IVXV architecture

Specification

Versioon 1.4.0

18.01.2019

39 lk

Dok IVXV-AR-EN-1.4.0

Contents

CHAPTER 1

Introduction

The online voting information system has been created based on the e-voting framework [ÜK2016] and the technical specification of the public procurement 171780 [TK2016]. This document specifies the architectural solution of IVXV. The online voting information system consists of offline applications and online components. The information system is additionally dependent on external information systems and affects the components directly used to vote online / verify votes.

The architecture document specifies the IVXV components, their interfaces with each other as well as external systems, and the protocols implemented by the components.

1.1 IVXV Concept

The general, but comprehensive overview of the technical and organizational side of the online voting system (IVXV) and its implementation at Estonian state-level elections is given in the general specification of the e-voting framework [ÜK2016].

IVXV as an information system implements the i-voting protocol based on the envelope scheme. IVXV works in the pre-voting stage, the voting stage, the processing stage, and the counting stage, offering options to participate in the online voting process to the organizer, the counter, the voter, the collector, the processor, the mixer, the auditor, customer support, the voter list compiler and the editor.

Information system components are the collector service, the processing application, the key application and the audit application. The voter application, the verification application and the mixing application are closely linked to the information system.

In its operation, the information system uses external services: the identification service, the signing service and the registration service.

1.2 IVXV Cryptographic Protocol

To achieve a situation where online voting is safe and verifiable, voting is secret and correct, and voters are independent, the cryptographic protocol of online voting [HMVW16] is strict. The protocol gives an essential and adequate overview of the structure and security aspects of IVXV. The components of IVXV implement the subparts of the cryptographic protocol.

The IVXV cryptographic protocol is also specified using the notation that allows formal verification with the protocol security aspects system [ProVerif].

1.3 Notation

In this document, we use UML schemas to illustrate the draft architectural solution, where we differentiate between the following aspects of features – actors, interfaces, components – with colors and <<>> markings:

- Marking <<IVXV>> (yellow) – the interface or component of the information system is defined/implemented in the course of work done for a specific procurement
- Marking <<External>> (red) – in implementing some functionality, the information system depends on the component of a third party or an existing interface, redefining which requires efforts by third parties
- Marking <<NEC>> (brown) – similar to the previous one, but NEC is the owner of the interface/component
- Marking <<Not defined>> (black) – an interface important to the information system has not been defined

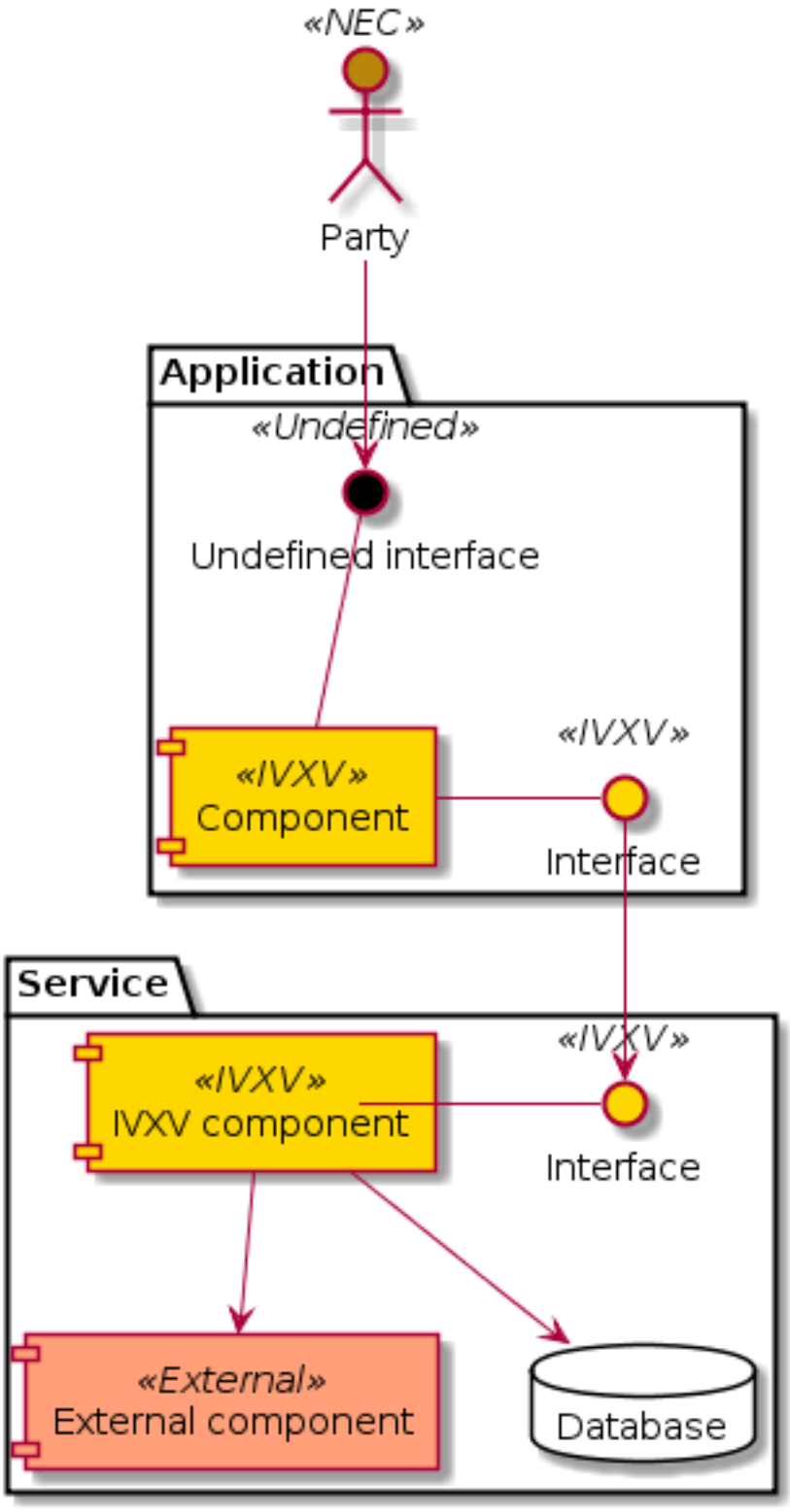


Fig. 1.1. Example schema

CHAPTER 2

Collector Service

Based on the general specification [ÜK2016], the collector service is a central element in the system, run by the collector. The service helps the voter compile an i-vote and registers it in the ballot box before storing it. The collector service uses external services (identification, signing, registering). The collector service has other administrators apart from the collector (organizer, customer support), for whom the collector service has separate administration interfaces.

The collector service works online, and at least the interfaces directed towards the voter and verification applications are open to the internet. Thus, the collector service handles requests that potentially originate from an untrustworthy source. Due to the security requirement of the software, and the requirements of high availability, scalability, multitier implementation and extendability, the collector service itself is divided into microservices providing one specific service that can be implemented flexibly.

All components of the collector service are programmed in Go[1]. The Go programming language has:

- Static typing, which allows finding type errors before the program is launched
- Automatic memory management, which avoids security risks arising from poor memory management
- Compiler with an open source code and
- Concurrent programming features allowing (inherent) parallelism in multicore systems

Generally, JSON is used as the data transmission format of the collector service, except in situations where external circumstances require the use of another data format – the BDOC format, for example, is based on XML.

[1] https://golang.org

The collector service supports Riigikogu elections, local government council elections, European Parliament elections and referendums.

The components of the collector service consider the use of virtualization technologies, and the collector service can be implemented on both one virtual hardware instance as well as by microservices on different instances. Collector service components can be implemented in the Ubuntu LTS 16.04 operation system 64-bit architecture.

Data retention has been implemented using the key-value database (etcd). For test purposes, data has also been stored in the file system and memory, but it is not recommended to use these in the product environment. The collector service also has an interface for adding new storage protocols. The final decision regarding the solution to be used is made by the collector service administrators when configuring the service.

2.1 Microservices

The collector service is divided into main services and support services. The main services – proxy service, choices service, voting service, verification service and storage service – are limited to one election in the interests of the simplicity of the architecture, but the microservices of several elections can go on one hardware, in one operation system. Additional support services can be used for the collector service – the identification service to identify the voter's person, and the signature service to make it easier for the voter application to sign a vote.

The services can be implemented separately as well as together in various configurations, which makes multitier architecture possible. Depending on the function, it is practical to keep proxy and storage services separate from the others.

The services use TLS and all connections are authenticated on both sides. The application layer uses the JSON-RPC (protocol).

All services create an action log that is kept both locally and logged using the rsyslog interface.

Proxy Service: Function and Technical Interface

The main function of the proxy service is to offer one entry point (port 443) to the voter application and the verification application. The proxy service is a dispatch service between other components that allows implementing the collector service internally as microservices, while having only one entry point to the system. In addition, in case of double implementation, it can act as a load balancer.

The proxy service does not terminate the TLS connection, but uses the TLS *Server Name Indication* (SNI) extension to identify the target. Clients put an SNI extension in the TLS `ClientHello` message, where they use open text to determine the service with which they want to communicate: the proxy service sees that, contacts the entity providing the relevant service, and starts sending messages between the client and the service. The proxy service does NOT terminate TLS and cannot see the content

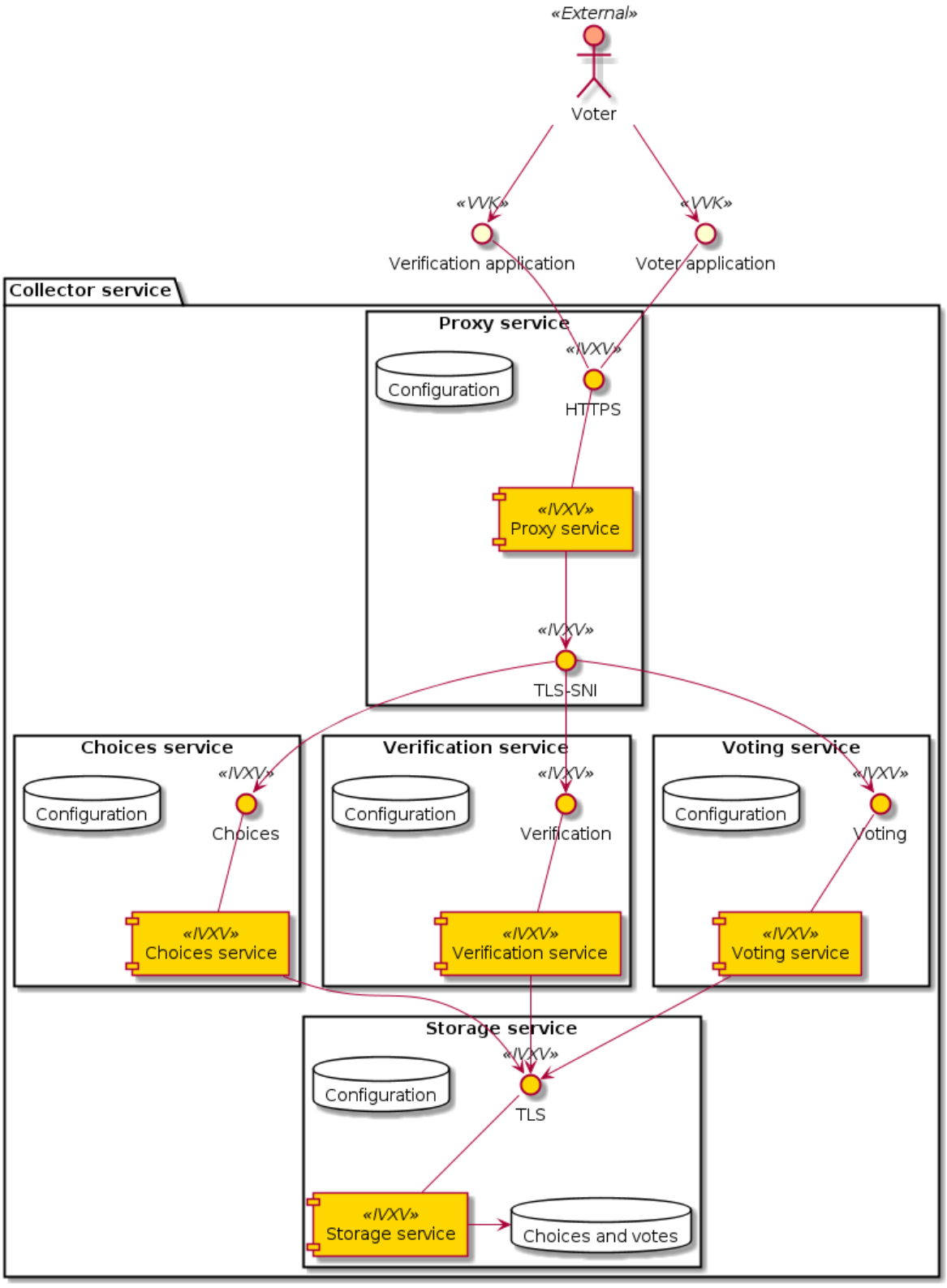


Fig. 2.2. The collector service divided into microservices

of the messages. The proxy service has data on the locations of all other services (aadress:port) and the service sends messages between all parties.

The proxy service is a status-free component, which can be scaled horizontally.

Application of the Proxy Service

The proxy service is implemented using the free software HAProxy server, which is a common load balancer and proxy. Since the proxy service is the first access point for connections coming from the public internet, it is sensible to use software that has proven to be reliable.

Although HAProxy is often used in HTTP, where it analyses traffic, in the role of the proxy service it operates in the TCP mode and cannot see inside the encrypted TLS channel it proxies.

IVXV settings are used to generate the HAProxy configuration file, which contains the locations of other services, and the task of relaying connections is left to the latter. In addition, HAProxy can also be configured to limit the connection frequencies on the basis of the source address or another denominator. However, this is the system administrator's task.

Even though HAProxy is able to function as a load balancer, it can be implemented behind other, potentially hardware-type load balancers, where it performs only the task of SNI-based relaying.

The HAProxy source code is public under GPL v2 and version 1.6.3 is packaged in the official storage of Ubuntu 16.04 (see *Technology Used*).

Choices Service: Function and Technical Interface

The main function of the choices service is to send lists of choices to the voter application. Information on the identified voter reaches the choices service and the choices service issues the list of choices that corresponds to the voter's district from the storage service to the voter application.

The choices service is a status-free component that can be scaled horizontally.

Verification Service: Function and Technical Interface

The main function of the verification service is to process verification requests and issue the vote to be verified from the storage service to the verification application.

The verification service is a status-free component that can be scaled horizontally.

Voting Service: Function and Technical Interface

The main function of the voting service is to process voting requests. The voting service verifies the incoming vote, registers it in the registering service and stores it in the storage service.

The voting service is a status-free component that can be scaled horizontally.

Storage Service: Function and Technical Interface

The main function of the storage service is to execute the long-term retention of choices, voter lists and votes.

Storage technology that allows for distributed data retention has to be used for the horizontal scaling of the storage service.

Application of the Storage Service

The storage service is not aware of the IVXV protocol or the specifics of the data to be retained; it is a general-use key-value database for storing binary data. All the knowledge on the structure of the data to be retained and the hierarchy of keys is in other services that use the storage service and act as “smart” clients.

This approach allows using any common key-value database as the storage service without much trouble: the only tasks are to convert IVXV settings into a format suitable for the database and to launch the service. The database software only has to allow retention and reading based on the key, listing the keys by a prefix, and an automatic (*compare-and-swap*) operation.

The storage service is an important determiner of the working speed of the collector service; that is why the hardware providing that service affects the performance of the entire system and should be dimensioned according to the database used.

At the moment, the only storage service application that is intended as a product uses the distributed key-value database etcd. The recommendations[2] of the authors of the etcd hardware should be followed.

Identification Service: Function and Technical Interface

The main function of the identification service is to identify the voter’s identity. The identification service is necessary when the user uses mobile ID authentication, for example.

[2] https://coreos.com/etcd/docs/latest/op-guide/hardware.html

Signing Service: Function and Technical Interface

The function of the signing service is to support the voter application in signing a vote. The signing service is necessary when the user uses mobile ID to sign, for example.

Mobile ID Support Service Execution

The composition of IVXV includes the mobile ID support service, which acts as both the identification service and signing service for mobile ID. The voter application sends IVXV requests to the mobile ID support service, which converts them into mobile ID requests and sends them to the mobile ID service provider.

In case of successful mobile ID identification, the support service issues to the voter application a ticket that can be used to confirm the voter's identity to other services. A user can only vote once with each ticket.

In case of signing, the voter application only sends the hash of the vote to be signed to the mobile ID support service, and uses the signature received in return in the same way as a signature created with an ID card.

The mobile ID support service does contain a status on unfinished identification sessions, but otherwise, it is a status-free component. Thanks to this, the mobile ID support service can be scaled horizontally, provided that all requests from one identification session are sent to the same entity.

Application of Collector Service Microservices

The collector service microservices are minimally dependent on external packages. The necessary dependencies:

- SSH server to conduct administrative activities (the administrative service uses it to manage microservices)
- rsyslog service to collect logs to the log collection service

The collector service microservices are packaged in the .deb format, and can also be implemented as docker-type containers.

2.2 External Services and Extendability

The collector service microservices use extension modules to execute various mechanisms to identify the voter and verify and edit digital signatures, incl. to register a vote. Extension modules can, in order to enable the execution, use external services. In the interests of the extendability of microservices, Go API has been defined that can be used to implement additional modules as well. At the moment, the following modules are implemented:

- Authentication with a TLS certificate (ID card)

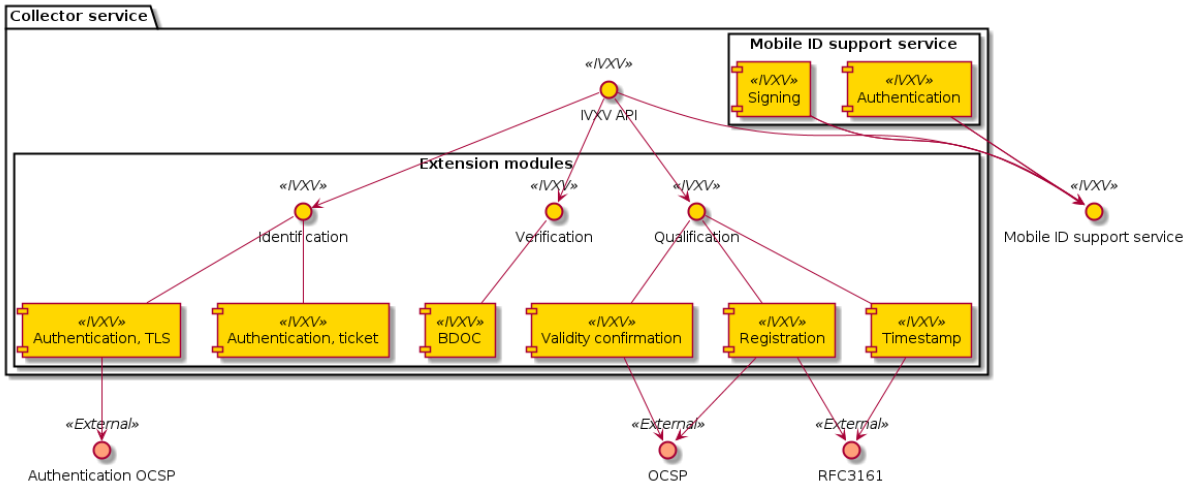


Fig. 2.3. Collector service extension modules and external services

- Authentication with an identification service ticket (mobile ID)
- BDOC verification
- Certificate status service OCSP
- Timestamp service RFC 3161
- Registration service OCSP
- Registration service RFC 3161

The registration service takes center stage in the IVXV cryptographic protocol, as it also takes part in the long-term retention of votes.

Registration Service: Function

The main function of the registration service is to accept signed registration requests from the voting service, confirm them with its own signed response, and store them at least until the end of the voting period for auditing at a later stage.

When resolving potential differences that arise during auditing, it is important that

- The registration service is able to prove that each confirmation it issued was preceded by a registration request from the storage service
- The storage service is able to prove that there is a registration service confirmation for every vote it has retained

Sufficient protocol to achieve such a level of verification occurs when both parties have a key pair for signing, the requests and responses are signed, and each party keeps a register of the messages of the other party. Such a protocol can be implemented, for example, in case of an OCSP-based registration service. However, there may be cases where it is impossible to sign registration requests with standard means – RFC 3161-based registration – and then the proof necessary for the registration service has to be provided with other organizational and technical means.

At the moment, the registration service has two different implementations:

- The OCSP requires the use of the OCSP-based timestamping service implemented in Estonia, where the nonce of the signed OCSP request is the hash of the vote issued by the voting service. The request is signed using standard OCSP means
- The RFC 3161 component, in case of which, as a non-standard solution, the nonce of the timestamp request is the hash of the vote signed by the voting service

Adding Collector Service Extension Modules

The API of the collector service defines six types of extension modules:

- Personal identification (Go package `ivxv.ee/auth`, e.g. `tls`),
- Deriving the voter identifier from the identified person certificate (Go package `ivxv.ee/identity`, e.g. `serialnumber`),
- Deriving the voter's age from the identifier (Go package `ivxv.ee/age`, e.g. `estpic`),
- Signed container verification (Go package `ivxv.ee/container`, e.g. `bdoc`),
- Signature verification (Go package `ivxv.ee/q11n`, e.g. `tspreg`) and
- Data retention protocol (Go package `ivxv.ee/storage`, e.g. `etcd`).

To add a new module, a new module identifier has to be added to the module package, as well as a module execution sub-package. At the initial loading of the sub-package, the `Register` function of the module package has to be invoked to register the module.

To use a new module, its identifier has to be added to the settings under the relevant module type with the sub-module setting. The extension module is given a configuration block referred to by its identifier, and the block will be processed further within the module.

The module packages and the interfaces required from their modules are specified in more detail in the document `IVXV API`. In addition, there is at least one application per module that can be used as an example.

2.3 Monitoring

Logging

The log generated by each microservice is defined systematically based on the protocol specification and the status diagram of service provision. At a minimum, the following is logged:

- The fact of receiving each request and the start of processing
- Giving processing over to an external component

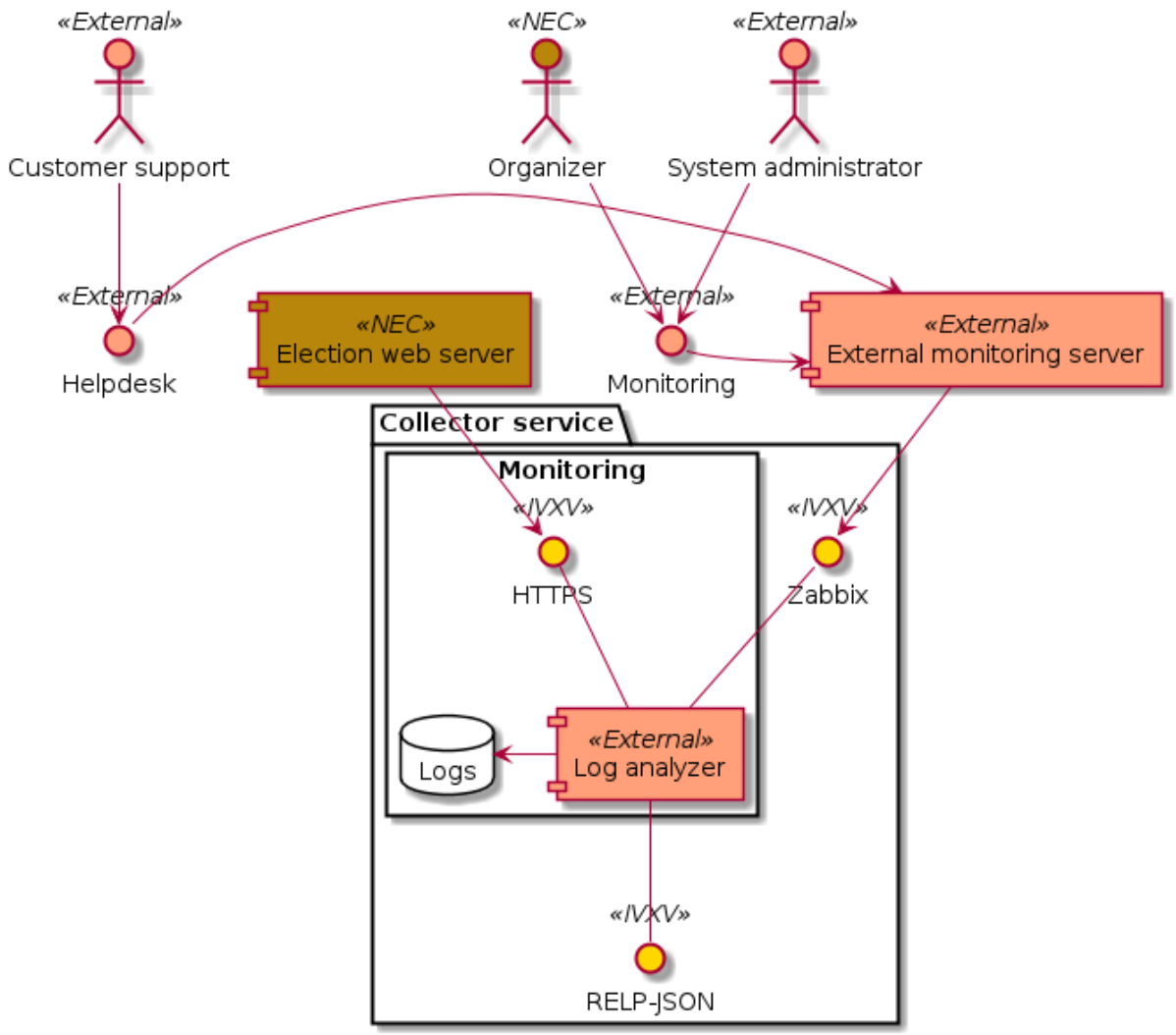


Fig. 2.4. Monitoring solution

- The return of processing to the components
- End of processing the request and the result
- Going through additional important stages in the process status model

The following principles are adhered to when logging:

- The rsyslog service is used for logging; it only takes a millisecond to register the moment when a log message is written
- Once a session begins, the system generates a unique identifier that the client application uses for its requests when addressing the central system
- All log entries under one session contain the same session identifier
- The log entry can be uniquely identified
- For each logged message, the unique identifier allows identifying the location where the message was created in the monitored system
- The log entry is in the JSON schema format; for automatic monitoring, machine-readability is primary and human-readability secondary

- The information to be logged is sanitized (urlencode), and given a length limit (total length and by parameters)
- Information coming from outside the system perimeter is only logged in a sanitized form, only in the prescribed length

Since logging is done via rsyslog, it is possible to use the Guardtime module to ensure the integrity of logs.

General Statistics

The statistics web interface is used to monitor the following statistics:

- The number of successfully collected votes / number of voters
- Voters by gender, age group, operation system and means of authentication
- Number of successfully verified votes/voters
- Repeat votes
- Voters by country based on their IP addresses

Detailed Statistics

Detailed statistics are aggregated based on logs using the SCCEIV log analyzer, which analyses the action log of applications in relation to the predefined profile, and allows performing session/error code-based analysis.

Detailed statistics are available using the HTTPS interface.

2.4 Administration

The collector service administration is done using digitally signed configuration packages.

The collector service provides two interfaces for loading configuration packages:

- Command line interface – the application verifies the signature, validates that the commands are in line with and match the collector service status. The command is implemented using a separate utility
- Web interface – sends the configuration package to the command line interface and returns to the user information on the result of the loading process. If the loading was successful, the configuration package is implemented automatically and is based on the same principles

The functions of the web interface are:

- Monitoring the status of the collector service microservices
- Managing election lists

- Displaying statistics on the progression of e-voting
- Managing the users of the administration service
- Displaying the collector service administration log

All commands sent to the application are kept – even the ones that were not implemented. Faulty commands (those that cannot be validated) are not kept.

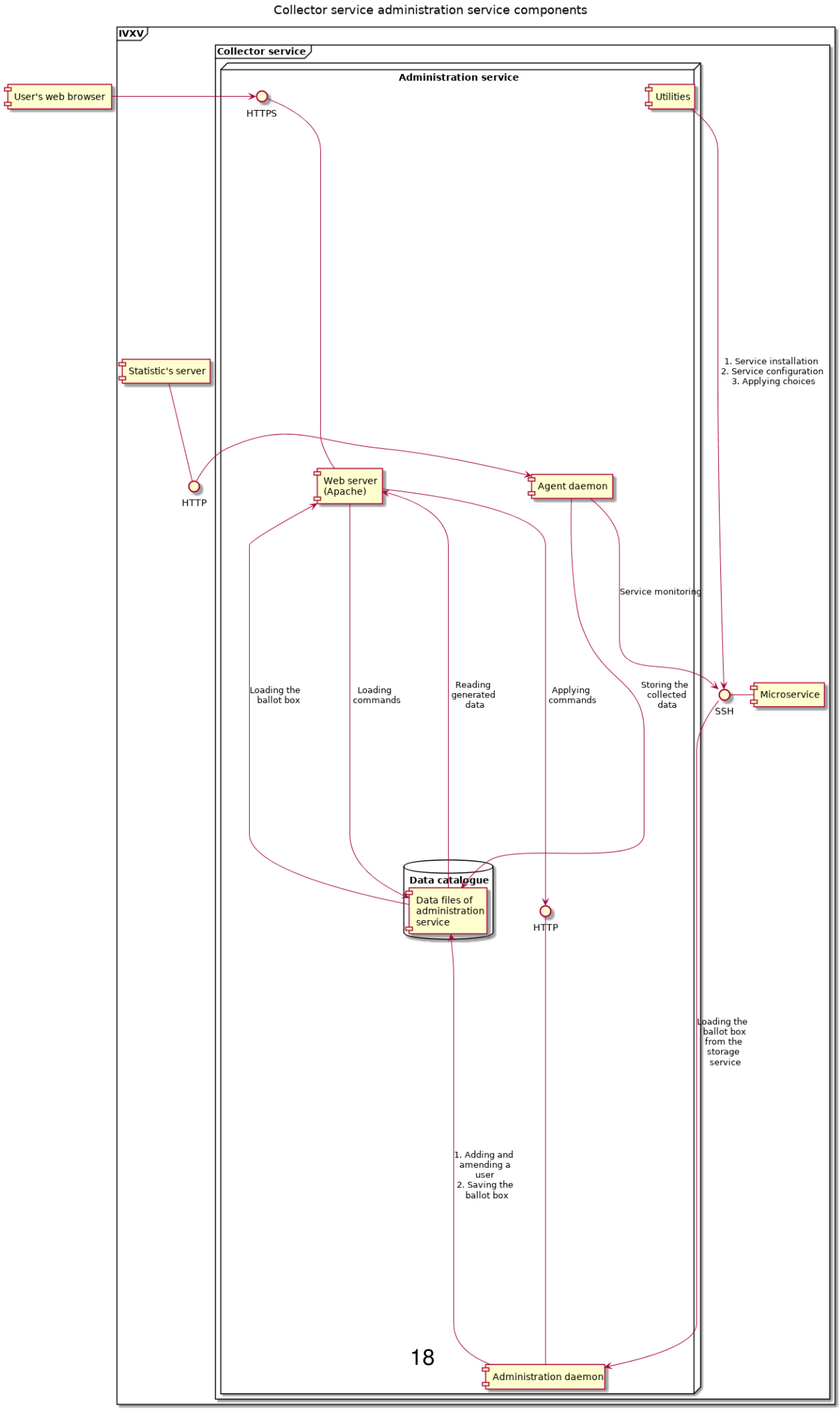
The collector service may perform the following actions automatically:

- Preparing stored votes, logs and settings for backup and archiving them in the backup service

Administration Service Components

1. **The administration web server** is an Apache server operating in the `www-data` rights of a system user, and its tasks are:
    1. Initial servicing of HTTPS requests from users:
        1. Proving the reliability of the administration service (TLS certificate)
        2. User authentication
    2. Serving generated webpages and data files from a data repository
    3. Filling a general background data request response with the data of the logged-in user (WSGI)
    4. Initial validation of uploaded commands and sending them to the administration daemon, and sending the administration daemon's relevant responses to the client (WSGI)
2. **The administration daemon** is a web server operating in the `ivxv-admin` admin rights of a user account and listening on the local (`localhost`) interface, and its tasks are:
    1. Validating the commands that are uploaded
    2. Directly applying the uploaded commands (user administration)
    3. Storing uploaded commands for later application (to apply settings and voting lists to the service)
    4. Proxy the downloading of the ballot box
3. **The agent daemon** is a web server operating in the `ivxv-admin` rights of a user account and its tasks are:
    1. Data collection and registration:
        1. The status of known microservices
        2. Downloading activity monitoring statistics
4. **The data repository** is a catalogue on the file system, where the administration service components keep the collected and generated data (for a more detailed specification, see the annexes to the `IVXV collector service administration guide`)

External components with which the administration service comes into contact:


Collector service administration service components
IVXV
Collector service
Administration service
User's web browser
HTTPS
Utilities
Statistic's server
1. Service installation
2. Service configuration
3. Applying choices
HTTP
Web server (Apache)
Agent daemon
Service monitoring
Loading the ballot box
Loading commands
Reading generated data
Applying commands
Storing the collected data
SSH
Microservice
Data catalogue
Data files of administration service
HTTP
Loading the ballot box from the storage service
1. Adding and amending a user
2. Saving the ballot box
Administration daemon

1. **Collector service subservices** - installation, configuration and status data collection is done via the agent daemon (SSH connection into the service machine)
2. **Monitoring server** - downloading general statistics data to display it in the administration service

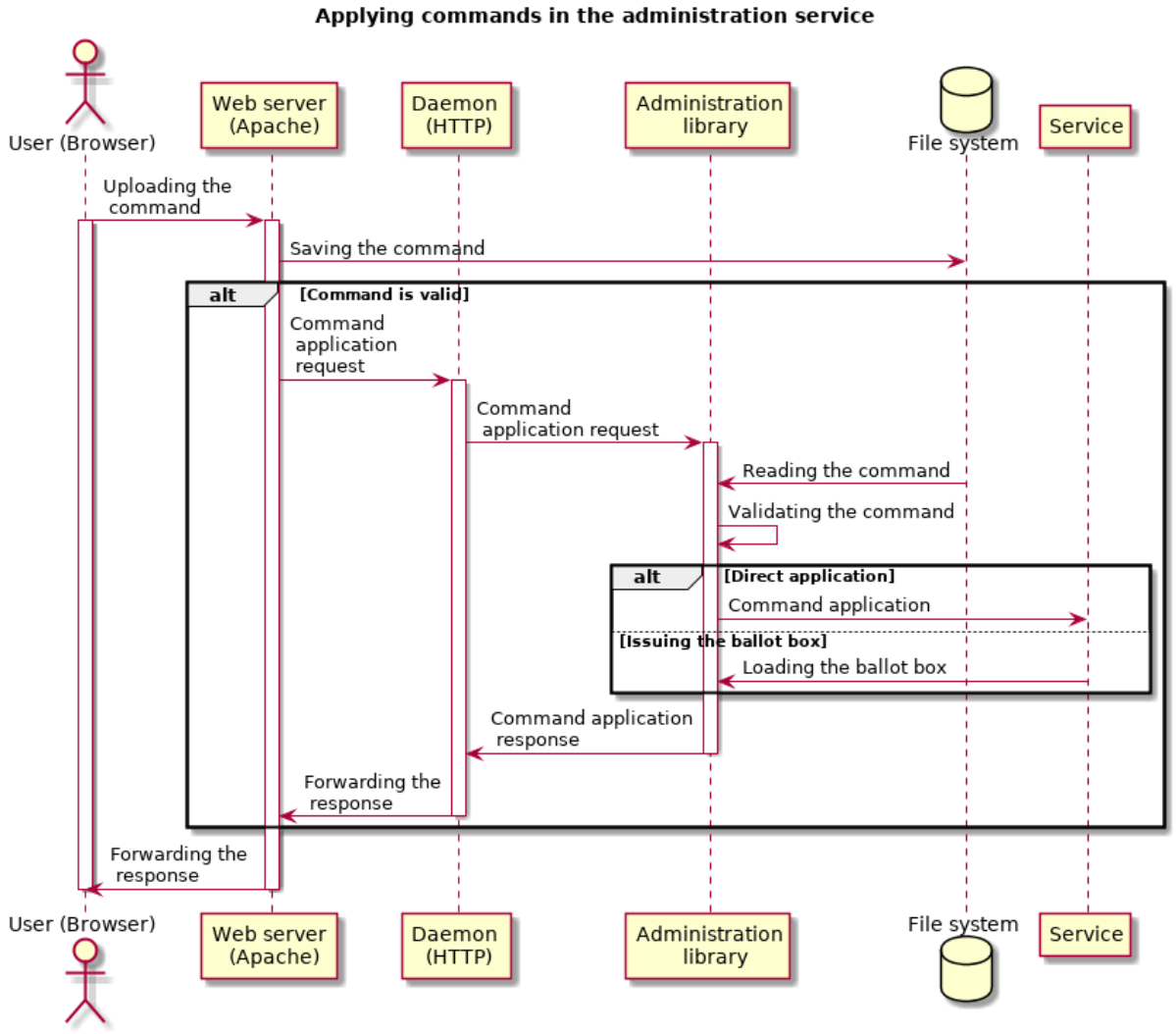


Fig. 2.6. Loading commands into the administration service

2.5 Collector Service Statuses

The status of the collector service reflects the status of all the subservices of the service, the status of the external services used, and the overall status derived from the above. The administration service is responsible for identifying the status of the collector service.

The overall situation statuses are:

1. **Not installed** - the status after the installation of the administration service until the installation of all subservices

2. **Installed** - all subservices are installed, technical settings have been applied, as well as the cryptokeys required for the operation of the service. Election settings have not been applied (but can be loaded into the administration service)
3. **Configured** - the collector service has been configured and is operational, it can be used to conduct voting and issue the ballot box
4. **Partial failure** - the collector service has been configured and is partially operational, some subservices are not operational, but it does not hinder the operation of the collector service
5. **Failure** - an important node of the collector service is not operational, it is impossible to provide the service as required

Statuses of the Subservices of the Collector Service

Changes in the Collector Service Status

The status of the collector service can be monitored as of the successful installation of the administration service, the original status is **Not installed**.

Not installed

The trust root and the technical configuration are applied to the collector service:

1. The settings are loaded into the collector service
2. The subservices specified in the technical configuration are installed
3. The trust root and the technical configuration are applied to the subservices

If the configuration is applied successfully, the new status of the system is **installed**.

Installed

The collector service settings are applied to all subservices, but the election settings are not applied. Election settings are loaded into the administration service and applied to the subservices.

If the election settings are applied successfully, the new system status is **configured**.

Configured

All subservices of the collector service are configured and operational. The administration service has fresh status reports from all subservices. The system can be used to conduct the voting and issue the ballot box.

If a failure is found in the system, the new status of the system is **partial failure**.

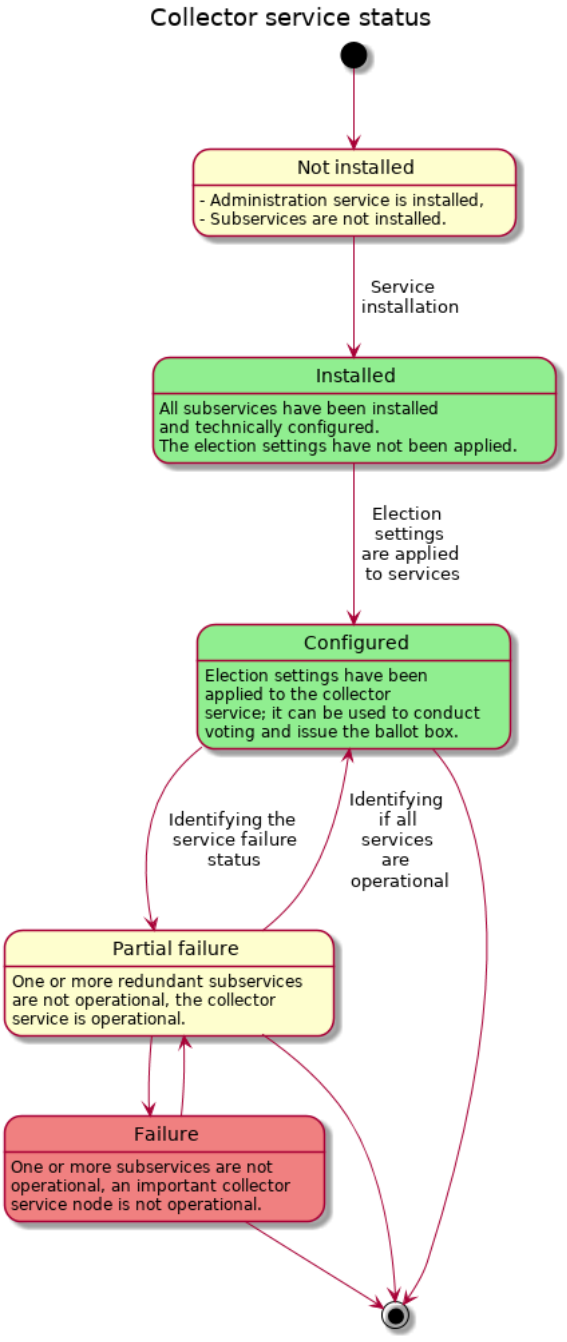


Fig. 2.7. Status diagram of the collector service. Statuses according to color: yellow – being configured, red – failure, green – operational

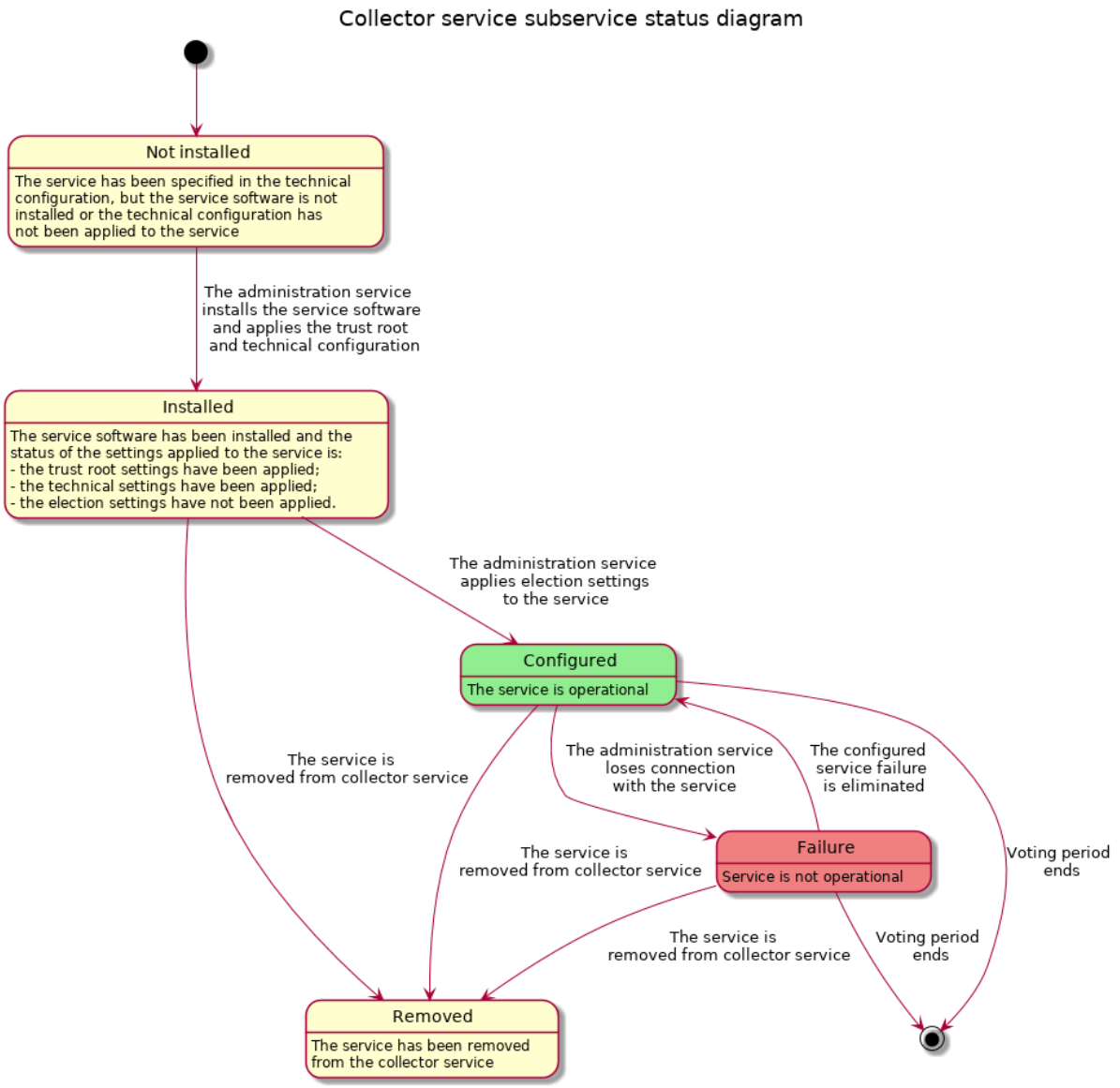


Fig. 2.8. Status diagram of a subservice registered by the administration service. Statuses according to color: yellow – being configured, red – failure, green – operational

The system can never go from the status **configured** back to the statuses **not installed** või **installed**, although if new subservices are added (until they are **not installed**/**installed**), the relevant criteria would be met.

Partial Failure

The system is configured and partially operational, some doubled parts of the system are not operational, but this does not keep the system from functioning.

If the failure worsens to the limit where the system is no longer able to provide the service, the new status of the system will be **failure**. Once all failures are eliminated, the new status of the system will be **configured**.

Failure

A failure is identified in the configured system that prevents service provision.

Once the failures are eliminated to the point where the system can be used to provide the service, the new status of the system will be **partial failure**.

Removed

The service has been removed from the configuration.

CHAPTER 3

Applications

3.1 General Principles

All applications are applications with a command line interface, packaged to function in the operation system Windows 7 (or newer). The user interfaces of components are single-language. The components are delivered in Estonian, and can be translated using a translation file.

The applications are programmed in Java.

Applications communicating with external information systems make maximum use of the existing interfaces/data structures.

The applications get their input from the application settings and from the file system files shown in the settings, and save their output into the folder specified by the user in the file system. The files can also be located on a main memory disk.

Relevant applications support the ElGamal cryptosystem in the multiplicative group of integers and the P-384 elliptic curve. The decryption proof is implemented on a protocol based on the Schnorr zero-knowledge proof.

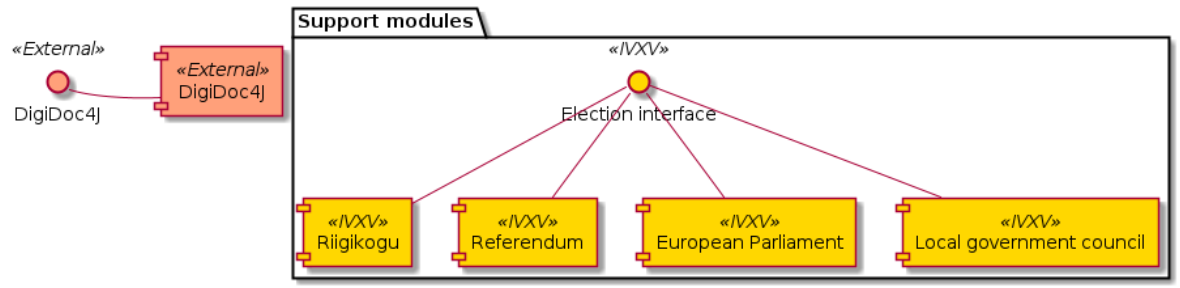


Fig. 3.9. Application support modules

The election interface is unified for applications that allow implementing various election types as modules. The functionality of verifying digital signatures is created using the digidoc4j[3] library. The use of support modules is not highlighted separately in the diagrams below.

Application Configuration

Applications are configured with either a digitally signed configuration package or with command line keys. Command line keys do not support entering a configuration with a hierarchical nature. The settings in a configuration package are specified in YAML:

```
check:
  ballotbox: votes.zip
  ballotbox_checksum: votes.zip.sha256sum.bdoc
  districts: TESTKOV2017.districts.json
  registrationlist: register.zip
  registrationlist_checksum: register.zip.sha256sum.bdoc
  tskey: ts.pub.key
  vlkey: test.gen.pub.key
  voterlists:
    -
      path: 00.TESTKOV2017.gen.voters
      signature: 00.TESTKOV2017.gen.voters.signature
    -
      path: 03.TESTKOV2017.gen.voters
      signature: 03.TESTKOV2017.gen.voters.signature
    -
      path: 06.TESTKOV2017.gen.voters
      signature: 06.TESTKOV2017.gen.voters.signature
    -
      path: 09.TESTKOV2017.gen.voters
      signature: 09.TESTKOV2017.gen.voters.signature
  election_start: 2017-05-01T12:00:00+03:00
  out: out-1
squash:
  ballotbox: out-1/bb-1.json
  ballotbox_checksum: out-1/bb-1.json.sha256sum.bdoc
  districts: TESTKOV2017.districts.json
  out: out-2
revoke:
  ballotbox: out-2/bb-2.json
  ballotbox_checksum: out-2/bb-2.json.sha256sum.bdoc
  districts: TESTKOV2017.districts.json
  revocationlists:
    - 12.TESTKOV2017.gen.revoke.json
    - 13.TESTKOV2017.gen.revoke.json
    - 14.TESTKOV2017.gen.revoke.json
```

(continues on next page)

[3] https://github.com/open-eid/digidoc4j

(continued from previous page)

```
    - 15.TESTKOV2017.gen.revoke.json
  out: out-3
```

Input Consistency Check

All applications perform an input consistency check for the configuration depending on the settings they use:

1. Loading certificate configuration
2. Verifying the digital signature of the configuration
3. Verifying the district list
4. Verifying the consistency of the district list
5. Loading the district list
6. Verifying the list of choices
7. Verifying the consistency of the list of choices
8. Loading the list of choices
9. Verifying the voter lists
10. Verifying the consistency of the voter lists
11. Loading the voter lists

3.2 Key Application

The key application is an application used to generate vote encryption and decryption key for each vote, and to count the votes and issue the result.

The key application uses the [DesmedtF89] threshold scheme, which is based on a trustworthy part distributor and uses the Shamir secret sharing, which is safe in an information theoretical sense in case of a $t < M$ party, where M is the threshold.

The key shares are generated in the main memory and saved on a chip card using the PKCS15 interface.

The key application input for generating the key is:

- The key pair identifier
- The cryptosystem ElGamal specification – the multiplicative group of integers or the P-384 elliptic curve and key length
- The M-N threshold scheme specification that has to meet the rule $N >= 2*M-1$
- N PKCS15-compatible chip cards

The key application output for generating the key is:

- A self-signed certificate

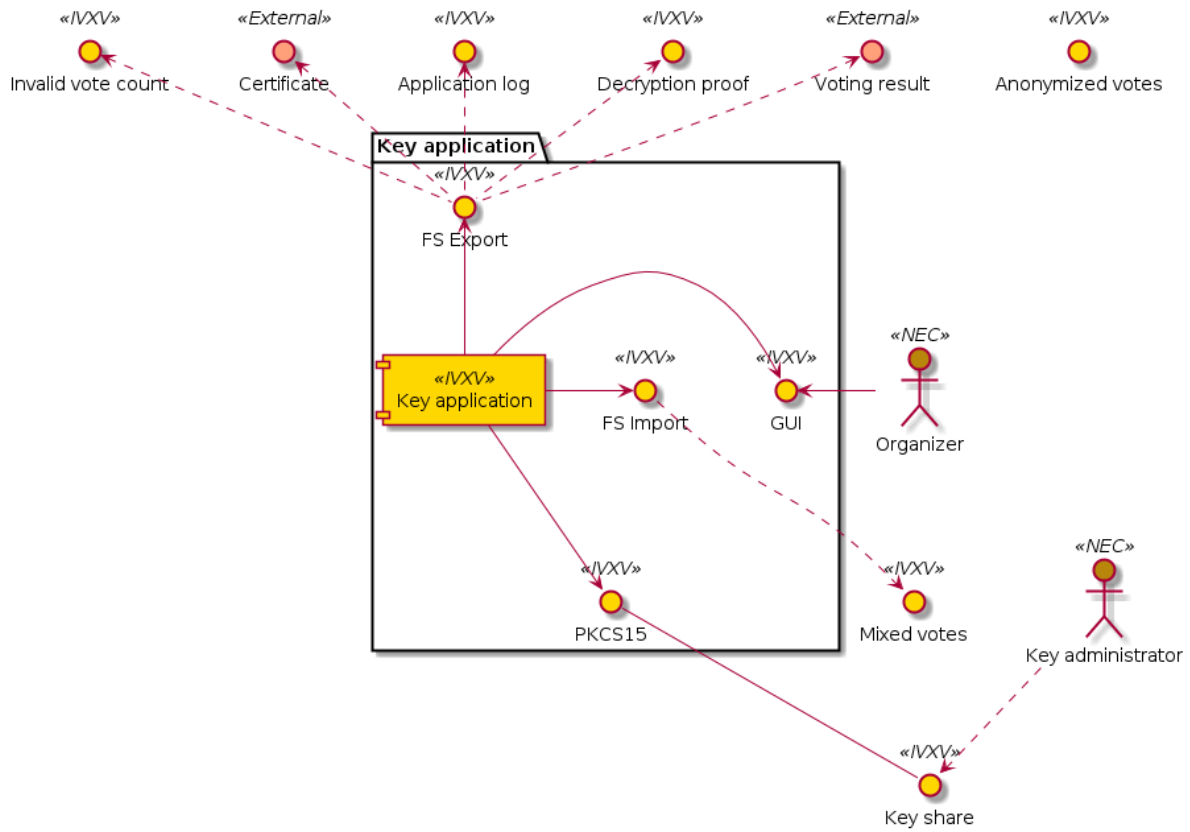


Fig. 3.10. Key application interfaces

- N key shares saved on chip cards
- The application's detailed action log
- The application's detailed error log

The key application input for counting votes is:

- Mixed votes
- Key pair identifier
- M key shares pursuant to the threshold scheme specification

The key application output for counting votes is:

- The signed voting result
- Invalid vote count
- The decryption proof (protocol based on the Schnorr zero-knowledge proof, referred to in the contract documents)
- The application's detailed action log
- The application's detailed error log

In addition to the interfaces and dependencies defined earlier, the processing application uses a third-party library to implement the PKCS15 interface. The specific library is selected in the design stage.

3.3 Processing Application

The processing application is an application used to verify, cancel and anonymize the votes collected over the voting period, which functions according to section 7.6 of the General Description.

The processing application input is:

- Electronic votes stored by the collector service
- Timestamps issued by the registration service
- Voter lists
- District list
- Revocation lists
- Restoration lists

The processing application output is:

- The application's detailed action log
- The application's detailed error log
- The list of online voters in a PDF format, depending on the processing stage
- The list of online voters in a format that can be processed by a machine, depending on the processing stage
- Anonymized votes

In addition to the interfaces and dependencies defined earlier, the processing application uses a third-party library to implement the functionality of issuing PDF files.

Fig. 3.11. Processing application interfaces

Full Processing of Electronic Votes

For the full processing of electronic votes, the processing application compares the number of votes stored by the collector service to the number of votes stored by the registration service, verifies that the stored votes comply with the election configuration, identifies the votes to be counted, and anonymizes them for delivery to the key application.

1. Loading application settings
2. Verifying the digital signatures of electronic votes
3. Verifying the registration service confirmations
4. Verifying the timestamps
5. Identifying the latest valid vote for each voter
6. Issuing an initial PDF list of the people who voted online
7. Verifying the revocation and restoration lists
8. Checking the consistency of revocation and restoration lists
9. Implementing the revocation and restoration lists
10. Generating a list of votes to be mixed, separating ciphertexts from digital signatures
11. Issuing a final list of people who voted online in a machine-readable format

Generating a List of Electronic Voters

1. Loading application settings
2. Verifying the digital signatures of electronic votes
3. Issuing an initial PDF list of the people who voted online

3.4 Audit Application

The audit application (Figure 9) is an application that mathematically verifies that the vote count is correct, and if mixing is used, that the mixing is also correct.

The audit application input is:

- Anonymized votes
- Mixed votes
- Shuffle proof (Terelius-Wikström, Verificatum)
- Voting result

The audit application output is the application's detailed action log, which also contains an assessment on the complete success of the audit. If necessary, the application's detailed error log is also issued.

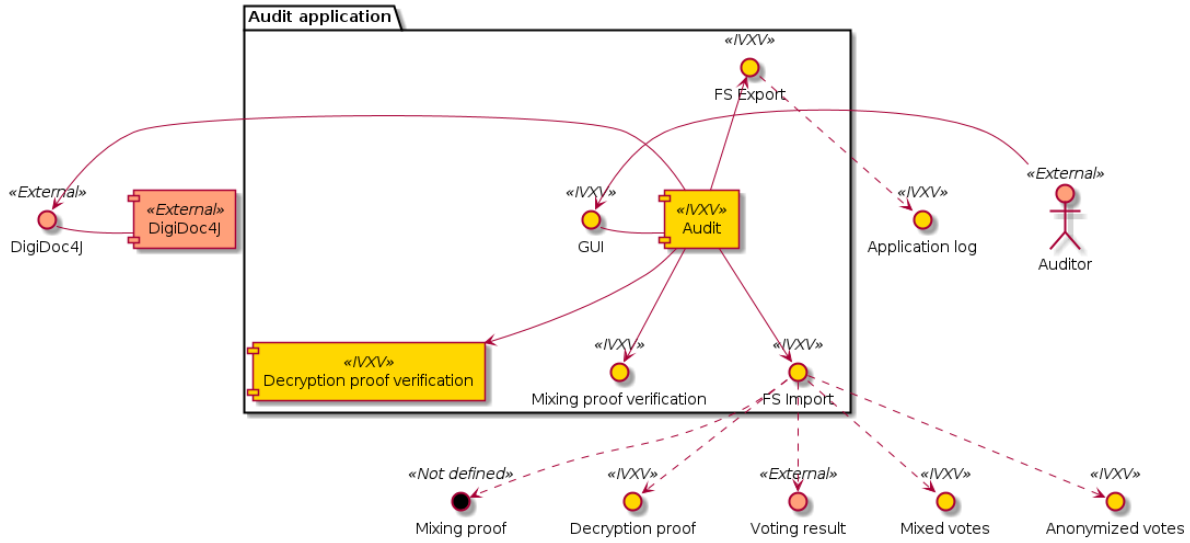


Fig. 3.12. Audit application interfaces

CHAPTER 4

Technology Used

4.1 Collector Service Programming Language

The core functionality of the collector service is programmed using the Go programming language, which meets the following procurement requirements:

- Static typing
- Automatic memory management
- Open source code compiler
- Concurrency

The collector service administration service is programmed in Python.

4.2 Programming Language of Applications

The applications are programmed in Java, which meets the procurement requirements regarding the widespread nature and sustainability of the programming language.

4.3 Project Dependencies

The third-party components used in the project together with their motivated usage need are listed in the tables below. There are separate tables for packaging and operating the framework as well as developing and testing the framework.

All external libraries used in the IVXV project are available in the `ivxv-external.git` repository or on the platform on which the application will be operating.

All components used in the collector service have an open source code.

Table 4.1. Third-party components used for the work of the IVXV framework

Name	Version	License	Usage need
Bootstrap[4]	3.3.7	MIT	Design of the user interface of the collector service administration service
Bouncy Castle	1.58	MIT	ASN1 handling, support functions of the data type BigInteger
Bottle[5]	0.12.13	MIT	Framework for executing the collector service administration service web interface
CAL10N	0.7.7	MIT	Multilanguage support, translation file validation
Digidoc 4j	2.1.0.	LGPL	BDoc container handling
Digidoc 4j DSS	5.2.d4j.3	LGPL	Digidoc 4j dependency
Apache Commons (cli 1.4, codec 1.10, collections 4.1, io 2.5, lang 3.6, logging 1.2, compress 1.3)	•	Apache License v2.0	Digidoc 4j and PDFBox dependencies
Apache HttpComponents	4.5.3	Apache License v2.0	Digidoc 4j dependency
Apache Santuario	2.0.9	Apache License v2.0	Digidoc 4j dependency
Google Guava	20.0	Apache License v2.0	Digidoc 4j dependency
JDigiDoc	3.12.1	LGPL	Digidoc 4j dependency
StaX	1.0-2	Apache License v2.0	Digidoc 4j dependency
log4j	1.2.6	Apache License 2.0	Digidoc 4j dependency
Woodstox	4.4.1	Apache License 2.0	Digidoc 4j dependency
Xalan-Java	2.7.2	Apache License 2.0	Digidoc 4j dependency
Xml Apis	1.3.04	Apache License 2.0	Digidoc 4j dependency

Continued on next page

Table 4.1 – continued from previous page

Name	Version	License	Usage need
Docopt[6]	0.6.2	MIT	Execution of the command line interface of the collector service administration utilities
etcd[7]	3.2.17	Apache License v2.0	Distributed key value database used as a storage service
github.com/ghodss/yaml[8]	73d445a	MIT	etcd client library dependency
gopkg.in/yaml.v2[9]	4c78c97	Apache License v2.0	github.com/ghodss/yaml dependency
github.com/golang/protobuf[10]	224aaba	BSD 2.0	etcd client library dependency
github.com/grpc-ecosystem/go-grpc-prometheus[11]	6b7015e	Apache License v2.0	etcd client library dependency
github.com/grpc-ecosystem/grpc-gateway[12]	6863684	BSD 2.0	etcd client library dependency
google.golang.org/grpc[13]	1.0.4	Apache License v2.0	etcd client library dependency
golang.org/x/net[14]	f249948	BSD 2.0	etcd client library dependency
Prometheuse klientteek[15]	0.8	Apache License v2.0	etcd client library dependency
github.com/beorn7/perks/quantile[16]	4c0e845	MIT	Prometheus client library dependency
github.com/matttproud/golang_protobuf_extensions[17]	1.0.0	Apache License v2.0	Prometheus client library dependency
Gradle	3.0	Apache License v2.0	Java applications build tool
HAProxy[18]	1.8.8	GPL v2	TCP proxy used as a proxy service
IvyPot	0.4	Apache License v2.0	A Gradle build tool extension for managing dependencies and building applications offline
Jackson	2.8.9	Apache License v2.0	Reading and writing JSON files
jQuery[19]	3.1.0	MIT	User interface of the collector service administration service
Logback	1.2.3	Eclipse Public License v1.0 or LGPL v2.1	Logging API SLF4J implementation

Continued on next page

Table 4.1 – continued from previous page

Name	Version	License	Usage need
Logback JSON	0.1.5	Eclipse Public License v1.0 or LGPL v2.1	Logback logger extension for compiling log entries in the JSON schema format using the Jackson library
metisMenu[20]	1.1.3	MIT	User interface of the collector service administration service
PDFBox	2.0.8	Apache License v2.0	PDF report generation support for Java applications
PyYAML[21]	3.12	MIT	Collector service configuration files' processing support for the administration service
Schematics[22]	2.0.1	BSD	Collector service configuration files' validation support for the administration service
SLF4J	1.7.25	MIT	Standard logging API
SnakeYAML	1.18	Apache License v2.0	Reading data in the YAML format
SB Admin 2[23]	3.3.7+1	MIT	Design of the user interface of the collector service administration service

Table 4.2: Third-party components used by the IVXV framework tests

Name	Version	License	Usage need
Hamcrest	1.3	BSD	A more readable use of assert-methods in Java unit tests
JUnit	4.12	Eclipse Public License v1.0	Java testing framework
JUnit-Params	1.1.0	Apache License v2.0	Test parameterization support
Mockito	2.10.0	MIT	Support for mocking the dependencies of the code being tested
Byte Buddy	1.6.14	Apache License v2.0	Mockito dependency
Objenesis	2.5	Apache License v2.0	Mockito dependency
libdigidoc2	3.10.4.1218	LGPL	Generating test data
libdigidocpp-tools	3.13.6.1371	LGPL	Generating test data

Table 4.3: Third-party tools used to develop and/or test the IVXV framework

Name	Version	License	Usage need
Behave[24]	1.2.6	BSD	Regression test driver (*Behavior-driven development*)
Docker[25]	18.06 (or newer)	Apache License 2.0	Environment for conducting regression tests – software containers
Docker Compose[26]	1.22.0	Apache License 2.0	Environment for conducting regression tests – software container management
Sphinx[27]	1.8.2	BSD	Environment for document generation

[4] http://getbootstrap.com
[5] https://bottlepy.org/
[6] http://docopt.org/
[7] https://coreos.com/etcd
[8] https://github.com/ghodss/yaml
[9] https://gopkg.in/yaml.v2
[10] https://github.com/golang/protobuf
[11] https://github.com/grpc-ecosystem/go-grpc-prometheus
[12] https://github.com/grpc-ecosystem/grpc-gateway
[13] https://google.golang.org/grpc
[14] https://golang.org/x/net
[15] https://prometheus.io
[16] https://github.com/beorn7/perks
[17] https://github.com/matttproud/golang_protobuf_extensions
[18] http://www.haproxy.org/
[19] https://jquery.org/
[20] https://github.com/onokumus/metisMenu
[21] http://pyyaml.org/
[22] https://github.com/schematics/schematics
[23] https://github.com/BlackrockDigital/startbootstrap-sb-admin-2

[24] https://github.com/behave/behave
[25] http://www.docker.com/
[26] http://www.docker.com/
[27] http://www.sphinx-doc.org/

CHAPTER 5

Annexes

5.1 Building the Collector Service with a Patched Go Standard Library

Several ID cards and digital IDs exist in Estonia whose certificates include an incorrectly encoded RSA public key. The Go standard library refuses to accept such faulty certificates. At the same time, there are too many to do nothing about it.

One solution would be to compile the IVXV collector service subservices using a patched Go standard library. The delivery comes with a `ivxv-golang` package, which contains patches for accepting faulty certificates of this type and the means for implementing them.

The patched standard library should be built in the same environment where the IVXV collector service is built, i.e. Ubuntu 18.04.

First, all the dependencies listed in the `README.rst` file in the `ivxv-golang` folder have to be installed. Then, with `make` command , the newest Go 1.9 source code is downloaded from Ubuntu depositories, patched, built and tested for the use of faulty certificates. If it is successful, the `source/` subfolder includes two necessary .deb packages:

- `golang-1.9-src_1.9.4-rubuntu1_amd64.deb` and
- `golang-1.7-go_1.9.4-1ubuntu1_amd64.deb`.

These have to be installed in the computer building the IVXV collector service before IVXV is compiled: then the Go standard library patched during preparation is used.

CHAPTER 6

References

Bibliography

[DesmedtF89] Desmedt, Y. & Frankel, Y. Brassard, G. (Ed.) Threshold Cryptosystems Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings, Springer, 1989, 435, 307-315.

[HMVW16] Sven Heiberg, Tarvi Martens, Priit Vinkel, Jan Willemson, Improving the verifiability of the Estonian Internet Voting scheme. In Robert Krimmer, Melanie Volkamer, Jordi Barrat, Josh Benaloh, Nicole Goodman, Peter Y.A. Ryan, Oliver Spycher, Vanessa Teague, Gregor Wenda (Eds.), The International Conference on Electronic Voting E-Vote-ID 2016, 18-21 October 2016, Lochau/Bregenz, Austria, TUT Press, pp. 213-229, ISBN 978-9949-83-022-0

[ProVerif] ProVerif: Cryptographic protocol verifier in the formal model, http://prosecco.gforge.inria.fr/personal/bblanche/proverif/

[TK2016] Tehniline kirjeldus. Elektroonilise hääletamise infosüsteemi arenduse hange, Vabariigi Valimiskomisjon, 2016

[ÜK2016] Elektroonilise hääletamise üldraamistik ja selle kasutamine Eesti riiklikel valimistel. Elektroonilise Hääletamise Komisjon, Tallinn 2016