IVXV protocols

Specification

Versioon 1.6.0

31.05.2020

53 lk

Dok IVXV-PR-EN-1.6.0

Contents

CHAPTER 1

Annotation

This document specifies the IVXV protocols of the online voting information system. The document gives an overview of the technical structure of the online voting system and the protocols used. The document defines the terminology and the data structures used in the protocols.

CHAPTER 2

Overview

The online voting protocol suite (hereinafter the protocol suite) defines communication between the components of the online voting system, the data structures used, the algorithms and interfaces to external systems. The communication of messages is presented in UML interaction diagrams, which define the sequence of messages. The specifications of data structures have representations in the Backus-Naur or JSON schema. The following are used to separate data structure fields: line break symbol `LF` with the ASCII code `0x0A` and the tabulator symbol `TAB` with the ASCII code `0x09`. The algorithms are presented as a pseudocode.

NB! All fields in the data structures of the protocol suite require strict adherence to the permitted symbols and the minimum and maximum lengths of the fields. Using additional spaces, tabulators, etc. is prohibited and applications implementing the specification have to refuse processing data that do not comply with the format.

The protocol suite defines the online voting protocol and the support structures necessary to implement this protocol.

2.1 Online Voting Protocol

The online voting protocol specifies:

1. The electronic vote format that allows defining the voter's intent unambiguously at a specific election
2. The encryption of the electronic vote to ensure secrecy
3. The digital signing of an electronic vote to ensure integrity and voter identification
4. Qualifying the electronic vote by a collector service to stand for vote acceptance

The protocol requires the election organizer to have defined the election and generated for vote encryption a key pair, whose public component has been made available to the voter application.

The voter's intent moves via the protocol to the ballot box stored in the collector service and is taken into consideration in the shaping of the result as follows:

1. The voter uses the voter application to formulate their intent electronically
    1. the expression of intent is shown as an electronic vote
    2. the vote is encrypted
    3. the encrypted vote is digitally signed
2. The collector service stores the electronic vote
    1. elements confirming the validity of the voter certificate are requested for the digitally signed vote
    2. the electronic vote is registered in the external registering service
    3. the voter can check the qualified electronic vote with the verification application
3. The voter can use the verification application to make sure that their vote is handled correctly by the collector service
4. At the end of the voting period the collector service issues the ballot box to the organizer of the election, and the registration service issues a list of the votes registered by the collector service
5. The organizer of the election calculates the voting result
    1. it is made sure that all the votes registered in the registration service have been delivered in the composition of the ballot box
    2. encrypted votes and digital signatures are separated
    3. encrypted votes are decrypted
    4. the voting result is calculated based on the decrypted votes

The protocol is analogous to the protocol of postal voting, where the voter's intent travels to the National Electoral Committee in two envelopes – the outer envelope has inside it another envelope, which in turn contains the voting ballot stating the voter's intent. The outer envelope carries the information that allows identifying the voter, thus making it possible to check the voter's right to vote, among other things. The inner envelope is anonymous to protect the secrecy of the vote. The inner envelopes and the outer envelopes are separated before the votes are counted.

In online voting, the inner envelope is an encrypted vote and the outer envelope is a digitally signed document.

CHAPTER 3

Election Definition

The organizer of the election defines the election. At Estonian national elections, all people who have the right to vote are divided into one or several electoral districts. A voter who belongs to a specific electoral district can only choose between the candidates of that district.

To define an election, at least the following have to be defined:

1. The unique identifiers of the election and the unique identifiers of the questions
2. A complete list of electoral districts and polling divisions
3. A list of people with the right to vote and the electoral districts to which they belong
4. A list of candidates and the electoral districts to which they belong

3.1 Election and Question Identifiers

One data set relating to the elections is integrated using a unique election identifier. Typically, people vote on one specific question at one election. However, it is possible to pose several questions at an election. All questions are separated using a unique identifier.

The length of identifiers is limited to 28 characters from the ASCII code table. The identifiers to be used for a specific election are specified each time in the election settings. The applications implementing the specification have to refuse to process data that identifies an election/question that is not included in the list of elections/questions set out for the application.

```
election-identifier := 1*28CHAR

question-identifier := 1*28CHAR
```

3.2 List of Electoral Districts and Polling Stations

Candidates can be set up for elections only in specific electoral districts. Districts are used to give voters voting choices:

1. The districts are divided into polling stations
2. Each voter belongs to a pre-determined polling station and hence a district
3. In all polling stations of one district, voters can only choose between the candidates of this district

At Estonian national elections, the following are distinguished between: local government elections, Riigikogu (Estonian parliament) elections, European Parliament elections, and referendums.

Local government elections are organized pursuant to the Local Government Council Election Act. The elections are organized at the local government level and each local government has its own voting result. The electoral districts are specified at the local government level pursuant to the rules specified in the election act.

Riigikogu elections are organized pursuant to the Riigikogu Election Act. The elections are organized at state level and the voting result is the same for all local governments. The state is divided into 12 electoral districts.

European Parliament elections are organized pursuant to the European Parliament Election Act. The elections are organized at state level and the voting result is the same for all local governments. The entire country is one large electoral district.

Referendums are organized pursuant to the Referendum Act. The elections are organized at state level and the voting result is the same for all local governments. The entire country is one large electoral district.

Various elections are not different on the basis of online voting data forms and procedures. Various district distributions are handled by the Elections Infosystem.

Candidates can be set up for elections only in a specific electoral district. Districts are divided into polling stations and voters are divided between stations. Voters who belong to a specific polling station can vote in all polling stations of one district and have a choice of all the candidates in this district. The polling station to which the voter belongs determines the district to which they belong. The voter can only choose between the candidates running in their district.

In local government council elections, voting happens at the level of Estonian local governments (parishes, cities), and thus the classification of Estonian administrative units and settlements (EHAK)[1] is used in the online voting protocol suite to specify

[1] http://ads.maaamet.ee/

electoral districts and polling stations and to show to which district voters and choices belong.

For example:

- The EHAK code for the Pirita city district in Tallinn City is 0596
- The EHAK code for Aegviidu Parish is 0112

As agreed, for state-level elections, the EHAK code for the district is 0. The EHAK code for polling stations is the code of the local government under which the specific polling station operates.

At Riigikogu and European Parliament elections and referendums, a fictitious polling station is set up in each electoral district and added to the list of electoral districts and polling stations for people voting abroad. The polling station number for those voters is 0 and the EHAK code is also 0.

```
ehak-code = 1*10DIGIT

ehak-district = ehak-code
no-district = 1*10DIGIT

ehak-station = ehak-code
no-station = 1*10 DIGIT

district = ehak-district '.' no-district
district-legacy = ehak-district TAB no-district

station = ehak-station '.' no-station
station-legacy = ehak-station TAB no-station TAB district-legacy
```

The JSON schema of the electoral district list is defined as follows. The elements of the object `region_dict` are indexed with the element type `ehak-code`. The elements of the object `district_dict` are indexed with the element type `district`. The elements of the array `stations` are type `station`.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "region" : {
            "type": "object",
            "properties": {
                "state": { "type": "string" },
                "county": { "type": "string" },
                "parish": { "type": "string" }
            },
            "additionalProperties": false,
            "minProperties": 1
        },

```

(continues on next page)

(continued from previous page)

```
        "region_dict": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+$": {
                    "$ref": "#/definitions/region"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },

        "station": {
            "type": "string",
            "pattern": "^[0-9]+\\.[0-9]+$"
        },

        "district": {
            "type": "object",
            "properties": {
                "name": { "type": "string" },
                "stations": {
                    "type": "array",
                    "items": {
                        "$ref": "#/definitions/station"
                    }
                }
            },
            "required": ["stations"]
        },

        "district_dict": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/district"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        }

    },

     "type": "object",
     "properties": {
         "election": {"type": "string"},
         "districts": {
             "$ref": "#/definitions/district_dict"
         },
```

(continues on next page)

(continued from previous page)

```
        "regions": {
            "$ref": "#/definitions/region_dict"
        }
    },
    "required": ["districts", "regions", "election"],
    "additionalProperties": false
}
```

Example:

```
{
  "districts": {
    "164.1": {
      "name": "Valimisringkond nr. 1",
      "stations": [
        "164.1"
      ]
    },
    "296.1": {
      "name": "Valimisringkond nr. 1",
      "stations": [
        "296.1",
        "296.2"
      ]
    },
    "784.6": {
      "name": "Valimisringkond nr. 6",
      "stations": [
        "524.68"
      ]
    },
    "784.8": {
      "name": "Valimisringkond nr. 8",
      "stations": [
        "614.85",
        "614.86",
        "614.87"
      ]
    },
    "795.1": {
      "name": "Valimisringkond nr. 1",
      "stations": [
        "795.1",
        "795.2"
      ]
    }
  },
  "election": "TESTKOV",
  "regions": {
```

(continues on next page)

(continued from previous page)


```
    "164": {
      "county": "Ida-Viru maakond",
      "parish": "Avinurme vald",
      "state": "Eesti Vabariik"
    },
    "296": {
      "county": "Harju maakond",
      "parish": "Keila linn",
      "state": "Eesti Vabariik"
    },
    "524": {
      "county": "Tallinn",
      "parish": "N\u00f5mme linnaosa",
      "state": "Eesti Vabariik"
    },
    "614": {
      "county": "Tallinn",
      "parish": "P\u00f5hja-Tallinna linnaosa",
      "state": "Eesti Vabariik"
    },
    "784": {
      "county": "Tallinn"
    },
    "795": {
      "county": "Tartu linn",
      "state": "Eesti Vabariik"
    }
  }
}
```


The list of electoral districts is received from the Election Infosystem and the JSON file is delivered to the online voting system as a digitally signed BDOC file.

3.3 List of Voters

The voter list includes the voters' names and personal identification codes, their polling station and the row number in the polling station voter list under which the voter votes. The voter list is uploaded to the system in the following format:

```
voter-personalcode = 11DIGIT
voter-name = 1*100UTF-8-CHAR
action = "lisamine" | "kustutamine"
line-no = "" | 1*11DIGIT
reason = "" | "tokend" | "jaoskonna vahetus" | "muu"

voter = voter-personalcode TAB voter-name TAB action TAB station-
↪legacy TAB line-no TAB reason LF
```

(continues on next page)

(continued from previous page)

```
version-no = "1"
list-type = "algne" | "muudatused"
voter-list = version-no LF election-identifier LF list-type LF␣
↪*voter
```

The legacy systems' data structures include the field version number, whose length is limited to 2 characters. The value of the field is 1.

The voter list can either be original or amended. The original list only allows adding voters; the amended list also allows removing voters from the list. The voter entry can also include additional information – the row number of the voter entry in the polling station's list and a reason to be in the specific amended list.

The data contain the following:

1. The type (`list-type`) "`original`" means the original large list uploaded to the system before e-voting starts, and "`amended`" means the cumulative updates made later
2. The action (`action`) "`adding`" means adding a new voter to the specific polling station, and "`deleting`" means removing. When a voter moves from one polling station to another, then one deleting entry is made in the amendments of the voter list to delete the voter from their previous polling station, and one adding entry is made to add the voter to the voter list in their new polling station. In the original list, all entries are the "adding" type
3. The polling station (`station-legacy`) identifies the station, the district and the local government where the voter votes
4. The row number (`line-no`) is the row number of the person in the polling station list. It is only filled for the original list; when there are amendments, this field is left empty
5. The reason (`reason`) is used in deleting entries to note the reason for the deletion. The reason field has to be empty for adding entries. If the reason is `preventive measure`, this means that from the moment the amendment is implemented, the voter with this personal identification code will not be allowed to vote anymore. If the reason is a `station switch`, it means that the voter is deleted from one polling station, because they are added to another. In this case, a deleting entry has to be accompanied by an adding entry (this is checked). If the voter is removed from the list for some other reason (death, moving to a district that is not part of the elections), the reason has to be `other` or can be left empty. This field is informative.

3.3.1 Signing the Voter List

The voter list is retrieved from the Population Register, which is run by the IT and Development Centre of the Ministry of the Interior. The legacy format text file comes

with a signature file made by the Population Register by taking the SHA256 hash from the original voter list and signing that hash with a 2048-bit RSA key. The public key generated by the Population Register is made available to the online voting information system, and this key is used to check the integrity of the voter lists. The schema has been used since the 2015 Riigikogu elections.

3.3.2 Version of the Applied List

The applied voter list at a certain point in time depends on the original list and the amendments made – what kind of amendments have been made and in which order. So as to identify this situation unambiguously, a list version has to be calculated.

NB! This version is not related to the version number in the list file that determines the list format version.

The version is calculated as follows:

```
v_0 = ""
v_n = base64(sha256(v_{n-1} | base64(sha256(nk_n))))
```

where `nk_n` is the list loaded in `n` (counting starts from one, i.e. the original list is `nk_1`), `v_n` is the voter list version after it has been loaded, `""` is an empty string and `|` is the string connection operation.

Records about the implemented list version are kept by the collector service and the processing application, which guarantee that the specific vote is counted in the right district.

3.4 List of Choices

The list of choices includes data on the candidates (at elections) or answers (at referendums). At elections, the list includes not only the candidate's data but also the name of their electoral list.

There are two systemic differences at elections that are visible to voters during online voting:

1. At referendums, voters do not choose between the candidates of political parties, but answer "yes" or "no" to specific questions
2. At Riigikogu, local government and European Parliament elections, voters vote for one candidate, who may or may not belong to a larger party/list

The protocol suite encodes the voter's possible choices in the district as a numerical value of up to 11 characters, which is encoded in the list of choices with the EHAK code of the district. Only the choices of their electoral district can be available to a voter. The voter application has to ensure this function, and the application calculating the voting result has to check it.

```
choice-no = 1*11DIGIT
district-choice = ehak-district '.' choice-no
```

The JSON schema of the list of choices has been defined as follows. The elements of the object `district_dict` are indexed with the `district` type element. The elements of the object `list-choices` are indexed with the `district-choice` type element.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "choice": {
            "type": "string"
        },
        "list_choices": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/choice"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },
        "district_choices" : {
            "type": "object",
            "additionalProperties": {
                "$ref": "#/definitions/list_choices"
            },
            "minProperties": 1
        },
        "district_dict": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/district_choices"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        }
    },

    "type": "object",
    "properties": {
        "election": {"type": "string"},
        "choices": {
            "$ref": "#/definitions/district_dict"
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "required": ["election", "choices"],
    "additionalProperties": false
}
```

Example:

```
{
  "choices": {
    "164.1": {
      "Nimi Valimisliit": {
        "164.126": "Nimi Kandidaat",
        "164.127": "Nimi Kandidaat"
      }
    },
    "296.1": {
      "Nimi Erakond": {
        "296.198": "Nimi Kandidaat",
        "296.199": "Nimi Kandidaat",
        "296.200": "Nimi Kandidaat"
      },
      "Nimi Valimisliit": {
        "296.115": "Nimi Kandidaat",
        "296.116": "Nimi Kandidaat",
        "296.117": "Nimi Kandidaat"
      },
      "Üksikkandidaadid": {
        "296.101": "Nimi Kandidaat",
        "296.102": "Nimi Kandidaat"
      }
    }
  },
  "election": "TESTKOV"
}
```

CHAPTER 4

Electronic Vote

The IVXV voting protocol is based on the double envelope system, which means that the voter's true plaintext intent is encrypted with the public key issued by the organizer of the election. The encrypted intent is signed digitally with signing means available to the voter, and delivered to the collector service in an agreed upon container format. The collector service may additionally qualify the vote signed by the voter, for example by making sure that the signing certificate is valid. The IVXV protocol suite requires, among other things, that the votes received by the collector service be registered in an external registration service.

The vote stored by the collector service along with its qualifying elements are made available to both the voter application and the verification application, which perform the same checks on a single vote that are later performed on all votes by the processing application of the election organizer. The option to check qualifying elements gives the voter confidence that their voice can be processed properly in the later stages.

4.1 Voter's Plaintext Intent

The voter's plaintext intent exists in the voter application and later also in the verification application. The voter's intent includes the code of the choice in the district, the EHAK code of the district, the name of the choice list, and the name of the specific choice in the list.

```
choice-name = 1*100UTF-8-CHAR
choicelist-name = 1*100UTF-8-CHAR

ballot = district-choice '\x1F' choicelist-name '\x1F' choice-name
```

4.2 Encrypted Ballot

The voter's intent in the form of a plaintext `ballot` is encrypted by the voter application using the public key generated by the election organizer. For encryption, IVXV needs a non-deterministic, homomorphic public key cryptosystem. The ElGamal cryptosystem is good for that purpose; it is now used in the IVXV context for a residue class set.

The public key of ElGamal is encoded with the ElGamal cryptosystem parameters and the identifier that characterizes the specific election. The cryptosystem parameters are part of the algorithm identifier structure, the public key is encoded into the `SubjectPublicKeyInfo` structure.

```
elGamalEncryption OBJECT IDENTIFIER ::= {
    {iso(1) org(3) dod(6) internet(1) private(4) enterprise(1)␣
↪dds(3029) asymmetric-encryption(2) 1}
}

elGamal-Params-IVXV ::= SEQUENCE {
    p                   INTEGER,
    g                   INTEGER,
    election-identifier GeneralString
}

elGamalPublicKey ::= SEQUENCE {
    y           INTEGER,
}

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm   AlgorithmIdentifier,
    subjectPublicKey    BIT STRING
}
```

In order to encrypt the voter's intent, the UTF-8 encoded data structure `ballot` is taken and converted into an element in a set specified by the ElGamal parameters. We presume the parameter `p` to be 256 bytes. In that case, the length of the structure `ballot` can be 253 bytes. The plaintext intent is padded to match the length of the parameter `p`.

```
padded-ballot = ballot '\x00' '\x01' *'\xff' '\x00'
```

The padded intent is interpreted as an integer, encoded as a quadratic residue in the set specified by parameter `p`. Encoding is bijective and important for the further mixing of the ciphertext.

The intent is encrypted pursuant to the ElGamal method with a public key.

```
elGamalEncryptedMessage ::= SEQUENCE {
    a           INTEGER,
    b           INTEGER
}
```

(continues on next page)

(continued from previous page)

```
encryptedBallot ::= SEQUENCE {
    algorithm   AlgorithmIdentifier,
    cipher  ANY
}
```

The DER encoding of the data structure `encryptedBallot` is an encrypted ballot i.e. the inner envelope in the double envelope system.

In the course of encrypting the voter's intent, the voter application generates a random number used by the ElGamal for encryption. The same random number is later revealed to the verification application. Pursuant to the idiosyncrasy of the ElGamal cryptosystem, this random number functions as a so-called second key and allows for the decoding of the ciphertext in the verification application.

4.3 Vote Signed by the Voter

Before it is sent to be stored by the collector service, the encrypted ballot has to be signed digitally, which can be done using all the digital signing means valid in the Republic of Estonia: the ID card, digital ID, and mobile ID.

This specification foresees using the BDOC signature format defined in the Republic of Estonia Draft Standard [BDOC2.1]. The BDOC signature format is made up of the ETSI standard TS 101 903 (XadES) profile and the OpenDocument container format. The IVXV protocol suite also allows using alternative signature and container formats.

Depending on the number of questions posed at the current election, a digitally signed vote can contain one or several data files with the MIME type `application/octet-stream`. Each data file contains an encrypted ballot. To hash the data file and the other data objects to be signed before signing, the hash function SHA-256 is used. The name of the data file is made up of the extension `ballot` and the election and question identifiers. All referenced data files have to be included in the signature container. A digitally signed vote cannot contain data files other than those which include votes cast in the context of the ongoing election. The collector service has to refuse to accept, store and process votes that to not match the settings.

```
extension = "ballot"

encrypted-ballot-name = election-identifier '.' question-identifier
↪'.' extension
```

The vote signed by the voter in the voter application is generated so that it could be further qualified in the collector service. This specification foresees getting both the OCSP validity certification and the PKIX timestamp to qualify a vote. This means that the final qualified vote is in the BDOC-TS format.

If a vote is signed using an ID card or digital ID, then the original signed container is generated in the voter application. If the vote is signed with mobile ID, the container

is generated in cooperation between the voter application and the mobile ID service relayed by the collector service. In case of mobile ID, the collector service uses the mobile ID service only to get a signature for the encrypted ballot. All elements necessary to qualify a vote are requested from relevant services only when the voter application has sent a signed vote to be stored. The qualified vote is delivered by the collector service to the voter application for verification, only a qualified vote has to meet the requirements of the BDOC 2.1 standard – the vote generated by the voter application is an intermediate step in reaching a qualified vote.

A vote signed in the voter application can only have one signature, which is kept in the signature file `META-INF/signature0.xml`. The container containing the vote and the signature is generated using the method specified in the BDOC 2.1 standard.

Below is a specification for a single-question vote signed in the voter application.

For the hash algorithm `DIGEST_ALG`, the SHA-256 (http://www.w3.org/2001/04/xmlenc#sha256) is used. To canonicalize XML (`CANON_ALG`), the method c14n11 (http://www.w3.org/2006/12/xml-c14n11) is used.

In case of RSA keys (ID card, digital ID), the signing method is http://www.w3.org/2001/04/xmldsig-more#rsa-sha256. In case of ECC keys (mobile ID), it is http://www.w3.org/2001/04/xmldsig-more#ecdsa-sha256.

The exact value of the identifiers *VOTE_REF*, *SP_URI* ning *SV_URI* is not important.

4.3.1 Element *SignedProperties*

The element `SignedProperties` is generated according to the BDOC 2.1 standard. If a timestamp is used for qualification, then the element `SignaturePolicyIdentifier` is not used. No non-obligatory elements are used. The time of signing is fixed by the computer filling in the data structure, and the voter's X509 certificate is retrieved from the ID card or using the mobile ID service.

```
1  <xades:SignedProperties xmlns:asic="http://uri.etsi.org/02918/v1.2.1
   ↪#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xades=
   ↪"http://uri.etsi.org/01903/v1.3.2#" Id="%SP_URI%">
2  <xades:SignedSignatureProperties>
3    <xades:SigningTime>%SIGNING_TIME%</xades:SigningTime>
4    <xades:SigningCertificate>
5      <xades:Cert>
6        <xades:CertDigest>
7          <ds:DigestMethod Algorithm="%DIGEST_ALG%"></ds:DigestMethod>
8          <ds:DigestValue>%CERT_DIGEST%</ds:DigestValue>
9        </xades:CertDigest>
10       <xades:IssuerSerial>
11         <ds:X509IssuerName>%ISSUER_NAME%</ds:X509IssuerName>
12         <ds:X509SerialNumber>%ISSUER_SERIAL%</ds:X509SerialNumber>
13       </xades:IssuerSerial>
14     </xades:Cert>
15   </xades:SigningCertificate>
```

(continues on next page)

(continued from previous page)

```
</xades:SignedSignatureProperties>
<xades:SignedDataObjectProperties>
  <xades:DataObjectFormat ObjectReference="#%VOTE_REF%">
    <xades:MimeType>application/octet-stream</xades:MimeType>
  </xades:DataObjectFormat>
</xades:SignedDataObjectProperties>
</xades:SignedProperties>
```

4.3.2 Element *SignedInfo*

The element `SignedInfo` is generated according to the BDOC 2.1 standard, referring to both the encrypted ballot (`VOTE_DIGEST`) and the element `SignedProperties` (`SP_DIGEST`).

```
<ds:SignedInfo xmlns:asic="http://uri.etsi.org/02918/v1.2.1#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xades="http://uri.etsi.org/01903/v1.3.2#">
  <ds:CanonicalizationMethod Algorithm="%CANON_ALG%"></ds:CanonicalizationMethod>
  <ds:SignatureMethod Algorithm="%SIG_ALG%"></ds:SignatureMethod>
  <ds:Reference Id="%VOTE_REF%" URI="%VOTE_URI%">
    <ds:DigestMethod Algorithm="%DIGEST_ALG%"></ds:DigestMethod>
    <ds:DigestValue>%VOTE_DIGEST%</ds:DigestValue>
  </ds:Reference>
  <ds:Reference Id="%SP_REF%" Type="http://uri.etsi.org/01903#SignedProperties" URI="#%SP_URI%">
    <ds:Transforms>
      <ds:Transform Algorithm="%CANON_ALG%"></ds:Transform>
    </ds:Transforms>
    <ds:DigestMethod Algorithm="%DIGEST_ALG%"></ds:DigestMethod>
    <ds:DigestValue>%SP_DIGEST%</ds:DigestValue>
  </ds:Reference>
</ds:SignedInfo>
```

4.3.3 Element *SignatureValue*

The element `SignatureValue` is generated according to the BDOC 2.1 standard. The canonicalized element `SignedInfo` is the basis for calculating the hash, which is signed using the PKCS1 method.

```
<ds:SignatureValue xmlns:asic="http://uri.etsi.org/02918/v1.2.1#" xmlns:ds="http://www.w3.org/2000/09/xmldsig#" xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Id="%SV_URI%">%SIG_VALUE%</ds:SignatureValue>
```

4.3.4 Element *XAdESSignatures*

The element `XAdESSignatures` contains one `Signature` element, generated on the basis of all the previous elements and the voter's X509 certificate. The element `UnsignedProperties` is not used.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<asic:XAdESSignatures xmlns:asic="http://uri.etsi.org/02918/v1.2.1#">
    <ds:Signature xmlns:ds="http://www.w3.org/2000/09/xmldsig#" Id="S0">
        %SI_XML%
        %SV_XML%
    <ds:KeyInfo>
      <ds:X509Data>
          <ds:X509Certificate>%X509_CERT%</ds:X509Certificate>
      </ds:X509Data>
    </ds:KeyInfo>
    <ds:Object>
        <xades:QualifyingProperties xmlns:xades="http://uri.etsi.org/01903/v1.3.2#" Target="#S0">
            %SP_XML%
      </xades:QualifyingProperties>
    </ds:Object>
  </ds:Signature>
</asic:XAdESSignatures>
```

CHAPTER 5

Qualifying an Electronic Vote for Storage

5.1 Qualified Vote

The work of the voter application results in sending a double envelope to the collector service; that double envelope includes the voter's intent in an encrypted form, the voter's signature on the encrypted intent in the signature and container format agreed upon, and the voter's signing certificate in the X509 format.

So as to store a vote successfully, the IVXV protocol foresees registering the vote with an external registering service provider, and making the registering certificate available to the voter application. The election organizer can prescribe steps in addition to registering in order to quality a vote – such as getting a certificate status concerning the certificate used to sign the vote.

All qualifying elements requested by the collector service that determine the status of the vote in the later stages of processing have to be presented to the voter application, and if demanded, also to the verification application to make sure that the voter learns about the proper processing of their vote in due course.

5.1.1 OCSP Certificate Status

OCSP (*Online Certificate Status Protocol*) is a standard protocol to request the validity information of X509 certificates. The collector service can use this protocol to learn the validity of the certificate used to sign a vote. The OCSP response says that the certificate was valid at the moment the request was made, but does not connect the OCSP response with a specific source.

5.1.2 OCSP-TM Certificate Status

The BDOC 2.1 standard specifies the BDOC-TM profile, where the certificate status retrieved with the OCSP protocol also functions as a timestamp, which confirms that the specific signature existed before the OCSP certificate status was retrieved.

5.1.3 RFC3161 Timestamp

The RFC3161 timestamp protocol is used to retrieve confirmation from the trust service that some set of data already existed before a certain point in time. In the BDOC-TS context, the signature element `SignatureValue` is timestamped in its canonicalized form. The classic OCSP response with a timestamp in an RFC 3161 format qualifies the BDOC-TS signature.

5.2 Vote Registration

The IVXV registration protocol has been specified in the document “Registration service for the electronic voting information system IVXV”. The registration service functions on the basis of the RFC3161 timestamp protocol. The protocol has been extended so that the collector service could give its signature to the timestamp request, which makes it possible to later retrieve a comparative extract from the registration service. The existence of an independent registration service decreases the risk of votes being ‘lost’ by the collector service.

There is no inherent need to bind the registration protocol to the timestamp protocol.

5.3 Storage

Storing an electronic vote in the collector service means:

1. Accepting a vote from the voter application and verifying the voter’s signature
2. Potential qualification of the vote – such as verifying the validity of the certificate at a time close to the moment the vote was signed
3. Registering the vote in an independent registration service
4. Relaying the elements that qualify the vote to the voter application

Various combinations of the signature format and the vote-qualifying service may generate various IVXV profiles. In a specific document, the IVXV profile looks as follows:

1. The signed vote format is BDOC-TS
2. The certificate status protocol is a standard OCSP
3. The RFC3161 timestamp used to qualify BDOC-TS is also used as registration verification

CHAPTER 6

Electronic Vote Verification

An electronic vote is verified in the processing application, in the collector service, in the voter application and in the verification application. An electronic vote is checked the most thoroughly inside the ballot box in the processing application, where it is decided whether this specific vote is to be sent to counting or not. For each single vote, checks are made in the voter application analogously to the processing application to make sure that the collector service has qualified the vote in a way that allows the checks made in the processing application to be performed successfully. The checks analogous to the voter application are performed by the verification application.

6.1 Checks in the Collector Service

The voter application sends to the collector service in the set of the signed vote the following:

1. An encrypted ballot
2. The voter's signature on the encrypted ballot
3. The voter's signature certificate

The collector service performs the following checks as a minimum:

1. The signer of the vote is included in the voter list
2. The signed vote is presented in the correct container format
3. The digital signature on the encrypted ballot is correct
4. The vote signer's certificate was valid at the moment the vote was accepted

To verify the validity of the vote signer's certificate, the collector service requests information from the certificate status service. The collector service verifies the response

of the certificate status service on the status of the certificate and adds the response to the elements that qualify a vote.

The collector service registers the fact of storing the vote in the external registration service by signing the registration request and storing the registration certificate signed by the registration service as one of the vote-qualifying elements.

The collector service sends all of the elements qualifying the vote collected by it to the voter application with the unique vote identifier.

6.2 Checks in the Voter Application

On the basis of the voter's plaintext will, the voter application prepares an encrypted ballot and signs it with the voter's chosen digital signing means.

The role of the voter application after the vote is signed is to make sure that the collector service has acted according to protocol when getting vote-qualifying elements and that the vote has been stored so that it will be taken into account by the processing application.

The voter application performs at a minimum the following checks:

1. The collector service got the certificate status from the certificate status service authorized for the voter's certificate. The voter application checks the signature on the certificate status response
2. The collector service registered the vote signed by the voter in the authorized registration service. The voter application checks that the request made by the collector service was signed by the collector service and referred to a correctly signed vote. The voter application checks that the registration service response has been signed by the correct registration service provider and includes the request signed by the collector service

If the elements needed to qualify a vote cannot be checked, the voter application will notify the user.

6.3 Checks in the Verification Application

The verification application receives the following information from the voter application:

1. The randomness used to generate an encrypted ballot
2. The unique identifier of a signed vote in the collector service

The verification application uses the unique identifier of the vote from the collector service to retrieve the following information:

1. The encrypted ballot

2. The voter's signature on the encrypted ballot
3. The voter's signing certificate
4. Vote-qualifying elements, incl. the certificate status and the registration certificate

The verification application makes the following checks:

1. The signed vote is presented in the correct container format
2. The digital signature on the encrypted ballot is correct
3. The certificate of the vote signer was valid at the moment the vote was accepted, which is confirmed by a correct certificate status
4. The vote is correctly registered in the correct registration service

After these checks have been performed, the verification application displays the data of the person who signed the vote.

In addition, the verification application uses the randomness used to generate the encrypted ballot also to decrypt the encrypted ballot.

NB! The randomness used to encrypt one vote can be used to decrypt only that vote. To decrypt several different votes, the private component of the vote secrecy key pair is needed.

The verification application makes sure that the plaintext retrieved as a result of decryption matches the required syntax of the plaintext intent.

The verification application displays the form-compliant intent to allow the verifier to make sure that the intent is correct.

6.4 Checks in the Processing Application

The processing application checks each vote separately, making sure among other things that the views on the content of the ballot box given by each collector service and registration service are consistent. The processing application then decides, which vote is the last in terms of time and is to be directed to the next stage of processing, as a result of which, the vote may make it to counting.

The input of the processing application:

1. A list of registration requests received by the registration service
2. A list of voter lists implemented in the collector service
3. The ballot box delivered by the collector service that includes per each vote an encrypted ballot, the voter's signature on an encrypted ballot, the voter's signing certificate, the status of the certificate, and the registration certificate

The processing application checks the consistency of the registration service and the collector service and issues the differences:

1. Votes for which there is a registration request in the collector service, but the response for which has not made it to the collector service
2. Votes for which there is a registration request in the collector service, but which have not been delivered by the collector service

The processing application checks every single vote for the following:

1. The signer of the vote is in the voter list
2. The signed vote is presented in the correct container format
3. The digital signature on the encrypted ballot is correct
4. The certificate of the signer of the vote was valid at the moment the vote was accepted, which is verified by the correct certificate status
5. The vote is registered correctly in the correct registration service

The processing application decides which of the voter's votes was last and thus moves on to the next stage of processing. I.e. one of the vote-qualifying elements plays the role of a fixer when the vote is stored, and on the basis of that element, the sequence of single votes in time is generated. Depending on the IVXV profile, this element can be in the composition of the certificate status (BDOC-TM), as a separate timestamp (BDOC-TS), or in the composition of the registration certificate (BDOC-TS).

CHAPTER 7

Communications Protocols

7.1 Interface

The microservices of the collector service directed at the voter communicate with the voter application and the verification application via the JSON-RPC protocol.

id JSON-RPC information request identifier
method RPC method
params Parameters of a specific RPC method

```
{
    "id": 0.0,
    "method": "RPC.Method",
    "params": [
        {
            "MethodParam": "value",
            "SessionID": "ec3a0cab353d552952289f2c7ad52e27"
        }
    ]
}
```

error Possible error code or `null` when there is no error
id JSON-RPC information request identifier, has to match the id used in the request
result Method-based response data structure

```
{
    "error": null,
```

(continues on next page)

(continued from previous page)

```
    "id": 0.0,
    "result": {
        "ResultParam": "value",
        "SessionID": "ec3a0cab353d552952289f2c7ad52e27"
    }
}
```

During the first information request exchange with an IVXV microservice, a HEX-coded unique session identifier (`result.SessionID`), is issued to the communicating application; this identifier will henceforth be used by the application in all requests sent to the collector service (`params.SessionID`). The session identifier is used to integrate various RPC requests related to voting into one session. This integration is informative and its purpose is to simplify log analysis; decisions related to the electoral district to which the vote belongs and/or other essential aspects are made on the basis of digitally signed data.

TLS is used as the transport protocol. The encrypted channel is terminated in a specific microservice. To enable sharing the load and implementing microservices flexibly, a TLS SNI extension is used that allows the proxy services to direct TLS into the correct microservice instance without terminating the flow. The proxy service is typically available in port 443 of the external interface of the collector service.

7.2 Retrieving the List of Choices

Retrieving the list of choices means that the voter application has to communicate with the list service (SNI `choices.ivxv.invalid`). Retrieving the list of choices means that the voter has to be authenticated and their electoral district has to be identified.

The voter application makes the request `RPC.VoterChoices` to retrieve the lists.

params.AuthMethod The supported choices are the methods `tls` and `ticket`.

params.OS The operation system in which the voter application is used.

Request `RPC.VoterChoices` when authenticating with an ID card – authentication happens at the TLS protocol level during the processing of the request using the authentication certificate of the ID card.

```
{
    "id": 0.0,
    "method": "RPC.VoterChoices",
    "params": [
        {
            "AuthMethod": "tls",
            "OS": "Operating System,2,0"
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

Request `RPC.VoterChoices` when authenticating with mobile ID – before making the request, the mobile ID proxy service (SNI `mid.ivxv.invalid`) has to be used to retrieve a signed authentication certificate.

params.AuthToken Certificate signed using the authentication service, which includes the voter's unique identifier.

params.SessionID Since in the case of mobile ID, an interaction to retrieve the authentication certificate has preceded list retrieval, there exists a session identifier that needs to be used.

```
{
    "id": 0.0,
    "method": "RPC.VoterChoices",
    "params": [
        {
            "AuthMethod": "ticket",
            "AuthToken":
↪"G1RTZqBSBKrzqReuKYrmFUFXWFPvaxhJjdiZi6zqAnaK3OvrT...",
            "OS": "Operating System,2,0",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
        }
    ]
}
```

Response of the list service to the request `RPC.VoterChoices`.

result.Choices The voter's district identifier `VoterDistrict`

result.List BASE64-encoded district choice list `DistrictChoices`

result.Voted If the voter has already cast their vote, it is `true`; if not, it will not be displayed among the field responses.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "Choices": "0140.1",
        "List":
↪"ew0KICAgICAgICAgICAgIkVyYWtvbmQgMSI6IHsNCiAgICAgICAgICAgIC...",
        "SessionID": "ec3a0cab353d552952289f2c7ad52e27",
        "Voted": true
    }
}
```

Possible error codes to the request `RPC.VoterChoices`.

BAD_CERTIFICATE An error in the voter's personal identification certificate.

BAD_REQUEST There is an error in the request.

INELIGIBLE_VOTER The voter does not have the right to vote.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

UNAUTHENTICATED An unauthenticated request.

VOTER_TOO_YOUNG The voter is too young to vote.

VOTING_END The voting period has ended.

7.3 Sending a Signed Vote to Storage

Sending a signed vote to storage means that the voter application has to communicate with the voting application (SNI `voting.ivxv.invalid`).

The voter application makes the request `RPC.Vote` to send the signed vote to be stored.

params.AuthMethod The supported choices are the methods `tls` and `ticket`.

params.Choices The voter's district identifier `VoterDistrict`, which was valid when the list of choices was retrieved. Correct use of the parameter allows the collector service to warn the voter if their district has changed compared to the start of voting.

params.OS The operation system in which the voter application is used.

params.Type The format of a signed vote. At the moment, the only supported value is `bdoc`.

params.Vote BASE64-encoded vote `SignedVote` in the format specified above.

Request `RPC.Vote` when authenticating with an ID card.

```
{
    "id": 0.0,
    "method": "RPC.Vote",
    "params": [
        {
            "AuthMethod": "tls",
            "Choices": "0140.1",
            "SessionID": "ec3a0cab353d552952289f2c7ad52e27",
            "OS": "Operating System,2,0",
            "Type": "bdoc",
            "Vote":
↪"UEsDBAoABgAAAAIAAAAbWltZXR5cGVhcHBsaWNhdGlv\nbi92bmQuZX..."
        }
    ]
}
```

Request `RPC.Vote` when authenticating with mobile ID.

```
{
    "id": 0.0,
    "method": "RPC.Vote",
    "params": [
        {
            "AuthMethod": "ticket",
            "AuthToken":
↪"G1RTZqBSBKrzqReuKYrmFUFXWFPvaxhJjdiZi6zqAnaK3OvrT...",
            "Choices": "0919.1",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b",
            "OS": "Operating System,2,0",
            "Type": "bdoc",
            "Vote":
↪"UEsDBAoAAAAAAAAAACKIflFHwAAAB8AAAAIAAAAbWltZXR5cGVhcHB..."
        }
    ]
}
```


The voting service's response to the request `RPC.Vote`.

result.Qualification.ocsp

result.Qualification.tspreg Additional proof retrieved by the collector service to qualify and correctly register the vote `SignedVote` created by the voter application. The composition of the response depends on the specific settings of the collector service; in this case the standard OCSP protocol is used to check that the voter's signature certificate is valid, and the PKIX timestamp protocol based registration service is used to both fix the time of casting the vote and registering the electronic vote in an external independent service. Both the OCSP response and the timestamp in the PKIX format with any additions necessary for the registration service are sent to the voter application to be checked.

result.TestVote If the vote was cast before voting started and was counted as a test vote, then `true`; if not, this field will not be included in the response. In case of a test vote, the voter application will display a relevant warning to the voter.

result.VoteID The vote's identifier in the storage service; based on that, the verification application can demand access to the vote for later analysis.


```
{
    "error": null,
    "id": 0.0,
    "result": {
        "Qualification": {
            "ocsp":
↪"MIIFTAoBAKCCBUUwggVBBgkrBgEFBQcwAQEEggUyMIIFLjCB5qFMME...",
            "tspreg":
↪"MIIDsAYJKoZIhvcNAQcCoIIDoTCCA50CAQMxCzAJBgUrDgMCGgQS..."
```


(continues on next page)

(continued from previous page)

```
        },
        "SessionID": "ec3a0cab353d552952289f2c7ad52e27",
        "TestVote": true,
        "VoteID": "VM/cUIU4n7VjxpUx1fC00Q=="
    }
}
```

Possible error codes in case of the request `RPC.Vote`.

BAD_CERTIFICATE An error in the voter's personal identification or signing certificate.

BAD_REQUEST Error in the request.

IDENTITY_MISMATCH The personal identification codes in the personal identification certificate and the signing certificate do not match.

INELIGIBLE_VOTER The voter does not have the right to vote.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

OUTDATED_CHOICES The district to which the voter belongs has changed since the moment the list was retrieved.

UNAUTHENTICATED An unauthenticated request.

VOTER_TOO_YOUNG The voter is too young to vote.

VOTING_END The voting period has ended.

7.4 Voting with Mobile ID

Using mobile ID as a means of signing and authentication means that a support service (SNI `mid.ivxv.invalid`) that integrates with the mobile ID service has to be used to retrieve an authentication certificate before the list of choices is retrieved and to sign the vote before storing it.

7.4.1 Retrieving an Authentication Certificate

The voter application makes the request `RPC.Authenticate` to start mobile ID authentication.

params.OS The operation system in which the voter application is used.

params.IDCode The personal identification code of the person using the mobile ID.

params.PhoneNo The mobile phone number of the person using the mobile ID.

```
{
    "id": 0.0,
    "method": "RPC.Authenticate",
    "params": [
        {
            "OS": "Operating System,2,0",
            "IDCode": "60001019906",
            "PhoneNo": "+37200000766"
        }
    ]
}
```

result.ChallengeID The mobile ID verification code to be displayed in the voter application.

result.SessionCode The mobile ID session identifier for further poll requests.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "Challenge": "EtsTur4XV7xEGS9LBjHSfF9Cc5PQxtYW+YAOysRIt2r...
↪",
        "SessionCode": "2127729011",
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
    }
}
```

Possible error codes in case of the request `RPC.Authenticate`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

MID_BAD_CERTIFICATE An error in the voter's mobile ID personal identification certificate.

MID_NOT_USER The phone number does not belong to the mobile ID client.

VOTING_END The voting period has ended.

The voter application makes the request `RPC.AuthenticateStatus` to assess the status of the authentication process.

params.OS The operation system in which the voter application is used.

params.SessionCode Authentication session identifier.

```
{
    "id": 0.0,
    "method": "RPC.AuthenticateStatus",
```

(continues on next page)

(continued from previous page)

```
    "params": [
        {
            "OS": "Operating System,2,0",
            "SessionCode": "2127729011",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
        }
    ]
}
```

result.AuthToken The authentication certification to be presented to other IVXV services, or `null`, if the request is still being processed.

result.GivenName The voter's given name in case of successful authentication.

result.PersonalCode The voter's personal identification code in case of successful authentication.

result.Status Request status – `POLL` means the request has to be repeated, `OK` means the authentication was successful. Other fields of the response only contain information if the value is `OK`.

result.Surname The voter's surname in case of successful authentication.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "AuthToken": null,
        "GivenName": "",
        "PersonalCode": "",
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b",
        "Status": "POLL",
        "Surname": ""
    }
}
```

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "AuthToken":
↪"G1RTZqBSBKrzqReuKYrmFUFXWFPvaxhJjdiZi6zqAnaK3OvrT2Qu6...",
        "GivenName": "MARY \u00c4NN",
        "PersonalCode": "60001019906",
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b",
        "Status": "OK",
        "Surname": "O\u2019CONNE\u017d-\u0160USLIK"
    }
}
```

Possible error codes in case of the request `RPC.AuthenticateStatus`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

MID_ABSENT The voter's mobile phone is not available.

MID_CANCELED The voter cancelled the mobile ID session.

MID_EXPIRED The mobile ID session has expired.

MID_GENERAL An error in the functioning of the mobile ID service.

VOTING_END The voting period has ended.

7.4.2 Signing a Vote

The voter application makes the request `RPC.GetCertificate` to get the signing certificate.

params.AuthMethod Only the authentication method `ticket` is supported.

params.AuthToken Mobile ID authentication certificate.

params.OS The operation system in which the voter application is used.

params.PhoneNo The phone number of the person signing the vote.

```
{
    "id": 0.0,
    "method": "RPC.GetCertificate",
    "params": [
        {
            "AuthMethod": "ticket",
            "AuthToken":
↪"G1RTZqBSBKrzqReuKYrmFUFXWFPvaxhJjdiZi6zqAnaK3OvrT...",
            "OS": "Operating System,2,0",
            "PhoneNo": "+37200000766",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
        }
    ]
}
```

result.Certificate Signing certificate in the X509 format.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "Certificate":
↪"MIIEVjCCAz6gAwIBAgIQRfmbsIcpkQ9UhxScCwG6VDANBgkqhki...",
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
    }
}
```

Possible error codes in case of the request `RPC.GetCertificate`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

MID_BAD_CERTIFICATE An error in the voter's mobile ID signing certificate.

MID_GENERAL An error in the functioning of the mobile ID service.

MID_NOT_USER The phone number does not belong to the mobile ID client.

VOTING_END The voting period has ended.

The voter application makes the request `RPC.Sign` to initiate vote signing.

params.AuthMethod Only the authentication method `ticket` is supported.

params.AuthToken Mobile ID authentication certificate.

params.Hash BASE64-encoded electronic vote SHA-256 hash.

params.OS The operation system in which the voter application is used.

params.PhoneNo The phone number of the person signing the vote.

```
{
    "id": 0.0,
    "method": "RPC.Sign",
    "params": [
        {
            "AuthMethod": "ticket",
            "AuthToken":
↪"G1RTZqBSBKrzqReuKYrmFUFXWFPvaxhJjdiZi6zqAnaK3OvrT...",
            "Hash": "9IBrA05ylt2StdjxKkSTYMW/rQXY3Vub4upzShdfEzo=",
            "HashType": "SHA256",
            "OS": "Operating System,2,0",
            "PhoneNo": "+37200000766",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
        }
    ]
}
```

result.ChallengeID The mobile ID verification code to be displayed in the voter application.

result.SessionCode The mobile ID session identifier for further poll requests.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "SessionCode": "E663A711BB9447EAD82491F9372F4CA",
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
    }
}
```

Possible error codes in case of the request `RPC.Sign`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

MID_BAD_CERTIFICATE An error in the voter's mobile ID signing certificate.

MID_NOT_USER The phone number does not belong to the mobile ID client.

VOTING_END The voting period has ended.

The voter application makes the request `RPC.SignStatus` to assess the status of the signing process.

params.OS The operation system in which the voter application is used.

params.SessionCode Mobile ID session identifier.

```
{
    "id": 0.0,
    "method": "RPC.SignStatus",
    "params": [
        {
            "OS": "Operating System,2,0",
            "SessionCode": "E663A711BB9447EAD82491F9372F4CA",
            "SessionID": "057229fdfa2df7d3c7f4ced81b02760b"
        }
    ]
}
```

result.Signature If the response `Status` field is `OK`, BASE64-encoded PKCS1 signature, otherwise `null`.

result.Status Request status – `POLL` means the request has to be repeated, `OK` means the signing was successful. Other fields of the response only contain information if the value is `OK`.

```
{
    "error": null,
    "id": 0.0,
    "result": {
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b",
        "Signature": null,
        "Status": "POLL"
    }
}
```

```
{
    "error": null,
    "id": 0.0,
```

(continues on next page)

(continued from previous page)

```
    "result": {
        "SessionID": "057229fdfa2df7d3c7f4ced81b02760b",
        "Signature": "MOj+8xQ9DmZPr/
↪ItHlm0tHNMCuTgn6dT9jcXjPLf0+2sVjsS11jRI...",
        "Algorithm": "SHA256WithECEncryption",
        "Status": "OK"
    }
}
```

Possible error codes in case of the request `RPC.SignStatus`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

MID_ABSENT The voter's mobile phone is not available.

MID_BAD_CERTIFICATE An error in the voter's mobile ID signing certificate.

MID_CANCELED The voter cancelled the mobile ID session.

MID_EXPIRED The mobile ID session has expired.

MID_GENERAL An error in the functioning of the mobile ID service.

VOTING_END The voting period has ended.

7.5 Vote Verification

The verification application makes the request `RPC.Verify` to download the signed vote and the certificates qualifying the vote from the collector service.

params.OS The operation system in which the verification application is used.

params.VoteID The identifier of the vote in the storage service retrieved from the voter application via a QR code.

```
{
    "id": 1,
    "method": "RPC.Verify",
    "params": [
        {
            "OS": "Operating System,2,0",
            "SessionID": "ec3a0cab353d552952289f2c7ad52e27",
            "VoteID": "VM/cUIU4n7VjxpUx1fC00Q=="
        }
    ]
}
```

result.Qualification.ocsp

result.Qualification.tspreg See the chapter on vote verification.

result.Type The signed vote format. At the moment, the only supported value is `bdoc`.

result.Vote BASE64-encoded vote `SignedVote` in the format specified above.

```
{
    "error": null,
    "id": 1,
    "result": {
        "Qualification": {
            "ocsp":
↪"MIIG8woBAKCCBuwwggboBgkrBgEFBQcwAQEEggbZMIIG1TCCASehgY...",
            "tspreg":
↪"MIIE0QYJKoZIhvcNAQcCoIIEwjCCBL4CAQMxDzANBglghkgBDQEJE..."
        },
        "SessionID": "027ab451969d9d3f044ea2cb2675b503",
        "Type": "bdoc",
        "Vote":
↪"UEsDBAoAAAAAAAAAACKIflFHwAAAB8AAAAIAAAAbWltZXR5cGVhcHB..."
    }
}
```

Possible error codes in case of the request `RPC.Verify`.

BAD_REQUEST An error in the request.

INTERNAL_SERVER_ERROR An error in the functioning of the internal server.

VOTING_END The voting period has ended.

CHAPTER 8

Processing the Ballot box

8.1 Revocation and Restoration List

The revocation and restoration list contains data on persons whose e-vote needs to be revoked (will not be counted when calculating the voting results) or restored (i.e. the previous revocation is cancelled and the restored e-vote is taken into consideration when votes are recounted). The list is uploaded into the system as a digitally signed document in the following data file format:

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "rev_entry": {
            "type": "string",
            "pattern": "^[0-9]+\\.[0-9]+$"
        }
    },

    "type": "object",
    "properties": {
        "election": {"type": "string"},
        "type": {"enum": ["revoke", "restore"]},
        "persons": {
            "type": "array",
            "items": {
                "$ref": "#/definitions/rev_entry"
            }
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "required": ["election", "persons", "type"],
    "additionalProperties": false
}
```

Example:

```
{
  "election": "TESTKOV",
  "persons": [
    "11412090004",
    "11412090005",
    "11412090006"
  ],
  "type": "revoke"
}
```

8.2 List of i-voters

The list of i-voters is a list of people who have cast their vote online, issued after the end of e-voting and sorted by polling stations. The document is generated by the processing application.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "onlinevoters": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+$": {
                    "type": "array",
                    "items": [
                        {
                            "type": "string"
                        },
                        {
                            "type": "number"
                        }
                    ],
                    "additionalItems": false
                }
            },
            "additionalProperties": false
        },
        "stations": {
```

(continues on next page)

(continued from previous page)

```
            "type": "object",
            "patternProperties": {
                "^[0-9]+.[0-9]+$": {
                    "$ref": "#/definitions/onlinevoters"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },
        "districts": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+.[0-9]+$": {
                    "$ref": "#/definitions/stations"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        }
    },

    "type": "object",
    "properties": {
        "election": {"type": "string"},
        "onlinevoters": {
            "$ref": "#/definitions/districts"
        }
    },
    "required": ["election", "onlinevoters"],
    "additionalProperties": false
}
```

Example:

```
{
  "election": "TESTKOV",
  "onlinevoters": {
    "164.1": {
      "164.1": {
        "11412090001": [
          "Nimi Nimeste",
          1
        ],
        "11412090002": [
          "Nimi Nimeste",
          1
        ],
        "11412090003": [
          "Nimi Nimeste",
```

(continues on next page)

(continued from previous page)


```
            1
          ]
        }
      },
      "296.1": {
        "296.1": {
          "11412090004": [
            "Nimi Nimeste",
            1
          ],
          "11412090005": [
            "Nimi Nimeste",
            1
          ],
          "11412090006": [
            "Nimi Nimeste",
            1
          ]
        },
        "296.2": {
          "11412090007": [
            "Nimi Nimeste",
            1
          ],
          "11412090008": [
            "Nimi Nimeste",
            1
          ],
          "11412090009": [
            "Nimi Nimeste",
            1
          ]
        }
      }
    }
}
```


8.3 Voting Result

The votes, decrypted by the key application and summed up, sorted by electoral districts and polling stations.

The voting result file has to contain the following data for every polling station.

1. An entry that shows the number of spoilt and invalid votes. If there were no spoilt or invalid votes in the polling station, the number of votes is zero
2. An entry that shows the number of votes given in favor of each choice. If no votes

were cast in favor of a choice in the polling station, the number of votes is zero

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "results": {
            "type": "object",
            "properties": {
                "invalid": {
                    "type": "integer"
                }
            },
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "type": "integer"
                }
            },
            "additionalProperties": false,
            "required": ["invalid"]
        },
        "stations": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/results"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },
        "district_dict": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/results"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },
        "stations_dict": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/stations"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
```

(continues on next page)

(continued from previous page)

```
        }
    },

    "type": "object",
    "properties": {
        "election": {"type": "string"},
        "bydistrict": {
            "$ref": "#/definitions/district_dict"
        },
        "bystation": {
            "$ref": "#/definitions/stations_dict"
        }
    },
    "required": ["election", "bydistrict", "bystation"],
    "additionalProperties": false
}
```

Example:

```
{
  "bydistrict": {
    "164.1": {
      "164.126": 0,
      "164.127": 0,
      "invalid": 0
    },
    "296.1": {
      "296.101": 0,
      "296.102": 0,
      "296.115": 0,
      "296.116": 0,
      "296.117": 0,
      "296.198": 0,
      "296.199": 0,
      "296.200": 0,
      "invalid": 0
    }
  },
  "bystation": {
    "164.1": {
      "164.1": {
        "164.126": 0,
        "164.127": 0,
        "invalid": 0
      }
    },
    "296.1": {
      "296.1": {
        "296.101": 0,
```

(continues on next page)

(continued from previous page)


```
        "296.102": 0,
        "296.115": 0,
        "296.116": 0,
        "296.117": 0,
        "296.198": 0,
        "296.199": 0,
        "296.200": 0,
        "invalid": 0
      },
      "296.2": {
        "296.101": 0,
        "296.102": 0,
        "296.115": 0,
        "296.116": 0,
        "296.117": 0,
        "296.198": 0,
        "296.199": 0,
        "296.200": 0,
        "invalid": 0
      }
    }
  },
  "election": "TESTKOV"
}
```


8.4 Ballot box

The file contains the votes received by the collector service along with the data accompanying the votes.

The file is in Zip64 container format.

The voter-specific folders are located immediately under a *votes* root directory.

File contents:

- votes/<voter id>/
- <timestamp>.version
- <timestamp>.<vote type>
- <timestamp>.<qualifier>*

where:

- <voter id> is the voter's identifier; in case of Estonia it is a personal identification code
- <timestamp> is the time of presenting the vote in the format yyyymmddhhmmssmmm±zzzz;

  - this is the moment when the request was sent to the collector service, and is simply given to improve the human-readability of the urn; the actual timestamp of the vote is inside one of the qualifying responses
- `<vote type>` is the choice container type; in case of Estonia it is BDOC
  - BDOC itself has a basic profile and does not contain any qualifying parameters (validity certifications, timestamps, etc.)
- `<qualifier>` is the type of vote-qualifying protocol; the following are currently possible:
  - `ocsp` - *Online Certificate Status Protocol* (validity certificate, RFC 6960[2]) confirms that the voter's signing certificate was valid at the moment of voting
  - `ocsptm` - the same as `ocsp`, but uses the BDOC[3] specification extension specified in Section 6.1, where the nonce is the vote signature hash in order to timestamp the vote
  - `tsp` - *Time-Stamp Protocol* (timestamp, RFC 3161[4]), which confirms that at the time of making the request the vote existed
  - `tspreg` - the same as `tsp`, but the nonce is the collector service signature on the request `MessageImprint` element to register the vote
- The files that exist for each vote are:
  - `<timestamp>.version` - the version of the voter list that was valid at the moment the vote was cast
  - `<timestamp>.<vote type>` - the choice container that includes the choice identifier in `<valimise id>.<küsimuse id>.ballot` form. In case of Estonia, the relevantly named file in the BDOC container
  - `<timestamp>.<qualifier>` - response to the vote-qualifying protocol request; there can be several, but no more than one for each protocol

8.5 Anonymized ballot box

Encrypted votes grouped by election districts and stations. Anonymized ballot box does not contain voter information.

The anonymized ballot box is the output of the processing application and input for the decryption application.

```
{
    "$schema": "http://json-schema.org/draft-04/schema#",

    "definitions": {
        "results": {
            "type": "array",
            "items": {
```

(continues on next page)

[2] https://tools.ietf.org/html/rfc6960
[3] http://www.id.ee/public/bdoc-spec212-est.pdf
[4] https://tools.ietf.org/html/rfc3161

(continued from previous page)

```
                "type": "string"
            },
            "additionalItems": false
        },
        "questions": {
            "type": "object",
            "additionalProperties": {
                "$ref": "#/definitions/results"
            },
            "minProperties": 1
        },
        "stations": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/questions"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        },
        "districts": {
            "type": "object",
            "patternProperties": {
                "^[0-9]+\\.[0-9]+$": {
                    "$ref": "#/definitions/stations"
                }
            },
            "additionalProperties": false,
            "minProperties": 1
        }
    },

    "type": "object",
    "properties": {
        "election": {"type": "string"},
        "districts": {
            "$ref": "#/definitions/districts"
        }
    },
    "required": ["election", "districts"],
    "additionalProperties": false
}
```

Example:

```
{
  "election": "TESTKOV",
  "districts": {
```

(continues on next page)

(continued from previous page)

```
    "164.1": {
      "164.1": {
        "TESTKOV.1": [ "MDkxOS4xMDUK", "MDkxOS4xMDQK", "MDkxOS4xMDEK
↪", "MDkxOS4xMDMK" ]
      }
    },
    "296.1": {
      "296.1": {
        "TESTKOV.1": [ "MDkxOS4xMDQK", "MDkxOS4xMDQK" ]
      },
      "296.2": {
        "TESTKOV.1": [ "MDExMi4xMDEK", "MDExMi4xMDQK", "MDExMi4xMDMK
↪" ]
      }
    }
  }
}
```

CHAPTER 9

Voting Result Audit

9.1 Shuffle Proof Verification

An algorithm is used to verify the shuffle proof, as defined in the Verificatum verifier implementing manual[5].

We would like to point out that when the shuffle proof is verified, the ciphertext is edited with data on the election, the district, the station and the question identifier. For addition, the relevant field is encoded as a group element, using the random factor 0 to ensure obfuscation. For example: if at first the ciphertext is $c_0 = (c_0 0, c_0 1)$, using the public key $pk = (g, y)$, then for Verificatum's input the wide ciphertext $C = (c_{id}, c_d, c_s, c_q, c_0)$, is used, where:

- the election identifier pseudo-ciphertext is given as $c_{id} = (1, encode(id))$, where the function $encode$ encodes the string as an element in the relevant group and *id* is the election identifier string
- the electoral district identifier pseudo-ciphertext is given as $c_d = (1, encode(d))$, where *d* is the district identifier string
- the polling station identifier pseudo-ciphertext is given as $c_s = (1, encode(s))$, where *s* is the station identifier string
- the question identifier pseudo-ciphertext is given as $c_q = (1, encode(q))$, where *q* is the question identifier string

In this case, the public key corresponding to the wide ciphertext is defined as $((g, 1), (g, 1), (g, 1), (g, 1), (g, y))$.

[5] https://www.verificatum.org/files/vmnv-3.0.3.pdf

9.2 Verification of the Decryption Proof

Let us have ciphertext $c = (c_0, c_1)$, which is decrypted into the value d with the given public key pk over the parameters (p, g) and with the decryption proof (a, b, s).

To check that the decryption was correct, the challenge $k = H("DECRYPTION"||pk||c||d||a||b)$ is calculated, and where H is the SHA-256 hash function. Then it is verified that $c_0^s = a * (c_1/d)^k$ and $g^s = b * y^k$.

9.3 Correct Conversion Check

To make sure that the conversion between the IVXV ballot box and the Verificatum ciphertexts has been performed correctly, the conversion has to be repeated independently. After an independent conversion, the outputs have to be compared. Since conversion is a deterministic procedure, a repeat conversion will guarantee the correctness of the action.