POLYAS

POLYAS verification tool handout developer
Version 1.3.2

December 4th, 2015

German Version: click here

1. Subject

The description contained in this document is intended to enable software developers to independently implement a software, here called *verification tool*, which allows to verify the correctness of the election results by recounting. It uses the election data (database contents in XML format from the electoral roll and ballot box), which are published by the respective election committee after a completed online election using POLYAS voting software, and recounts the data.

2. Preparation

2.1. Obtaining the election data

You will receive the necessary election data and further information about the election from your election committee. If you have any questions about the verification tool, please contact POLYAS GmbH at the following email address: info@polyas.com

2.2. Checking the provided election data for completeness

First, make sure that the election committee provides you with the following data / files:
1. the election data of the electoral roll as an XML file in a ZIP archive (auszaehlung-wvz.zip),
2. the election data of the ballot box as an XML file in a ZIP archive (auszaehlung-urne.zip),
3. the keystore (destKeyStore.jks),
4. the checksum via the ZIP archive with the election data from the electoral roll, the checksum via the ZIP archive with the election data of the ballot box as well as the checksum of the keystore, checksum.csv (see the description of *checksum* in the glossary),
5. the password for the keystore (keystorepassphrase) and the password of the private key (countkeypassphrase) for the ballot box (passphrases.csv). This private key is stored in the keystore.

You can verify the authenticity and integrity of the files received (election data and keystore)
by re-creating the checksums (SHA-256) for the individual files and comparing them to the checksums issued by the election committee. For this purpose, the following programs are available for various operating systems that allow you to easily create the checksums.

- Windows - for example the *Fsum frontend* (http://fsumfe.sourceforge.net/) or *md5deep* (http://md5deep.sourceforge.net/#sha256)
- Linux and MacOS - the command line command sha256sum or shasum -a 256

2.3. Required software and libraries

POLYAS voting system is implemented in the Java programming language and is particularly effective for the implementation of the cryptographic functionality on various free Java libraries. It is therefore advisable to use Java to be able to implement the verification tool. The following packages are required:

- Java Development Kit (JDK) 1.7 or higher.
- Java Cryptographic Extension (JCE): To use the strong encryption, the JCE package, which is subject to American export regulations, must be installed subsequently. The version compatible with the JDK version is to be used here (e.g. “Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7”). The JCE consists of a single ZIP UnlimitedJCEPolicyJDK7.zip This has to be unpacked and the two .jar files it contains have to be copied over the existing files in the directory jre/lib/security/ of the installed Java distribution (thus, the files there will be overwritten).

A trusted, standards-compliant implementation of a security provider for the *Cryptography API of* the Java Security Framework. We recommend the implementation of BouncyCastle, which is also used by POLYAS (see http://www.bouncycastle.org/).

To provide its safety-critical functionality, the POLYAS voting system uses various cryptographic functions and algorithms. The following algorithms must therefore also be available for verification:

- AES (256-bit) for symmetrical encryption of data,
- RSA for asymmetrical encryption of data (private / public key),
- SHA-256 for forming of checksums over data,
- SHA-256 with RSA for the verification of digital signatures, (encrypted and then secure signed data).

Note: Make sure to use the corresponding 32- or 64-bit versions.

2.4. Keystore

The keystore of the ballot box, which is needed to decrypt the votes in the ballot box, is delivered in Java-keystore format and usually has the .jks file extension.

See http://docs.oracle.com/javase/7/docs/api/java/security/KeyStore.html

The keystore contains the following public keys

- Alias validator, the public key of the validator,
- Alias registry, the public key of the electoral roll,
- Alias vote, the public key of the ballot box. The private part of the key was used to sign the votes.

The keystore also contains the private key

• Alias votecount, the private key for counting. The public key part of this key was used to encrypt the votes.

To verify this, you can display the contents of the keystore using the command line keytool supplied with the Java distribution. Assuming the keystore you want to use is called vote.jks, then execute the following command:

keytool -list -v -keystore destKeyStore.jks

You will be asked to enter the password for the keystore and after correct entry you will be shown detailed information of all keys contained. The output should look like the following:

Aliasname: voteCount
Creation date: 10/31/2012
Entry type: PrivateKeyEntry
Certificate chain length: 1

Certificate [1]:
Owner: CN = key urn, OU = Polyas, O = Polyas GmbH, L = Kassel, ST = Hessen, C = DE
Issuer: CN = key urn, OU = Polyas, O = Polyas GmbH, L = Kassel, ST = Hessen, C = DE
Serial number: 509106bb
Valid from: Wed Oct 31 12:08:43 CET 2012 to: Sat Oct 29 13:08:43 CEST 2022
Digital fingerprint of the certificate:
MD5:
F5: 26: D4: C7: E0: 1C: BB: 94: 7A: E2: 98: 1F: 11: 42: 1F: A5
SHA1: 49: 22: EB: 57: 54: 7D: D4: 0B: 0A: 9A: 83: F7: 60: B0: 8C: DE: 97: 9E: CD: 59
Signature algorithm name: SHA256withRSA
Version: 3

3. Election data

The election data of the electoral roll and the ballot box are delivered as an XML file. The file contains the election data of the respective subsystem at the time the election is shut down (electoral stop). The files must be in UTF-8 encoding.

The expected formats and contents of the XML files are described in detail below and illustrated with examples. Please note the examples:

1. The string “{JSON}” inserted in the examples indicates that data in JSON format is expected at this point in the actual XML document.
2. The string “{x | y | z}” inserted in the examples indicates that in the actual XML document at this point either the value x or y or z is expected at this point, where x , y , z are placeholders for the actual values.
3. The character string "..." inserted in the examples indicates that additional characters are expected in the actual XML document. The data format of the expected character string is specified in the respective description of the elements / attributes below.

3.1. Data format of the election data in the electoral roll

Example of an XML file with election data from the electoral roll

```
<registry-result date = "yyyy-MM-ddTHH: mm: ssZZZ" num-voters = "..." has-voted = "..." >
<voters>
<voter pin = "..." key-present = "{true | false}" has-voted = "{0 | 1 | 2}" ballots = "..."
sign-registry = "..." sign-validator = "..." data = "{JSON}" />
<voter pin = "..." key-present = "{true | false}" has-voted = "{0 | 1 | 2}" ballots = "..."
sign-registry = "..." sign-validator = "..." data = "{JSON}" />
</voters>
<ballot-templates>
<ballot-template id = "101" name = "..." data = "{JSON}" />
<ballot-template id = "102" name = "..." data = "{JSON}" />
</ballot-templates>
<checksums>
```

<checksum> ... </checksum>
<checksum> ... </checksum>
</checksums>
</registry-result>

The structure of the XML is described in a table below.

- 1st column: Heading contains the name of the element described in the table. The values list the elements and attributes.
- 2nd column: Type. In the case of an attribute, the type of the data is specified here. In the case of an element, this column remains empty.
- 3rd column: M. Indicates the multiplicity of the element or attribute.
- 4th column: Description.

Election result electoral roll

registry result	Type	M	Description
date	String	1	Timestamp of the count; in xsd:date format: date format (ISO 8601)
num-voters	Long	1	Total number of registered voters
has Voted	Long	1	Number of voters who cast their vote
voters	_	1	Container for the list of voter representations
ballot-templates	_	1	Container for the list of ballot templates
checksums	_	1	Container for the list of hash sums

Voter representation

Voters	Type	M	Description

voter	_	0+	voter representations

Voter	Type	M	Description
pin	String	1	PIN
key-present	Boolean	1	Note whether the registered voter did not cast his vote: true == voter registered, but did not cast a vote; required for statistics in PDF.
has-voted	Integer	1	Identification of the vote; possible values: 0 == not submitted, 1 == not successfully submitted, 2 == successfully submitted
ballots	String	1	List of the ballot IDs separated by "|" characters
sign registry	String	1	TAN signature of the electoral roll ($Sig_{SKregistry}$ (*pin*,*saltedtan*)) as Base64 coding of the SHA256_RSA signature, i.e. pin and tan are concatenated as string and the bytes of the result string are signed.
sign-validator	String	1	Validator signature of the TAN signature of the electoral roll: $Sig_{SKvalidator}$ ($Sig_{SKregistry}$ (*pin*,*saltedtan*)) as Base64 coding of the SHA256_RSA signature
Tan	String	0/1	Salted TAN to the PIN
Data	JSON	0/1	Information about voters in JSON format, such as Last name, First name, Adress, etc.

Ballot templates of an electoral group

ballot templates	Type	M	Description

ballot template	_	1+	ballot template
ID	Integer	1	ID of the ballot template
name	String	1	Title of the ballots
data	JSON	1	Election rules and other information about ballot papers in JSON format, content is not relevant for data verification.

Block checksums of the urn

checksums	Type	M	Description
checksum	_	0+	Element for checksum of the ballot box from the database of the electoral roll

checksum	Type	M	Description
__	CDATA	1	Checksum of the ballot box from the electoral roll database, transmitted with each complete block of 30 from the ballot box, must match the checksums from the ballot box file.

3.2. Data format of the election data from the ballot box

Example of an XML file with election data from the ballot box

```
<vote-result date="yyyy-MM-ddTHH:mm:ssZZZ">
  <ballot-templates>
    <ballot-template id="101" method="..." data="{JSON}">
      <sheet-template id="10101" method="..." data="{JSON}">
```

```
            <candidate-template id="1" data="{JSON}" />
            <candidate-template id="2" data="{JSON}" />
          </sheet-template>
          <sheet-template id="10102" method="..." data="{JSON}">
            <candidate-template id="3" data="{JSON}" />
          </sheet-template>
        </ballot-template>
    </ballot-templates>
    <block id="1">
      <vote key="..." commited="{0|1|2|3}" signed-object="...">
        <ballot reference="101" valid="{true|false}" abstain="{true|false}"
intentionally-invalid="{true|false}">
          <sheet reference="10101" valid="{true|false}">
            <candidate reference="1" count-yes="2" count-no="0" />
            <candidate reference="2" count-yes="0" count-no="0" />
          </sheet>
          <sheet reference="10102" valid="{true|false}">
            <candidate reference="3" count-yes="1" count-no="0" />
          </sheet>
        </ballot>
      </vote>
    </block>
    <checksums>
      <checksum>...</checksum>
      <checksum>...</checksum>
    </checksums>
</vote-result>
```

Election result of the ballot box (vote-result)

The top element of the XML file of the election data of the ballot box is **vote-result**.

vote-result	Type	M	Description
date	String	1	Timestamp of the count; in xsd:date format (ISO 8601)
ballot templates	__	1	List of ballot paper templates
block	__	0+	Blocks of votes
checksums	__	1	Checksums

Ballot Templates

Ballot templates	Type	M	Description
Ballot template	1	0+	Definition of a ballot template

ballot template	Type	M	Description
id	Integer	1	ID
method	String	1	Not in use
data	JSON	1	Election rules and other information about ballot papers in JSON format. See description of the data attribute further below. See ballot-template
sheet-template		1+	

Data-Attribut (JSON-String) of the Elements ballot-template

Keyword	Type	M	Description
name	String	1	Name of this ballot
sheets	String-List	0+	List of Sheet IDs, which apply for this ballot

Example:
{"sheets": [1,2],
"name": "Ballot 1",
...
}

The JSON string in the data attribute can also contain other entries, but they do are all not relevant to the enumeration.

Sheet-template	Type	M	Description
id	Integer	1	ID
method	String	1	Name of the election method / mode (simple election, yes / no vote, list election etc.), see. Election rules section
data	JSON	1	Election rules and other information about ballot papers in JSON-format. See description of the data attribute further below. sheet-template
candidate-template	__	1+	

The data attribute (JSON-string) of the element *sheet-template*

Keyword	Type	M	Description
Max-votes	Integer	1	Number of maximum votes to be cast; must be a positive integer.
columns	String	n	Column title of the columns with the candidates

Example:
{"max-votes": 8,
"columns": ["first name", "last name"],
...
}

The JSON-string in the data attribute can also contain other entries, none of which are relevant to the vote count.

candidate-template	Type	M	Description
id	Integer	1	ID
data	JSON	1	JSON Representation of the candidate, key column1, column2, ... in the order of the column caption from the parent columns element of the sheet-template, value is the respective content.

Example:
{"column1": "Erika",
"column2": "Mustermann"}

Blocks of 30 votes (block)

Block	Type	M	Description
id	Integer	1	ID
vote	__	1-30	Vote in the block of votes

Votes (vote)

vote	Type	M	Description
key	String	1	Unique name of the vote from the database. Random value after active deletion of the election token.
commited	Integer	1	Identification of voting; possible values: [0,1,2,3]

			where only 3 == cast successfully
signed-object	String	1	Representation of the encrypted vote
ballot	__	1+	Vote of the voters

Completed ballot (ballot)

ballot	**Type**	**M**	**Description**
reference	Integer	1	Reference to the ID of the ballot template
valid	Boolean	1	Note whether the ballot paper is valid according to the election rules
abstain	Boolean	1	Note whether there is an abstention
intentionally-invalid	Boolean	1	Note whether the voter explicitly (via button) has cast an invalid vote
sheet	__	1+	filled in candidate list

Completed candidate lists (sheet)

sheet	**Type**	**M**	**Description**
reference	Integer	1	Reference to ID of the candidate list template
valid	Boolean	1	Note whether votes in the candidate list are valid
candidate	__	1+	Vote for a candidate on the candidate list

Candidates with precise number of votes (candidate)

Candidate	Type	M	Description
reference	Integer	1	Reference to ID of the candidate list template
count-yes	Integer	1	Number of votes cast
count-no	Integer	0-1	Not used, always 0

Block checksums of the ballot box

checksums	Type	M	Description
checksum	_	1+	List of ballot box checksums from the ballot box database

checksum	Type	M	Description
__	**CDATA**	**1**	Checksum of a block of 30 votes of the ballot box from the ballot box database

4. Processing steps

4.1. Reading the election data

First of all, the data contained in the XML representations of the election data must be read from the corresponding files according to the data formats described in the previous section (see section 3) (note: the encoding should be UTF-8) and converted into suitable representations of the programming language used. It is assumed in the following, that the Java language should be used for the implementation of a verification tool. Thus, the XML elements could be represented by objects of the classes: Candidate, HashcodeSequence (block checksum sequence), Voter and Votum (ballot paper cast / vote cast).

4.2. Decoding and counting votes

The most important part of the verification is the counting of the votes cast and the comparison of the results with the published election results. For this purpose, the votes still encrypted in the attribute signed-object of the vote element must be decrypted and compared with the vote counts already available in plain text in XML format. The data thus

compared is finally counted. If the data contained in the vote element has been read in correctly and completely, the representation of each vote entry contains the attribute committed and the attribute signed-object. An entry in the ballot box is initially only considered to be a validly cast vote if the committed attribute has the value 3.

The character string stored in signed-object is a string representing the vote as an object (serialized Java object, zipped), which is encrypted with the public key with the alias 'voteCount', signed with the private (signature) key of the ballot box, alias 'vote', and finally Base64 encoded. It should be noted that the signature was performed using an object of type Object[] (Java Object Array) of length two, the first entry containing the AESS key (symmetric encryption of the data stream) encrypted with the private key using RSA and the second entry containing the actual content, i.e. the serialized object with the vote information.

The following steps are necessary to decrypt the vote:
1. Decoding of the character string according to Base64 encoding,

2. Deserialization of the bytes in an instance of the class java.security.SignedObject. The instance of the SignedObject contains a Java array of Object type with a length of two. The entry with index 0 contains the encrypted AES key (see step 4). The entry with index 1 contains the AES-encrypted zipped data of the votes

3. Verification of the signature of the decoded character string with the public key of the ballot box (alias vote in the keystore of the ballot box) according to SHA-256 with RSA ("SHA256withRSA"),

4. Decryption of the *256 bit* AES key using RSA with the private key, alias 'voteCount' and finally

5. Decryption of the actual content via AES with the AES key.

6. Unzip and deserialize the contained Java object of type java.util.Map <Integer, Object []>

This process is illustrated by the code in Section 4.2.3.

The result of this procedure is a Java object of the type java.util.Map <Integer, Object []>, see JSON example. The Map contains an illustration of the ballot number - the ballot on which the voter cast their vote - on a ballot structure of type Object [], which ultimately has the following structure:

• Index 0: an object of type java.util.Collection, which contains the string representation of the candidates, as described under Candidate Representation, for which the voter cast their vote. If there are several votes for a candidate, this will be repeated according to the number of votes.

• Index 1: a boolean that indicates whether the ballot structure has been voted invalid, as well as
• Index 2: an object of the type java.util.Map attribute name on attribute type, this can be used for additional information in the future.

Note: Depending on the election rules, a decision may have to be made on whether a ballot will be considered invalid if one of several ballots within the ballot is invalid.

Candidate representation
A candidate is represented as a string within a ballot. This string contains 3 components, which are separated by the pipe symbol (vertical line). These parts correspond to a database representation of a candidate.

1. Candidate-ID, the unique number of this candidate
2. Sheet-ID, the unique number of the sheet containing the candidate, must match the key of the Map
3. JSON structure with the attributes of the candidate, for example, the displayed column values column1, ...

Example

Example of a decoded vote in a Javascript representation:

```
{
   1: [
      [
         "11|1|{'name':'Max Müller','databaseId':'saas_id128 Ja'}",
         "21|1|{'name':'Erika Mustermann','databaseId':'saas_id129 Ja'}",
         "32|1|{'name':'Lisa Meier','databaseId':'saas_id130 Nein'}",
         "41|1|{'name':'Matthias Meyer','databaseId':'saas_id131 Ja'}"
      ],
      false,
      {}
   ],
   2: [
      [
         "70|2|{'name':'Carsten Schulz','databaseId':'saas_id1238'}",
         "50|2|{'name':'Karsten Schmidt','databaseId':''saas_id1239''}"
      ],
      false,
      {}
   ]
}
```

This vote represents a vote for 2 ballots, sheet-template id 1 and 2. The first sheet has the voting mode CHECKBOX_YES_NO, the second CHECKBOX_SINGLE_YES, see below

4.2.1 Election rules / election modes

POLYAS allows you to define an election mode for entire ballots as well as for individual candidate lists within a ballot (sheet). The following election modes are supported:

Keyword:	**Description**
CHECKBOX_SINGLE_YES	Checkboxes for individual candidates, i.e. you can assign one or more votes to each candidate
CHECKBOX_YES_NO	One YES and NO checkbox for each individual candidate. Like CHECKBOX_SINGLE_YES, except that you can also give "negative" votes (i.e. 'rejection').

In the case of the CHECKBOX_YES_NO method, 2 candidate entries are generated in the sheet template for each candidate. One for the “yes” option and one for the “no” option. You can recognize the assignment by the candidate ID, which has at least two digits. The right digit is coded as follows:

Right Number	**Description**
0	CHECKBOX_SINGLE_YES election rule.
1	CHECKBOX_YES_NO, 'YES' vote
2	CHECKBOX_YES_NO, 'NO' vote

The rest of the candidate ID uniquely describes the candidate.

The currently valid election mode for a ballot paper or a list of candidates can be taken from the Attribute method of the templates of the ballot papers or candidate lists (see the following section). The other election rules (for example, the maximum number of votes per ballot) are contained in the attribute data of the templates for ballots or candidate lists (also described in the following section).

4.2.2. Counting

To count the votes, the entries in the ballot box are considered invalid votes if they:

- are marked as invalid (Ballot),
- contain more than the maximum allowed number of votes to be cast (see the described list of selected candidates - above: the java.util.Collection).

Entries which, as described in Section 4.2, do not represent ballots that have not been finally submitted and whose Attribute committed is not equal to 3, and which should therefore also be regarded as not finally submitted, may not be taken into account in the counting.

Ballots with the CHECKBOX_YES_NO voting method are to be considered invalid if both a 'NO' vote and a 'YES' vote were cast for at least one candidate.

The counting of votes for the candidates nominated may therefore only include those entries in the ballot box for which none of the above-mentioned characteristics apply. To ensure a correct comparison of the candidates contained in a submitted ballot with the candidates actually nominated (see Section 4.2.1), the representation of the candidates on the submitted ballot must be known (see Section 3) and a string comparison (character-by-character comparison) must be performed.

It may be that the ballot papers marked as 'invalid' by the voter must be shown separately from those that are invalid due to the counting rules.

4.2.3. Code examples

The following code illustrates how to check the string in the signed-object attribute and decode the information about the ballot it contains:

```
import java.io.ByteArrayInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.ObjectInputStream;
import java.security.KeyStore;
import java.security.PrivateKey;
import java.security.Provider;
import java.security.PublicKey;
import java.security.Signature;
import java.security.SignedObject;
import java.util.Base64;
import java.util.Map;
import java.util.zip.GZIPInputStream;

import javax.crypto.Cipher;
import javax.crypto.spec.IvParameterSpec;
import javax.crypto.spec.SecretKeySpec;

import org.bouncycastle.jce.provider.BouncyCastleProvider;


public class SeleadSignedObjectDecrypt {
   private final static String SSO = "...";// signed-object
   private final static String KEYSTORE = "path/to/the/keystore.jks";
   private final static String DEC_KEYALIAS = "votecount";
   private final static String VER_KEYALIAS = "vote";
   private final static String JKS_PASS = "...";
```

```
    private final static String KEY_PASS = "...";
    private final static Provider PROVIDER = new BouncyCastleProvider();


    /***deserializes the BASE24 encoded Object, verifies the signature, decodes
the*BlockCipher Key and decrypts the ballot*/


    public static Map<Integer, Object[]> getContent(
            final PublicKey verifKey,
            final PrivateKey decodeKey,
            final String base64object) throws Exception {
        final ObjectInputStream oistream = new ObjectInputStream(new
ByteArrayInputStream(Base64.getDecoder().decode(base64object.getBytes())));
        final SignedObject so = (SignedObject) oistream.readObject();
        final boolean valid = so.verify(verifKey, Signature.getInstance("SHA256withRSA",
PROVIDER));
        if (valid == false) {
            throw new RuntimeException("Can't verify signature");
        }
        final Object votum = so.getObject();
        if (!(votum instanceof Object[])) {
            throw new RuntimeException("Corrupted vote");
        }
        final Object[] content = (Object[]) votum;
        final byte[] rsaEncKey = (byte[]) content[0];
        final Cipher rsaCipher = Cipher.getInstance("RSA", PROVIDER);
        rsaCipher.init(Cipher.DECRYPT_MODE, decodeKey);
        final byte[] rsaDecKey = rsaCipher.doFinal(rsaEncKey);
        final byte[] aesKey = new byte[32];
        System.arraycopy(rsaDecKey, 0, aesKey, 32 - rsaDecKey.length, rsaDecKey.length);
        final byte[] iv = new byte[16];
        System.arraycopy(rsaEncKey, 0, iv, 0, 16);
        final IvParameterSpec ivParam = new IvParameterSpec(iv);
        final Cipher streamCipher = Cipher.getInstance("AES/GCM/NoPadding", PROVIDER);
        streamCipher.init(Cipher.DECRYPT_MODE, new SecretKeySpec(aesKey,
"AES/GCM/NoPadding"), ivParam);
        final byte[] dec = streamCipher.doFinal((byte[]) content[1]);
        final ByteArrayInputStream bais = new ByteArrayInputStream(dec);
        final ObjectInputStream ois = new ObjectInputStream(new GZIPInputStream(bais));
        final Map<Integer, Object[]> loaded = (Map<Integer, Object[]>) ois.readObject();
        ois.close();
        bais.close();
        return loaded;
    }

    public static void main(final String[] args) throws Exception {
```

```
        // Extract the keys from keystore

        final File file = new File(KEYSTORE);
        final FileInputStream is = new FileInputStream(file);
        final KeyStore keystore = KeyStore.getInstance(KeyStore.getDefaultType());
        keystore.load(is, JKS_PASS.toCharArray());
        final PrivateKey privateKey = (PrivateKey) keystore.getKey(DEC_KEYALIAS,
KEY_PASS.toCharArray());
        final PublicKey publicKey = keystore.getCertificate(VER_KEYALIAS).getPublicKey();
        // decode the ballot
        final Map<Integer, Object[]> decodeVote = getContent(publicKey, privateKey, SSO);
    }
}
```

4.3. Calculation and comparison of the block checksums

In order to verify the correctness of the block checksums formed during the election, the checksums of the two XML representations (electoral roll, ballot box and element checksums) must be compared. They must match in order and value. Secondly, the block checksums should also be recalculated from the votes in the ballot box (signed-object) and then compared in pairs with those in the XML representations. To determine the block checksums from the votes in the ballot box, proceed as follows:

1. Sort all (finally) cast ballot papers in ascending order according to the id of the block in which they are located (structurally within a block element) and within a block in ascending order alphanumerically according to the key (G-. . .),

2. Form the hash value using SHA-256 for the string "(C) Micromata Polyas" (the quotation marks are not part of the string) and remember it as the initial hash value,

3. Form in ascending order of the block numbers for each voice block the hash value as follows:

    1. Take the hash value of the previous block - for the block with the number 1, take the initial hash value - as a character string and add to this character string for each ballot, one after the other, the character string contained in signed-object (without spaces or such like),
    2. Form the hash value through this character string using SHA-256,
    3. Remember the hash value as a checksum for the current block.

5. Verification steps

The following steps are necessary to check the integrity and recounting of the given data.

Data delivery

1. Check the data delivery for completeness, see section 2.2 .
2. Check the hash sums of the files contained, see section 2.2 .
3. Check the availability of the required keys in the keystore, see sections 2.2 and 2.4 .

Electoral roll

1. Check all signatures sign-registry with the public key 'registry', see Section 3.1 .
2. Verification of all signatures sign-validator using the public key 'validator', see Section 3.1 .
3. Check if the checksum elements match the order and equality of the values with the entries in the XML ballot boxes.

Ballot box

1. Check the hash chain in the element checksums by recalculating using the data given in the XML. Note that the last, possibly opened block does not have its own checksum, see Section 4.3.

2. Check the signature of the encrypted signed-object vote, see sections 4.2 and 4.2.3.

3. Decode the vote contained in the signed-object, see section 4.2 and 4.2.3.

4. Check that the votes decoded match the corresponding information in the XML. In particular, the assignment of candidates, sheets and ballots to their reference numbers in the element ballot-template, sheet-template and candidate-template should be checked, see sections 3.1, 3.2 and 4.2.

5. Start the vote count

6. Glossary

Checksum

A checksum is a character string that is created for a file or a data record and uniquely identifies it. Different files or data records always receive different checksums. POLYAS currently uses checksums from the SHA-2 family (Secure Hash Algorithm), more precisely: SHA-256. The checksums generated in their hexadecimal representation consist of exactly 64 characters.
An example of a checksum generated with SHA-256:
0739fdaee043869d6c481922bd780cd2cbb563c3cc6a1b15bd347820dd37dfd0

Block checksum

The block checksum is a checksum (see Checksum) that is formed over a block of cast ballots and the block checksum of the previous block. Ballots within a block are sorted uniquely for this purpose. However, this sorting is carried out according to random values assigned to you when you cast your vote. The random values ensure that the sequence in which the ballots were cast in the election and an assignment to the voters who cast them is

impossible.

Keystore
A keystore is a file which contains private and / or public digital keys (see keys, private key) for an (asymmetrical) encryption of data and is secured with a password.

Digital key
A private digital key is used to encrypt sensitive data in such a way that only the person who has encrypted the data with this key can decrypt it again. A private key can also be used to create a unique digital signature. With the help of such a signature, recipients of data signed in this way can verify the correct identity of the sender/originator. To prevent misuse of such a private key by third parties, it is provided with a password known only to the owner of the key. A public digital key is generated from a private key and can be issued to any third party, who can then encrypt data in such a way that only the owner of the corresponding private key can decrypt it again can.