openly licensed" documentation in the Commission's Open Source Voting Resolution.)

- [TODO: provide recommendations related to managing community feedback and contributions during project development. Also think about whether contributor license agreements (CLA's) should be required.]

5.7. Procurement

[TODO]

5.8. Software architecture and design

Listed below are recommendations related to the software architecture.

5.8.1. Modular design

- The software should be designed to be a collection of independent components rather than a monolithic application. Each major component should have well defined input and output file formats– the output of one component may serve as input to another component. Components would be designed to be developed in separate stages, and could be replaced in entirety in a future revision. Likewise, each major component might be subdivided into a collection of independent sub-component modules, performing a portion of the overall component function, using intermediate files (files other than the component input or output files) to interoperate.

- A GUI and/or process control application could be used to control the sequencing of component and sub-component application commands, prepare input and configuration files, view results, etc. Such a GUI application could give the appearance of a monolithic application, but underneath, processing occurs as a sequence of the independent module commands.

- To control the sequence of processing required when updating input data, a "build" system could be used, such as the Unix `make` command or equivalent (some programming language environments have alternatives), linking the dependencies of inputs, outputs, and application commands required for processing. Alternatively, the "build" system could be constructed with custom software and scripts. In either case, a small software application should be used to identify the general election configuration settings and available input files to create a customized build command file/script. This "build" subsystem can be considered an independent component, as with the individual modules, and independent of a GUI.

- When defining software components to develop, favor designs that promote reusing components. For example, a software library that can read a digital ballot picture and return the marked "votes" (what we are calling a "ballot picture interpreter" component) can be used in both precinct scanners and central scanners (as well as software applications for adjudication or

auditing). Favoring component reuse can mean having less code to write and test, which in turn can reduce required time and costs.

- Software that performs critical operations that rely on the integrity of the software, e.g. vote tabulation, vote direct recording, etc. should be classified as “sensitive” (in other words the correct operation can only be verified via auditing or independent replication). The amount of code in sensitive modules should be minimized to improve security, and to facilitate auditing of the code by inspection. Less sensitive parts of a component could be split into a separate application, converting input and configuration data to a form optimized to simplify the sensitive module. The intermediate files produced by the split-off insensitive code can be posted for inspection, with independent checking software. Likewise, incorporation of large amounts of third-party libraries should be minimized in sensitive modules– it may be preferable to copy-and-paste the specific code needed rather than embed a complete library (inspecting and auditing a large library might be infeasible).

5.8.2. Testing, Auditing, and Error Checking

Besides the basic set of commands that implement components, a collection of checking and testing applications should be included. The functions performed might include:

- Checking the data formats for input, output, and intermediate files for validity, and computing some basic statistics for error checking, e.g. number of contests and candidates.

- Cross-checking output files against input data. For example, a program to check input data configuring a ballot picture interpreter could extract the text from a ballot PDF file, match locations of contests, candidates, and mark targets with the extracted text or target image data. The candidate names and order of appearance can be cross-checked with the contest definitions, ballot type data, and candidate rotation definitions. Section headings and contest titles can be compared against the contest definition data. Likewise, a contest definition checker could cross check data with raw imports from an external EMS.

- Auto-generating test data. Rather than relying on manual testing of components, software could be used to generate sample input data that should produce a known (or independently calculated) result. Auto-generated data might also be used to generate all possible cases of input data to be handled.

- A testing methodology should be developed as part of the usual software development process. For example, a special test mode might be used to enable testing of individual software routines. When changes are made, a set of “regression tests” should apply a collection of test input data and verify the output data produced matches what is expected.

5.8.3. Data-Driven Architectural Design

The overall architecture of components and sub-components should be developed by defining the data formats for input and output files. A component could be split into a set of sub-components by defining intermediate data file formats. The input/output data definitions, along with processing requirements then lead to a definition of the individual modules that need to be developed. Also, checking and testing components that are possible follow from the data definitions.

Using a data-driven design facilitates incremental development, independent implementations and checking, and easy replacement if needed. This contrasts with functional design, simply defining processing required, or architectural design, specifying programming languages, development frameworks, database systems, etc.

5.8.4. Input/Output File Formats

Besides defining the content in individual input and output files, a data format should be defined, optimized to meet the following criteria:

- Aside for inherently binary data such as image files, data should be represented in a printable text format vs a specialized binary format. This facilitates easy inspection, human viewing/editing, and debugging.
- Data files should be assumed to be public records (aside from data that contains private information), and posted in an electronically accessible manner, e.g. on an election department web site.
- The data format should work well with revision control systems and line-based “diff” software that can compare changes made over time. (A text file format is required that includes line breaks at logical spots, e.g. a data record.)
- Data formats that can be read and written with minimal software (in any programming language) should be preferred over formats that require a large complex library to read/write or use a format built into a specific programming language.
- Simpler data formats suitable for easy human viewing and editing should be preferred over more verbose formats that rely on software to extract.
- Simpler data formats less sensitive to errors should be preferred. [Though widely used, most libraries used to read XML or JSON, for example, are very sensitive to even slight inconsequential errors anywhere in the input data, returning nothing if a data file format is not perfect (as well as no meaningful error message).]
- Where practical, data formats that can interoperate with ordinary office software (e.g. a spreadsheet) should be preferred.

- The data formats should ideally separate data that remains invariant from data that changes regularly or independently during an election cycle. Data files are assumed to have cryptographic hash and digital signatures, which also identify and track changes.

- A data format might be designed to simplify sensitive software modules, reducing the number of lines of code required, including use of third-party libraries.

The primary input and output files designed with the above criteria are assumed to be publicly posted with digital signatures for validation and change tracking, and interoperate with a revision control system (e.g. git). This implies that use of a monolithic database could not be used. However, this does not preclude a database from being used as an intermediate file, but the database must be a slave to the official digitally signed and validated input data loaded into the database, and final output data should be exported into a format that meets the above criteria.

Likewise an import/export file used with external systems might not meet the criteria above. For example, a collection of input data files created by various components might be converted into a single NIST SP-1500-100 XML data file used for reporting election results– the SP-1500-100 containing election definition data, precint-district definition data, result subtotals and totals. The SP-1500-100 format would be a required part of interoperable computer-readable results reporting (being a standard format), but that format is not conducive to the modular software architecture described above, with data update and revision control.

5.8.5. Internationalization

Part of the software architecture design includes how multiple languages can be handled– support for multi-lingual ballot design, printing, and voting machine displays are a basic requirement. The design might also include provisions for multi-lingual support for voter information systems, including election results reporting.

5.8.6. Open-source Linux Host OS

Other than small embedded systems, the host OS should be based on an open-source Linux distribution. Some considerations include:

- The host OS should be open-source and freely redistributable. Members of the public should be able to download an image or reproduce the installation.

- A selection of a distribution must be made, probably either the Ubuntu/Debian or Centos/Redhat distribution.

- A deliverable should be a virtual machine image (either virtualbox or vmware). This would allow members of the public to run the system on any OS. A bare OS image as well as image

with all applications would be best.

- The host OS should have minimal software, configured only to securely run applications packaged in containers (see below).
- Deliverables should include scripts to install the OS from a standard distribution, configure the system, adding or removing software packages as needed.
- For enhanced security, a customized kernel could be built that excludes drivers for unused hardware, e.g. WiFi, audio, possibly USB.

5.8.7. Packaging applications into containers

A recent development in software development and deployment is the use of “containers” (e.g. Docker), a form of virtual machine, but running a virtual OS rather than virtual hardware. Containers combine a base OS interface, a set of libraries, and set of application software packages into one image file. The container is configured with specific versions for each of the libraries and applications included, so any complex version dependencies are resolved. The packaged software running within a container appears as if it’s running within a separate virtual machine. Software running in a separate container appears as a separate machine. Containers can start quickly from a fresh read-only image, so any corruption of software is lost when the container stops. A container built with linux application will generally run on any linux host with no specific application software installation required.

- All application software should be packaged into containers for release and deployment. The host OS should only contain scripts required to configure the container environment (disk, networking communications, access to security devices, etc.), and control a sequence of container operations.
- For enhanced security, components of the voting system can be packaged in separate containers. Sensitive applications (e.g. vote totaler) should be run within a separate container with minimized software. Data integrity across container operations should be maintained with digital signatures and cryptographic hashes.
- The software development environment used to build the voting application components and containers should be provided as a container and included in deliverables.
- Part of the design includes determining how the host OS initiates a sequence of container operations, and how data is stored within the host OS and passed into the containers. This includes the use of read-only file systems to prevent alteration of input and configuration files by the container application.

5.8.8. Multi-computer Configuration Design

Besides configuring a host OS to run applications as container operations, part of the design may include how multiple computers can be used and interconnected.

- The voting system might be divided into 2 machines, one running a system with outside (internet) connectivity, and another highly secure system with no outside connections other than a limited one-way connection to the other machine.
- A set of machines might be used in a central scanner operation, one machine per scan station. The scanner application (presumably running within a container) would need to use networking communications to deliver raw scanned images and processed cast vote records to a primary machine.
- Voting stations at a precinct or vote center might require networking (e.g. to communicate with a print station).

5.9. Software development

- The project should not depend on volunteers for the successful completion or security of the project. However, useful volunteer contributions should be encouraged and not turned away.

5.10. Hardware design

5.10.1. Hardware Security Issues

5.10.1.1. Read-only file system support

Maintaining security and integrity of data and software, especially OS software, can be greatly simplified if data can be made read-only at the hardware level. Some devices are inherently read-only, e.g. finalized DVD-ROM. General can be configured at the OS-software level. For containers or virtual machines, software read-only protection is reliable, but if a host OS is compromized “read-only” data could be altered. Many seemingly hardware-based read-only protections are actually software based using hardware switches or sensors, for example camera memory card write protect switches, or (for those old enough to remember) floppy disk write-protect tabs. A hardware issue that might need to be addressed is if data can be stored write-only at a true hardware level, independent of an OS or firmware.

5.10.1.2. Hardware secure boot capabilities

The capabilities or limitations at a hardware level of a secure boot process can affect the inherent security of a computer system. This may affect the selection of hardware. In a completely secure system, boot firmware is immutably stored within a microchip and linked with a digital signature private key. The firmware cannot be changed without replacing the chip in entirety, also replacing the

protected digital signature key and revealing alteration of the hardware. However, other “secure boot” system hardware might be changeable by altering BIOS settings, or reprogramming firmware with specialized hardware devices. More details are discussed in the security section.

5.10.1.3. Hardware Cryptographic Devices

Digital signatures on output data files are an important aspect of the entire voting system integrity. Storing private digital signature and encryption keys within a hardware device is the only means to insure keys cannot be stolen and used elsewhere. The capabilities of a hardware security device may be important in hardware selection.

The US Department of Defense requires TPM (Trusted Platform Module) in much of it’s computer procurement. It would make sense to require TPM chips in voting system computers both for secure boot and as the hardware cryptographic device. For remote access (e.g. within the election department local network) to voting application servers, the client computers should also have TPM chips for hardware-based client authentication.

USB Hardware Security Modules (HSM) could be used for digital signatures, decrypting escrowed data, or enabling access to a secure system, only at certain times. The HSM could be physically stored in a safe at other times. [Some HSMs support cryptographic operations that support multiple devices, e.g. require 3 of 5 USB keys.]

5.10.2. Other Hardware Design Topics… [TODO]

5.11. Documentation

[TODO]

5.12. Security

5.12.1. Primary security is provided by auditable paper records

The primary basis for election integrity and security is the use of voter verified paper records, using electronics for enhanced security, and auditing to validate paper with automated processing.

5.12.2. Security architecture and design should be a separate voting system project and deliverable

The requirements, methodology, and solutions needed for computer security are complex, and should be considered a separate component of the voting system, and like other components possibly implemented incrementally. Part of this design and implementation includes:

- Evaluation of a secure boot process by which firmware, boot loaders, host OS, containers, and application software is authenticated with digital signatures. Ideally, only software stacks signed by a certification authority (e.g Secretary of State) could be loaded and run on a machine.

- How can cryptographic hardware be used– HSM (Hardware Security Modules), TPM (Trusted Platform Module) chips, and USB HSMs that can be stored in a safe until used.

- What format should be used to digitally sign collections of data files, either packaged for internal (e.g. cross-container) use, or public distribution. The solution could be based on an existing format (e.g. JAR), a custom design based on scripts using standard commands, or some combination. Signatures on data collections are also inherently part of a secure boot process (booting might require a different validation format than input/output files).

- How should software be securely updated– firmware and boot loaders, host operating system, virtual machines (if used), containers, application software, and data files.

- What access and networking provisions are needed. Can a secure system be constructed with no general (sysadmin) access, i.e. system administration occurs via secure signed updates and secure boot. (If a sysadmin has access to a machine, even over secure communications, a compromise of the sysadmin's machine means a compromise of the secure system.)

- What methodology should be used to reproduce software builds for binary comparison.

5.12.3. Primary security of software is via public scrutiny and reproducibility.

All software used in the voting system (including the OS and supporting software) should be open-source, freely downloadable by the public, and copies included for redistribution packaged as part of the voting system deliverables.

Certification authorities as well as members of the public should be able to reproduce builds of the application software executables, containers, and host OS, such that a binary comparison can be made on digitally signed images used in the voting system deliverabes and secure boot chain. Either some scripts or container configuration would be used to reproduce identical binary build results, and/or using specialized software to perform a comparison excluding time stamps or other build-system identifiers.

The software source code and documented security methodology is subject to scrutiny by certification authorities as well as the public at large. Public comments should lead to identification of weaknesses and improvements over time.

Most election data should be posted electronically and accessible to the public. Validation of this data can be made by reproducing data analysis and data checking, possibly using independent software.

Members of the public would be free to develop their own independent electronic auditing. (Scanned ballot image data would probably be too large for internet download, so provisions for a copy on hard disk would be required for public access.)

5.12.4. All software should be validated with digital signatures

The integrity of software running on a voting system must begin with a secure boot process– digital signature validation of an initial boot loader begins at the hardware level, then an authenticated loader can validate the digital signatures of the operating system image and/or OS file system prior to booting the OS, then the OS might need to validate the digital signatures of software applications or containers. A full “secure boot” means the complete chain of firmware, booting, OS, and application software is authenticated.

Updates of software is also an important part of computer security. Updates need to be possible, but only with authenticated software/firmware changes, and only during approved times.

Besides the executable images of software, digital signatures should be used with source code as well. Source code repositories and revision control systems such as `git` have provisions to use cryptographic hashes and digital signatures.

Some issues related to software validation include:

- Is secure boot immutable at the hardware level or can boot options be changed by BIOS setting or reflashing firmware.
- Can secure boot be limited to signatures of an approved certification authority or limited to vendors set by the hardware manufacturer. [A related issue is, can the required signature be changed/updated via digitally signed update data.]
- How should digital signatures be used with software source-code repositories. How can certification authorities use digital signatures (or cryptographic hashes) to approve specific source code releases.
- Should software build logs include cryptographic hashes of source code and configuration data plus hashes of binary executables, be digitally signed and included with a distribution to link a specific source code version with the built executables.
- Containers could be validated with a digital signature of the image, but as with software application builds, it may be important to digitally sign a log of the container build that includes the cryptographic hash of base containers layered in a build, and hash of files within the container. This log then links a signed container image with specific versions of reference containers and

5.12.5. Software updates should be possible only via a controlled process

Besides a secure boot to a known certified set of software, it must be possible to make updates to the software (as well as firmware and authorized signatures). However, updates should only be made during a controlled process.

If a file system disk has hardware write protection, then physical access to a machine is required to enable writing and updates.

Updates and in turn, secure boot, would only be possible with digitally signed certified software. However, it might be useful to require some additional step to limit update times, for example, inserting an HSM or Security Token (normally stored in a safe) to enable updates.

It might be possible to alter contents with physical access to a hard disk– it would boot if the content included secure boot signatures. However, changes might be protected using a TPM chip or disk encryption with keys held in a TPM.

5.12.6. All data files should be validated with digital signatures

Besides software validation, the integrity of data processing should be insured by checking the digital signature or cryptographic hash of all input files (including configuration files). Data files imported from an EMS, other system, or GUI input data should be digitally signed as part of the import. (A collection of input data could be signed as a batch.) All output data files produced should be digitally signed. (Likewise, a collection of output data could be signed as a batch.)

General validation of input data files could be performed during a secure boot, container start, or even by a host OS initiating a sequence of container operations. A sensitive application should explicitly validate input data and signatures, ideally by computing the cryptographic hash of input data as it is being read.

Output data files created in a processing step should be digitally signed. This could be done in a post-processing phase, or for sensitive applications, could be done using a cryptographic hash computed as part of writing a file.

Most digital signatures would be created using a private key stored on the machine performing the processing. [The signature verifying data was created on that machine.] For certain components of the voting system, certified versions of data input might be required (for example, contest definitions in vote tabulation, or ballots within voting machines). To identify certified data, a separate digital signature key should be used to differentiate from the usual system hardware identification key, for example a key held by the election official inside a USB hardware security module (HSM) normally stored in a safe, and used by that election official to manually create a digital signature. (Typically, a password would be required to activate the HSM in addition to physical access.)

5.12.7. Log files should include hash codes of input/output files

To link specific input files with output files created by a voting system component, the application should create a log file of the processing that includes a listing of the cryptographic hashes of input data files read (including configuration files), the cryptographic hash of the software executable used (could be the hash of a container), and hash of the output files created. This log file would be included in the digitally signed data set.

A read-only file system might simplify digital signature checks. A container application could be constructed with the assumption that the contents of a read-only file system be externally validated prior to the container start. The application would only need to retrieve the hash codes for use in a log file rather than the full process of computing hash codes and validating a signature. However, sensitive applications might not want to rely on external input data validation.

It may be useful to design a log file format that would standardize the notation/format for listing input and output file names and hashes in the log files so a generic log analysis application could validate data across a sequence of processing steps, and also automatically derive the processing sequence required when data updates occur.

5.12.8. Digital signature keys should be held in hardware

If practical, keys used for digital signatures or decryption should be held within a hardware security module (HSM) or Trusted Platform Module (TPM) chip built into some computer hardware. The alternative (commonly used with ordinary internet servers) is a key held in a file, readable to anyone or any software program with OS (root) access.

While some HSMs cost 10-100x the cost of a server computer (generally designed for high performance, high volume), the recommendation is to utilize relatively low cost (e.g. <$100) TPM chips or USB HSMs for general purpose data signing.

As a staged release of security measures, early versions of a voting system could use the usual software keys stored in a data file, then later upgraded to use an HSM. The open-source software should be designed to work with HSMs or software-stored keys.

5.12.9. Hardware digital signatures created should be tracked

While a hardware security module can safely hold keys without the possibility of being stolen, a compromise of the host OS of a machine with an HSM means any software on that machine might be able to use that HSM to sign any data.

To detect unauthorized uses of an HSM (and validate authorized use), an HSM with a serial number (counter) can be used. If a log is kept of signatures created by serial number, then audit software could

compare logged signatures with the signatures included in processing or release logs.

5.12.10. An external digital notary service could be used for an audit trail

The public release of certified data could be logged and validated with a third-party digital notary service. For unreleased non-public data, a summary of cryptographic hashes could be digitally signed and notarized. The notary signatures are then available during automated or manual audits to prove that data signed during an election cycle has not been altered after-the-fact.

Use of a notary or even an appropriate notary service might not be immediately availabe, but could be part of an overall phased security implementation.

5.12.11. Recommended cryptographic software

The recommended software for cryptographic operations is the `openssl` command line tool and application software library. Openssl is open-source and the most widely used cryptographic software and subject to a great deal of public scrutiny. It is also FIPS-140-2 certified. Openssl may be used with software keys or using a plug-in driver, a hardware security module.

For computing and checking cryptographic hashes on individual or batches of files, the `sha256sum` command can be used (part of the core utilities hat comes standard with linux distributions).

5.12.12. All networking communications should be encrypted

For any client-server or other networking used between machines, either over ethernet cable or wireless communications, data should be separately encrypted rather than rely upon the security of wired networks or WiFi encryption, etc. Either a VPN protocol could be used, or a secure communications protocol such as HTTPS, SSH, or TLS (for others) could be used.

5.12.13. Secure protocols should authenticate both sender and receiver

With the usual HTTPS “secure” web protocol, only the server is authenticated, and by default using a registry of public certification authorities (e.g. LetsEncrypt.org). For secure voting systems, it may be useful to use a separate built-in certificate authory to limit access.

To allow communications only between authorized machines, the secure protocol should require certificates at both ends. An https server should use the SSLVerifyClient (or equivalent) directive to require public key certificates to be installed in browsers, ideally utilizing private keys stored in a hardware device.

Other protocols (e.g. SSH) can use public/private key encryption by default– access can be limited to a fixed set of keys.

5.12.14. Limit use of name and password access control

For some non-sensitive applications (like ballot layout software), a web (https) interface could be used and accessed by an office computer, using a usual name and password authentication with cookies for access control. The servers might be configured to allow only machines on an internal network, but in general password-based authentication should be considered weak and limited to parts of the system where output files can be analyzed cross-checked.

The basic password authentication could be improved by using a “security token”, a hardware device that generates a semi-random code (usually based on the time), that can be used for access. To login, a user would need the security token device to get a code.

Certification and finalization should be restricted to specific machines with public key access and/or digital signatures. A USB Hardware Security Module could be used to enable certain important operations, or access.

5.12.15. Limiting access to secure systems

For the most sensitive applications, it may be possible to eliminate command-line access for anyone, including sysadmins. If sysadmin access is allowed, the a compromise of a sysadmin’s computer, probably leads to compromise of a computer that sysadmin can access.

As an alternative to command line access, the system would secure boot to a set of applications. Outside access would only be available via the installed applications. Any change to the software would be done via secure upgrade process (digital signatures on new software would be verified, then used in the secure boot). Some provisions would be required to replace input or configuration files in a secure manner. Normal import of data input files, export of output files, and initiating processing would be part of the installed applications (presumably running within a container).

To prevent any command-line access, such a secure system might have keyboard and serial port input drivers removed from the OS.

5.13. Testing

1. **Gather real election data.** Datasets of real election data (e.g. a couple past elections in San Francisco of different types) should be compiled in a structured format for product prototyping and testing. This includes not just vote totals but also candidate and contest data. This will help in establishing requirements and designing the system.

2. **Gather real digital ballot pictures.** Starting with the June 2018 election, during each election the Department should gather and save large numbers (e.g. thousands) of digital ballot pictures for future testing purposes. The Director has already expressed a willingness to do this in the