that the prover known encryption exponent was used for the ballot generation.

For privacy reasons, the plain text is not included, to avoid that anyone can access the content of the encrypted ballot by simply verifying the proof.

A Schnorr proof is a proof of knowledge of a discrete logarithm. The only place where it's generated in the client is in the following line (231) of the VerifiableMixer.js

```
const generatedProof = this.schnorrProofHandler.generate(secret,
      ciphertext.gamma,{ voterId: tokenUsername,
      electionId: ballot.electionId, preComputation: this.precomputation });
```

This proof of correct *encryption* ensures that an encrypted vote from another voter is not reused. In other words, it's used to make sure that the voting device knows a secret r such that the first component of ciphertext is g^r. After the proof is formed, the encrypted vote is digitally signed together with the proof and its associated metadata using the pair of Voter Signing Keys obtained during the authentication. The encrypted vote, signature, and proof are sent to the Voting Service. This signature prevents any attempts to modify the encrypted vote.

The proof of correct encryption is verified both when the vote cast arrives at the voting server and when the cleansing operation is performed during the counting. The output of the cleansing does not contain this proof, only ciphertexts (the votes without signature, proofs or any other information), thus after this step the proof is not forwarded in the next steps of the counting. In order to make sure, that this vote has not been changed during the cleansing, mixing and decryption processes, an auditor should check the correctness of the cleansing process (this process is deterministic and can be reproduced) and verify proofs of correct mixing and decryption (which are not the encryption proofs we are discussing here).

If we include selected plaintext into this proof, everyone (Voting Service, someone who intercepted the message, someone who reads the database) would be able to say which message is inside without breaking the encryption.

As for the proof of correct decryption, that is generated during the decryption phase (takes place after mixing), it is based on the Chaum-Pedersen protocol, which is schnorr-like indeed. A non-interactive challenge for that proof is computed as follows: h = H(pk|c2/m|gs|(c1)s|“DecryptionProof”). The code can be found in ProofUtil.java

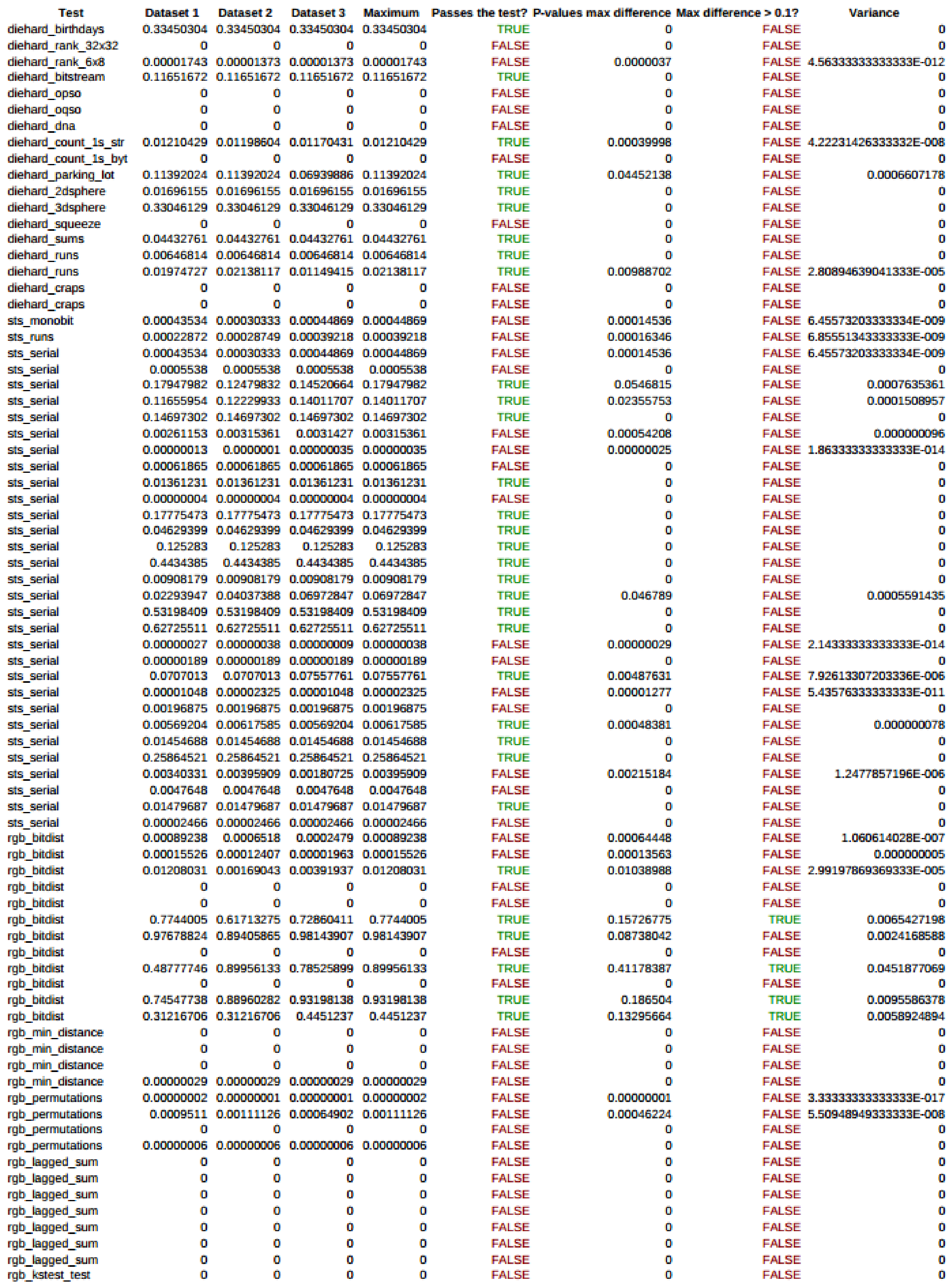
4 Attached

Annex1.dieharder_tests_prng_flawed

Annex2.dieharder_tests_prng_gold_standard

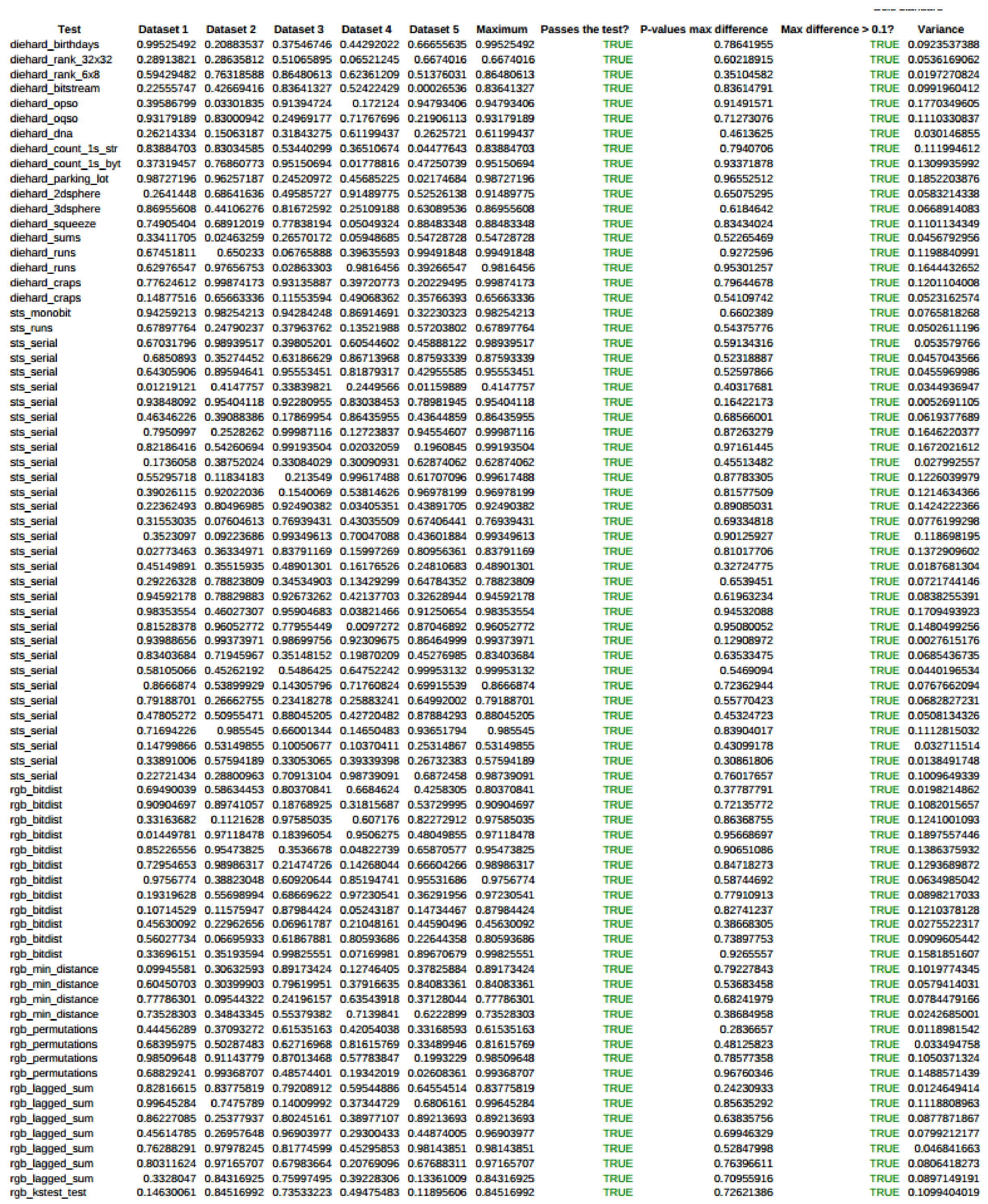
Annex3. dieharder_tests_prng_scytl

4.1 Annex 1 – PRNG flawed



Test	Dataset 1	Dataset 2	Dataset 3	Maximum	Passes the test?	P-values max difference	Max difference > 0.1?	Variance
diehard_birthdays	0.33450304	0.33450304	0.33450304	0.33450304	TRUE	0	FALSE	0
diehard_rank_32x32	0	0	0	0	FALSE	0	FALSE	0
diehard_rank_6x8	0.00001743	0.00001373	0.00001373	0.00001743	FALSE	0.0000037	FALSE	4.56333333333333E-012
diehard_bitstream	0.11651672	0.11651672	0.11651672	0.11651672	TRUE	0	FALSE	0
diehard_opso	0	0	0	0	FALSE	0	FALSE	0
diehard_oqso	0	0	0	0	FALSE	0	FALSE	0
diehard_dna	0	0	0	0	FALSE	0	FALSE	0
diehard_count_1s_str	0.01210429	0.01198604	0.01170431	0.01210429	TRUE	0.00039998	FALSE	4.22231426333332E-008
diehard_count_1s_byt	0	0	0	0	FALSE	0	FALSE	0
diehard_parking_lot	0.11392024	0.11392024	0.06939886	0.11392024	TRUE	0.04452138	FALSE	0.0006607178
diehard_2dsphere	0.01696155	0.01696155	0.01696155	0.01696155	TRUE	0	FALSE	0
diehard_3dsphere	0.33046129	0.33046129	0.33046129	0.33046129	TRUE	0	FALSE	0
diehard_squeeze	0	0	0	0	FALSE	0	FALSE	0
diehard_sums	0.04432761	0.04432761	0.04432761	0.04432761	TRUE	0	FALSE	0
diehard_runs	0.00646814	0.00646814	0.00646814	0.00646814	TRUE	0	FALSE	0
diehard_runs	0.01974727	0.02138117	0.01149415	0.02138117	TRUE	0.00988702	FALSE	2.80894639041333E-005
diehard_craps	0	0	0	0	FALSE	0	FALSE	0
diehard_craps	0	0	0	0	FALSE	0	FALSE	0
sts_monobit	0.00043534	0.00030333	0.00044869	0.00044869	FALSE	0.00014536	FALSE	6.45573203333334E-009
sts_runs	0.00022872	0.00028749	0.00039218	0.00039218	FALSE	0.00016346	FALSE	6.85551343333333E-009
sts_serial	0.00043534	0.00030333	0.00044869	0.00044869	FALSE	0.00014536	FALSE	6.45573203333334E-009
sts_serial	0.0005538	0.0005538	0.0005538	0.0005538	FALSE	0	FALSE	0
sts_serial	0.17947982	0.12479832	0.14520664	0.17947982	TRUE	0.0546815	FALSE	0.0007635361
sts_serial	0.11655954	0.12229933	0.14011707	0.14011707	TRUE	0.02355753	FALSE	0.0001508957
sts_serial	0.14697302	0.14697302	0.14697302	0.14697302	TRUE	0	FALSE	0
sts_serial	0.00261153	0.00315361	0.0031427	0.00315361	FALSE	0.00054208	FALSE	0.000000096
sts_serial	0.00000013	0.0000001	0.00000035	0.00000035	FALSE	0.00000025	FALSE	1.86333333333333E-014
sts_serial	0.00061865	0.00061865	0.00061865	0.00061865	FALSE	0	FALSE	0
sts_serial	0.01361231	0.01361231	0.01361231	0.01361231	TRUE	0	FALSE	0
sts_serial	0.00000004	0.00000004	0.00000004	0.00000004	FALSE	0	FALSE	0
sts_serial	0.17775473	0.17775473	0.17775473	0.17775473	TRUE	0	FALSE	0
sts_serial	0.04629399	0.04629399	0.04629399	0.04629399	TRUE	0	FALSE	0
sts_serial	0.125283	0.125283	0.125283	0.125283	TRUE	0	FALSE	0
sts_serial	0.4434385	0.4434385	0.4434385	0.4434385	TRUE	0	FALSE	0
sts_serial	0.00908179	0.00908179	0.00908179	0.00908179	TRUE	0	FALSE	0
sts_serial	0.02293947	0.04037388	0.06972847	0.06972847	TRUE	0.046789	FALSE	0.0005591435
sts_serial	0.53198409	0.53198409	0.53198409	0.53198409	TRUE	0	FALSE	0
sts_serial	0.62725511	0.62725511	0.62725511	0.62725511	TRUE	0	FALSE	0
sts_serial	0.00000027	0.00000038	0.00000009	0.00000038	FALSE	0.00000029	FALSE	2.14333333333333E-014
sts_serial	0.00000189	0.00000189	0.00000189	0.00000189	FALSE	0	FALSE	0
sts_serial	0.0707013	0.0707013	0.07557761	0.07557761	TRUE	0.00487631	FALSE	7.92613307203336E-006
sts_serial	0.00001048	0.00002325	0.00001048	0.00002325	FALSE	0.00001277	FALSE	5.43576333333333E-011
sts_serial	0.00196875	0.00196875	0.00196875	0.00196875	FALSE	0	FALSE	0
sts_serial	0.00569204	0.00617585	0.00569204	0.00617585	TRUE	0.00048381	FALSE	0.000000078
sts_serial	0.01454688	0.01454688	0.01454688	0.01454688	TRUE	0	FALSE	0
sts_serial	0.25864521	0.25864521	0.25864521	0.25864521	TRUE	0	FALSE	0
sts_serial	0.00340331	0.00395909	0.00180725	0.00395909	FALSE	0.00215184	FALSE	1.2477857196E-006
sts_serial	0.0047648	0.0047648	0.0047648	0.0047648	FALSE	0	FALSE	0
sts_serial	0.01479687	0.01479687	0.01479687	0.01479687	TRUE	0	FALSE	0
sts_serial	0.00002466	0.00002466	0.00002466	0.00002466	FALSE	0	FALSE	0
rgb_bitdist	0.00089238	0.0006518	0.0002479	0.00089238	FALSE	0.00064448	FALSE	1.060614028E-007
rgb_bitdist	0.00015526	0.00012407	0.00001963	0.00015526	FALSE	0.00013563	FALSE	0.000000005
rgb_bitdist	0.01208031	0.00169043	0.00391937	0.01208031	TRUE	0.01038988	FALSE	2.99197869369333E-005
rgb_bitdist	0	0	0	0	FALSE	0	FALSE	0
rgb_bitdist	0	0	0	0	FALSE	0	FALSE	0
rgb_bitdist	0.7744005	0.61713275	0.72860411	0.7744005	TRUE	0.15726775	TRUE	0.0065427198
rgb_bitdist	0.97678824	0.89405865	0.98143907	0.98143907	TRUE	0.08738042	FALSE	0.0024168588
rgb_bitdist	0	0	0	0	FALSE	0	FALSE	0
rgb_bitdist	0.48777746	0.89956133	0.78525899	0.89956133	TRUE	0.41178387	TRUE	0.0451877069
rgb_bitdist	0	0	0	0	FALSE	0	FALSE	0
rgb_bitdist	0.74547738	0.88960282	0.93198138	0.93198138	TRUE	0.186504	TRUE	0.0095586378
rgb_bitdist	0.31216706	0.31216706	0.4451237	0.4451237	TRUE	0.13295664	TRUE	0.0058924894
rgb_min_distance	0	0	0	0	FALSE	0	FALSE	0
rgb_min_distance	0	0	0	0	FALSE	0	FALSE	0
rgb_min_distance	0	0	0	0	FALSE	0	FALSE	0
rgb_min_distance	0.00000029	0.00000029	0.00000029	0.00000029	FALSE	0	FALSE	0
rgb_permutations	0.00000002	0.00000001	0.00000001	0.00000002	FALSE	0.00000001	FALSE	3.33333333333333E-017
rgb_permutations	0.0009511	0.00111126	0.00064902	0.00111126	FALSE	0.00046224	FALSE	5.50948949333333E-008
rgb_permutations	0	0	0	0	FALSE	0	FALSE	0
rgb_permutations	0.00000006	0.00000006	0.00000006	0.00000006	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_lagged_sum	0	0	0	0	FALSE	0	FALSE	0
rgb_kstest_test	0	0	0	0	FALSE	0	FALSE	0

4.2 Annex 2 – PRNG gold standard



Test	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5	Maximum	Passes the test?	P-values max difference	Max difference > 0.1?	Variance
diehard_birthdays	0.99525492	0.20883537	0.37546746	0.44292022	0.66655635	0.99525492	TRUE	0.78641955	TRUE	0.0923537388
diehard_rank_32x32	0.28913821	0.28635812	0.51065895	0.06521245	0.6674016	0.6674016	TRUE	0.60218915	TRUE	0.0536169062
diehard_rank_6x8	0.59429482	0.76318588	0.86480613	0.62361209	0.51376031	0.86480613	TRUE	0.35104582	TRUE	0.0197270824
diehard_bitstream	0.22555747	0.42669416	0.83641327	0.52422429	0.00026536	0.83641327	TRUE	0.83614791	TRUE	0.0991960412
diehard_opso	0.39586799	0.03301835	0.91394724	0.172124	0.94793406	0.94793406	TRUE	0.91491571	TRUE	0.1770349605
diehard_oqso	0.93179189	0.83000942	0.24969177	0.71767696	0.21906113	0.93179189	TRUE	0.71273076	TRUE	0.1110330837
diehard_dna	0.26214334	0.15063187	0.31843275	0.61199437	0.2625721	0.61199437	TRUE	0.4613625	TRUE	0.030146855
diehard_count_1s_str	0.83884703	0.83034585	0.53440299	0.36510674	0.04477643	0.83884703	TRUE	0.7940706	TRUE	0.111994612
diehard_count_1s_byt	0.37319457	0.76860773	0.95150694	0.01778816	0.47250739	0.95150694	TRUE	0.93371878	TRUE	0.1309935992
diehard_parking_lot	0.98727196	0.96257187	0.24520972	0.45685225	0.02174684	0.98727196	TRUE	0.96552512	TRUE	0.1852203876
diehard_2dsphere	0.2641448	0.68641636	0.49585727	0.91489775	0.52526138	0.91489775	TRUE	0.65075295	TRUE	0.0583214338
diehard_3dsphere	0.86955608	0.44106276	0.81672592	0.25109188	0.63089536	0.86955608	TRUE	0.6184642	TRUE	0.0668914083
diehard_squeeze	0.74905404	0.68912019	0.77838194	0.05049324	0.88483348	0.88483348	TRUE	0.83434024	TRUE	0.1101134349
diehard_sums	0.33411705	0.02463259	0.26570172	0.05948685	0.54728728	0.54728728	TRUE	0.52265469	TRUE	0.0456792956
diehard_runs	0.67451811	0.650233	0.06765888	0.39635593	0.99491848	0.99491848	TRUE	0.9272596	TRUE	0.1198840991
diehard_runs	0.62976547	0.97656753	0.02863303	0.9816456	0.39266547	0.9816456	TRUE	0.95301257	TRUE	0.1644432652
diehard_craps	0.77624612	0.99874173	0.93135887	0.39720773	0.20229495	0.99874173	TRUE	0.79644678	TRUE	0.1201104008
diehard_craps	0.14877516	0.65663336	0.11553594	0.49068362	0.35766393	0.65663336	TRUE	0.54109742	TRUE	0.0523162574
sts_monobit	0.94259213	0.98254213	0.94284248	0.86914691	0.32230323	0.98254213	TRUE	0.6602389	TRUE	0.0765818268
sts_runs	0.67897764	0.24790237	0.37963762	0.13521988	0.57203802	0.67897764	TRUE	0.54375776	TRUE	0.0502611196
sts_serial	0.67031796	0.98939517	0.39805201	0.60544602	0.45888122	0.98939517	TRUE	0.59134316	TRUE	0.053579766
sts_serial	0.6850893	0.35274452	0.63186629	0.86713968	0.87593339	0.87593339	TRUE	0.52318887	TRUE	0.0457043566
sts_serial	0.64305906	0.89594641	0.95553451	0.81879317	0.42955585	0.95553451	TRUE	0.52597866	TRUE	0.0455969986
sts_serial	0.01219121	0.4147757	0.33839821	0.2449566	0.01159889	0.4147757	TRUE	0.40317681	TRUE	0.0344936947
sts_serial	0.93848092	0.95404118	0.92280955	0.83038453	0.78981945	0.95404118	TRUE	0.16422173	TRUE	0.0052691105
sts_serial	0.46346226	0.39088386	0.17869954	0.86435955	0.43644859	0.86435955	TRUE	0.68566001	TRUE	0.0619377689
sts_serial	0.7950997	0.2528262	0.99987116	0.12723837	0.94554607	0.99987116	TRUE	0.87263279	TRUE	0.1646220377
sts_serial	0.82186416	0.54260694	0.99193504	0.02032059	0.1960845	0.99193504	TRUE	0.97161445	TRUE	0.1672021612
sts_serial	0.1736058	0.38752024	0.33084029	0.30090931	0.62874062	0.62874062	TRUE	0.45513482	TRUE	0.027992557
sts_serial	0.55295718	0.11834183	0.213549	0.99617488	0.61707096	0.99617488	TRUE	0.87783305	TRUE	0.1226039979
sts_serial	0.39026115	0.92022036	0.1540069	0.53814626	0.96978199	0.96978199	TRUE	0.81577509	TRUE	0.1214634366
sts_serial	0.22362493	0.80496985	0.92490382	0.03405351	0.43891705	0.92490382	TRUE	0.89085031	TRUE	0.1424222366
sts_serial	0.31553035	0.07604613	0.76939431	0.43035509	0.67406441	0.76939431	TRUE	0.69334818	TRUE	0.0776199298
sts_serial	0.3523097	0.09223686	0.99349613	0.70047088	0.43601884	0.99349613	TRUE	0.90125927	TRUE	0.118698195
sts_serial	0.02773463	0.36334971	0.83791169	0.15997269	0.80956361	0.83791169	TRUE	0.81017706	TRUE	0.1372909602
sts_serial	0.45149891	0.35515935	0.48901301	0.16176526	0.24810683	0.48901301	TRUE	0.32724775	TRUE	0.0187681304
sts_serial	0.29226328	0.78823809	0.34534903	0.13429299	0.64784352	0.78823809	TRUE	0.6539451	TRUE	0.0721744146
sts_serial	0.94592178	0.78829883	0.92673262	0.42137703	0.32628944	0.94592178	TRUE	0.61963234	TRUE	0.0838255391
sts_serial	0.98353554	0.46027307	0.95904683	0.03821466	0.91250654	0.98353554	TRUE	0.94532088	TRUE	0.1709493923
sts_serial	0.81528378	0.96052772	0.77955449	0.0097272	0.87046892	0.96052772	TRUE	0.95080052	TRUE	0.1480499256
sts_serial	0.93988656	0.99373971	0.98699756	0.92309675	0.86464999	0.99373971	TRUE	0.12908972	TRUE	0.0027615176
sts_serial	0.83403684	0.71945967	0.35148152	0.19870209	0.45276985	0.83403684	TRUE	0.63533475	TRUE	0.0685436735
sts_serial	0.58105066	0.45262192	0.5486425	0.64752242	0.99953132	0.99953132	TRUE	0.5469094	TRUE	0.0440196534
sts_serial	0.8666874	0.53899929	0.14305796	0.71760824	0.69915539	0.8666874	TRUE	0.72362944	TRUE	0.0767662094
sts_serial	0.79188701	0.26662755	0.23418278	0.25883241	0.64992002	0.79188701	TRUE	0.55770423	TRUE	0.0682827231
sts_serial	0.47805272	0.50955471	0.88045205	0.42720482	0.87884293	0.88045205	TRUE	0.45324723	TRUE	0.0508134326
sts_serial	0.71694226	0.985545	0.66001344	0.14650483	0.93651794	0.985545	TRUE	0.83904017	TRUE	0.1112815032
sts_serial	0.14799866	0.53149855	0.10050677	0.10370411	0.25314867	0.53149855	TRUE	0.43099178	TRUE	0.032711514
sts_serial	0.33891006	0.57594189	0.33053065	0.39339398	0.26732383	0.57594189	TRUE	0.30861806	TRUE	0.0138491748
sts_serial	0.22721434	0.28800963	0.70913104	0.98739091	0.6872458	0.98739091	TRUE	0.76017657	TRUE	0.1009649339
rgb_bitdist	0.69490039	0.58634453	0.80370841	0.6684624	0.4258305	0.80370841	TRUE	0.37787791	TRUE	0.0198214862
rgb_bitdist	0.90904697	0.89741057	0.18768925	0.31815687	0.53729995	0.90904697	TRUE	0.72135772	TRUE	0.1082015657
rgb_bitdist	0.33163682	0.1121628	0.97585035	0.607176	0.82272912	0.97585035	TRUE	0.86368755	TRUE	0.1241001093
rgb_bitdist	0.01449781	0.97118478	0.18396054	0.9506275	0.48049855	0.97118478	TRUE	0.95668697	TRUE	0.1897557446
rgb_bitdist	0.85226556	0.95473825	0.3536678	0.04822739	0.65870577	0.95473825	TRUE	0.90651086	TRUE	0.1386375932
rgb_bitdist	0.72954653	0.98986317	0.21474726	0.14268044	0.66604266	0.98986317	TRUE	0.84718273	TRUE	0.1293689872
rgb_bitdist	0.9756774	0.38823048	0.60920644	0.85194741	0.95531686	0.9756774	TRUE	0.58744692	TRUE	0.0634985042
rgb_bitdist	0.19319628	0.55698994	0.68669622	0.97230541	0.36291956	0.97230541	TRUE	0.77910913	TRUE	0.0898217033
rgb_bitdist	0.10714529	0.11575947	0.87984424	0.05243187	0.14734467	0.87984424	TRUE	0.82741237	TRUE	0.1210378128
rgb_bitdist	0.45630092	0.22962656	0.06961787	0.21048161	0.44590496	0.45630092	TRUE	0.38668305	TRUE	0.0275522317
rgb_bitdist	0.56027734	0.06695933	0.61867881	0.80593686	0.22644358	0.80593686	TRUE	0.73897753	TRUE	0.0909605442
rgb_bitdist	0.33696151	0.35193594	0.99825551	0.07169981	0.89670679	0.99825551	TRUE	0.9265557	TRUE	0.1581851607
rgb_min_distance	0.09945581	0.30632593	0.89173424	0.12746405	0.37825884	0.89173424	TRUE	0.79227843	TRUE	0.1019774345
rgb_min_distance	0.60450703	0.30399903	0.79619951	0.37916635	0.84083361	0.84083361	TRUE	0.53683458	TRUE	0.0579414031
rgb_min_distance	0.77786301	0.09544322	0.24196157	0.63543918	0.37128044	0.77786301	TRUE	0.68241979	TRUE	0.0784479166
rgb_min_distance	0.73528303	0.34843345	0.55379382	0.7139841	0.6222899	0.73528303	TRUE	0.38684958	TRUE	0.0242685001
rgb_permutations	0.44456289	0.37093272	0.61535163	0.42054038	0.33168593	0.61535163	TRUE	0.2836657	TRUE	0.0118981542
rgb_permutations	0.68395975	0.50287483	0.62716968	0.81615769	0.33489946	0.81615769	TRUE	0.48125823	TRUE	0.033494758
rgb_permutations	0.98509648	0.91143779	0.87013468	0.57783847	0.1993229	0.98509648	TRUE	0.78577358	TRUE	0.1050371324
rgb_permutations	0.68829241	0.99368707	0.48574401	0.19342019	0.02608361	0.99368707	TRUE	0.96760346	TRUE	0.1488571439
rgb_lagged_sum	0.82816615	0.83775819	0.79208912	0.59544886	0.64554514	0.83775819	TRUE	0.24230933	TRUE	0.0124649414
rgb_lagged_sum	0.99645284	0.7475789	0.14009992	0.37344729	0.6806161	0.99645284	TRUE	0.85635292	TRUE	0.1118808963
rgb_lagged_sum	0.86227085	0.25377937	0.80245161	0.38977107	0.89213693	0.89213693	TRUE	0.63835756	TRUE	0.0877871867
rgb_lagged_sum	0.45614785	0.26957648	0.96903977	0.29300433	0.44874005	0.96903977	TRUE	0.69946329	TRUE	0.0799212177
rgb_lagged_sum	0.76288291	0.97978245	0.81774599	0.45295853	0.98143851	0.98143851	TRUE	0.52847998	TRUE	0.046841663
rgb_lagged_sum	0.80311624	0.97165707	0.67983664	0.20769096	0.67688311	0.97165707	TRUE	0.76396611	TRUE	0.0806418273
rgb_lagged_sum	0.3328047	0.84316925	0.75997495	0.39228306	0.13361009	0.84316925	TRUE	0.70955916	TRUE	0.0897149191
rgb_kstest_test	0.14630061	0.84516992	0.73533223	0.49475483	0.11895606	0.84516992	TRUE	0.72621386	TRUE	0.1099404019

4.3 Annex 3 – PRNG Scytl

Test	Dataset 1	Dataset 2	Dataset 3	Dataset 4	Dataset 5	Dataset 6	Dataset 7	Dataset 8	Dataset 9	Dataset 10	Dataset 11	Maximum	Passes the test?	P-values max difference	Max difference > 0.1?	Variance
diehard_birthdays	0.65742816	0.99346854	0.87433827	0.79695184	0.49734932	0.51639456	0.91433851	0.72851752	0.61146317	0.85345915	0.39852511	0.99346854	TRUE	0.59494343	TRUE	0.0369698588
diehard_rank_32x32	0.67666646	0.0824705	0.91009904	0.86411819	0.90775623	0.7089944	0.51690781	0.7053571	0.97735722	0.2304836	0.47054183	0.97735722	TRUE	0.89488672	TRUE	0.0841462028
diehard_rank_6x8	0.01241533	0.56035068	0.56510347	0.66480295	0.67045299	0.01783682	0.3526054	0.058164	0.84400059	0.92551272	0.90987029	0.92551272	TRUE	0.91309739	TRUE	0.1218052245
diehard_bitstream	0.20914819	0.94910568	0.99937068	0.01987356	0.83772849	0.98521555	0.42566885	0.93744997	0.27066239	0.45293843	0.16007045	0.99937068	TRUE	0.97949712	TRUE	0.1432168412
diehard_opso	0.40504667	0.30462652	0.76978152	0.79953043	0.74434375	0.854469	0.90149754	0.14652836	0.87062052	0.60940585	0.6977432	0.90149754	TRUE	0.75496918	TRUE	0.0635839889
diehard_oqso	0.82359171	0.51692541	0.54260133	0.98845119	0.93429947	0.75718183	0.96684454	0.72345039	0.4710806	0.0389227	0.55434426	0.98845119	TRUE	0.94952849	TRUE	0.0784047343
diehard_dna	0.17040425	0.97579964	0.68300486	0.72681892	0.63381468	0.38033601	0.35875465	0.79301844	0.29529544	0.61543056	0.74063209	0.97579964	TRUE	0.80539539	TRUE	0.0601790529
diehard_count_1s_str	0.97024182	0.1586434	0.60259836	0.66126637	0.02006871	0.40655879	0.57615593	0.38382387	0.62197755	0.50850163	0.41998422	0.97024182	TRUE	0.95017311	TRUE	0.0651267391
diehard_count_1s_byt	0.85267254	0.22048686	0.78850822	0.99456558	0.94213445	0.77560755	0.87206614	0.34334985	0.56321448	0.87049109	0.74785756	0.99456558	TRUE	0.77407872	TRUE	0.0612389654
diehard_parking_lot	0.94334354	0.15298494	0.10511838	0.97166083	0.2254751	0.01170409	0.92282635	0.5022859	0.7733542	0.71266869	0.60921267	0.97166083	TRUE	0.95995674	TRUE	0.1302971692
diehard_2dsphere	0.99116939	0.94396292	0.96064534	0.99546742	0.77506181	0.44549237	0.90443146	0.98840772	0.99063525	0.31414039	0.87279309	0.99546742	TRUE	0.68132703	TRUE	0.055840645
diehard_3dsphere	0.62076674	0.84061677	0.07533222	0.99912885	0.99418613	0.22105541	0.98473216	0.70888442	0.85477535	0.94895365	0.02411245	0.99912885	TRUE	0.9750164	TRUE	0.1428970161
diehard_squeeze	0.15963934	0.98472051	0.45726064	0.75172127	0.83163491	0.4054089	0.0366757	0.92649195	0.90445387	0.12727395	0.49113692	0.98472051	TRUE	0.94804481	TRUE	0.1203864536
diehard_sums	0.06984881	0.11438474	0.06860969	0.03469112	0.82301819	0.01915397	0.00960945	0.1671129	0.00777327	0.13667209	0.25921118	0.82301819	TRUE	0.81524492	TRUE	0.0549694658
diehard_runs	0.96316475	0.4043532	0.69886605	0.19144578	0.92068581	0.41497184	0.49988452	0.92261006	0.63293509	0.64586021	0.45428441	0.96316475	TRUE	0.77171897	TRUE	0.0620433488
diehard_runs	0.14053913	0.90083415	0.09111485	0.16907519	0.11597269	0.91145507	0.04035857	0.59860994	0.35071318	0.71576689	0.53567732	0.91145507	TRUE	0.8710965	TRUE	0.1095852125
diehard_craps	0.33067725	0.40794686	0.99182421	0.57271633	0.62578585	0.8970615	0.00777512	0.50388833	0.18186607	0.94984042	0.86476658	0.99182421	TRUE	0.98404909	TRUE	0.1073468431
diehard_craps	0.70891788	0.23139349	0.28249486	0.27567839	0.42077959	0.27335414	0.81920116	0.38206434	0.10555725	0.85661686	0.49161099	0.85661686	TRUE	0.75105961	TRUE	0.0631013199
sts_monobit	0.78858822	0.67482562	0.30870564	0.35433845	0.91957836	0.241589	0.99284276	0.23046353	0.89935066	0.14539761	0.11389389	0.99284276	TRUE	0.87894887	TRUE	0.1161711736
sts_runs	0.42510829	0.38327968	0.161461	0.82698927	0.59959534	0.22930088	0.87511678	0.57813149	0.158769	0.05922672	0.3919297	0.87511678	TRUE	0.81589006	TRUE	0.0733369919
sts_serial	0.78858822	0.67482562	0.30870564	0.35433845	0.91957836	0.241589	0.99284276	0.23046353	0.89935066	0.14539761	0.11389389	0.99284276	TRUE	0.87894887	TRUE	0.1161711736
sts_serial	0.0830099	0.70173049	0.72765881	0.58494982	0.94382589	0.91894992	0.91280471	0.32860719	0.82553767	0.99158402	0.95204215	0.99158402	TRUE	0.90857412	TRUE	0.0842934412
sts_serial	0.50732002	0.88162495	0.81866227	0.44064081	0.53096086	0.25958215	0.52518595	0.19715497	0.59668061	0.21841563	0.86586755	0.88162495	TRUE	0.68446998	TRUE	0.0613726018
sts_serial	0.45191547	0.47968157	0.80340807	0.68301318	0.10311147	0.68552643	0.09496261	0.74911954	0.38576857	0.27257352	0.98869363	0.98869363	TRUE	0.89373102	TRUE	0.0846368852
sts_serial	0.97195697	0.31948976	0.73503586	0.98846578	0.85942315	0.99016704	0.43399148	0.6736627	0.84382692	0.16138486	0.75821528	0.99016704	TRUE	0.82878218	TRUE	0.079832162
sts_serial	0.11200806	0.46260563	0.80694501	0.26367956	0.18679672	0.97765358	0.44684953	0.24269226	0.23011128	0.95115411	0.20007842	0.97765358	TRUE	0.86564552	TRUE	0.1028732302
sts_serial	0.70109632	0.56480609	0.90248127	0.57274157	0.90430962	0.60245761	0.88163526	0.89643633	0.4431076	0.79640676	0.62736044	0.90430962	TRUE	0.46120202	TRUE	0.0275491359
sts_serial	0.35572906	0.60148198	0.88666075	0.11195108	0.97820643	0.79059305	0.94399728	0.98440593	0.50129291	0.91671726	0.23816859	0.98440593	TRUE	0.87245485	TRUE	0.1020754751
sts_serial	0.52669466	0.99860227	0.23073288	0.61183429	0.65171041	0.49112162	0.40746734	0.94109149	0.50918868	0.12230573	0.86361262	0.99860227	TRUE	0.87629654	TRUE	0.0769141246
sts_serial	0.19525056	0.43472106	0.2876228	0.72927355	0.92973994	0.47264276	0.8678904	0.99092674	0.50727596	0.14341535	0.92587058	0.99092674	TRUE	0.84751139	TRUE	0.0977016577
sts_serial	0.99332849	0.40496626	0.44602301	0.27285109	0.79797264	0.47538051	0.82125502	0.76069857	0.31887267	0.99357352	0.34100736	0.99357352	TRUE	0.72072243	TRUE	0.075393054
sts_serial	0.9927894	0.79501412	0.80024859	0.37782597	0.60309875	0.21863594	0.91763066	0.06596377	0.21600981	0.8269299	0.34078306	0.9927894	TRUE	0.92682563	TRUE	0.1061438025
sts_serial	0.98838609	0.7886213	0.72552142	0.98538439	0.99100883	0.19455227	0.41037547	0.38890786	0.64810434	0.77462244	0.71373235	0.99100883	TRUE	0.79645656	TRUE	0.0701641109
sts_serial	0.96519505	0.91552635	0.5789413	0.29354721	0.86056491	0.58456072	0.15866462	0.87290641	0.24248926	0.40397098	0.68980765	0.96519505	TRUE	0.80653043	TRUE	0.0838871058
sts_serial	0.43024379	0.23998624	0.86975624	0.91241051	0.97833927	0.1878529	0.78751849	0.83237636	0.39763886	0.88264019	0.02975923	0.97833927	TRUE	0.94858004	TRUE	0.1176732763
sts_serial	0.99977775	0.28234913	0.88090315	0.6414837	0.59266582	0.06891716	0.09914464	0.90603503	0.76767706	0.77121568	0.31289118	0.99977775	TRUE	0.93086059	TRUE	0.1100857659
sts_serial	0.39301829	0.14305599	0.7488626	0.24103858	0.8287241	0.35608619	0.86281261	0.52095504	0.24292291	0.87731839	0.55209072	0.87731839	TRUE	0.7342624	TRUE	0.0733386529
sts_serial	0.95167195	0.26398881	0.61448642	0.49568879	0.44432134	0.40097781	0.48899495	0.82178057	0.29155285	0.29886495	0.62284219	0.95167195	TRUE	0.68768314	TRUE	0.0484936769
sts_serial	0.71130348	0.83305378	0.82156774	0.68958195	0.4249439	0.40826761	0.99840042	0.22166885	0.37983216	0.66088154	0.80154062	0.99840042	TRUE	0.77673157	TRUE	0.0574679306
sts_serial	0.54396465	0.21783836	0.36883662	0.91084135	0.40653965	0.89970584	0.68042539	0.91629935	0.20256688	0.09597231	0.79823828	0.91629935	TRUE	0.82032704	TRUE	0.0955921639
sts_serial	0.11189163	0.79961048	0.82876527	0.96206833	0.47661674	0.26171073	0.26199155	0.39998771	0.34673778	0.99684463	0.5413312	0.99684463	TRUE	0.884953	TRUE	0.0936409405
sts_serial	0.58893849	0.58575984	0.42479004	0.9887192	0.0424656	0.0931568	0.08736622	0.90987117	0.63436083	0.86361874	0.07731653	0.9887192	TRUE	0.9462536	TRUE	0.1298050862
sts_serial	0.20792282	0.97485497	0.80214072	0.54750539	0.33562096	0.973841	0.10128429	0.61404856	0.5203258	0.79910845	0.31227086	0.97485497	TRUE	0.87357068	TRUE	0.0909843473
sts_serial	0.76992953	0.83984383	0.9734374	0.77508586	0.64784062	0.70892504	0.72036946	0.02818518	0.96748065	0.21152993	0.99416077	0.99416077	TRUE	0.96597559	TRUE	0.0955718877
sts_serial	0.65539709	0.52887604	0.29653277	0.85156169	0.8355101	0.76711783	0.52116783	0.30934907	0.28999255	0.77499784	0.19586022	0.85156169	TRUE	0.65570147	TRUE	0.0597732286
sts_serial	0.84948425	0.29556024	0.99225811	0.97759748	0.80949098	0.56870508	0.63175779	0.45885501	0.55500467	0.9429079	0.97172036	0.99225811	TRUE	0.69669787	TRUE	0.0582936565
sts_serial	0.01955664	0.994941	0.6926179	0.09615156	0.13836832	0.36600111	0.27821284	0.74746636	0.26749339	0.51944956	0.65948005	0.994941	TRUE	0.97538436	TRUE	0.0967859966
sts_serial	0.11229906	0.98610912	0.94392242	0.05142067	0.42854313	0.74887767	0.28433425	0.54597675	0.75285964	0.77300987	0.34195942	0.98610912	TRUE	0.93468845	TRUE	0.1044002525
sts_serial	0.12558624	0.43131965	0.67052706	0.25236754	0.93091118	0.20797148	0.48494385	0.89777168	0.2403196	0.80571792	0.65309804	0.93091118	TRUE	0.80532494	TRUE	0.0845398616
sts_serial	0.97932523	0.2448193	0.16113796	0.41150594	0.85519336	0.28720408	0.93236035	0.87485765	0.9654656	0.89576609	0.76254929	0.97932523	TRUE	0.81818727	TRUE	0.1040685003
rgb_bitdist	0.91447676	0.73780629	0.96949434	0.64939551	0.77563519	0.44251849	0.16792829	0.27710009	0.54335745	0.70110281	0.25555974	0.96949434	TRUE	0.80156605	TRUE	0.0733026113
rgb_bitdist	0.23414884	0.98388753	0.34700663	0.29317736	0.89355748	0.42789544	0.71412075	0.29833951	0.99607207	0.64853379	0.76319001	0.99607207	TRUE	0.76192323	TRUE	0.0845040647
rgb_bitdist	0.50139091	0.11573329	0.13264761	0.047116	0.64974195	0.92758738	0.98292064	0.41447615	0.83465882	0.88569538	0.88092261	0.98292064	TRUE	0.93580464	TRUE	0.1270087201
rgb_bitdist	0.01753168	0.60051871	0.09725073	0.90091921	0.82905085	0.9560124	0.29684728	0.01804382	0.43746056	0.55069386	0.64292838	0.9560124	TRUE	0.93848072	TRUE	0.1180386073
rgb_bitdist	0.46891858	0.00294661	0.62920392	0.41779648	0.37607632	0.84616441	0.18483422	0.35129331	0.40685958	0.04427679	0.95107817	0.95107817	TRUE	0.94813156	TRUE	0.0886729982
rgb_bitdist	0.40058425	0.95274092	0.99283992	0.68790102	0.12648577	0.43337725	0.7344036	0.52094913	0.67721001	0.99334106	0.9906705	0.99334106	TRUE	0.86685529	TRUE	0.0840471703
rgb_bitdist	0.69230037	0.17646462	0.71151915	0.69149841	0.67447198	0.9452608	0.38414052	0.26440341	0.4935335	0.67849642	0.57061639	0.9452608	TRUE	0.76879618	TRUE	0.0501889397
rgb_bitdist	0.55229019	0.93925114	0.48039872	0.21661858	0.06152319	0.85987263	0.40735509	0.26801918	0.4991796	0.59124919	0.98887501	0.98887501	TRUE	0.92735182	TRUE	0.0895772345
rgb_bitdist	0.47268902	0.26248526	0.82018865	0.76510376	0.94570101	0.26011189	0.5511425	0.47125722	0.78254664	0.4804671	0.06568014	0.94570101	TRUE	0.88002087	TRUE	0.0745498782
rgb_bitdist	0.72420607	0.37724364	0.09952428	0.06168118	0.16093175	0.69744856	0.91683936	0.87351877	0.9686358	0.51403019	0.91779991	0.9686358	TRUE	0.90695462	TRUE	0.1219519712
rgb_bitdist	0.00679692	0.06001252	0.04721747	0.8076999	0.95506362	0.88492484	0.25134305	0.47398237	0.84829131	0.39710922	0.60204912	0.95506362	TRUE	0.9482667	TRUE	0.1294103794
rgb_bitdist	0.22410256	0.37759777	0.57753008	0.99107699	0.74667269	0.35855332	0.21491817	0.08667209	0.6840353	0.9103435	0.82950578	0.99107699	TRUE	0.9044049	TRUE	0.0959241231
rgb_min_distance	0.96070529	0.74347069	0.55868076	0.2526295	0.21557736	0.33001895	0.37540118	0.17994061	0.36834934	0.52885292	0.04486943	0.96070529	TRUE	0.91583586	TRUE	0.0708639314
rgb_min_distance	0.26159695	0.99896119	0.05187394	0.80150228	0.90650643	0.12465641	0.83240768	0.79936596	0.72112025	0.22924781	0.25831464	0.99896119	TRUE	0.94708725	TRUE	0.1262943801
rgb_min_distance	0.54435195	0.31302252	0.91628771	0.72358192	0.17325494	0.60362805	0.21873724	0.73460591	0.03362064	0.6196529	0.57226635	0.91628771	TRUE	0.88266707	TRUE	0.0748710004
rgb_min_distance	0.8045697	0.68382986	0.41819832	0.41598244	0.21076608	0.1844985	0.6445686	0.15808047	0.13309981	0.58488052	0.23407459	0.8045697	TRUE	0.67146989	TRUE	0.0577881155
rgb_permutations	0.94089981	0.8899092	0.59213936	0.72800841	0.72584563	0.99745969	0.57354691	0.99802597	0.21898084	0.52083033	0.43042765	0.99802597	TRUE	0.77904513	TRUE	0.0635986865
rgb_permutations	0.85634915	0.81668631	0.31744725	0.7986548	0.19088969	0.33400536	0.65015506	0.90899888	0.98012144	0.66089057	0.67838456	0.98012144	TRUE	0.78923175	TRUE	0.0690508818
rgb_permutations	0.75631496	0.98844181	0.57649976	0.84892352	0.10902231	0.36927514	0.37427656	0.67914825	0.73174153	0.91235691	0.32289912	0.98844181	TRUE	0.8794195	TRUE	0.0780621161
rgb_permutations	0.47157972	0.06765117	0.79588525	0.60889001	0.93976964	0.68390964	0.8819782	0.12874305	0.29869333	0.99255832	0.80677777	0.99255832	TRUE	0.92490715	TRUE	0.1049529819
rgb_lagged_sum	0.6837465	0.69280042	0.86551006	0.75188849	0.99255977	0.55957449	0.3496064	0.66255888	0.86448242	0.51393847	0.94093685	0.99255977	TRUE	0.64295337	TRUE	0.0376746464
rgb_lagged_sum	0.12617015	0.98528812	0.05273035	0.79988408	0.20049322	0.94336503	0.78097923	0.35607959	0.75495295	0.74221365	0.67076837	0.98528812	TRUE	0.93255777	TRUE	0.1128236978
rgb_lagged_sum	0.44430036	0.65225953	0.21120255	0.27803484	0.68583341	0.35392264	0.54802006	0.2185125	0.42790975	0.31235462	0.53634867	0.68583341	TRUE	0.47463086	TRUE	0.0275276953
rgb_lagged_sum	0.54115862	0.22859374	0.68945852	0.02550344	0.00392844	0.15388887	0.20278173	0.03078007	0.52252542	0.13283542	0.8002287	0.8002287	TRUE	0.79630026	TRUE	0.0808819284
rgb_lagged_sum	0.06075826	0.54505197	0.79714769	0.19784399	0.00219311	0.38590676	0.89229895	0.92323037	0.95602811	0.29900312	0.37834266	0.95602811	TRUE	0.953835	TRUE	0.1231219605
rgb_lagged_sum	0.54765758	0.52726689	0.79724949	0.22285196	0.19158426	0.98090164	0.70726167	0.47244809	0.62268703	0.95403631	0.60709949	0.98090164	TRUE	0.78931738	TRUE	0.065466107
rgb_lagged_sum	0.2732678	0.92210268	0.90191758	0.70547632	0.49606033	0.57567811	0.57414633	0.87287462	0.61552621	0.7718328	0.52310069	0.92210268	TRUE	0.64883488	TRUE	0.0398679231
rgb_kstest_test	0.26895749	0.55414375	0.72410759	0.06959659	0.00259409	0.93125838	0.02944141	0.19194577	0.27717129	0.56936124	0.64076936	0.93125838	TRUE	0.92866429	TRUE	0.097585326