Deployment and User Manual

Introduction

This accompanying document provides the deployment manual and user manual for the vVote Verifier which is a stand-alone independent reference implementation of a verifier for the vVote system [1]. The vVote System is an end-to-end verifiable electronic election system based on the Prêt à Voter election scheme [2]. vVote was designed and built to be used as part of the State Election taking place in Victoria, Australia in November 2014.

The verification process of the vVote system, as carried out by the vVote Verifier, focuses specifically on a number of protocols or processes carried out including:

- Ballot Generation – The ballot generation process takes place during the pre-voting stage of the election and involves the production of generic ballots containing randomly permuted candidate names along with the candidate permutations. The random candidate permutation is constructed using combined randomness values generated and communicated by a number of independent randomness generation servers. The verification process involves the auditing of a randomly selected set of generic ballots. Each audit involves the recalculation and verification of the generic ballot.
- Vote Packing – The vote packing process is used to reduce the amount of work carried out by the Mixnet during the tallying phase of the election. The vVote system uses a Mixnet to shuffle the voter preferences represented by re-encrypted candidate identifiers sorted into preference order. The vVote system uses a method of packing together multiple re-encrypted candidate identifiers before passing them to the Mixnet meaning far fewer ciphertexts require shuffling and decryption thereby producing significant efficiency benefits. The verification process is two-fold:
    1. The first check is used to verify that the ciphertexts provided to the Mixnet have been correctly packed according to the set of underlying voter preferences and that the packings match those that were input to the Mixnet. It involves the recalculation of the vote packing for each vote cast and verifying that the packed values match those provided to the Mixnet input.
    2. The second check is used to verify that the decryptions from the Mixnet, corresponding to packed plaintexts, correctly match the packing of the plaintext candidate identifiers using the claimed preferences from the Mixnet output. This step verifies that the values match what would have been looked up to determine

the voter preferences and avoids having to perform a full recalculation of the lookup tables requiring only the specific entries (those actually used in the lookup) to be recalculated. This check relies upon the homomorphic encryption property of ElGamal Elliptic Curve encryption.

- Public Web Bulletin Board (WBB) Commits – The Public WBB is used by the system to make available publicly relevant information which can be used in the verification process for other components such as the vote packing and ballot generation processes. The Public WBB is simply updated once per day as separate commitments. Each commitment consists of a number of JSON messages, an attachment ZIP file and a JSON signature file. The JSON signature file contains a signed hash of the important protocol information from both the JSON and ZIP files. The signing will be carried out by the Public WBB and will serve as its authenticity and can be used to verify the commitment. The verification process is to recalculate the hash of the data, which was signed by the Public WBB during the commit, and to verify that the hashed data is the same data that was signed by the signature contained in the Signature JSON file using the Public WBB's public key.

Additional details regarding the specifics for each of these verifiable protocols can be seen in Chapter 3 – 3.5 Verifiable Election Stages and Chapter 4 – 4.3.5 vVote Verifier in the accompanying dissertation. Additionally concrete examples of the Ballot Generation process and Print on Demand process can be seen within Appendix H – vVote Verifiable Component Examples of the dissertation.

vVote Verifier Explained

The vVote Verifier consists of a number of specific component focused verifiers however it can be run and considered as a single verifier which performs a number of election wide checks on the various underlying components. The primary focus of the vVote Verifier project was to build an independent reference implementation of a verification tool able to verify the output produced by the portion of the vVote code developed by the University of Surrey consisting of the ballot generation process, vote packing process and commits made to the Public WBB.

The verification process, carried out by the vVote Verifier, can be broken into a number of smaller discrete verification steps:

Ballot Generation Verification – The step by step details of the algorithm used to verify the ballot generation process can be seen in Chapter 5 – 5.5.1.1 Ballot Generation in the accompanying dissertation:

- Verification of the Fiat-Shamir signature calculation for each PoD Printer which is used to determine the ballots chosen for auditing.
    - For each PoD Printer, we construct and compare a re-calculation of the Fiat-Shamir signature with the Fiat-Shamir signature included in the ballot submission response message corresponding to that PoD Printer's ballot audit commit message.
- For each PoD Printer, verify that the validated Fiat-Shamir signature was correctly used for selecting the ballots for auditing.
    - The Fiat-Shamir signature is used as the seed for a random selection of generic ballots which should be audited for the PoD Printer. Using the re-calculated Fiat-Shamir signature we verify that the new list of ballots chosen for auditing match those ballots included in the ballot audit commit data for the corresponding PoD Printer.
- Verify that for each ballot chosen for auditing the witness value from each pair of randomness values opens the corresponding commitment made to the random value posted on the Public WBB by the corresponding randomness generation server.
    - Each ballot chosen for auditing will have had their secret randomness values revealed and published on the Public WBB.
    - For each pair of randomness values, concatenate the random value and witness value together and hash the result using SHA256. Compare the hash with what was posted, by a corresponding randomness generation server, to the Public WBB during the Ballot Generation process.
- For each ballot chosen for auditing, carry out the re-encryption of the base candidate identifiers using a different combined randomness value for each of the candidate identifiers and sort the resulting ciphertexts in a per race basis. Verify that the sorted, re-encrypted base candidate identifiers match the list of re-encrypted base candidate identifiers contained in the matching generic ballot with the same serial number, published on the Public WBB.
- For each ballot chosen for auditing, check that the permutation commitment contained in the corresponding generic ballot from the Public WBB can be opened by using the final combined randomness value as a witness value and the permutation constructed using the original order of the base encrypted candidate identifiers on the recalculated ballot as the random value.
    - Concatenate the permutation value and the final combined randomness value together and hash the result using SHA256. Compare the hash with what was is

included in the matching generic ballot with the same serial number, published on the Public WBB.

Vote Packing Verification – The step by step details of the algorithm used to verify the vote packing process can be seen in Chapter 5 – 5.5.1.2 Vote Packing in the accompanying dissertation:

- Check the pre-Mixnet vote packing process. Check that for each of the cast votes all of the candidate ciphertexts have been correctly packed according to the preferences provided. This checks that the packed encryptions match those that were provided as input to the Mixnet.
- Check the post-Mixnet vote packing process. Check that the decrypted packings from the Mixnet correctly match the packing of the plaintext candidate identifiers using the claimed preferences.

Public WBB Commit Verification – The step by step details of the algorithm used to verify the public WBB commitment process can be seen in Chapter 5 – 5.5.1.3 Public WBB Commits in the accompanying dissertation:

- Verify that the signatures over a hash of the commitments made to the Public WBB are valid and have been correctly calculated. The verification process is to recalculate the hash of the data, which was signed by the Public WBB during the commit, and to verify that the hashed data is the same data that was signed by the signature contained in the Signature JSON file using the Public WBB's public key.

How to use vVote Verifier

This section provides details for both simply running the provided JAVA Archive File (JAR) or for compiling and running the source code within the Eclipse [3] Integrated Development Environment (IDE).

Using the vVote Verifier JAR

The vVote Verifier has been exported to a JAR file for ease of use, however, the required specification and schema files as well as a cryptographic parameters file and the compiled c++ code utilising the c based OpenSSL library were not exported to the JAR file and therefore to run the vVote Verifier these libraries must be present when the JAR is run.

To run the vVote Verifier, first the OpenSSL libraries must be present on the operating system running the verification process. The c++ based library, utilising the c based OpenSSL libraries, has been compiled for either a 64-bit Linux machine or a 32-bit Windows machine. The implementation was initially designed to be platform independent and to have no dependencies as to the operating system or system configuration; however, due to the use of a c++ compiled library using the OpenSSL library the vVote Verifier requires that the system used to run the tool is either (64-bit) Linux or (32-bit) Windows based.

To run the vVote Verifier the OpenSSL libraries must be installed as follows:

Windows (tested on Windows 8.1 Pro 64-bit edition):

- Install a c++ compiler such as MinGW, which includes a port of the GNU Compiler Collection (GCC).
    - MinGW can be downloaded from http://www.mingw.org/
- Install the OpenSSL libraries.
    - OpenSSL can be installed using the Win32 OpenSSL installation project at https://slproweb.com/products/Win32OpenSSL.html and downloading and installing the file Win32 OpenSSL v1.0.1i or similar.
- Install the JAVA Platform, Standard Edition Development Kit (JDK) from http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

Linux (tested using Ubuntu 14.04):

- Install the GNU Compiler Collection (GCC).
    - On Ubuntu 14.04 this can be carried out by using the command: "`sudo apt-get install gcc`".
- Install the OpenSSL libraries.
    - On Ubuntu 14.04 this can be carried out by using the command: "`sudo apt-get install openssl`".
- Install the LibSSL libraries.
    - On Ubuntu 14.04 this can be carried out by using the command: "`sudo apt-get install libssl-dev`".
- Install the JAVA Platform, Standard Edition Development Kit (JDK).
    - On Ubuntu 14.04 this can be carried out by using the command: "`sudo apt-get install default-jdk`".

The vVote Verifier delivery includes the vVote Verifier JAR file named vVoteVerifier.jar. The verification tool can be run from the command line by providing a single argument. The only requirement with regard to the input is that all data be present and in the required correct format. Configuration files do not need to be specifically within any folder and will be found automatically; this is a feature added to avoid any issues with files having their locations changed during the development of the vVote system. The system takes, as input, the path to a single folder which contains all the data and configuration files required.

The vVote Verifier is run using the command:

```
“java -Djava.library.path=jni -jar vVoteVerifier.jar <path to data>”
```

The results produced through the running of the verifier will be both output to command line and also saved to two log files within a “logs” folder created in the same directory as the tool being run. The first log file “results.log” will contain step by step information including the details of the verification step being carried out and the result of the verification step. The second log file “logfile.log” will contain more detailed information as to the exact steps taking during the verification. The second file will be significantly larger than the results.log file and should only be examined in the case of a failure occurring in the results.log file.

How to Interpret the Results

This section will provide a brief description of how to interpret the results of the vVote Verifier output to the results.log file. A typical line of the results.log file will show the time the verification took place, the level of the logging, the verification component and a short descriptive message as can be seen below.

```
17:36:55.030 INFO  ComponentVerifier - Successfully verified that the plaintext ids and base
candidate ids match
```

The level of the logging is the most important part of the log shown and indicates the result of the current operation. INFO denotes that the message is purely informational and that the verification has taken place correctly. WARN denotes that something has happened which should be investigated and needs to have questions asked and answers provided by the election officials as to why the issue has occurred. ERROR is the most important indication and denotes that something has gone very wrong in the verification. In the case of an ERROR being present within the results.log file the area of the vVote system which has caused this error should be heavily scrutinised and questions should be raised as to the correctness of its operation.

Compiling and Running vVote Verifier using Eclipse

The vVote Verifier source code can be compiled, run and analysed using Eclipse (or any other relevant IDE). A number of deployment steps need to be followed to successfully compile and run the code.

- Download and install Eclipse from https://www.eclipse.org/.
    - On Windows the 32 bit version must be installed so that the 32-bit JAVA Virtual Machine (JVM) is used.
    - On Ubuntu the 64-bit version should be installed.
- Import the vVote Verifier source code into Eclipse as an existing project from the archive file provided.
    - Import -> General -> Existing Projects into Workspace -> Select the accompanying vVoteVerifier archive file -> Choose to only import the vVote Verifier project.
- Add the JNI folder to the default virtual machine arguments.
    - Windows -> Preferences -> JAVA -> Installed JRE's -> Edit the existing JRE and add the following text to the default vm arguments "`-Djava.library.path=jni`".
    - The provided DLL (Windows) and so (Linux) files need to be in the JNI folder in the top level of the project.

Editing the c++ Code Utilising OpenSSL

To edit the c++ code utilising the c based OpenSSL library then a number of additional steps must be carried out:

Windows (tested on Windows 8.1 Pro 64-bit edition):

- The OpenSSL libraries need to be made available to the c++ compiler.
    - Copy the DLL libraries "libeay32.dll" and "libssl32.dll" from the installed OpenSSL directory such as "C:/OpenSSL-Win32" to the bin folder of the c++ compiler chosen such as "C:/MinGW/bin" folder.

Both Windows and Linux:

- To modify the JAVA side code used to call the c++ based library then a new header needs to be created:
    - When inside the folder "vVoteVerifer/bin" use the command "`javah com.vvote.verifierlibrary.CryptoUtils`".

To recompile the c++ code the steps are as follows:

Windows:

- Create .o object file:
  ```
  gcc -c -I"C:\Program Files\Java\jdk1.7.0_51\include" -I"C:\Program Files\Java\jdk1.7.0_51\include\win32" -I"C:\OpenSSL-Win32\include" CryptoUtils.cpp
  ```
- Create .dll shared object:
  ```
  gcc -Wl,--add-stdcall-alias -shared -o CryptoOpenSSL.dll cryptoUtils.o -lssl -lcrypto -lstdc++
  ```

Ubuntu:

- Create .o object file：
  ```
  gcc -c -fPIC -I/usr/lib/jvm/java-7-openjdk-amd64/include -I/usr/lib/jvm/java-7-openjdk-amd64/include/linux CryptoUtils.cpp
  ```
- Create .so shared object:
  ```
  gcc -Wl -shared -o libCryptoOpenSSL.so CryptoUtils.o -lssl -lcrypto -lstdc++
  ```

Bibliography

[1] C. Culnane, P. Y. A. Ryan, S. Schneider and V. Teague, “vVote: a Verifiable Voting System,” 24 June 2014. [Online]. Available: http://arxiv.org/abs/1404.6822. [Accessed 12 August 2014].

[2] P. Y. A. Ryan, D. Bismark, J. Heather, S. Schneider and Z. Xia, “The Pret a Voter Verifiable Election System,” 2010. [Online]. Available: http://www.pretavoter.com/publications/PretaVoter2010.pdf. [Accessed 19 August 2014].

[3] “Eclipse,” Eclipse.org, [Online]. Available: https://www.eclipse.org/. [Accessed 24 August 2014].