Victorian Electoral Commission comment on DemTech Report

Updated 5/3/15

This document accompanies the publication of a report written for the Victorian Electoral Commission (VEC) by DemTech, an e-voting expert group based in Denmark. This attachment provides VEC motivation for and responses to a system assessment requested for the new vVote software. Responses from vendors accompany this document.

Between –team code review

VEC ran an internal between-team code review for suVote and Ximix. These two software products make up the vote collection system and the vote shuffling and decryption mechanism, respectively. Teams were given source codes and build information and advised to pursue a number of goals. A person-week of work was allocated for this to each team. The outcomes of the internal review led to changes to software. The review and the outcomes were given to DemTech to also examine.

Commentary from VEC on general aspects of the report

Not an audit

The DemTech report (Report from here down) is not an audit and it does not comprise a certificate or binding statement of audit compliance of vVote software.

VEC has previously sought an audit and audit certificate for new or changed software forming critical services in elections. vVote was not be surrendered for such an audit for these reasons:

1. vVote is a software design for e-voting considering that e-voting should be “software independent”. This is a recent development first proposed in US elections by US expert Ron Rivest[nist]. Specifically “an undetected error or fault in the voting system’s software SHALL NOT be capable of causing an undetectable change in election results”[provwiki]. That is, even if the entire software system is replaced (“trojaned”), providing the system observes the implemented voting protocol, any malfeasance on the voting data themselves will always be self-evident via inconsistencies in emitted system proofs. It should be stated here that maintaining the secrecy of votes in the case of a compromised system is as such an unsolved problem. Still the vVote software provides many features to also protect secrecy.

2. The converse of point number 1 is that the system provides “provable security”[wiki]. That is, all security claims made about the system can be evidenced by what the system emits. It is not possible for the system to accept receipt data and emit different vote data than are contained in the receipts without also emitting inconsistent proofs, which are self-evident.

3. In 2010 VEC sought a software audit of the e-voting solution in place at that time, Scytl Pnyx.DRE and Pnyx.IVR. The audit scope was very large and included both source code analysis, auditor-controlled secure build and random auditor site visits (“spot audits”) and central system visits to get snapshots of the running system. This approach seeks to “lock” the system; prevent flaws or malware being introduced by the build environment; and to “catch out” any changed systems which are deployed. This approach was discarded in favour for software independence for these reasons and others:
a. In spot-audits, general purpose computer systems being interrogated for their contents can easily obscure their contents. The audits provided above were clearly actions a system could identify as outside its normal mode of operation. This means malware, or rogue operators of the system can make the system appear to be hosting one application (visible to audit) when in fact another is hosted (visible to users). There is one documented attack on computers which can cause even a binary examination of the entire system storage to reveal nothing [bluepill].

b. Secure build and source code controls are important but not adequate security over mission critical systems. This is because the 2010 system was very large (in excess of 2000 files of source codes) and that the secure build environment was a vendor-provided VM, and that software compilers themselves are extremely complex. The size and complexity of this arrangement makes it impossible for exhaustive audit. As an example, it is considered that a software application over even a fairly small size or low complexity cannot be exhaustively audited or declared “bug free”. There is a competition run regularly to demonstrate how unexpected software can be hidden in sources [iocc]. A textbook implementation of a 17 line long basic sorting routine was found to be buggy after 20 years and a published bug fix was found to be buggy [in bloch06].

c. The audit certificate an auditor can provide will never guarantee the audited system will behave in all circumstances as it should. This is because the auditor would in effect be guaranteeing a possibly infinite range of behaviours (in such a large system). No insurer will back an unlimited claim like this and indeed previous audits of VEC software provide no guarantees or indemnities at all [bmm12].

4. VEC sought expert testimony on the suitability of vVote for Victorian elections. Instead of resources expended in controlling the sources, build and deployment of the system instead, the source codes in early 2014 were given to DemTech to examine how well they support the Prêt à Voter protocol and mitigate a wide range of risks in network-based e-voting. This is a deep examination with both human observation and mechanical analysis to confirm the compliance of the software to the protocols.

It is important to add here the process of voting systems certification is evolving. It is the case that certification is desirable for any public system apparently adhering to standards. As it stands there are no standards for vVote but there are general standards in the Common Criteria for security and several proposals for Targets of Evaluation in electronic voting (for example see [cc]).

Software not complete

The DemTech examination of the system was commissioned before the system was complete and before the system had a gold image version for deployment. This was done so that flaws and gaps could be found and resolved during development. In fact a number of questions were raised and the report advises they should be examined. These were subsequently addressed (this is documented in the attached article from the University of Surrey).

It is the case the software has been changed since the DemTech audit. This has been done to improve the software and great care has been taken not to change fundamental behaviours forming
the basis of the DemTech write up. As above, the key evidence that the system performs correctly can be publicly, independently derived from the proofs the system emits.

**Procedural matters**

An attached document describes the operating environment for vVote. This was provided to DemTech to help it consider risks in the operation of vVote in addition to risks in its implementation.

**2014 SGE Implementation**

1. The design assumes the implementation is entirely heterogeneous. It is not: the central server (called MBB) is homogenously implemented and centrally housed and the shuffling and decryption mixnet (MIX) is homogeneously implemented. An heterogeneous implementation would see the MBB and MIX peers running on disparate 3rd party software, diverse hardware and network distributed installations under the operation of diverse organisations and people.

2. Separate services are sometimes on the same hardware: CancelStation (a means to mark votes to not be decrypted) is on the same tablet as PrintStation (the device that prints candidate lists (CLs)). Ideally they are separated and CancelStation has stronger physical security.

3. The verifiability ends at decryption and shuffling. It does not extend to the count.

4. Only the vVote votes have end-to-end verifiability, the rest of the election is conventional paper voting (with e-counting of Region votes). The vVote votes may not reach 2,000 for the whole election.

5. vVote has known privacy risks not present in paper voting. For example, the voting device is a networked computer that knows the clear vote and could theoretically publish it. A previous design was considered with voting device entirely offline that used a separate online “ScanStation” machine to scan the elector’s Preferences Receipt (PR). This design could not be made reliable enough because the limitations of OCR on a printing device trialled with the voting device. However even in this design, PrintStation also knows the CL sequence and could collude with the ScanStation. This risk does not affect integrity, but it does affect privacy.

6. The design does not encompass voter identification or mark-off so there are no integrity guarantees in vVote for voter authenticity.

7. The system does not provide any how-to-vote-cards (HTVCs) so blind users must (somehow) transfer HTVC advice in to their vVote voting session.

8. No unsupervised voting. The protocol assumes meaningful, rigorous human supervision in a controlled environment and strong procedural security over the staging and operation of vVote devices.

**Overall design limitations**

1. E2e does not allow the identification of individual damaged or changed votes in all cases of fraud or errors. It is possible to quarantine the votes taken from a known bad device (also from a specific point in time) but it is not possible to identify or assess arbitrary votes as being part of a broad compromise to the system. The broad compromise itself will be known with high confidence due to audits, but it will not be possible to segregate the
affected votes since only the voters are allowed to know how they voted and thus are the only ones who can perform individual the audits of their own voting choices that can detect trouble.

2. E2e does not provide recovery of damaged elections. It may provide redundancy and resilience, but this does not mean it is able to undo any and all damaged or deleted votes in a serious attack or failure.

3. If there are low numbers of voters, e2e cannot be publicly verifiable since raw vote data will be withheld from publication due to privacy risks. If votes are “tagged” (such as with voting location or with the district a region vote was cast in) then more votes may be withheld since tagging increases privacy risks.

Comments on the Report

Formal methods – the scope of the report did not include formal methods as a tool to determine the correctness of implemented algorithms. The scope went as far as functional matching to confirm the implemented algorithms matched the key literature on which they were based. Actual formal methods analysis is a very laborious and time consuming process. This should eventually be attempted on any mission critical software but for this system we rely on the audit design to detect integrity gaps. Before formal methods are used, it would be desirable to expand the infrastructure to use heterogeneous implementations of the same protocol. See item below on homogeneous implementation.

Users cannot directly verify algorithms (hence system is not verifiable) – p22 This report refers to the gap between verifiable collected receipts and verifiable shuffle and decryption proofs. It also makes the more general observation that any voter is not actually capable of verifying computer algorithms and must trust software. Since the report, VEC has provide a reference verification programme for the shuffle and decryption of the votes and Surrey has provided reference implementations of the BLS signature checker (for cooperatively signed votes, audits and cancels) and a verifier which checks ballot generation audits and receipt packing to ASN1 objects. It is the case that the general public can run these verifiers on their own PC but they must trust that the verifiers actually do what they claim to do. To gain trust in these components is to gain trust in the entire system. It is expected that a number of specialist (commentators, the media, and academics) will also examine the verification tools described here. Such is the case with elections that the presence of scrutiny and the absence of serious complaints comprise a successful election. It is the case that only one specialist needs to find one flaw in the published source codes that built to the verifier software in order to call the e-voting in to question. This may seems like a considerable gamble, but in fact it is extremely important that this transparency is present because other forms of malfeasance, poor quality coding and poor revision control lead to undetectable flaws if such transparency is not in place.

Availability concerns – P23 – Availability is indeed a risk for this implementation. The risks concern the newness of the system, its complexity and the centralised, homogenous implementation (mentioned above). To measure and mitigate these risks VEC has pursued a large programme of testing between 5 December 2013 and 25 September 2014. Such testing has been executed by dedicated professional testers. In addition, SuVote was separately tested by (non-project) software specialists in the United Kingdom in October and November 2013. VEC Testing did reveal bugs and
so retesting and regression tests were duly run. As the Report advises, VEC will indeed provide a paper voting failover for vVote.

**Ximix not well defined** – The project had need of an RPC mixnet and the available academic literature provides considerable detail on how this tool can be built. A reference list is provided with ximix documentation. General requirements were provided by Surrey for VEC to issue in the Ximix contract (for example, that RPC be extended to handle tuples). It emerged during development that there were details that were missing or contradictory and that publicly available source codes for implementations in academic papers were incomplete. To resolve this, either the academic contractors or academic volunteers provided advice; and/or Cryptoworkshop derived its own implementation based on its knowledge of the field. Cryptoworkshop has considerable experience in cryptographic implementations being the authors of BouncyCastle Cryptographic API for Java [bc] and cryptographic services for the Google Android operating system. Ideally a full specification could have been written but it was decided between project members and academic volunteers that the scientific papers and other public domain material were the most reliable and direct source for the design.

**FindBugs output:** All issues were fixed or annotated as appropriate in the sources.

**Functional requirement 6:** Mixing and decryption are not two separate actions. In the present version, one cannot shuffle without decrypting.

VEC: This is a constraint of the project scope. See below: there is a stand-alone decryption library but it was not deployed this time.

**Functional requirement 7:** Because of the kind of mixing used (Randomized Partial Checking), no zero-knowledge proofs are generated during mixing but only during decryption.

VEC: This is a constraint of RPC.

**P13 The computation of h...**

DemTech confirmed subsequently that the software had been corrected:

(28/3/13) We spot checked the Ximix code: The updated code appears to reflect our recommendations as outlined in our report regarding the implementation of Chaum-Pedersen commitments and [14].

**P14 The paper [12] describes two options to shuffle ciphertexts...**

CW: the advice from Surrey was that from a security point of view it was equivalent, I'd also say that if you pair on individual nodes (ie. do 2 shuffles on one node in order to break the input/output correspondence), I believe the Surrey position is correct as is the implementation.

**P14 The implementation of computeProof appears to follow [14]...**

DemTech followed up with another check of the changed source codes for Ximix.

*ProofGenerator.java*
- We checked method computeProof.
- We checked method computeChallenge. Stylistic remark: The fourth argument is an ECPair. This should probably be an ECPoint, to make it consistent with the function computeChallenge implemented in ECDecryptionProof.java. Also the naming of variables should be made consistent.

**ECDecryptionProof.java**

- We checked method computeChallenge.
- We checked method isVerified.

**ClientCommandService.java**

- We checked method isVerified.

**P18 – ServiceConnection does not implement recovery actions:**

CW: With the ServicesConnection class the framework will support some modes of node failure, but for the most part if a node falls over operator intervention is required - this is in line with the original spec. ... where it was possible to put in place at least some work that would contribute to making it easier to put in better recovery down the track it was done so...

**References**

[cc]  [https://www.commoncriteriaportal.org/iccc/9iccc/pdf/B2303.pdf](https://www.commoncriteriaportal.org/iccc/9iccc/pdf/B2303.pdf)


[bb]  [http://bitbucket.org/vvote](http://bitbucket.org/vvote)


[ioccc]  [http://www.ioccc.org/years.html](http://www.ioccc.org/years.html)


Deployment 2014 and considerations for vvote design

This document is being written as supplementary information for the Demtech review of the vVote system. Specifically information was requested about technical and procedural measures that may impact the Pret a voter / vVote design and were not provided with source codes and technical information to Demtech.

This document should provide a means for Demtech to better weigh up reported issues in vVote against larger opportunities or compromises in procedures deploying and supporting vVote.

This document is a report and summary and what it describes may change since final deployment of vVote is some months off and end-to-end testing is still under way.

The document is broken down in to procedural controls for vVote starting with build, then to technical support, operational staging, operational deployment, support, issues management and then post-processing of collected votes.

Some acronyms

EVC – Early Voting Centre or Election Office set up with vVote gear

VPS – vVote Print Station, an administrative tablet for printing candidate lists and getting reconciliation reports of the day’s voting

CL – Candidate list – Pret a voter shuffled candidate list part

PR – Preference receipt – Pret a voter shuffled preferences list (to align with CL)

EVM – Electronic voting machine – voting tablet

MBB, WBB, WBBM, MIX, RAND, VVA – services: Private Web Bullentin Board peer system, Public Web Bulletin Board, internal Public Web Bulletin Board mirror, RPC mixnet peer system and console machine, Random number generator peer system, vVote Administrative system.

UoS – University of Surrey, developers

CW – Cryptoworkshop.com, developers

UAT – User acceptance testing – activity or technical environment for testing

BA- business analyst

Build

vVote has been built by three separate teams – VEC, CW and UoS. The teams manage their own source control, documentation and build pipelines. Two teams use GIT, VEC uses Microsoft Team
Foundation Server (TFS). Version control, unit testing, separate environments and other controls have been managed within-teams.

VEC manages seven environments. vVote uses five of them. They are development, systest, UAT, pre-production and production. vVote does not use a staging environment nor a break-fix environment. The attachment [1] gives more details of these environments and current status of builds of them.

vVote BA and scrum manager has been the vVote Project Manager. vVote has a test manager and has access to VEC IT Quality Manager and two BAs. vVote development has used two developers and one remains.

Configuration control

Client devices are “rooted” Android tablets. The root process is via generic tool to modify the boot partition. The tablets were rooted to allow a cut-down version of the open source Android (CyanogenMod) to be used. This was chosen as it allowed the clients to connect to a camera and a printer. The official operating system did not allow this.

VVA configures the clients using ADB. VVA also controls some parts of the configuration for MBB, WBBM, WBB and RAND. Specifically:

- VVA imports and converts VEC candidate, party and affiliation data to JSON for vVote
- VVA reads candidate and party names and generates synthetic audio MP3 placeholder files
- VVA accepts human agent candidate and party audio files to replace placeholders before the final version of the configuration bundle is signed
- VVA writes certificate files such as filemessage.bks, evm.bks, vps.bks and cancel.bks which are used on MBBs to authorise transactions with client devices
- VVA writes WBBPeerConfig.json for each MBB peer specifying the config given in Technical Support Manual (see Surrey submission)
- VVA writes RAND config
- VVA writes ClientConfig.json for client devices (see Surrey submission)

Tablet devices carry version numbers that can be read off the software interface for both on-board software and on-board configuration bundle. The configuration bundle is signed in an offline ceremony before being published to where tablets can download it.

All software and configuration on field devices is signed. The configuration is publicly available albeit at an orphan URL on the WBB server.

Build Environments

CW and UoS software is Java. In 2014 Ubuntu server or desktop LTE12.04 is used for MBB, RAND and MIX systems. This is for performance reasons (as there are native implementations of certain cryptographic libraries on this operating system not yet available to Windows). Windows server 2008 is used for WBB, WBBM and VVA. VVA is .NET. Each environment provides separate virtual machines for all of these services.
Production and Systest use the peered systems in their correct configurations for performance and reliability. Otherwise, peered systems are run on single VMs.

Tablets are built in one of the environments and labelled as that environment. Tablets being built for production will be built in production.

**Testing**

UoS and CW have performed their own testing. They have shared unit test coverage and other testing between teams in a between-team review of software.

VEC performs end-to-end functional and non-functional testing in all environments and commenced on the 1 February 2014. This testing ends in pre-production on the 31 May 2014.

The testing and quality plans for VEC can be provided.

**Network**

All environments are on the same network except pre-prod and production. These two environments are virtual LANs (VLAN – a VPN and routing rules within a physical LAN). All devices in the field will use physical Ethernet and be routed on two VLANs from routers at all sites (except London) back to VEC LAN VLANs for VPS and EVM devices. Routers are BDSL with 4G failover.

There is SSL bi-directional signed traffic between vVote clients and vVote servers. A CA signs all client keys and the servers have the CA public key (CRLs are manually freshened).

Routes (connections can be initiated in one direction):

- EVM devices route to MBB and WBBM (to get sessions, post votes and to get the configuration (from WBBM)).
- VPS devices route to MBB, WBBM and VVA (to perform POD, to get configuration and to make cancel requests)
- VVA routes to EMS (the VEC election management system, to get nominations and ballot draw info)
- MBB routes to WBBM and VVA (to perform the commit protocol (WBBM) and to push telemetry data (VVA))
- RAND routes to MBB (to post randomness commitments)
- WBB routes to VAA (telemetry)
- WBBM routes to WBB (to mirror commit files and client configuration)
- MIX is offline
- CA is offline
- Imaging service is offline (the service that is used to flash Cyanogenmod operating system ROM on to android tablets)

London will provide a VPN via a local ISP and a 3G router. This also will enter the right VLAN for production.

Transfer of all other files and configuration is either manual or via VVA (for example using Android Debug Bridge – ADB push and pull). For example
Keystore files assembled by VVA when setting up devices are manually installed on MBBs
Keystore files for other services (like RAND) are manually installed on MBB peers
An offline CA is used to sign SSL CRLs from client devices
VVA prepares commit files for MIX and these files are manually transported to the MIX network
Forcing the MBB peers to perform an “extra commit” is a manual task for each MBB console
Adding CRLs to MBB is a manual task

Voting locations and procedures

There are 25 voting locations using vVote in Victoria and one in London. In 2010, a deployment of 101 locations took 961 votes (740 in London).

There is no vVote for Australian interstate sites nor for Victorian mobile sites where mobile gear is used (such as hospitals, nursing homes, jails etc).

Please see attached [2] which is specimen advice and instructions for VEC to incorporate in to training and support documentation. Training content and manuals have not been written and final wording, procedures and advice may vary from what are proposed.

At the time of providing information to DemTech the following processes anticipated in staff training and voter support were not confirmed (we also list some other transparency measures), specifically

1. Candidate lists (CL) and preference receipts (PR) must be in all languages and English (LOTE). This can be provided, but is not approved. Now not provided.
2. WBB must be LOTE. This is provided by Google translate for some but not all languages.
3. CL Audit must be trained for and offered. CL Audit will be trained and anyone asking for one will be handled. Not provided.
4. Shredders at all sites for use by voters themselves. Security boxes with seals. CLs all shredded at end of early voting by VEC staff. Not provided.
5. Blind user readback deployed and offered to electors. Present, but not approved. Provided.
6. Coloured CL slip print paper. Declined. All slips are white, CL slips are thicker. Not provided.
7. vVote brochure at all sites in English and LOTE. Not approved. Provided, not confirmed if LOTE.
8. WA “offline” (that is, machines revert to just printing out marked ballots) mode of operation. Declined. Not provided.
9. PR readback via mobile version of WBB (that is, QR code on the PR directs user phone browser to WBB site with query string for receipt lookup). Declined. Not provided.

Public information

Public information about vVote is
• Verbal via suggested training to our staff, as above.
• In English, via the WBB
• Promoted in mainstream media (campaign not yet designed)
• Promoted via local and specialised media such as Chinese local newspapers (campaign not designed)
• Direct contact to all blind citizens via mailout
• Promoted directly to disability and language organisations (campaign not designed)

Provided from WBB. WBB provides
• Orphan URL to the signed configuration bundle for client devices
• Orphan URL to candidate name pronunciation checking
• A page of information
• A page to look up a receipt (and to download commit files)
• A page of verification information. This page provides
  o Sets of zipped and signed files for each major race
  o Shuffle proof files
  o NZKP files
  o Unpacking check and conversion to CSV check
  o EXE file to do accept the above and input receipt files
  o Instructions. This should include link to the source codes on bitbucket.

Bitbucket documentation should be updated to provide

• Information
• Summary of the roles of all sub-repos
• Documents on the setup of built applications

Live monitors

The server systems are connected as above via routes. VVA provides two pages for monitoring at all times. They provide these

• Status of any tablet during staging such as what staging step it is at (start up, ballot generation, etc). This is known directly from staff using VVA during staging.
• Status of spare tablets such as if they are deployed, and to where. Again, known from manual settings on VVA Staging.
• Status of live tablets as having downloaded the configuration bundle. This is from telemetry from WBBM to VVA.
• Status of WBB as in having been used to visit vVote pages or lookup-receipt (telemetry from WBB to VVA)
• Status of live tablets as having performed POD, post-vote etc (telemetry from MBB to VVA)
• System errors are pushed to VVA via telemetry

Post processing
The server systems are connected to each other for various purposes but the system is not interconnected for all tasks. Post processing consists of these steps

1. MBBs are shut down manually to prevent more voting after last day of early voting
2. Commit files on WBB are downloaded to VVA via VVA request
3. VVA packages commit files by performing the vote packing in Surrey technical manual
4. Packed votes are moved via USB stick to the MIX network, which is on an isolated LAN
5. The MIX console uploads the votes and some or all can be chosen to be mixed and decrypted at once, or it can be done in batches.
6. The MIX system mixes and decrypts.
7. All the files it emits are taken to VVA
8. VVA converts emitted CSV files to PDFs
9. PDFs are given on USB to Elections Branch who either print them for addition to the handling of postal votes or they send the PDFs to remote sites to print there.
10. VVA reports that emitted files from the MIX have certain numbers of votes per CSV file. If there are few, the Commissioner can choose that certain emitted files are not to be published (for example, if there is one vote only from a voting location).
11. VVA uploads all the emitted MIX files (with no privacy risk due to single votes or few-votes) to the WBB Verify page

All printed PDFs have a page footer identifying the matching row of the vote to the CSV file published. This is intended for audit at voting sites and to resolve printing problems where printers print duplicates or jam.

References

[1] vVote VMs and machines for all envs.xlsx Excel sheet of current built environments for vVote.
[2] EVC staff support for EAV and EAV users 0.9.docx Word document of information for training
UNIVERSITY OF SURREY
RESPONSE TO “REVIEW OF THE VVOTE SYSTEM”

for

VEC vVote System

Version 1.0

Tuesday 30th September, 2014

Prepared by Steve Schneider,
Chris Culnane, Matthew Casey
University of Surrey Response to the “Review of the vVote System”

Steve Schneider, Chris Culnane, Matthew Casey

30 September 2014

The DemTech Review of the vVote System [R1] considered the entire verifiable voting system being developed by VEC for the November 2014 Victorian State Election. The response provided here is with respect to the Surrey contribution: the back-end system developed by the University of Surrey.

Overall we consider the report as very positive for us and we are pleased with the overall tone of the report and the overall recommendation. There are of course a number of detailed comments, as one would expect for a project of this size and complexity, but having reviewed the comments we do not consider any of these to be major issues for the current implementation, and they have now been addressed in response to the report. Some comments are concerned with future maintainability and they can be dealt with on a longer timescale than those concerned specifically with the current implementation intended for November 2014.

The report made three specific recommendations with respect to the back-end system:

vVote: Ensure that message sequence charts and protocol implementation match. We have gone through the documentation and updated the Technical Report [R4] with additional message sequence charts, and these are now in alignment.

vVote: Use JSON as a data-interchange format only. Serialize and deserialize as Java objects. This is an appropriate suggestion for long-term maintainability, but it is not essential for the current implementation for November 2014, which we prefer to keep stable. There is a substantial amount of work required in this proposed refactoring, and evaluation required with respect to efficiency.

vVote: Replace MongoDB with a database that offers transaction management. The main concern appears to be that faults might be introduced in future modifications of the code base, resulting in livelocks or deadlocks; the review did not identify any faults in the current implementation. In fact our use of MongoDB does not introduce deadlocks or livelocks, as all threads which are processing messages are finite and never hold more than one lock. These considerations may require more explicit documentation for future developers, but the current implementation is robust for November 2014. Our approach using MongoDB is motivated primarily by performance considerations, and use of a DBMS will reduce performance significantly. Since this is not a practical problem for November 2014 we will defer revisiting this until future versions.

The rest of this document provides responses to the detailed comments of the report:

Documentation: We have reviewed the Surrey Technical Documents for consistency. We will also include further explanation in the journal version of [10] currently in preparation, in the light of the review’s comments. A peer-reviewed version of [10] has now appeared as [R3].
There is some confusion as to which documentation is with respect to the Surrey back-end, and which is applicable to the whole system. In fact all technical and project documentation delivered by Surrey is concerned only with the Surrey back-end. Background research papers (e.g. [7]) are typically concerned with the design of the whole system.

**Code:** The code review comments have now been worked through and considered carefully, and modifications have been made where appropriate.

Our responses to the specific issues raised in the report are as follows:

### 3.1: vVote Requirements Document

“Moreover, requirements should be given for hardware as well as software.”

Baseline hardware requirements are already specified in PF7.

“...all system requirements should be given in a single document.”

It was never the intent of this SRS to cover all components as each are specified separately.

“Some of the supplementary papers, e.g., §6 of [7], state security properties that the vVote system should fulfill that are not listed §3 of the SRS.”

Requirements SF1 and AT4 provide overall summary constraints on the security protocols used such that the rigorous detail of each protocol required is covered within peer reviewed publications.

“...properties relevant for security are mixed in with other functional properties”

This is deliberate as the “document follows the IEEE ‘Recommended Practice for Software Requirements Specifications’ using the template organisation for features [IEE98]” (SRS, P9). Organising by features is an IEEE standard approach to requirements specification.

### 3.2: vVote Requirements Satisfaction

“IR1–2 require CSV files which were not present in the sources we received; they should be added when the system is deployed”

The configuration files are particular to an election and are not therefore included within the open source repository as they will differ with each use.

“SF1 requires end-to-end verifiability. The vVote design achieves this in general. However, see the limitations discussed in §5.2.1.”
Noted. Depends upon runtime operation.

“PR20 may be violated. See the comment on thresholds §4.”
The protocol as implemented meets requirement PR20.

“PU5 should be updated: the vote receipt also contains the district number.”
SRS updated.

“IR4 is false: Ximix does not use JSON.”
SRS updated to state explicit components.

“AT7 requires that the software shall be written using Java 7. For the overall system this is false as some parts are written in Javascript and C#, e.g., VVA is programmed in C#.”
This SRS only covers the components listed and not Ximix. SRS updated to state explicit components and Java versions.

“PP10 requires that all ballot printers have unique identifiers.”
SRS updated to include PoD Printer identifier in uniqueness.

“PP13 is false. The method buildBallot in ballot_generation.js does not appear to check the authenticity of the received podResponse.”
An error response is sent if the ballot is invalid.

“PR16 is false. We interpret mixed votes as the votes that were mixed by Ximix. Ximix does not send the votes back to the privateWBB.”
Mixed votes are committed via the Private WBB Peers.

“PR22 appears to be true. It should be true by design and we have found no evidence contrary to this claim. However, there might be other external factors effecting it, the evaluation of which is beyond the scope of this report.”

Noted.
4.1 Programming Structure
We use JSON throughout for transferring messages. The JSONObjects are not directly passed around; they are encapsulated in a JSONWBBMessage or a subclass of JSONWBBMessage. When generating a response we generate it via a JSONObject.

4.3 Thresholds
The protocol expects all peers to be online at the end of the day for commit. In a worst case scenario, i.e. a server is irretrievably lost; all peers can be manually updated to remove that peer from their configuration file and restarted. They will then perform the commit with all remaining peers – the threshold will remain the same. Whilst the protocol can run with only a threshold of peers the proof is considerably harder and the advantage of requiring all online is that you can recover from larger data losses (i.e. it strictly only requires 1 peer to still have a valid copy).

4.6 Refactoring
The Ximix code duplication is due to Ximix not providing a standalone crypto library. We have to extract the public key and private key shares from Ximix files, which requires we reference some of their code. We do not want to reference the entire Ximix distribution just to convert a single key file. Hence we were provided with a minimum subset for extracting that public key. Ideally we would have a Ximix crypto library, but since we don't, this is the best option.

BallotGenAudit and BallotGenMix
The code is different because they are constructing different zip files. The ballot gen audit also now includes an additional file, creating distinct code. This was anticipated from the start and hence the two methods are separate and were not refactored into the superclass.

4.7 Fault Tolerance
MongoDB is suitable and has been extensive tested within the system to ensure that it is robust and trustworthy. This tested has also looked at potential concurrency issues. Switching to a transaction based database would increase response times and scalability issues.

4.8 Exception Handling
SecurityExceptions are subclasses of RuntimeException. It is generally accepted that a RuntimeException is non-recoverable and hence should not be caught. The system will have to terminate if a RuntimeException occurs – for example, if the system does not have permission to access the storage device. As such, it is a fatal error and should be handled as such.

4.9 Libraries
The reviewed code was built against BouncyCastle 1.49, with 1.50 only released during the final week before delivery and hence this version was not used within the tested baseline. ECUtils pre-exists the implementation inside BouncyCastle and the normalize method was only introduced in 1.50. We have since moved the code to 1.50 and addressed the deprecated methods it introduced, as well as the point normalization.
5.2.1 Design
The vote packing can be independently checked for any single result by reconstructing the packing. For example, take the candidate Ids and pack them back up (it does not need to be performed in the encrypted domain and does not require the whole table to be rebuilt). To check an entry in the table requires performing EC point multiplication and addition. That is similar to the type of calculations that would be needed to check the proof of decryption. The use of the vote packing table does not place any additional burden on a voter who wishes to independently verify the output.

6 Code scanning
We have run CodePro and Findbugs analysis on our codebase and addressed the issues they raised where appropriate.

6.1
This has been addressed by moving the thread.start call out of the constructor.

6.2
The only reference to readLine is on a BufferedReader that wraps System.in. We have modified it, although this is a wrapper of System.in, so should never return null anyway.

6.3
We have checked all cited occurrences of missing return checks and corrected the code as needed (not all return checks are needed, for example for mkdirs): Have updated to throw an exception when mkdirs/delete fails – thus checking the return value. Findbugs has been run on the code and all ignored return values addressed

6.4
CandidateCipherText: will override equals method

RemovedCipher: will override equals method

6.5
There should only ever be a single instance of Client running in an execution and its type is fixed.

6.6
Fixed for TableBuilder and KeyGeneration

6.7
The comment actually discusses removing of the synchronization for performance reasons, not for the reason claimed. The purpose of the synchronization is to allow the removal of an empty queue from the parent collection. Whilst the synchronization does indeed operate orthogonally to the concurrent collections class, it operates correctly when all access to that object is synchronized. Due to the nature of how the code runs (all serial numbers that are the same are queued and processed sequentially) we can guarantee that the inner queue is only being accessed by a single thread. However, our concern was the constructing thread creating a new queue or adding to the inner queue whilst it is being removed. We have rewritten this portion of the code anyway, to make use of a weak key map to avoid the problem of having to explicitly remove key references and the associated synchronization.
6.8  
The value is not used and will be removed.

6.9  
We have a global catch exception in the send error message to prevent the act of sending an error message causing a crash. For example, if there is some form of runtime exception on a particular network link we want to catch it to stop the whole peer being taken down by us trying to send an error message. The Exception ensures we catch RuntimeExceptions as well, since we can log them and want to try and continue

The remainder of 6.9 and 6.10-6.12 are concerned with Ximix.

6.13  
This has now been addressed. maxZipEntry is now of type long and multiplication is done with long arithmetic, as suggested in the report.

B2 Code Review

Client  
• Will look at in the future – switch is adequate for the moment  
• Private Key File: Access control is assumed to be controlled by the OS – there is no benefit to adding a password to a config file.  
• CRYPTO: Have fixed variable typo  
• getJSONPublicKey – Exception string – have updated string  
• getJSONPublicKey – There is an internal key store used for storing the keys and a second key store containing just the public key so that it can be easily exported for inclusion on the peers. Empty password: this is because we are not storing a password

BallotGenAudit  
• If it fails it returns an exception with “An exception has occurred whilst conducting a BallotGen audit”. It is fatal and it is not going to be consumed by the user, but the underlying exception is included as well for logging.  
• Audit specification – This is given in [R2] which was provided for the review.  
• doAudit: This method does what is required for the audit– it reveals the randomness.  
• submitAuditToWBB – this is a filename, it is appropriately documented in JavaDoc with an appropriate explanation.

BallotGenMix  
• There is no attack. As noted by the second bullet point, the files have signatures attached from the WBB and are verified before being processed. The zip files cannot be altered without detection.  
• We agree that verifyAllFilesAndSignatures is critical, and it is treated as such.  
• Noted  
• SecurityException is a RuntimeException that is unrecoverable; there is no reason to explicitly catch it.
• The message sequence chart has been updated, however the actual audit process is covered in [R2]; the message sequence chart only shows the communication, not the underlying processing.

**CandidateCipherText**

• The variable name “my” conforms with standard practice in cryptographic code. Furthermore, all variables and methods have appropriate JavaDoc (including private variables) to avoid any confusion, for example, getMYREncoded has the JavaDoc: “Gets the BigInteger my^r cache value (encoded ECPoint)”.

**CandidateID**

• The method returns a new instance of CandidateCipherText for re-encryption. This will be called many times during ballot generation (once per candidateID, per ballot). We have the appropriate JUnit tests that test both CandidateID and CandidateCipherText. We are not writing a library package or framework, this is an internal component within our code.

**CombinedRandomness**

• keyPair – KeyPair variable is loaded via the readJSONObjectFromFile method in either the BallotGenMix or BallotGenAudit, depending on context.

• Invariants: The RandomnessFiles class constructs the arrays and we can therefore be certain they are all the same size, otherwise an exception will already have been thrown.

• A class constructor will never return null, it might throw an exception, but it won’t ever return null. The reason you check for null in the finally is in case an exception was thrown during construction, in which case the finally cannot close the reader because it was not constructed. However, if your code is executing beyond the construct line it is guaranteed not to be null.

• SecurityException: we avoid catching runtime exceptions as they are fatal to the system and dealt with appropriately by Java.

• searchInProgress: it is no longer needed, as a result of refactoring, it previously was required when processing took place after finding the serialNo. However it does no harm and provides greater flexibility in the future, so we prefer to leave it in.

**RandomnessFiles**

• verifyAllFilesAndSignatures: There is no valid case for a peer trying to send randomness more than once. Both could be valid, but only one should be used, but which one? It is clearly an exception separate from not having enough randomness.

• getIntParameter: The current code is consistent with the method signature and so we currently leave it unchanged. The suggestion to throw an exception will have a far-reaching impact across the codebase, and will be considered for the future.

• Invariant: They are kept in sync by the way the code is written.

• We never convert to an unchecked exception, which means any conversion to a checked exception is caught.

• Have changed “WBB” to a constant.
• findRandomnessFile: we avoid catching runtime exceptions as they are fatal to the system and dealt with appropriately by Java.
• A reset is not needed.

RandomnessRow
• It is not clear how RandomnessRow returns null for getSerialNo. That method calls getString on a JSONObject, which will throw an exception if the field does not exist or is not a string (including if it is set to null explicitly).

B3

RandomnessServerUI
• Style of config: RandomnessServerUI uses a JSON config file like all other components.
• Using super() as the first line of a constructor (even if the super class is Object) is idiomatic.
• EqualsIgnoreCase — this provides flexibility.
• getFailures allows it to process multiple keys and then list the ones that failed, it does not fail entirely if it gets an invalid key. As such, the method is started and can be left to run. On completion getFailures is checked to get a list of any keys that failed.

ProcessKeyFiles
• Unique IDs are managed by the VEC.

B4 PrivateWBB

CommandListener
• The only casting I can find is in setting up the log appender, which is required because it casts it from an abstract SLF logger to the concrete logback logger, which we then need to interact with.
• This is logging to the front-end, not logging in general. Since we know all of our classes have a top level package of “uk” we can guarantee to log all of our messages.
• The method shutdown is a void method. It shuts down either in an orderly fashion or an immediate shutdown. The shutdownExecutorService may shutdown with tasks remaining (if forced or timed out), but it is still shutdown. As such, the call to shutdown always results in a shutdown.

AuditMessage
• If the message has been reduced it has the internal field “_reducedParams” set and loads the reduced params from that. The class can handle both situations and when initially loaded there is no pre-condition.
• The appearances of this.id and this.getID() are a consequence of the refactoring of the code during development. Within this class they are interchangeable.
• The implicit checking on the array together with exception handling is sufficient: if an out of bounds request is made then an exception would have to be thrown in any case.

• Inconsistency removed from comment: we include commitTime in the internal signature, but not the external one, and the comment has been updated to reflect this.

BallotFileMessage
• Have updated logger message

CancelMessage
• The reason there could be multiple cancel messages from the client is in the comment above that line “Because the cancel procedure can be run as many times as desired there is a set of original message.” There is both an inline and class level comment noting that the cancel protocol can be run by the client multiple times.

• The Array and for loop are necessary, the data comes back as an Array because there could be multiple cancel requests from the client. The process does not know which is the most recent or active so it tries to respond to them all.

• Comment updated.

CommitR1Message
• The commitTime is the start point of the commit period for which the submission will be committed to the PublicWBB. It acts as both a time and identifier for that particular end of day commit.

• It checks there is a valid commitHash, if one does not exist an exception is thrown. The process of checking there is a commitHash in the database returns the commitHash value.

CommitR2Message
• All messages are subclasses of JSONWBBMessage. If a peer has sent invalid data (i.e. data that is cannot be loaded as a subclass of JSONWBBMessage) we do not want to continue loading it.

• Using super() as the first line of a constructor (even if the super class is Object) is idiomatic.

MixRandomCommitMessage
• They are overridden because the signatures are slightly different. In the commit of the random data the printerID is included, this is not the case for a generic FileMessage, and hence we have to override the methods from FileMessage in MixRandomCommitMessage to construct the appropriate signature to check.

PODMessage
• It checks the commitTime amongst the peers, but not from the client. Since the internal peer messages are abstracted none of these messages are shown in the technical report.
• It is a configurable option as to whether the check is done in real-time. It is a processor intensive task, since it requires re-encrypting multiple ciphers. As such, it would be possible to hand off that processing to separate machine – it is not really the job of the WBB to be doing that kind of crypto check. A standard Audit would also catch a cheating PoD, so it does not appear mandatory to perform the check in real-time.

StartEVMMessage
• The following explanation is given in the class level comment
  “Represents a StartEVM message received from an EVM/EBM when the voting session begins. The purpose of this message is to stop the timeout from occurring when a vote is submitted. Once a startEVM message is received the voter can take as long as they like to actually submit their vote. This message is handled slightly differently to other messages, since the purpose is slightly different. Normally we would seek a consensus on the message before responding to the client. This demonstrates that the message will end up on the PublicWBB because a threshold of peers must have seen it. However, the sharing of any missing messages is only performed at the end of the day. We need the agreement to take place before then. Otherwise it would be possible for a threshold of peers to see the StartEVM message, respond with a signature and then one of them go offline. In such situations another peer might reject a vote message because it hadn’t seen the startEVM message. The easiest way to avoid this is to require the joint StartEVM signature to be included within the VoteMessage. The peers can then agree on it in round 1 of the VoteMessage, thus achieving the consensus on it without requiring all the peers to have seen the original message.”

The inline comments which are incorrect have been corrected.

A Message Sequence Chart is now included in the revised Technical Report

• The last two bullet points are unclear: the first states there is a missing sequence chart, the second refers to the sequence chart which was missed.

VerifyPreferences
• It is unclear what additional comment is required.

VoteMessage
• The implementation currently requires a three race submission for all ballots, even when a vote is not being cast in one of the races. In future this will be generalised, but currently it is bound to the VEC election setup.
• performValidation – the EBMID is sent as the senderID – this was generalised in the implementation to provide better code reuse between different types of messages. [3] was provided for information only, the StartEVM message was requested by the VEC after the initial design to handle a particular use case.
• validateEBMSignature – this is because there are two classes with the name JSONWBBMessage, one with the static fields defined in MessageFields in the Library and the actual JSONWBBMessage class in the WBBPeer messages package. The explicit package identifier is the only way to handle this short of a cross-component refactoring.

• validateStartEVMSignature – Further documentation of startEVM has been provided as a Message Sequence Chart in the Technical Report. The if-then-else is required to handle the different types of possible signature schemes. If a genuine threshold scheme is used only a single (joint) signature will be sent as the startEVM signature. However, if standard DSA is used (with a threshold set of individual signatures) the startEVM signature will be sent as an array of individual DSA signatures.

WBBPeer

• shutdownExecutorService – the code is correct. The waitForFinish indicates that the code should attempt an orderly shutdown, with a timeout occurring if that is not possible. The call to shutdown on the executor service does not immediately shut it down; it just starts the orderly shutdown. If waitForFinish is false it means shut down immediately, which requires a call to shutdownNow instead of shutdown.

• The notion of truststores is not something we defined; it is a concept within SSL in Java: http://javarevisited.blogspot.co.uk/2012/09/difference-between-truststore-vs-keyStore-Java-SSL.html is one such example.

• 1000 is 1 second, since the timeout values in the config are based in seconds we hard code the interval.

• Have addressed ignored return values, these are now checked.

• The principal is the x500 entity attached to the x509 certificate. (The definition from the JavaDoc for the Principal object that is passed to the method is: “an X500Principal of the end-entity certificate for X509-based cipher suites”.) The validity of the certificates is checked by the underlying SSL classes – we are not re-implementing SSL. We then perform an additional, more stringent check, to verify that connection is not only valid, but from another peer with the appropriate IP address. If the SSL connection was invalid it would be rejected before it reaches this point.

• Setting up a default config filename is inappropriate, WBBPeers should only be invoked with respect to specific known config files.

B5 vVote Table Building

BinarySearchFile

• The rounding of integer division is fixed in Java to being “round towards zero”. This means the fraction will be discarded. As such, in positive integers the round will always be equivalent to taking the floor of the value. As such, the rounding is always the same.

PagedMappedByteBuffer

• Same as above.
ByteMergeSort

- There is comment about the fixed allocation, there is no need for it to be a config item, and the performance difference is small enough that it will not make a difference.

Whilst there are plenty of examples of MergeSort implementations available, there is no clear default. Many are in-memory merge sorts or have sub-optimal file handling. As such, there was no compelling reason to use an unknown version from the internet. We also significantly refactored it to provide additional functionality, specifically targeting the table building/VotePacking. Of particular note is that the MergeSort also checks that there are no duplicate values in the table. This is vitally important in the VotePacking setting, since a duplicate would result in an unrecoverable vote. Whilst the probability of getting a collision is negligible it cannot be guaranteed not to happen. Hence, we check whilst performing the sort to get a guarantee that it has not occurred.

DuplicateException

- It is “strongly recommended” in the Java Serializable interface that a class implementing Serializable explicitly declares a serialVersionUID. We generate our serialVersionUID using the built-in generator in Eclipse.

PermutationWorker

- The paper in [18] was provided for information only and describes an (informal) generalised algorithm for VotePacking. The implementation implements the concept but was not intended to be a direct implementation of the methods described in [18].
- PermutationWorker performs the actual ECPoint calculations, using previously cached values to improve performance. The allButLast indicate a value generated from a previous table. For example, if you have a table of size 2, with 4 preferences, you will have the following values already calculated:

  \[
  [1][2]-ECPoint \\
  [1][3]-ECPoint \\
  [1][4]-ECPoint \\
  \ldots
  \]

  When building a table of size 3, each of those values will be reusable for calculating further points, for example \([1][2][3]\) and \([1][2][4]\). We only need to perform a single point operation to add the additional permutation, which is a significant efficiency step. The allButLast is the key to retrieving the previously generated value, whilst the last is the new value to be packed into it. In the above two examples, allButLast would be \([1][2]\) in both cases and last would be \([3]\) and \([4]\) respectively. The code has been primarily written for efficiency – the building of a table is extremely computationally costly. Unfortunately, high efficiency and optimisation often has a negative impact on readability.
TableBuilder

- createSampleCandidates: the method signature says “Utility method for creating sample Candidate ID and storing them into the default file. In production we would expect the candidates ID to be generated externally.” This is extremely useful when building sample tables.
- buildTable: The blocks are referred to by their size, i.e. the zero based block (first one in the array and generated) is of size 1. There is no zero width blocks. All the arrays are zero based.

TVSPermutationGenerator

- This class has been extensively tested and we believe it to be error free in terms of generating permutations.

The table is not a single point of failure, it can be checked and evaluated and in a worst case scenario you can run everything through the mix without packing.

TVSPermutationGenerator does not return an invalid permutation after construction, it decrements the initial value, so the first call to getNextPermutation (the only way to retrieve the permutation) can correctly increment it to get the first value. This is explicitly stated in the class comments.

If we construct the following (which, does not have min=max):

TVSPermutationGenerator tvsPerm = new TVSPermutationGenerator(4,3,1,4);

That is a permutation generator with 4 candidates and a packing of size 3. The following is the first output from getNextPermutation():

[1, 2, 3]
Which is correct.

The complete output is as follows:

[1, 2, 3]
[1, 2, 4]
[1, 3, 2]
[1, 3, 4]
[1, 4, 2]
[1, 4, 3]
[2, 1, 3]
[2, 1, 4]
[2, 3, 1]
[2, 3, 4]
[2, 4, 1]
[2, 4, 3]
[3, 1, 2]
[3, 1, 4]
The number of permutations can be calculated by \( n!/(n-r)! \) which equals 24, which is how many permutations we have.

**VerifyBuiltTable**

- The arrays are zero based, they are not 1 based, but preferences range from 1 to \( n \). There is no such thing as a zero preference. As such the first preference (1) will be in the index zero in the zero based array.

**B7 vVote library**

**ECUtils**

- The normalization is only required when switching to BC 1.50 - there is no normalization available in 1.49 because of a different way of representing the ECPoints internally in BouncyCastle. Having changed over to 1.50 we have updated to code to handle the new methods provided in 1.50.
- ECPair our code was written before ECPair was included in BC.
- A zero check has now been added as suggested.
- BouncyCastle did not include those classes when the code was written.

**References**


[R2] Faster Print on Demand for Pret a Voter, Chris Culnane, James Heather, Rui Joaquim, Peter Y.A. Ryan, Steve Schneider and Vanessa Teague, USENIX Journal of Election Technology and Systems 2(1), 2014


[R4] vVote: a Verifiable Voting System version 2.0, Chris Culnane, Peter Y.A. Ryan, Steve Schneider and Vanessa Teague, arXiv 1404.6822v3, 30 September 2014,
Review of the vVote System

Carsten Schürmann, David Basin, Lorena Ronquillo
March 2014
Review of the vVote System

Carsten Schürmann, David Basin, Lorena Ronquillo
DemTech Group
Ribegade 19 st th
DK - 2100 Copenhagen
Denmark

Final Report
Disclaimer: The contents of this report are the Intellectual Property of the Victoria Election Commission (VEC). Citation of trade names does not constitute an official endorsement or approval of the use of such commercial products. All product names and trademarks cited are the property of their respective owners. DemTech is a registered trademark with the Danish trademark office.
Table of Contents

1 Executive Summary ........................................................................................................... 1

2 Background and Scope .................................................................................................... 4
  2.1 Materials Received .............................................................................................. 4
  2.2 Reviewing Methodology .................................................................................. 5
  2.3 Scope ............................................................................................................. 6

3 Requirements ............................................................................................................... 8
  3.1 vVote Requirements Document ...................................................................... 8
  3.2 vVote Requirements Satisfaction .................................................................. 9
  3.3 Ximix Requirements Satisfaction .................................................................. 10

4 Software .................................................................................................................... 11
  4.1 Programming Structure ............................................................................. 11
  4.2 Functional Matching .................................................................................. 12
  4.3 Thresholds ................................................................................................. 15
  4.4 Randomness ............................................................................................... 15
  4.5 Array Indexing ........................................................................................... 16
  4.6 Refactoring ................................................................................................. 16
  4.7 Fault Tolerance ......................................................................................... 18
  4.8 Exception Handling .................................................................................. 18
  4.9 Libraries ................................................................................................. 19

5 Security ...................................................................................................................... 20
  5.1 General Observations .............................................................................. 20
  5.2 Specific Findings ....................................................................................... 20

6 Code Scanning ............................................................................................................. 24

7 Project Management ................................................................................................. 29

8 Testing ........................................................................................................................ 31

9 Conclusions and Outlook ......................................................................................... 32

References ....................................................................................................................... 35

Appendix A: Requirements Review .................................................................................. 37

Appendix B: Code Review ............................................................................................. 38
  B.1 EVC Component ....................................................................................... 38
  B.2 vVote Client ............................................................................................. 42
  B.3 vVote Randomness Server ...................................................................... 44
  B.4 vVote Private WBB ............................................................................... 44
B.5 vVote Table Building................................................................................ 47
B.6 VVA Component .................................................................................... 49
B.7 vVote Library......................................................................................... 51
B.8 Ximix................................................................................................... 52

Appendix C: Project Management Plan Review ........................................... 59
1 Executive Summary

**Scope.** We conducted an analysis of the vVote election system designed for the Victorian Electoral Commission. Based on the materials received, comprising project documentation, background publications, source code, and configuration and test data, we performed a detailed review of the system’s design, implementation, and its supporting documentation. Our focus was on the functional correctness of the main subsystems and the security of the overall architecture. Our objective was to determine the suitability of the system for elections in Victoria, and to recommend improvements to the system and its development and deployment processes.

**State-of-the-art.** The vVote system is a sophisticated election system. It is impressive in its use of cryptography and distributed algorithms to support highly secure and fault tolerant elections. By using advanced cryptographic techniques, it offers an extremely high degree of transparency, auditability, privacy, and robustness. Transparency is provided by using open source software and making all algorithms and design documents public. Auditability is provided by cryptographic mechanisms, whereby voters receive evidence that their vote is cast as they intended, included unaltered in the tally, and everyone can check that the cast votes are consistent with the election outcome. Privacy is supported by using encryption and mixing (the cryptographic equivalent of shuffling) in various forms. Robustness is supported by fault-tolerant algorithms that distribute computation and trust in different ways so that no one component alone must be reliable or trustworthy, i.e., single points of failure are minimized. All of this is accomplished using sophisticated, state-of-the-art, peer-reviewed algorithms, which have been carefully scrutinized by the security community and an open-source implementation that can be examined by anyone or any organization that wishes to do so.

**User perspective and impact on democratic processes.** The vVote system’s design does an impressive job in hiding its cryptographic workings from the voters. Voters carry out processes and work with artefacts that are different from those in conventional paper-based voting, but nevertheless based on familiar notions like candidate lists and preferences. Moreover, as voters can audit the voting process at various stages, this encourages (and requires) their direct participation in this democratic process.
Complexity, security, and functional correctness. The vVote system’s design is impressive in its innovations and features. The flip side is that many of the algorithms it uses rely on advanced mathematics and are highly non-trivial. Moreover, while the system is designed to behave robustly and securely, even in the presence of different kinds of failures, this only holds when the algorithms are designed and implemented correctly. Even minute design or implementation flaws can result in unexpected system failures. While this is true for most software systems, we emphasize that the complexity of the algorithms used is quite high and the combination of cryptography and distributed computation leads to extremely subtle interactions. As a result, considerable care must be taken in the system’s design, implementation, and test.

We did not find major faults in the vVote system, either in its design or its implementation. Nevertheless, the quality and maturity of the different parts of the system varies considerably. We expand on the following points in the body of our report.

1. The vast majority of the code is in place. However, some minor parts are still under development.

2. Parts of the code would benefit from polishing and simplification.

3. Some protocols did not directly correspond to the provided documentation.

4. Parts employ homegrown solutions, rather than using existing libraries or components.

5. Finally, as it is currently structured and documented, the code will be difficult to maintain and evolve, especially if the development team later changes.

Discussions with the developers suggest that it will take 4-6 weeks to address (1)–(3), which are relevant for the upcoming election. The last two points, (4)–(5), are relevant for long-term system maintenance and evolution, and can be addressed at a later date.

Other Artefacts. The trustworthiness of the election results produced using the vVote system will depend on far more than the system’s design and implementation. The vVote system must interact with other voting infrastructure and the privacy of users’ votes and the integrity of its results depend on numerous human-driven processes. Examples include how the system is deployed, configured, and administered during elections, how cryptographic keys are distributed and protected, how the platforms it runs on are secured,
and even whether voters take the actions asked of them, such as shredding their candidate lists and taking relevant auditing actions. We received preliminary documentation on fallback and emergency procedures for handling system failures. The full details still need to be completed.

**Recommendation.** The vVote system is well-designed and offers advantages over traditional voting processes. Prior to its use, the first three points stated above should be addressed. Even then, as with any complex technical system, there is always the residual risk of failure. For the vVote system, our main concern is not the privacy or integrity of votes, but rather the system's availability. However, provided that appropriate fallback mechanisms are in place, it should be possible for the VEC to recover from any system failures and to gain valuable experience on using vVote to improve its electoral process. We therefore recommend using the system for the upcoming election with paper voting serving as a fallback mechanism.
2 Background and Scope

We reviewed the vVote election system designed for the Victorian Electoral Commission. This final report also incorporates feedback received from the implementation teams on a preliminary version. Where relevant, we highlight conclusions drawn from discussions on the previous version.

We describe below the materials we received, our reviewing methodology, and the scope of our review.

2.1 Materials Received

Our review is based on the following artefacts and documents that we received from the Victorian Election Commission.

- The components of the vVote system developed by the University of Surrey, written in Java. We downloaded this zip file from www.computing.surrey.ac.uk/personal/st/S.Schneider/VEC/vVoteDelivery20131211.zip. The zip file is 90,023,352 bytes and includes background literature, project documentation [1, 2, 3, 4, 5, 6, 7], test results, and the source code of the following components: Table Building, Randomness Server, Private WBB, Public WBB, Client, Library, and Third Party.

- The Ximix component developed by the Crypto Workshop, written in Java. We received this as a 725,472 byte gzipped tar ball from www.cryptoworkshop.com/demtech/ximix.tgz. We also received the functional requirements for the mixnet [8].

- The EVC electronic ballot marker component developed by the VEC, written in Javascript. We downloaded this from https://bitbucket.org/vvote/vvote_evc.

- The VVA component that contains the packing code developed by the VEC, written in C#. We downloaded this from bitbucket.org/vvote/vvote_vva.

- The WBB component that contains the source files for the website where voters can verify their receipts or download the file of all votes and proofs of decryption. The code was developed by the VEC, and contains HTML and Javascript files. We downloaded this from bitbucket.org/vvote/vvote_wbb.
• Emails and supplementary documentation provided in response to questions we had and the preliminary version of this report.

2.2 Reviewing Methodology

We conducted reviews of the vVote’s design, code, requirements, and tests, focusing our efforts primarily on the code itself. In the following, we describe the methodology we followed for these activities.

2.2.1 Design Review

We reviewed the design documents to determine which algorithms are used in the different system components, how components communicate and synchronize, and the overall flow of information. Given that vVote is a system under development, parts of which are described in multiple design documents and scientific papers, our review task required us to determine which parts of which documents were actually relevant.

2.2.2 Code Review

We carried out structured, line-by-line reviews of substantial parts of the code base. Our objectives were to:

• check if the implementation conforms to the design;
• detect common kinds of programming errors;
• reconstruct intended invariants and determine whether they are satisfied or violated; and
• assess the code quality, e.g., to what extent the code is comprehensible and maintainable.

We expand on each of these below.

Design Conformance: Much of our code review efforts were centered around functional matching: we compared the source code with the design specified in the materials received to determine whether the implementation matches the design. Mismatches point to deficiencies either in the code or documentation. In cases where the specification is incomplete or ambiguous, we tried to infer the omitted information.
**Programming Errors:** We checked each line of code for common programming errors, such as null pointer dereferences, uncaught exceptions, unused return values, missing cases, and possible concurrency errors.

We supplemented our manual code review by using two code scanners: FindBugs and PMD. Both try to detect common programming mistakes while traversing the source code of a given system. FindBugs works on Java and PMD can analyze both Java and JavaScript. Note that such tools often over-approximate program behavior or take heuristics in diagnosing possible faults. They thereby produce voluminous output, including many false positives. We reviewed the most severe kinds of errors and performed spot checks of the rest.

**Check Invariants and other Behavioral Properties:** For each class reviewed, we attempted to reconstruct invariants and check their satisfaction. Each module is implemented in an object-oriented language. The hallmark characteristic of any such language is a class hierarchy that describe the structure of objects such as keys, messages, candidate lists, and permutations. One way to assess the quality of object-oriented software is to assess if and how objects and their respective attributes (data fields) are related and if this relationship is violated anywhere in the code. In computer science jargon, the relationship is called an invariant, because it must be maintained throughout the running of the system.

**Code Quality:** The style of object-oriented code is usually influenced by basic design decisions, for example, the choice of a particular programming pattern or the mapping of data onto the class hierarchy. We inferred and evaluated these basic design decisions.

**Requirements Review:** [4] lists 77 requirements for the system and its components. These include functional requirements, performance requirements, design requirements, and deployment requirements. We checked each requirement against the implementation, when we had sufficient documentation for verification. [8] lists 11 functional requirements and 6 other requirements.

**Test Review:** For each project, we reviewed the testing infrastructure and spot checked the provided test cases.

### 2.3 Scope

We reviewed the vVote system over a two week period. We focused our review on the parts of the implementation that have been written specifically for
the vVote system, excluding standard libraries. We also excluded extensions to nanoHTTP and the MongoDB library. For the remaining code base, we covered all of the components listed in Table 1 on page 12. We did not run any tests.

As the vVote system is based on algorithms that themselves are based on peer-reviewed papers, we did not reconfirm their correctness. We did however check (functional matching) whether the implementation corresponds to the algorithms. We emphasize though that strong, precise guarantees about the implementation’s behavioral properties can only be given by formal methods.

Finally, note that the software reviewed is part of much larger system whose design and proper functioning we could not check. This includes other software, hardware, and a numerous human-oriented processes. Their proper operation is essential to the security of the vVote system. We expand on this in §5.
3 Requirements

The SRS [4] contains the software requirements specification for the part of the vVote system developed by the University of Surrey. The document [8] contains the software requirements specification for the Ximix component developed by Crypto Workshop. Below we review the SRS document and afterwards the satisfaction of the individual requirements specified in SRS and mixSRS.

3.1 vVote Requirements Document

The SRS follows the IEEE Recommended Practices for Software Requirements Specifications [9]. The SRS provides a good overview of some of the central system components, their functionality, and interfaces. Below we comment on several areas for improvement. Additional lower-level comments are given in Appendix A.

**Scope:** The SRS describes the requirements for software developed by the Surrey team, focusing on the back-end and administrative software. As noted in the SRS (§1.2), this is only part of the overall system, e.g., it excludes the mixnet and different user interface software. Moreover, requirements should be given for hardware as well as software.

While we appreciate that different requirements have been defined by different teams, all system requirements should be given in a single document.

**Completeness:** Under the IEEE recommendations, all normative (i.e., prescriptive) requirements must be in §3 (Requirements). Any information provided in other sections or other documents is just informative.

Some of the supplementary papers, e.g., §6 of [7], state security properties that the vVote system should fulfill that are not listed §3 of the SRS. Examples include non-repudiation of failures, receipt freeness, and resistance to kleptographic attacks. If these are intended as requirements for the vVote system, they should explicitly be included in the listed requirements.

**Organization:** Requirements are given both at the level of the system components presented in §2.2 and the system as a whole. In particular, prop-
properties relevant for security are mixed in with other functional properties. We suggest a separate section for these requirements, analogous to what has been done for performance requirements.

3.2 vVote Requirements Satisfaction

We examined all the listed requirements. Here we discuss those requirements that we failed to verify or for which we did not have enough information to check. All other requirements not listed are satisfied.

Insufficient information: The following requirements could not be checked as they depend on information outside of the scope of the documents received, such as configuration data and the runtime environment:

IR1–IR2, SF2, PP3–5, PR24, PF1–8, C01, AT1–AT5.

For example, IR1–2 require CSV files which were not present in the sources we received; they should be added when the system is deployed, as part of the setup process.

Incomplete or violated:

• SF1 requires end-to-end verifiability. The vVote design achieves this in general. However, see the limitations discussed in §5.2.1.
• PR20 may be violated. See the comment on thresholds §4.
• PU5 should be updated: the vote receipt also contains the district number.
• IR4 is false: Ximix does not use JSON.
• AT7 requires that the software shall be written using Java 7. For the overall system this is false as some parts are written in Javascript and C#, e.g., VVA is programmed in C#.
• PP10 requires that all ballot printers have unique identifiers.
• PP13 is false. The method buildBallot in ballot_generation.js does not appear to check the authenticity of the received podResponse.
• PR16 is false. We interpret mixed votes as the votes that were mixed by Ximix. Ximix does not send the votes back to the privateWBB.
• PR22 appears to be true. It should be true by design and we have found no evidence contrary to this claim. However, there might be other external factors effecting it, the evaluation of which is beyond the scope of this report.
3.3 Ximix Requirements Satisfaction

We examined all listed requirements. Here we discuss those requirements that we failed to verify or for which we did not have enough information to check. All other requirements not listed are satisfied.

Insufficient information: The performance requirements could not be checked as they depend on information outside of the scope of the documents received, such as the runtime environment.

Incomplete or violated:

• Functional requirement 6. Mixing and decryption are not two separate actions. In the present version, one cannot shuffle without decrypting.

• Functional requirement 7: Because of the kind of mixing used (Randomized Partial Checking), no zero-knowledge proofs are generated during mixing but only during decryption.

• Other requirements (d): It appears that a standalone decryption library is included in the distribution, but it is disabled in the vVote deployment of Ximix. Decryption can only be executed from the mixnet Java Applet, after the shuffling of ciphers has completed.

Conclusions: The developers agree with our assessment, but point out that Requirements 6, 7, (d) are unattainable. We concur.
4 Software

The software system under review consists of nine components that together implement the vVote system, the mixing network Ximix, the electronic ballot marker EVC, and the vote administration system VVA. Figure 1 summarizes these components together with their vendor and the programming language they are written in.

The vVote system components are complex software artefacts that implement sophisticated communication protocols and mathematics, based on elliptic curve cryptography. During early voting, it is critical that the EVC component and vVote system interact flawlessly to precompute randomness, generate ballots, and record and store vote preferences. After the election, other parts of the system must interact flawlessly to guarantee the timely production of printed paper ballots that will go into the manual count. These include the VVA system that reads the public bulletin board of the vVote system, packs the votes, and sends them to Ximix for further anonymization, decryption, and unpacking.

In this section, we summarize the result of our code review. For detailed explanations and comments, please see the appendix.

4.1 Programming Structure

The programming style of the source code of the vVote system appears to be governed by the idiosyncrasies of the JSON data interchange format. The implementation is riddled by code fragments such as the following one.

```java
JSONObject response = new JSONObject();
response.put(MessageFields.TYPE, CommitProcessor.FINAL_COMMIT_MESSAGE_TYPE);
response.put(MessageFields.CommitSK2.HASH, myHash);
response.put(MessageFields.CommitSK2.COMMIT_TIME, peer.getDB().getCurrentCommitField(DBFields.COMMITMENT_TIME));
response.put(MessageFields.CommitSK2.PEER_ID, peer.getID());
```

This affects the readability and maintainability of the source code.
It is possible to achieve better code quality by using encapsulation: JSON objects should only be used for generating output and parsing input and should otherwise remain hidden from the developer and the code base. This means that the associated functionality of such a class includes serialization, deserialization, and validity checking.

Conclusions: Surrey agrees this is a good idea for long-term maintainability, but not essential for 2014. We concur.

### 4.2 Functional Matching

One of our main tasks was functional matching. We followed the vVote protocol as specified by the message sequence charts described in [3, 7] and tried to match specification and implementation. What made this task particularly difficult for the preparation of this report is that the documents we received were written over a large time span and described irreconcilable versions of algorithms and protocols at different levels of details. One of our challenges was therefore to determine which formulation of the specification to compare the implementation against.

#### 4.2.1 vVote System

Despite these challenges, we identified several places where the implementation and message sequent charts differ. We would like to emphasize, however, that such differences does not necessarily mean that the implementa-
tion is broken; it merely says it does not adhere to the specification. Here are some examples.

**WBB/AuditMessage.java:** The message sequent chart 4.6 on page 40 specifies that the printer sends a nestedly signed message to the Web bulletin board: \( \text{Sign}_{\text{Printer}} \{ \text{Sign}_{\text{WBB}} \{ \text{SerialNo}, \text{district} \}, \text{CommitOpening} \} \). The inner signed message is correctly checked by the implementation. However, the outer message signed by the Printer is not. In particular, the signatures are not nested, and there are additional fields checked in the implementation.

**WBB/CancelMessage.java:** The message sequent chart from [7] Figure 4.7, page 44 (see also C1 in [1] Figure 2.1, Page 10) specifies that cancel station sends a nestedly signed message to the MBB peers (third message): \( \{ \text{"Cancel"}, \{ \text{"cancel.1"}, \text{SerialNo} \}_{\text{sigCancelAuth}} \}_{\text{sigVPS}} \). Therefore the implementation should check two signatures. The implementation, however, checks two entirely different signatures:

\[
\begin{align*}
&\{ \text{"cancel"}, \text{SerialNo} \}_{\text{sigCancelAuth}}, \\
&\{ \text{"cancel"}, \text{SerialNo} \}_{\text{sigVPS}}.
\end{align*}
\]

Hence, the implementation does not implement the specification. Also note the change in notation. What was previously referred to as the printer is now the VPS. What was previously called the Web bulletin boards is now the MBB peers, and the concrete syntax for cryptographic messages has changed. Furthermore, the explanation of C1 on page 24 of [1] does not match.

**PrivateWBB/CommitR1Message.java**, and **PrivateWBB/CommitR2Message.java:** There is inconsistent naming between rounds for the WBB commitment protocols in [3]. For example, the flowchart in Figure 4.10, whose caption reads **WBB Round 1**, gives in fact an account of all three rounds described in Figure 1 of the posting protocol in [10].

**Conclusions:** Surrey will ensure that these are brought into alignment.

### 4.2.2 Ximix

We observed similar discrepancies when trying to match the algorithms and protocols implemented in the Ximix system and the relevant literature:

1. The computation of \( h \) when generating keys should be computed in a distributed way to guarantee that the discrete logarithm problem is suffi-
ciently hard to solve [11]. In class KeyGenerationCommandService, however, it is not computed in a distributed way. Even worse, instead of computing \( h \) directly, the implementation computes \( k \), which is the solution of the discrete logarithm problem. This way of generating \( h \) is only acceptable if there is a trusted key generation process. Otherwise it is in conflict with the system’s privacy requirements.

**Conclusions:** According to Crypto Workshop, the computation of \( h \) has been fixed in the current version of the system. They also point out, however, that this computation is not yet fixed for BLS key generation due to some issues with the JPBC implementation.

2. [11] requires that in case one commitment does not match, the broadcast complaint protocol is executed. It appears that implementation of the ECKeyManager does not implement this protocol but raises an exception instead.

**Conclusions:** According to Crypto Workshop, the class ECKeyManager does not implement the broadcast complaint protocol as described in [11] because it was not required by the original specification. All parties agree that this would be a useful improvement to be considered in the future.

3. The paper [12] describes two options to shuffle ciphertexts in Sections 4.3 and 4.4, respectively. It appears that the implementation of the method isPairingEnabled follows Scheme 2 – Pairwise Dependent Selections. We believe that it must distinguish between even-numbered and odd-numbered mix nodes. In the implementation, the commitments produced are mappings of input elements to output elements, i.e. of the form \( \Gamma_j^n \), regardless of whether the node is odd- or even-numbered (see Section 4.4 in [12]). As such, the implementation does not implement the algorithm as required.

**Conclusions:** Crypto Workshop claims that pairing is implemented correctly. However, we observe that the Ximix implementation differs from what is described in [12].

4. The implementation of computeProof appears to follow [14]. The non-interactive zero-knowledge proof of correct partial decryption consists of a non-interactive version (through the Fiat-Shamir transformation [15]) of a proof of discrete logarithm equality [16] between the public key share of the current node (see Section 3.1 in [14]) and the element \( xC \) (denoted by \( \beta_{xi} \) in the aforementioned paper). An interactive version of this proof is described in Section 3.2 in [14]. We observe that the implementation differs significantly from what is described in [14].

**Conclusions:** Crypto Workshop is aware that in the current version the zero-knowledge proof of correct partial decryption only provides evidence
that the node is in possession of its secret key share $x_i$. They will investigate this further.

4.3 Thresholds

Thresholds are used extensively throughout the vVote system: threshold cryptography and threshold agreements between peers. What value to assign as the threshold is a central decision that we trust will be chosen responsibly when configuring the system. There is one location in the code, however, that looks particularly suspicious because the threshold is set to the number of peers, and we were unable to convince ourselves about the justification for this.

In PrivateWBB/CommitProcessor.java, the checkCommitThreshold appears to require that all peers (not just the threshold) agree before any post to the public bulletin board. Since this seems to be stringent requirement, we consulted the formal specification of the commit protocol in [10], which did not help us clarify this question. Despite its formality, it was unclear to us when $t$ (threshold) versus $n$ (number of peers) was intended. Here is why. First, in Event-B it is customary to have a context specifying carrier sets constants and axioms about them. That is missing here. Second, indexed sets of variables are represented by functions in Event-B. In the machine specifications given, it is not always clear what the index sets (functions) range over, e.g., what is $p$ in $D_j, p$ in BBProt2? Is it related to $p_j$? Finally in the different models, $j$ always ranges between 1 and the threshold number of peers, not over the set $n$. While expressing behavior in terms of the behavior of a threshold number of peers may be fine when formalizing the ideal functionality of the protocol in the early models, and we understand that the Dolev-Yao adversary is used to represent a minority of dishonest peers, we had nevertheless expected to see the protocol specified in terms of $n$ peers, at least in the later models, so that it corresponds to the less formal description from §2 of [10].

In the Ximix key generator KeyGenerator, the threshold of peers (variable threshold of type int) is simply the number of available nodes minus 1. This seems to violate the design goal of fault tolerance.

Conclusions: For the PrivateWBB, we now understand that all the peers are involved since this protocol runs once per day and is not time critical, which allows time for manual intervention in the case of failure. Surrey will review this since the protocol is also correct when a threshold of peers participate. We agree with this and also that the situation is analogous for KeyGenerator.
4.4 Randomness

We observe a general weakness of the Ximix implementation in that it does not appear to use a secure source of randomness for computing encryptions, commitments, and reencryptions. The developers of the system seem to be aware of this fact. We feel strongly that this weakness should be addressed.

Conclusions: All parties concur.

4.5 Array Indexing

The Javascript code that implements the electronic ballot marker uses various zero-based arrays in the definition of the user interface components. This can lead to unexpected user interface behaviors if one-based indexes are used. Moreover, such software must deploy index transformations that clutter the code, are error prone, and difficult to maintain. At several places in the Javascript and C# code, we noticed the use of $+1$ and $-1$. Without formal methods, it is extremely difficult to check that they are used consistently. We therefore strongly encourage the developers to use consistently either one-based or zero-based indices.

4.6 Refactoring

There is code duplication throughout the entire vVote system, which is in part due to the overloading of methods or cutting and pasting code from one place to the other. The problem with code duplication is that it will make the code base difficult to work with and to maintain and should therefore be avoided. An example is the creation of the wbbSubFile during ballot generation, which is to be submitted to the private WBB peers for storage.

We also note that there is some code duplication between Ximix and the Surrey code. This is problematic as changes made by one team may not be noticed and accounted for by the other team. For example, there are two occurrences of the file that implements critical parts of the threshold encryption infrastructure LagrangeWeightCalculator.java.

```
find . -name 'LagrangeWeightCalculator.java' -print
./vVote_Release/components/tvsproject-vvotethirdparty-bcc5dee46ab7/
vVoteThirdParty/src/ximix/util/LagrangeWeightCalculator.java
./ximix/common/src/main/java/org/cryptoworkshop/ximix/common/crypto/threshold/LagrangeWeightCalculator.java
```
We also found code duplication where the duplicate code was subtly different, for inexplicable reasons. The following is an example of this.

*Client/BallotGenAudit.java*

```java
logger.info("About to submit {} to the MBB. Creating file {}", filename, submitOutput);
File wbbSubFile = new File(submitOutput);

// Create zip file and add submission to it
ZipOutputStream zos = null;
MessageDigest md = MessageDigest.getInstance(ClientConstants.DEFAULT_MESSAGE_DIGEST);
try {
    zos = new ZipOutputStream(new FileOutputStream(wbbSubFile));
    ZipUtil.addFileToZip(new File(filename), md, zos);
} finally {
    zos.close();
}
```

*Client/BallotGenMix.java*

```java
logger.info("About to submit ballot to MBB");
// Get a reference to the cipher file
File wbbSubFile = new File(outputFile);

// Prepare a zip file for sending the data to the MBB
ZipOutputStream zos = new ZipOutputStream(new FileOutputStream(wbbSubFile));
MessageDigest md = MessageDigest.getInstance(ClientConstants.DEFAULT_MESSAGE_DIGEST);
logger.info("Creating zip file");
try {
    // Add the ciphers file to the zip and close it
    ZipUtil.addFileToZip(new File(filename), md, zos);
    logger.info("Added {} to zip", filename);
} finally {
    zos.close();
}
```

The C#-code that implements the VVA system would also benefit from code refactoring, which would simplify code maintenance and review. For example, in the vote Decrypter, implemented in file `VoteDecryptBL.cs`, lines 760–945 repeat the same lines of code for packing votes for LA, ATL and BTL races multiple times.

We found similar instances of code duplication in the Ximix system. We only give here one example: the methods `generateCoeff` and `generateH`
in ShamirSecretSplitter and KeyGenerationCommandService, respectively, are variants of each other and implement the same functionality.

4.7 Fault Tolerance

The distributed nature of the vVote system aims to lessen the likelihood of system failure and to increase the trustworthiness of the implementation regarding confidentiality and availability. In particular, the private WBB, randomness servers, and mixnet servers are all designed to run as a distributed system. The public WBB will probably be replicated using traditional infrastructure.

We have the following concern about how fault tolerance might fail. The private WBB extends MongoDB by a custom-designed lightweight concurrent transaction system. Although we did not find any faults in the implementation at this stage, we would like to emphasize the brittleness of this approach. Even if it all works correctly now, it can easily be broken by future modification of the code base, resulting in live-locks or dead-locks. There are simply too many unstated invariants in the code to guarantee that it will work properly.

In general, concurrency should be handled with extreme care and home-made solutions should be avoided in favor of standard proven ones. We therefore recommend replacing the MongoDB database system by a database system that provides proper transaction management as required.

Somewhat orthogonal to this discussion, we remark there is missing functionality Ximix system that may cause the system to behave poorly. For example, the ServicesConnection class does not implement recovery actions when it detects a dead node.

Conclusions: We understand Surrey’s motivation for using MongoDB as well as their argument of why the system, as designed, should not have concurrency bugs. Our reservations still stand. Surrey will revisit this for future versions.

4.8 Exception Handling

Many components of the vVote system are implemented in Java. Java’s exception mechanism distinguishes between checked and unchecked exceptions, which means that the Java compiler does not check for unchecked exceptions. The components that do not handle unchecked exception set them-
selves are vulnerable to ungraceful crashes. In particular, SecurityExceptions
are hardly ever caught. This means that if the system is run on a node where
permissions are not properly set, it may crash with an Uncaught Exception.

4.9 Libraries

Some of the source code uses deprecated libraries, e.g., X509.extension
in Client/GenerateSignatureKeyAndCSR.java.

It appears that the Surrey team reimplemented parts of the elliptic curve li-
brary (uk.ac.surrey.cs.tvs.utils.crypto.ECUtils) that is al-
ready provided by BouncyCastle. In particular, the Surrey implementation
differs slightly from the BouncyCastle implementation in that it does not nor-
malize ECPoint. We recommend that the Surrey implementation uses the
libraries already imported.
5 Security

5.1 General Observations

As mentioned in the executive summary, the vVote system uses cryptography and distributed algorithms in innovative ways to build a highly secure and fault tolerant election system. By offering end-to-end verifiability, it laudably goes far beyond what one generally sees in other voting systems — be it conventional or electronic. Moreover, the use of cryptography is based on algorithms that have been developed and were rigorously analyzed and openly published in peer-reviewed venues, or are currently under review by the scientific community. We share the designers’ view that this is a good prerequisite for implementing a secure voting system.

Security goes far beyond the vVote system’s design and even its implementation. We highlight three aspects of this here. First, the vVote system is a system within a system in that it is part of a much larger voting infrastructure. This larger infrastructure is comprised of both human and IT processes, which interact with the vVote system, and their operation must be compatible with the design and assurance assumptions of vVote. For example, for voter privacy, it is essential that the printed candidate list is irrecoverably shredded immediately after the vote is cast and, moreover, voters cannot be observed or filmed while in the voting booths. Second, the vVote system runs on (or with), and therefore depends on, subsidiary systems and auxiliary components including conventional hardware, standard operating systems, protocols, and libraries. The security of vVote therefore depends on the security of these components (which, incidentally, nation states are skilful at compromising). Finally, vVote’s security depends on people and processes, for example, for system deployment, configuration, administration, updates, and patch management. Figure 1 gives two standard examples of such dependencies.

These are general observations that hold for any system, but they bear repeating nonetheless. They highlight the critical importance of having appropriate processes and controls. Analyzing these is outside of the scope of this project. However, as noted in §7, we recommend that more attention should be given to this in project management activities.
Build, deployment, and configuration: The right system must be built, deployed, and configured. This requires numerous control processes. For example, one must ensure the correctness, confidentiality, and integrity of distributed system configuration data (e.g., thresholds and keys).

System security: If the attacker can compromise the hardware or operating systems upon which vVote runs, then he can control the system. Preventing this requires processes for system hardening, network security, access control, operating-system patch management, training and screening of administrative personal, etc.

Figure 1. Examples of dependencies on people, processes, and other systems.

5.2 Specific Findings

5.2.1 Design

The design of the vVote system has been carefully analyzed in [7], supported by associated technical papers. This includes an analysis of the system’s privacy and integrity properties, as well as resistance to particular kinds of attacks. We are in agreement with the analysis in [7].

Despite our agreement, we wish to highlight two issues concerning verifiability. First, for the initial use of the system in 2014, vVote will run alongside traditional paper-based voting. As explained in §1.5 of [7], full end-to-end verifiability is not possible then, as STV tallying requires that the system processes all of the votes. Hence the publicly verifiable output from vVote must go into the paper count and one must trust the election observers who monitor the paper count.

The second issue is more subtle and concerns the very meaning of verifiable. If we interpret verifiability as meaning no trust assumptions then we do not have end-to-end or universal verifiability because there are algorithms, respectively computations, involved that the voter is not capable of verifying, respectively computing, himself.

As an example of this, consider what is required for a voter to check that votes are properly counted as cast. Votes are packed before they enter the Ximix network and they are unpacked when they come out after decryption, see [18]. Correct election results require that these functions operate correctly. For unpacking (which is not covered by the zero-knowledge proofs that
Ximix produces), this means that the Vote Packing Table must have been computed correctly. The point is this: even though all information for computing this table is public, and therefore the correctness of unpacking does not require any hidden knowledge, the user must still trust that this table has been computed correctly. Said another way, even though such a computation is verifiable from the cryptographic perspective, unless it is extremely simple (e.g., adding two numbers), the user cannot directly check it himself and must trust a program to do this.

We see no way around such residual trust assumptions. This is an issue that has also concerned, for example, the formal methods and theorem proving communities. How do you even trust a proof? In our example, one can have one or more programs (which the voter can download and inspect) independently construct the table or, even better, formally verify an algorithm for table construction. By doing this, one minimizes, but does not completely eliminate, trust assumptions. While we do not intend to start a philosophical debate here, we do wish to highlight that verifiability and trust are subtle notions, which are extremely relevant for evaluating design decisions and assurance requirements for systems like vVote. In general, increasing program complexity reduces trustworthiness, unless the computed results can be easily checked. Conversely, assurance activities like formal methods can be used to increase trustworthiness.

5.2.2 Implementation

**Integrity:** In general, it is possible to have a design that is secure and an implementation that is insecure. For vVote, end-to-end and universal verifiability provide a safety-net for integrity properties: users can check results independent of possible bugs in the implementation. There are some provisos here, as noted above: users must actually perform the checks and the checking itself must work properly. For example, for vVote, the VECverifier.exe file can be downloaded from the VEC voting server and executed by the public\(^1\) to verify the entire electronic voting run. But modulo these provisos, verifiability means the system (with help from the users) can ensure the integrity of the election results.

**Privacy:** Verifiability does not ensure privacy properties and the improper use of cryptography can reveal voter preferences. We did not find any such problems during our code review.

---

\(^1\)Note that if this executable’s development, deployment, and storage is not carefully managed, an adversary can corrupt it. This would not only allow him to violate end-to-end verification guarantees, he could corrupt and control the computers of Victoria’s constituency.
**Availability:** We believe the largest security risks arising from the implementation concern availability, i.e., the successful completion of different voting functions. Software errors could cause vVote components to fail or to yield results that cannot be verified and must be rejected.
6 Code Scanning

In this section, we describe the results of our code scanning activities using the code scanning tools FindBugs and PMD. We applied these tools to the vVote system’s source code, including Ximix.

FindBugs reported several hundred issues of varying degrees of severity. PMD reported several tens of thousands issues, but most of them were of a stylistic nature which we did not feel particularly strongly about, although we were in basic agreement with them. For example, many of the complaints addressed violations of the “Law of Demeter” and method chaining, which we agree with. We recognize that beauty in programming is in the eye of the beholder and thus we leave it to the developers to install the PMD tool and follow its recommendations where appropriate.

In the remainder of this chapter we report on the findings that we obtained using FindBugs. FindBugs runs as an Eclipse plugin, which makes it particular easy to check for bugs during software development. FindBugs produced many high-quality diagnostics and, in some cases, it even confirmed suspicions by the programmer. For example, the comment in file ExternalMessageProcess.java line 99 precisely mirrors the bug found by FindBugs and is described in (7) below.

In the following, we list the most critical bugs found by FindBugs (up to level 17). We reproduce the 18 error message below, along with FindBugs’s explanation, and the relevant program lines. While we did not run tests to confirm these, we recommend that the developers carefully consider all of them, in particular (7) on synchronization and (17) on random object creation.

1. Constructor invokes Thread.start()
   “The constructor starts a thread. This is likely to be wrong if the class is ever extended/subclassed, since the thread will be started before the subclass constructor is started.”
   [Client.java:226] [CacheDataFile.java:76] [CacheDataFile.java:95] [CommandListener.java:135]

2. Dereference without nullcheck
“The result of invoking readLine() is dereferenced without checking to see if the result is null. If there are no more lines of text to read, readLine() will return null and dereferencing that will generate a null pointer exception.”

[ProxyServer.java:233]

3. **Return value ignored**

“This method returns a value that is not checked. The return value should be checked since it can indicate an unusual or unexpected function execution. For example, the File.delete() method returns false if the file could not be successfully deleted (rather than throwing an Exception). If the result is not checked, one would not notice if the method invocation signals unexpected behavior by returning an atypical return value.”

[BallotGentAudit.java:210] [BallotGenMix.java:304] [Client.java:593] [TempMergeFile.java:83, 117] [TableBuilder.java:134] [ProcessKeyFiles.java:240] [RandomnessWorker.java:140] [RGKeyGeneration.java:106] [PublicWBB.java:134] [CommitMessageThread.java:186, 352] [PublicWBBKeyGeneration.java:68] [CommitProcessor.java:321] [WBBPeerKeyGeneration.java:83] [XimixNodeContext.java:291] [VVoteNanoHTTPD.java:128] [FileBackedTimeoutManager.java:76] [KeyGeneration.java:262] [TVSKeyStore.java:169] [IOUtils.java:154,353] [BoundedBufferedInputStream.java:134] [ClientCommandService.java:1124]

4. **Use of Object.equals**

“This class defines a compareTo(...) method but inherits its equals() method from java.lang.Object. Generally, the value of compareTo should return zero if and only if equals() returns true. If this is violated, weird and unpredictable failures will occur in classes such as PriorityQueue. In Java 5 the PriorityQueue.remove method uses the compareTo() method, while in Java 6 it uses the equals() method.”

[CandidateCipherText.java:97] [RemovedCipher.java:59] [AdapterInfo.java:66]

5. **Write to static field**

“This instance method writes to a static field. This is tricky to get correct if multiple instances are being manipulated, and generally bad practice.”

[Client.java:213]

6. **Possible null pointer dereference**

“A reference value which is null on some exception control path is dereferenced here. This may lead to a NullPointerException when the code is executed. Note that because FindBugs currently does not prune infeasible exception paths, this may be a false warning.”

[Tablebuilder.java:253] [KeyGeneration.java:439] [MainFrame.java:313]
7. **Synchronization performed**

“This method performs synchronization on an object that is an instance of a class from the java.util.concurrent package (or its subclasses). Instances of these classes have their own concurrency control mechanisms that are orthogonal to the synchronization provided by the Java keyword synchronized. For example, synchronizing on an AtomicBoolean will not prevent other threads from modifying the AtomicBoolean.”

[ExternalMessageProcessor.java:100, 101]

8. **Dead Store**

“This instruction assigns a value to a local variable, but the value is not read or used in any subsequent instruction. Often, this indicates an error, because the value computed is never used.”

[BallotGenVerifier.java:260] [BLSKeyManager.java:173, 238, 239] [RemoteServicesCache.java:278] [InstallerConfig.java:122] [CommandApplet.java:1507] [ClientCommandService.java:665]

9. **Exception is caught when Exception is not thrown**

“This method uses a try-catch block that catches Exception objects, but Exception is not thrown within the try block, and RuntimeException is not explicitly caught. It is a common bug pattern to say try ... catch (Exception e) something as a shorthand for catching a number of types of exception each of whose catch blocks is identical, but this construct also accidentally catches RuntimeException as well, masking potential bugs.”

[SendErrorMessage.java:163] [NodeShuffledBoardDecryptionService.java:322] [BoardHostingService.java:705] [CommandApplet.java:980, 1540] [ConsoleConfigFactory:66] [ClientCommandService.java:818, 1107] [NodeServicesConnection.java:278] [SignedDataVerifier.java:82, 118]

10. **Null passed for nonnull parameter**

“This method call passes a null value for a nonnull method parameter. Either the parameter is annotated as a parameter that should always be nonnull, or analysis has shown that it will always be dereferenced.”

[XimixNodeBuilder.java:79]

11. **May fail to close stream**

“The method creates an IO stream object, does not assign it to any fields, pass it to other methods that might close it, or return it, and does not appear to close the stream on all paths out of the method. This may result in a file descriptor leak. It is generally a good idea to use a finally block to ensure that streams are closed.”
12. **Does not call super.clone()**

“This non-final class defines a clone() method that does not call super.clone(). If this class ("A") is extended by a subclass ("B"), and the subclass B calls super.clone(), then it is likely that B’s clone() method will return an object of type A, which violates the standard contract for clone().”

13. **Result of integer multiplication cast**

“If the multiplication is done using long arithmetic, you can avoid the possibility that the result will overflow.” [ZipUtil.java:63]

14. **Possible shutdown of the entire virtual machine**

“Invoking System.exit shuts down the entire Java virtual machine. This should only be done when it is appropriate. Such calls make it hard or impossible for your code to be invoked by other code. Consider throwing a RuntimeException instead.” [Installer.java:69]

15. **Invocation of toString on lookup**

“The code invokes toString on an array, which will generate a fairly useless result such as [C@16f0472. Consider using Arrays.toString to convert the array into a readable String that gives the contents of the array. See Programming Puzzlers, chapter 3, puzzle 12.”

16. **Boxing/unboxing to parse a primitive new**

“A boxed primitive is created from a String, just to extract the unboxed primitive value. It is more efficient to just call the static parseXXX method.”

17. **Random object created and used only once**

“This code creates a java.util.Random object, uses it to generate one random number, and then discards the Random object. This produces mediocre quality random numbers and is inefficient. If possible, rewrite the code so that the Random object is created once and saved, and each time a new random number is required invoke a method on the existing Random object to obtain it.

If it is important that the generated Random numbers not be guessable, you must not create a new Random for each random number; the values
are too easily guessable. You should strongly consider using a \texttt{java.security.SecureRandom} instead (and avoid allocating a new \texttt{SecureRandom} for each random number needed).” [ServicesConnectionImpl.java:98]

18. \texttt{Thread.sleep()} called with a lock held FindBugs says “This method calls \texttt{Thread.sleep()} with a lock held. This may result in very poor performance and scalability, or a deadlock, since other threads may be waiting to acquire the lock. It is a much better idea to call \texttt{wait()} on the lock, which releases the lock and allows other threads to run.” [NodeServicesConnection.java:165]
7 Project Management

The PMP document [2] describes project management objectives and processes for producing the vVote system. In Appendix C we provide several detailed comments on the plan. Here we describe the main areas requiring either updating or improvement.

Timing: The project time schedule (§1.6 and Figure 3.2) must be updated. In particular, the project is behind schedule both for software and for documentation and this in turn delays quality assurance activities. Missing software cannot be reviewed and it delays all integration, load, and soak testing, which are essential prior to the actual use of the vVote system. Missing or misaligned documentation make software reviews difficult or impossible (see next point).

Project Documentation: §1.5 of the PMP lists the 6 project documents. It is critical that these documents are mutually consistent and consistent with the implementation. We have given numerous examples in §4 where this is not the case. Moreover, to the extent that they are not self-contained, they must unambiguously state which (parts of) other documents should be consulted, i.e., what constitutes the project documents?

Although documentation may appear to be a secondary concern, we nevertheless strongly recommend improvements here as it directly impacts implementation, quality assurance, and maintenance. In its current state, we found it difficult, and in some cases, impossible, to match parts of the implementation with the received design documents. Moreover, in the existing documentation, essential requirements and design aspects are not given in the 6 project documents, but rather in supplementary material like [7].

As a side remark, note that should certification under Common Criteria ever be desired, this would also necessitates complete, consistent design documents matching the implementation.

Conclusions: Surrey has acknowledged this and will correct ambiguities and inconsistencies.
Development Dependencies: The project depends on separate developments by different teams. The Surrey code relies on code implemented by Crypto Workshop in the Mixnet (e.g., LagrangeWeightCalculator) and the operation of the Mixnet. Moreover the VEC is developing in house components like the Ballot Marker and table building and packing code. The PMP §5.1 states that progress will be monitored by weekly meetings held with the project team; however, it is not clear if this team is just the Surrey team or all project participants. Clearly project activities must be closely coordinated between all development teams.

Associated Processes: Some critical activities were not listed in §3.2 of the PMP. In particular, it is unclear where administrative processes will be defined and how they will be documented and audited. Management of cryptographic keys is one example of this. [5] describes where keys will be stored. However, trusted individuals must be involved to generate them, to serve as a CA to sign public key certificates, etc. Another example, even more banal, is defining and documenting procedures for handling and recovering from error messages produced by the vVote system.

Conclusions: We received additional documentation of these processes.
8 Testing

The system test plan [6] describes how testing will take place for the back-end of the vVote system. The test plan on the whole is sensible and hence we have only minor comments in this section.

The approach taken to unit testing is systematic and follows a good methodology. CodePro is used to generate empty test cases that are manually completed to define preconditions, execute the method under test, and test the outcomes using assertions. Coverage is checked with EclEmma. We manually inspected a selection of test cases by hand, and the test case selection was reasonable.

One of the objectives of integration testing is to find faults resulting from the interaction of different modules, such as interface problems or unanticipated side-effects. The test plan presented, however, is limited to testing the subsystems developed at the University of Surrey and excludes Ximix. It is essential that tests are generated involving all system components.

Another objective of integration testing is to test performance and reliability requirements. The scenarios described in §3.3 do a good job of this, however it appears that PF3–PF5, concerning the minimum numbers of regions, districts, and candidates, have been neglected.

Finally, we note that testing’s benefits (the ability to detect faults or provide confidence for their absence) depend of course on the quality of the test cases selected. When testing performance and reliability, test case selection is particularly difficult as the system must be extensively tested under different representative loads and configurations. The results will also depend on the timing of different events, which is difficult or impossible to control for during testing. The Project Management Plan (pg. 16) leaves only ca. 3 weeks for full-system testing. This seems a bit tight. Moreover, it will be insufficient if testing uncovers problems that require redesign and reimplementation and the development waterfall then degenerates into a development spiral.
9 Conclusions and Outlook

The vVote system is a well-designed, state-of-the-art voting system, whose design is based on distributed cryptographic algorithms that provide strong privacy and integrity guarantees. In our review, we have focused primarily on the system’s implementation. Figure 2 summarizes the most critical issues we encountered and our recommendations for addressing them.

- **vVote**: Ensure that message sequence charts and protocol implementation match.
- **vVote**: Use JSON as a data-interchange format only. Serialize and deserialize as Java objects.
- **vVote**: Replace MongoDB with a database that offers transaction management.
- **Ximix**: Ensure that the implementation of non-interactive zero-knowledge proofs matches the algorithm described in [14].
- **VVA**: The decryption service, which prints the completed ballots for inclusion in the manual count, is particularly critical as it falls outside of end-to-end verifiability. Consider the application of formal methods for higher assurance here.

Figure 2. Most critical issues and recommendations.

Ideally, the vVote system will have a long lifetime where it evolves from a system running alongside paper-based voting to one that eventually replaces it. This requires further engineering to support change. Future changes may be in the requirements, the underlying technologies (cryptographic primitives, protocols, databases), and even the development teams. Here we make some general recommendations that may help pave the way for future developments.

**Traceability.** vVote is a sophisticated system, but it is also a complex one. Its complexity stems from:

1. the complexity of the different scientific papers, cryptographic algorithms, and protocols that underpin its design;
2. its distributed design; and
3. complexity within the implementation itself.

To keep this complexity manageable as the system evolves and its developers change, it is essential that these three parts are always synchronized. This means the design document must precisely state the algorithms to be used (not merely, which set of algorithms it is based on), and the code must clearly follow the design. It should always be possible to track all algorithms (and requirements) through both the design and code. As noted in the report, this was not always possible in the current state. Note that clearly documenting such correspondences is important too, should certification under Common Criteria ever be desired.

**Diversity.** We recognize that the vVote system is intended to be used in conjunction with randomness and WBB peers that are produced by third party developers (parties and other stakeholders). While this is a laudable goal, we doubt this can be done before the November elections. It is nontrivial to write a client that implements all the required protocols and the documentation does not adequately support this. This means that, in the short-term, vVote's distributed architecture adds considerable complexity but has questionable gain. Given that the servers are all mono-culturally written, they will likely run on identical machines, and they will all likely be administrated by the same person, it is difficult for us to imagine an adversary who can compromise only some, but not all, of the servers. This observation holds for the randomness servers, private WBB peers, and mixnet peers.

**Formal methods.** Traceability and functional matching require detailed system models. The Project Management Plan [2] calls for modeling using languages like UML. The consistent use of such semi-formal modeling languages would definitely help with these tasks.

The goal however, is to match behavior, not just structure. For this, there is no alternative to the use of formal methods. The application of formal methods is increasingly common, especially for critical systems, and we would categorize the vVote system as such a system. Note in this regard that end-to-end verifiability provides a safety net for integrity, but not against the system throwing unexpected exceptions, or delivering results that fail to check. Formal methods could be used to rule out such behaviors.

A light-weight starting point would be to document methods and interfaces with contracts, e.g., pre-conditions, post-conditions, and invariants. We often found ourselves in need of these when doing functional matching. In the long
run, one would ideally use verification tools such as extended static checkers, software model-checkers, or theorem provers, especially for the more difficult parts or those parts falling outside the scope of end-to-end verification guarantees.
References


Appendix A: Requirements Review

The following are specific comments on [4].

• The product perspective should be updated and synchronized with the Technical Report [7] and the High-level Design Specification [1]. Examples include: differences in the description of the ballots, whether the PoD printer and Audit Station are separate or together, missing procedures in the product perspective such as no description of cancellation, etc.

• The constraints listed in §2.5 are not constraints in the sense of [9]. In particular, constraints should describe items that limit the developers’ options. The first paragraph contains high-level requirements for securing and administrating the platforms and networks. The second paragraph contains assumptions about appropriate procedures.

• The assumptions and dependencies listed in §2.6 are not assumptions and dependencies in the sense of [9]. Statements about following coding standards, open source software and keys, can all be turned into requirements. Indeed, most of these are requirements already.

• Requirement PR18 is vague. A comparison period should also be listed.
Appendix B: Code Review

B.1 EVC Component

We have reviewed the `vvote_evc` system. The main code base of the system is written in Javascript. We walked the code and report on all observations below.

While we did not find any serious problems in the source code, the code would nevertheless benefit from comments on the intended meaning of arguments passed to methods. We would like to encourage the programmers to add those, to make the code more readable and classes more self contained.

`above_line_council_ballot.js`:

- Order in which methods are defined are different from the order in `Ballot`.
- Code duplication in `reset` and `init`.
- `shuffleOrder` and `partyGroups` are not reset during `reset` or `init`.
- Missing invariant on method `setPartyGroupListing`. What type is `data`? We presume an array of partylists.
- `groupBallotPosition` is an array that is also offset by 1 from the information found in `data`. This may be a source of bugs.
- `getPartiesInShuffleOrder` is not declared in abstract class `Ballot`.
- Class invariant missing for `shuffleOrder`. Without this information, it is difficult to see if the `index += 2` in `getPartiesInShuffleOrder` is correct. It looks like `shuffleOrder` is a string, with two characters at position `n` and `n + 1` (n even). We remark that the system cannot be used for contests with more than 99 parties without changing the software.
- In [4], Section 2.2.1, the CVS files also specify district and region, which seem to be absent from the code.
- in `getPartiesInShuffleOrder`, it is possible that the returned object named `parties` is not updated in the case that the party in question is ungrouped or does not have a ballot box. The relevant information stems directly from the `data` object passed as arguments to
setPartyGroupListing. It does not appear that the present function
conducts any consistency checks.

• The method getCandidateByShuffleOrder reuses code from the
previous method getPartiesInShuffleOrder. We suggest to up-
date the code so that ballot receipt becomes its own class. This would
reduce the amount of code duplication.

• getGroupByBallotId and getBallotBoxId are simple iterative
search procedures through all partyGroups, which retrieve a group
identified by some BallotBoxId or the BallotBoxId from group, re-
spectively. The control structure of this method uses the exception mech-
anism to recover in case that a key was not a valid partyGroup. It is un-
clear how this could happen. Also the mode of error recovery is dubi-
ous because it simply skips party groups. We suggest removing the try-
catch construction and raise an exception with a proper error message.

• selectGroup implements that a voter can only vote for one group,
which is consistent with the explanation in Section 2.1 of [4].

• swapSelections seems to assume that if this.groupSelection
is not equal to group1, then it is equal to group2. We cannot judge if
this is really the case. If ballotSwapped in this file is the only other
method that swaps these, then it relies on the invariant that
ballotOptionSwapped is only called when the two are not equal.

• isinformal and other methods of this kind could be written more suc-
cinctly as simply return this.groupSelection == null.

• isSelected and isLastGroupSelection seem to be identical func-
tions. Why are they both there? Why are they not identical? Why is not
one calling the other?

• setCurrentGroupPosition. Avoid headaches and transform the
code into everything zero-based or everything one-based.

• moveToNextGroup. If maxCouncilGroups is one-based and, more-
over, this.currentGroupPosition is zero-based, then it seems
that the condition of the fourth line of this method is ok. The use of
++this.currentGroupPosition in an array lookup may be elegant,
but it is difficult to reason about and hence error prone. It would be help-
ful if there were more invariants available.

• moveToPreviousGroup is dual to moveToNextGroup, except it does
not rely on nextGroup being ungrouped. Here too, it would be helpful to
see more invariants.

• getCurrentGroup may have omitted an index shift?
• **goToLastSelected Candidate** is a search procedure without exception handling. This is fine, but please state the assumed invariant: If \( \text{this.getSelection}() \neq \text{null} \) then there must be an index \( i \), such that \( \text{this.partyGroups}[i].\text{getBallotBoxId()} == \text{this.getSelection}().\text{getBallotBoxId()} \).

• The function **getPreferenceNumber** does not type check. Either it returns a number or a string. To avoid problems down the line, we suggest that methods always return objects of the same type.

• **getPreferencesInShuffleOrder** has duplicate code; see also getPartiesInShuffleOrder and getCandidateByShuffleOrder above. Note too that the preferenceList contains precisely 1. The last loop in this method is buggy. Namely

```javascript
for (var preferenceIndex = 0; preferenceIndex < ticketedGroupCount; preferenceIndex++) {
    if (preferenceList[at1ShuffleOrderArray[preferenceIndex]] == null) {
        preferenceList[at1ShuffleOrderArray[preferenceIndex]] = null;
    }
}
```

is virtually a NOOP. This code cannot be intended.

• **getTextualPreferencesInShuffleOrder** has duplicate code. Moreover, the use of `instanceOf` is bad style and the data structures could be engineered in such a way that this is unnecessary.

• **getShuffledPositionInPreferenceOrder** again has duplicate code. The preceding comment seems to imply that there is more than one preference. The body only defines `array[1]`. The behavior

```javascript
if (this.getHighestSelection() > 0) {
    var currentlySelected = this.getSelection();
    this.unselectGroup();
}
```

is not described in the preceding comment. Why is this needed?

• **checkForTimeout 2000** is not mentioned in the requirements.

**ballot_generation.js**

• In **createQRCode**, the QR code error correction level is set to code.ERROR_CORRECTION.L, contradicting the comment.

• The number of argument to **CandidateRow** does not match what is described in the comment.
• In several `addToPDF` methods, the `CellPaddingX` is computed in closely related ways. We suggest refactoring the code to remove redundancies.

• `content instanceof Array` requires reflection! The `else` branch kicks in, even if `content` is not a `String`. This can be avoided by splitting `HeadingRow`, `ArrowRow`, or `SubHeadingRow` into two respective functions.

• In `HeadingRowUnicode`, `content` is just a `String`. We suggest fixing the comment.

• In `isUpperCase`, space and comma are considered to be lowercase. Why?

• In `getExactWidthText`, many parameters are hardcoded. It would be better to put them into a configuration file to make them explicit.

• In `buildBallot`, what are the types of the arguments? All strings? We note that search algorithms for district and region are brute force. We suggest considering alternative data structures such as redblack trees or hash tables.

• Population of `laCandidates`, `shuffledAtlCouncilCandidates` and `shuffledBtlCouncilCandidates`: Not clear that the sanity checks are complete. It is critical that they are, otherwise informal ballots may not be recognized as such. We suggest adding the completeness argument in the comments. Nationalization: "english" is hardcoded, which should be avoided.

• In the functions relating unicode, we did not check the document elements, such as `foreign` and `foreign2`.

candidate.js

• Write an invariant for `groupPosition` and `ballotPosition` in a comment. It will help the reviewer of this code tremendously.

party_group.js

• The method `addCandidate` does not use an if-then-else. It should.

• Positions in the array `ccandidates` and the position stored within a candidate differ by 1. This may introduce bugs. We suggest using position indices uniformly throughout the code.

voting_session.js
• getUncontestedMessage: note that we are not looking into Ajax.

• In setMaximumNumberOfAssemblyCandidates, we assume that this is the real number of candidates and need not be index transformed.

• setVote and getVote. Class invariants? What kind of object is data?

B.2 vVote Client

Client/Client.java

• Stylistic: Subclassing is preferred over a switch over client types.

• The private key is saved in a file called CRYPTO_KP_FILENAME. This requires access control.

• In generateCryptoKeyPair, the spelling CRYPTO incorrect.

• In getJSONPublicKey, there is an error message misnomer. It should be “public key”.

• In getJOSONPublic, give the design rationale behind the two keys identified by SIGNING_KEY_FILENAME and SIGNING_KEY_ALIAS.

• importSSLCertificate, initComms, and importCertificates in key generation use hardwired "" as a password.

Client/BallotGenAudit.java

• If the audit fails, the exception message is unclear.

• We miss a specification of what should be audited.

• The method doAudit is misnamed. It promises to do the audit, but it only extracts the relevant randomness and stores it into a file. Such mislabeling is misleading and will be a source of frustration for anyone who has to understand or maintain this code.

• submitAuditToWBB: Choice of variable name filename could be more informative, e.g., auditfile.

Client/BallotGenMix.java

• Possible attack: If you can modify the zip files (RandomnessFiles) you control the randomness. Are the commitments enough to save us?

• Call to verifyAllFilesAndSignatures is critical code.

• Stylistic: JSON Object called combinedRandomness as opposed to the class CombinedRandomness.
- `generateBallots`: unchecked `SecurityException`.

- In `submitAuditToWBB`, the client will audit a fraction of the generated votes stored on the private WBB peers. It does this by interacting with the private WBB peers through `submitAuditToWBB` requesting a signature to check. The message sequence chart 3.2 describes this auditing, but does not mention signing or checking of signatures.

`Client/CandidateCipherText.java`

- “my” is confusing. Variable `my` versus `m × y`.

`Client/CandidateId.java`

- `CandidateCipherText` is missing an invariant on what the newly generated ciphertexts are.

`Client/CombinedRandomness.java`

- How is `keyPair` stored?
- Invariant with sizes needed. Invariant:
  ```java
  this.randFiles.getRandomnessFiles().size() = this.randFiles.getCommitFiles().size() = this.randFiles.getAESKeys().size()
  ```
- `CombinedRandomness`: If you create a new buffered reader, check immediately if it is `null`, not just upon closing the function in `closeAllFiles`. Otherwise there is a risk of null pointer dereferencing in `findSerial`.
- Check for `SecurityException` when handling files.
- `findSerial`. The variable `searchInProgress` is not needed.

`Client/RandomnessFiles.java`

- `verifyAllFilesAndSignatures` specification. We note that in the case that a randomness server sends randomness twice, the exception is raised independent of whether you have enough randomness.

- `if (peerFiles.keySet().size() >= clientConf.getIntParameter(ClientConfig.RANDOMNESS_THRESHOLD))` does not check if `getIntParameter` returns `-1`.

- Invariant: We note that the variables `configs`, `aesKeys`, `commitFiles` and `peerIDS` must always be in sync. Spell out invariants.

- Explain which kinds of failure modes are used throughout the implementation and when they are used. For example, `Exception` versus `Boolean`
versus –1. If you convert from Boolean to Exception, make sure that all exceptions are handled.

- The principal “WBB” is identified by a string. This may cause problems if misspelled or miscapitalized.

- `findRandomnessFiles` may raise a security exception if the read permission is not present.

- `reset`. Unlike the other arrays, `configs` is not cleared.

`Client/RandomnessRow.java`

- The code uses invariants in different (ambiguous) ways: sometimes invariants are clear and taken advantage of, and other times they are checked dynamically in the code. For example, `row.getSerialNo()` may return `null` (see `nextRow`).

B.3 vVote Randomness Server

`RandomnessServer/RandomnessServerUI.java`

- Different styles are used for configuring the system. Make system configuration uniform and as simple as possible.

- The constructor for `RandomnessServerUI` calls `super()`, which is the constructor of class `Object`. Why?

- The use of `equalsIgnoreCase` in `processMessage` is strange: Server commands and their spelling should be consistent throughout the system, especially if the commands are generated by the programs.

- `getFailures` case does not appear to be documented.

`RandomnessServer/ProcessKeyFiles.java`

- The name of the printer is communicated to the randomness server as part of a filename. This file is designed to store the printer’s public key. How do you guarantee that the printer names are unique? We know it is a requirement, but we wonder how this is actually implemented.

B.4 vVote Private WBB

`PrivateWBB/CommandListener.java`

- Explicit type casting (unnecessary).
• Why only logging messages from classnames and package names starting with “uk”?

• In the case for CLI_SHUTDOWN, the logic is opaque: Shouldn’t there be an else for this.peer.shutdown(true). Moreover, see the comment in PrivateWBB/WBBPeer.java shutdownExecutorService.

PrivateWBB/AuditMessage.java

• Precondition: Message not reduced yet.
• Inconsistent use of methods: this.id versus this.getID().
• Stylistic: (!(cipherRecord == null) && podMessage != null).
• The array index should be checked to avoid out of bounds exception:

```java
fullPerms[ballotReductions.getJSONArray(raceCounter).getJSONObject(i)
        .getInt(MessageFields.PODMessage.BALLOT_REDUCTIONS_PERMUTATION_INDEX)]
        = null;
```
• getExternalSignableContent does not include commitTime despite comment. Compare with getInternalSignableContent.

PrivateWBB/BallotFileMessage.java

• preprocessAsPartofCommitR2: It appears that the logger error message reports errors in MixRandomCommit. It should say preprocessAsPartofCommitR2.

PrivateWBB/CancelMessage.java

• In processMessage, it is not clear how there could be more than one message with the same id.
• Array and for loop unnecessary.
• Incorrectly duplicated comment in performValidation about cancellation machine.

PrivateWBB/CommitR1Message.java

• processMessage: What is the commit time?
• Multiple calls to getCurrentCommitHash returns a string that is used nowhere. It appears that this method simply checks if the CommitHash exists.

PrivateWBB/CommitR2Message.java
• importCommitR2Messages  It is important that msgs contains message objects of subclasses of JSONWBMessage. Otherwise an exception is raised. This is an important invariant that can easily be broken.

• Constructor calls super() from object. It appears that this call is unnecessary.

• parsePeerMessage: Laudable comment: No class conversion good.

PrivateWBB/MixRandomCommitMessage.java

• It appears that signatures are verified in MixRandomCommit with the class specific methods validateSenderSignature and validateSenderSignatureWithLocalDigest overriding the methods declared in FileMessage.java. This indicates that there are possibilities for refactoring the code.

PrivateWBB/PODMessage.java

• getInternalSignableContent: When comparing the message described in [7] in Figure 4.5 where the Printer sends a request for authorization to a peer, it appears that the implementation checks for a message that also contains the commit time. This is an inconsistency.

• It appears that in the method validateBallotReduction, the implementation checks for condition peer.getConfig().doLivePODCheck(). Note that according to Figure 4.5, such a check is mandatory. Furthermore, following the chain of definitions, it appears that doLivePODCheck is set to false.

PrivateWBB/StartEVMMessage.java

• We could not properly review this file because neither a message sequence chart nor another specification could be found. This message is insufficiently documented. It appears that comments and implementation are out of sync. For example, the comment speaks about a constructor CancelMethod, while the constructor is for StartEVMMessage.

• We observe that the message describes \{Type,SerialID,District\}_{Sig(EBM)}, which is different from the one specified by the message sequence chart.

PrivateWBB/VerifyPreferences.java

• PrefsVerification looks fine, but document the handling of the String “ “ in the two methods checkSinglePref and checkPrefs.
**PrivateWBB/VoteMessage.java**

- **init**: The for loop iterates over an array of races but only works for three. The use of cashing should be documented.

- **performValidation**: In the vote submission protocol [3], Figure 4.6, the EBM sends five fields, the fifth being an EBMID. In the code, we see all fields being sent except for EBMID. Moreover the StartEVM signature is checked in the code, which is not documented in [3].

- **validateEBMSignature**: The use of explicit package identifiers inside the code is stylistically questionable. We recommend to import this explicitly as in *Library/VerifyPreferences.java*.

- **validateStartEVMSignature**: The purpose of this undocumented code is unclear. In particular, what is the logic behind the if-then-else? The if part chooses to check a signature with SK2, whereas the else uses a BLS key. Please update the documentation.

**PrivateWBB/WBBPeer.java**

- **shutdownExecutorService** It appears that waitForFinish should be negated in the first if.

- Notion of truststore is not documented.

- 1000 is hardwired in the creation of the timeout manager.

- Boolean return value of mkdirs is ignored.

- **verifyConnection**: We had expected X509 certificates to be used but did not see them.

- **main** Could add a default config filename.

**B.5 vVote Table Building**

**Tablebuilding/BinarySearchFile.java**

- The method binarySearch follows the standard implementation of binary search. It is notoriously difficult to implement binarySearch correctly if the midpoint position is not clearly computed. We see a potential problem here because the correctness of the implementation depends on if (this.previousEnd - this.previousStart) / 2 is rounded up or down.

**Tablebuilding/PagedMappedByteBuffer.java**
• This class is used during binary search. There are several uses of integer division that may or may not round in the expected direction.

Tablebuilding/ByteMergeSort.java

• The size of `inMemoryByteThreshold` is hardcoded. Use configuration files like in the rest of the vVote system. Also, why not use a standard off-the-shelf library implementation of merge sort?

Tablebuilding/DuplicateException.java

• The `serialVersionUID` is hard-coded.

Tablebuilding/PermutationWorker.java

• `call` implements the table-building algorithm according to Section 5.4 in [18]. We observe that the implementation does not directly correspond to the informal algorithm, which makes functional matching hard.

• The juggling with indices and `allButlast` and `last` makes it nearly impossible to check compliance. This code be simplified for better readability.

Tablebuilding/TableBuilder.java

• `createSampleCandidates`: The code creates sample candidates at the beginning of a table-building run. The code is therefore not production ready and should not be considered for deployment in its current form.

• `TableBuilder`: The method `buildTable` uses one-based indexing for blocks and zero-based indexing for datafiles and blockSizes. This opens the implementation up for future maintenance problems.

Tablebuilding/TVSPermutationGenerator.java

• The code to generate all permutations of an array is convoluted and may contain errors regarding generating all permutations. If the table is incorrectly generated, because entries were not generated for some permutations, then votes may not be packed or unpacked. The table is a single point of failure. After its creation, the `TVSPermutationGenerator` returns an invalid permutation. Only in the case that `min = max` will the `getNextPermutation` correctly loop back to the first declaration.

Tablebuilding/VerifyBuiltTable.java
• Also in this file, preferences and candidates must be shifted by one to compensate for zero-based indices. This is worrisome for production code.

B.6 VVA Component

VVA/BtlVoteDecrypter.cs, VVA/AtlVoteDecrypter.cs and VVA/DistrictVoteDecrypter.cs

• GeneratePDF: No documentation was provided about the expected ballot form layout, the order of parties and candidates. This made functional matching with the implementation impossible. Nevertheless, we have some high-level remarks.

This code is highly critical for the vVote system since it runs outside the process that we associate with end-to-end verifiability. Under the assumption that there are no observers that compare the printed pdf files that are produced by VVA, these three files present a single point of failure for the integrity of the entire election. It is in the vVote project’s best interest that this code is checked mechanically against its specification.

Reviewing the code with this kind of rigor in mind, we note many issues in the code that a mechanical verification system would complain about. We mention a few. First, without explicit invariants that state that parties in ContextParty are pairwise distinct, the verifier would not convince itself that party is not uniquely determined. Second, it appears that the ballot form will be populated in some arbitrary order, which might violate the requirements. Third, and more serious, the relationship between the indices \( j + 1 \) and \( i \) and \( k \) in line 144 of VVA/BtlVoteDecrypter.cs must be made explicit so that the verification system can reason about the relationship between line 143 and line 144:

```csharp
contextCandidates.Single(c =>
    c.Id == fileItem.VoteDecryptRows[i].Votes[k].CandidateId)
```

and

```csharp
fileItem.VoteDecryptRows[i].Votes[k].Vote.HasValue
? fileItem.VoteDecryptRows[i].Votes[k].Vote.Value
```

VVA/BtlVoteDecrypter.cs The method DownloadDecryptVotes looks suspicious as it seems to remove the entire contextDirectory, even if any exception arose during any of the operations in the try block.

VVA/VoteDecryptBL.cs
• **PackProcessedVotes**: Note that, as in many other system components, many of the parameters defining VEC elections are hard-coded into the system. The array `maxVotes` is limited to 3 races.

• **PackageProcess**: When we matched the implementation of this method against the peer reviewed paper [18], we observed that the implementation followed the paper closely. The cipher texts were sorted as described on page 6, Section 4.2 of [18]. However:
  
  – The variable `pc.GYR` should be renamed to `pc.GR` to conform with the notation that the variable stored $g \times r$.

  – The implementation differs from [18] in the way votes are being padded. The implementation uses a predefined “null” point on the curve stored in `EncryptedPaddingPoint.json`, whereas the paper recommends: “We can simply append the ciphertext list by repeating the list from the left side until it exactly divides $\alpha + \beta$.” No information was provided about how this point is computed or what other properties it possess.

  – Some loops are programmed in idiosyncratic ways. As an example, consider the following code fragment (lines 1137–1164):

```csharp
PackCiphers pcs = new PackCiphers()
    { BallotLength = votePref.Count,
      PrefCount = sequence.Count };
PackCipher pc = default(PackCipher);
for (int i = 0, p = 1; i < sequence.Count; i++, p++)
{
    ECPoint[] ecPoint = ECHelper.CipherToECPoint(sequence[i].Value);
    // Pack the vote
    if (pc == null)
    {
        pc = new PackCipher();
        pc.GYR = ecPoint[0].Multiply(BigInteger.ValueOf(p));
        pc.MYR = ecPoint[1].Multiply(BigInteger.ValueOf(p));
    }
    else
    {
        pcs.IsPacked = true;
        pc.GYR = pc.GYR.Add(ecPoint[0].Multiply(BigInteger.ValueOf(p)));
        pc.MYR = pc.MYR.Add(ecPoint[1].Multiply(BigInteger.ValueOf(p)));
    }
    if (!pcs.Ciphers.Contains(pc))
        pcs.Ciphers.Add(pc);
    if ((i + 1) % blockSize == 0)
    {
        pc = null;
        p = 0;
    }
}```
The code quality could be much improved by using two nested loops, an inner loop that creates one PackCipher after the other, and an outer loop that populates the collection pcs with \((i+1) \mod \text{blocksize}\)-many pack ciphers, accordingly.

- The method PackProcessedVotes splits a vote into the three races by executing the following lines of code.

```csharp
laCiphers.RemoveRange(maxVotes[0], maxVotes.Sum() - maxVotes[0]);
atlCiphers.RemoveRange(0, maxVotes[0]);
atlCiphers.RemoveRange(maxVotes[1], atlCiphers.Count - maxVotes[1]);
btlCiphers.RemoveRange(0, maxVotes.Sum() - maxVotes[2]);
```

This certainly looks suspicious, but without additional information or invariants, we cannot verify if these splitting operations are (in)correct.

- We were surprised that the method ReadCommitFile marks quarantined ballots by resetting their commit time back to “1.1.1970”.

### B.7 vVote Library

```csharp
class uk.ac.surrey.cs.tvs.utils.crypto.ECUtils
```

One should always normalize an ECPoint after generation to ensure that the \(x\) and \(y\) coordinates correspond to the coordinates of the equivalent point in an affine coordinate system (i.e. any projective coordinate is 1). This applies to methods like `encrypt(ECPoint plaintext, ECPoint publicKey, BigInteger randomness)` for both components of an elliptic curve ElGamal ciphertext. It also applies to other methods like `reencrypt(ECPoint pk, BigInteger rand)` in class `uk.ac.surrey.cs.tvs.ballotgen.CandidateCipherText`. There may be other places too where this change makes sense.

Instead of using an array of objects of type `org.bouncycastle.math.ec.ECPoint`, and to avoid possible errors (ElGamal ciphertexts consist only of pairs), considering using instead the class `org.bouncycastle.crypto.ec.ECPair`. This class is already used in Ximix.

When encrypting, the randomness used should be checked to ensure it is not zero. In the class `uk.ac.surrey.cs.tvs.utils.crypto.ECUtils` the method `getRandomInteger(BigInteger n, SecureRandom rand)` allows zero values as randomness.
In general, try to reuse the classes of BouncyCastle. For example, consider the class `org.bouncycastle.crypto.ec.ECElGamalEncryptor`, which contains many of the methods implemented in `uk.ac.surrey.cs.tvs.utils.crypto.ECUtils`.

### B.8 Ximix

#### B.8.1 Elliptic Curve Key Generation

*class org.cryptoworkshop.ximix.client.connection.KeyGenerationCommandService*

- **generatePublicKey(String keyID, KeyGenerationOptions keyGenOptions):**
  
  There is a “ToDo” comment that states that the if might not be needed. This suggests that the only algorithm that must be handled is BLS. However, this method is called, for example, from the command line tool in class `org.cryptoworkshop.ximix.tool.KeyGenerator`, where the algorithm being passed can also be ECDSA.

- **generateH(BigInteger g, SecureRandom random):**
  
  The name of this method suggests the generation of an element $h$, generated by the elliptic curve base point $g$. This value $h$ will be used in the perfectly secret commitments described in [11]. Method `generateH` actually generates a random nonzero value called $k$, which is checked to be less than the order of the generator $g$ (base point) of the elliptic curve. This ensures that a generator $g$ can actually generate $k$.

  There are several problems. First, the name of the variable $g$ in this method is misleading because it does not contain the value of the generator $g$, but rather the order (denoted in the code by $n$) of $g$. The name of the `generateH` is also confusing because this method does not appear to compute $h$, but rather the exponent $k$ which the generator $g$ will be later multiplied to obtain $h$, i.e $k = \log_g(h)$. This multiplication $h = kg$ actually takes place inside the constructor of the class `org.cryptoworkshop.ximix.common.crypto.threshold.ECNewDKGSecretSplitter`.

  Similarly, the value returned by method `generateH` should not be stored in a field called $h$ (of type BigInteger), inside the class `org.cryptoworkshop.ximix.common.asn1.message.NamedKeyGenParams`, at least if the notation from [11] is to be followed.

  Much more serious than the above notational concerns is that this way of generating $h$ is only acceptable if we assume there is a trusted process, which would weaken the system’s privacy properties. Instead, $h$
should be generated jointly by all the mixnet peers in a preliminary phase of the key generation protocol. Only then can one be sure that no one peer knows the discrete logarithm $\log_g(h)$. 

**Conclusions:** Crypto Workshop agrees and states that they fixed the computation of $h$ in the current version of the system. They remark, however, that the BLS key generation has yet to be fixed.

class org.cryptoworkshop.ximix.tool.KeyGenerator

This class implements a command line from which the key generation process can be started. It seems, however, that the threshold of peers (variable threshold of type int) to be used is set here to the number of available nodes minus 1. This means that all the nodes must be present.

**Conclusions:** Crypto Workshop states that there is a requirement regarding the choice of threshold peers. We are not aware of such a requirement.

class org.cryptoworkshop.ximix.node.crypto.key.ECNewDKGGenerator

• generateThresholdKey(String keyID, NamedKeyGenParams ecKeyGenParams):

There is a highly relevant "ToDo" comment stating that this method should have a source of randomness.

class org.cryptoworkshop.ximix.common.crypto.threshold.ShamirSecretSplitter

The constructor of the class org.cryptoworkshop.ximix.common.crypto.threshold.ECNewDKGSecretSplitter calls the constructor of the ShamirSecretSplitter by passing the parameter $N$ from an object of type org.bouncycastle.crypto.params.ECDomainParameters, which corresponds to the order of the generator $g$. However, the constructor of ShamirSecretSplitter expects as a second argument the size of the field instead. There is some confusion here between the following two cases.

1. The ShamirSecretSplitter constructor does need the size of the field to be passed, and then ECNewDKGSecretSplitter should actually send the value of the domainParams.getCurve().getFieldSize() instead of the order of $g$.

2. Alternatively, ShamirSecretSplitter actually expects the order of the elliptic curve (which coincides with the order of the generator $g$ only
if the cofactor of the elliptic curve used is 1). In this case, we suggest changing the name of the variable fieldSize.

**Conclusions:** Crypto Workshop agrees that there was a problem in the constructor of the class ShamirSecretSplitter. The name of the variable fieldSize was confusing and it should refer instead to the order of the elliptic curve base point generator.

class org.cryptoworkshop.ximix.common.crypto.threshold.ShamirSecretSplitter

- **generateCoeff(BigInteger n, SecureRandom random):**
  
  There is a comment at the beginning of this method stating that the (coefficient of the) highest order term of the polynomial should not be zero if we want the resulting polynomial to have degree equal to the threshold of peers. This is reasonable. However, the code actually generates all coefficients, and not just the one of the highest degree term. This means that no coefficient in the polynomial is ever allowed to be zero and this deviates even further from Shamir’s original paper [19] than what is warned in the comment.

**Conclusions:** Crypto Workshop acknowledges this concern.

class org.cryptoworkshop.ximix.common.crypto.threshold.ECNewDKGSecretSplitter

- **getRandomInteger(BigInteger n, SecureRandom rand):**
  
  This method does the same as many other methods, like `generateCoeff` of the class org.cryptoworkshop.ximix.common.crypto.threshold.ShamirSecretSplitter. It is also similar to the method `generateH` of the class org.cryptoworkshop.ximix.client.connection.KeyGenerationCommandService. It might be better to just write a function that works for all these cases, i.e. a function that given a SecureRandom object and an upper bound for the random value we are looking for, returns a random value in the range \((0, \text{upperbound})\). Whether the generated random value can be zero or not, could be passed to the function as a boolean variable, for example.

Besides this, the method `getRandomInteger`, which is called to generate the \(b_0\) coefficient of the polynomial \(f'\) (see Figure 2 of [11]), allows the generated random value to be zero. It is unclear to us why this random value can be zero, while in method `generateCoeff` (see the previous comment) none of the \(a_i\) coefficients of polynomial \(f\) were allowed to be zero.
Conclusions: Cryptoworshop acknowledges that it is a possible bug that the generated random value in `getRandomInteger` was allowed to be zero.

```java
class org.cryptoworkshop.ximix.node.crypto.key.ECKeyManager

• buildSharedKey(String keyID, ECCommittedSecretShareMessage message):

  This method checks the commitment of the received share, and if the check fails then the following exception is thrown.

  `IllegalStateException("Commitment for " + keyID + " failed!")`
```

From our reading of the code, this exception does not appear to generate any specific action. According to Fig. 2 in [11], the peer receiving an invalid share should at this point broadcast a complaint against the peer that sent the commitment being checked. When the party that sent the wrong share receives the complaint, then it should broadcast the shares that satisfy the check (item c of Figure 2). None of these actions appear to take place.

Conclusions: Crypto Workshop acknowledges this concern and agrees that this would be useful improvement to be considered in the future.

B.8.2 Communication with the Ximix services

```java
class org.cryptoworkshop.ximix.client.connection.ServicesConnectionImpl

This class implements a general ServicesConnection. Note the relevant “ToDo” comment that says that there is no code yet to swap out a dead node for a live one.
```

B.8.3 Shuffle process

There is a problem in the Ximix documentation [20]. The section "The Command Applet and the Shuffle Process" mentions a Runnable class called `StartShuffleTask` inside `org.cryptoworkshop.ximix.node.mixnet.service.BoardHostingService` that should start the shuffling. This class does not exist in the Ximix system, not even outside the `BoardHostingService` class.

Conclusions: The Ximix documentation will be fixed by Crypto Workshop.

```java
class org.cryptoworkshop.ximix.node.mixnet.shuffle.TransformShuffleAndMoveTask
```
The shuffling or permutation is done by using the class \texttt{org.cryptoworkshop.ximix.node.mixnet.shuffle.RandomIndexNumberGenerator}, and the ballots already taken are marked in a variable of type \texttt{BitSet}, so that no ballot is taken twice for re-encryption. Both the permutations performed and the randomness used in the re-encryption are committed using a hashing commitment, as suggested in [12]. However, the commitments produced here seem to be mappings of input elements to output elements, i.e. of the form $\Gamma_{In}^j$, which would only correspond to an odd numbered mixing node (see Section 4.4 in [12]).

Following [13], we expect to find in the system a check that the permutation performed on the ballots actually is a permutation. However, the only check we found is the one that ensures that all ballots are taken and re-encrypted once. There is no code checking that the new arrangement of the ballots is different from the initial order in which they came in. We assume this check will be done when opening the corresponding permutation commitments.

\textit{Conclusions:} Crypto Workshop acknowledges our concern about permutation checking and will consider this further.

class \texttt{org.cryptoworkshop.ximix.node.mixnet.service.BoardHostingService}

The are more than 25 different cases in the \texttt{switch} statement. This makes it difficult to read, navigate through and, in particular, find a particular case in the code. Arranging the cases alphabetically, or using some other structuring, e.g., by categories, would be helpful.

The challenges that determine which links will be asked to open to each mixing node are generated inside the \texttt{switch-case \_\_\_DOWNLOAD_SHUFFLE_TRANSCRIPT\_\_\_}, and this code seems to run after the nodes have already shuffled their inputs.

The challenge appears to be generated before the pairing of nodes. This is confusing, especially taking into account that at this point the nodes have already done the shuffling and generated their commitments. We see here a problem because if the nodes are not aware yet of whether they are odd or even nodes at runtime, then they cannot generate the right kind of commitment: $\Gamma_{In}^j$ for odd numbered nodes, and $\Gamma_{Out}^j$ for even numbered ones (see Section 4.4 in [12]).

\textit{Conclusions:} Crypto Workshop claims the pairing of nodes is handled correctly, interpreting [12] in a particular way, because it is not necessary for the
nodes to know whether they are even or odd nodes. We disagree: being an odd or even node does not determine which commitments will be requested to be opened, but rather the kind of commitments (input-output, or output-input) they must to compute. We encourage Crypto Workshop to follow the algorithm specified in the peer reviewed papers.

class org.cryptoworkshop.ximix.client.ShuffleTranscriptOptions

• isPairingEnabled():

The value returned by this method is checked, for example, when pairing mixnet nodes (inside the class BoardHostingService). The system seems to have been implemented so that the decision of following either a Pairwise Dependent Selection or an Independent Random Selection (no pairing) can be taken later. However, we think the system is not ready yet to be run following a Pairwise Dependent Selection, as we commented in the point above.

Ximix documentation [20] states that the class SeededChallenger is in package org.cryptoworkshop.ximix.node.mixnet.challenge. This is wrong. It is in package org.cryptoworkshop.ximix.common.util.challenge.

class org.cryptoworkshop.ximix.client.verify.ECShuffledTranscriptVerifier

There is a relevant “ToDo” comment in the constructor of this class: "we should incorporate the challenge seed for the witness transcript into this."

According to [13], the opened commitments should be checked to verify that the permutation commitments are consistent with a permutation, but this check seems not to have been done. The re-encryptions of the ballots, though, seem to be properly checked (by re-computing them).

Conclusions: Crypto Workshop points out that the “ToDo” comment in class ECShuffledTranscriptVerifier stating that the challenge seed should be incorporated for the witness transcript was out of date.

class org.cryptoworkshop.ximix.node.crypto.service.ProofGenerator

This is one of the most crucial classes in Ximix, because it is where the challenges of the non-interactive zero-knowledge proofs of correct partial decryption, as well as the proofs themselves, are computed.
We expect this method to compute the actual non-interactive zero-knowledge proofs of partial decryption. There is also a comment at the beginning of the current class, stating that the paper [14] was followed here. The non-interactive zero-knowledge proof of correct partial decryption consists of a non-interactive version (through the Fiat-Shamir transformation [15]) of a proof of discrete logarithm equality [16] between the public key share of the current node (see Section 3.1 in [14]) and the element $xC$ (denoted by $\beta_{x_i}$ in the mentioned paper). An interactive version of this proof is described in Section 3.2 of the paper. We observe that the implementation differs from what is described in [14].

Conclusions: Crypto Workshop is aware that, in the current version, the zero knowledge proof of correct partial decryption only provides evidence that the node is in possession of its secret key share $x_i$. They will investigate this further.

class org.cryptoworkshop.ximix.client.verify.ECDecryptionChallengeVerifier

There is a relevant “ToDo” comment: “this should probably also take the final output, and verify that the partial decrypts match the final output as well”. This check can be done publicly by anyone, since it does not require any private key to do so. However, we recommend that the system does it as well at this point.

Conclusions: Crypto Workshop reports they have finished the implementation referred to by the “ToDo” in class ECDecryptionChallengeVerifier.
Appendix C: Project Management Plan Review

The following are specific comments on [2].

- In §1.4, open source is required for developed software. While we find open source laudable, note that the fact that the software can be independently verified does not mean it will be. In particular, such verification is less likely to occur for specialized systems (e.g., those requiring specialized hardware or operational requirements) and rapidly evolving systems.

- §2.4 states that software will be developed in Java and PHP. This list is incomplete, e.g., C# is also used. It is stated that software will be modeled using appropriate languages such as UML. Is this a recommendation or requirement? The current design models are limited.