POLYAS Verificationtool Handreichung Entwickler

Version 1.3.2

4. Dezember 2015

Inhaltsverzeichnis

1. Gegenstand

Die in dem vorliegenden Dokument enthaltene Beschreibung soll es Softwareentwicklern ermöglichen, eigenständig eine Software zu implementieren, hier *Verifikationtool* genannt, welche es erlaubt anhand der Wahldaten (Datenbankinhalte als XML-Dateien des Wählerverzeichnisses und der Urne), die nach einer abgeschlossenen Online-Wahl mit der Wahlsoftware POLYAS vom Wahlvorstand veröffentlicht werden, die Korrektheit des Wahlergebnisses durch erneute Auszählung zu überprüfen.

2. Vorbereitung

2.1. Bezug der Wahldaten

Die notwendigen Wahldaten und weitere Informationen über die Wahl erhalten Sie vom Wahlvorstand. Falls Sie Fragen über das Verifikationstool haben, wenden Sie sich bitte an die POLYAS GmbH unter folgender E-Mail Adresse: support@polyas.de.

2.2. Überprüfung der bereitgestellten Wahldaten auf Vollständigkeit

Stellen Sie zunächst sicher, dass Ihnen der Wahlvorstand die folgenden Daten/Dateien ausgeliefert hat:

1. die Wahldaten des Wählerverzeichnisses als XML-Datei in einem ZIP-Archiv (`auszaehlung-wvz.zip`),
2. die Wahldaten der Urne als XML-Datei in einem ZIP-Archiv (`auszaehlung-urne.zip`),
3. der Keystore (`destKeyStore.jks`),
4. die Prüfsumme über das ZIP-Archiv mit den Wahldaten des Wählerverzeichnisses, die Prüfsumme über das ZIP-Archiv mit den Wahldaten der Urne sowie die Prüfsumme des Keystores, `checksum.csv` (vgl. Beschreibung *Prüfsumme* im Glossar),
5. das Passwort für den Keystore (`keystorepassphrase`), sowie das Passwort des privaten Schlüssels (`countkeypassphrase`) der Urne (`passphrases.csv`). Dieser private Schlüssel ist im Keystore abgelegt.

Die Überprüfung der Authentizität und Integrität der erhaltenen Dateien (Wahldaten und Keystore) können Sie durch erneute Bildung der Prüfsummen (SHA-256) über die einzelnen Dateien und den Abgleich mit den vom Wahlvorstand ausgegebenen Prüfsummen durchführen. Hierzu stehen Ihnen für die verschiedenen Betriebsysteme Programme zur Verfügung, die eine einfache Erstellung der Prüfsummen ermöglichen.

- Windows – beispielsweise das *Fsum Frontend* (http://fsumfe.sourceforge.net/) oder *md5deep* (http://md5deep.sourceforge.net/#sha256)
- Linux und MacOS — der Kommandozeilen-Befehl `sha256sum` bzw. `shasum -a 256`

2.3. Benötigte Software und Bibliotheken

Das Wahlsystem POLYAS ist in der Programmiersprache Java implementiert und greift insbesondere für die Realisierung der kryptographischen Funktionalität auf verschiedene freie Java-Bibliotheken zurück. Es ist daher anzuraten für die Implementierung eines Verifikationstools ebenfalls Java zu verwenden. Vorauszusetzen sind somit folgende Pakete:

- Java Development Kit (JDK) 1.7 oder höher.
- Java Cryptographic Extension (JCE): Für die Verwendung der starken Verschlüsselung ist das - den amerikanischen Exportbestimmungen unterliegende - JCE Paket nachträglich zu installieren. Hier ist die zur JDK-Version kompatible Version zu verwenden (z. B. "Java Cryptography Extension (JCE) Unlimited Strength Jurisdiction Policy Files 7"). Die JCE besteht aus einem einzelnen ZIP `UnlimitedJCEPolicyJDK7.zip`. Dieses muss entpackt werden und die beiden darin enthaltenen `.jar`-Dateien müssen über die bestehenden Dateien im Verzeichnis `jre/lib/security/` der installierten Java Distribution kopiert werden (d.h. die dort vorhandenen Dateien werden überschrieben).
- Eine vertrauenswürdige, standardkonforme Implementierung eines Security Providers für die *Cryptography API* des Java Security Frameworks. Empfehlenswert ist hier die auch von POLYAS eingesetzte Implementierung von BouncyCastle (siehe http://www.bouncycastle.org/).

Das POLYAS-Wahlsystem setzt zur Bereitstellung seiner sicherheitskritischen Funktionalität verschiedene kryptographische Funktionen und Algorithmen ein. Die folgenden Algorithmen müssen daher auch zur Verifikation bereitstehen:

- AES (256-bit) für eine symmetrische Verschlüsselung von Daten,
- RSA für eine asymmetrische Verschlüsselung von Daten (Private-/Public-Key),
- SHA-256 für die Bildung von Prüfsummen über Daten,
- SHA-256 mit RSA für die Überprüfung digitaler Signaturen, (verschlüsselte und dann signierte Daten).

Hinweis: Es ist darauf zu achten, die einander entsprechenden 32 bzw. 64 bit Versionen zu benutzen.

2.4. Keystore

Der Keystore der Urne, welcher zur Entschlüsselung der Stimmen in der Urne benötigt wird, wird im Java-Keystore-Format ausgeliefert und hat in der Regel die Dateiendung .jks .

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

Der Keystore enthält die folgenden öffentlichen Schlüssel

- Alias `validator`, der öffentliche Schlüssel des Validators,
- Alias `registry`, der öffentliche Schlüssel des Wählerverzeichnisses,

- Alias `vote` der öffentliche Schlüssel der Urne. Mit dem privaten Teil des Schlüssels wurden die Stimmen signiert

Der Keystore enthält darüber hinaus den privaten Schlüssel

- Alias `votecount`, der private Schlüssel zum Auszählen. Mit den öffentliche Schlüsselteil dieses Schlüssels wurden die Stimmen verschlüsselt.

Um dies zu verifizieren können Sie sich den Inhalt des Keystores mittels des mit der Java-Distribution ausgelieferten Kommandozeilen-Befehls keytool anzeigen lassen. Angenommen der Ihnen zur Verfügung gestellte Keystore heiße vote.jks, dann führen Sie folgenden Befehl aus:

```
keytool -list -v -keystore destKeyStore.jks
```

Sie werden dazu aufgefordert, das Passwort für den Keystore einzugeben und bekommen nach korrekter Eingabe detaillierte Informationen aller enthaltenen Schlüssel angezeigt. Die Ausgabe sollte etwa wie folgt aussehen:

```
Aliasname: voteCount
Erstellungsdatum: 31.10.2012
Eintragstyp: PrivateKeyEntry
Zertifikatskettenlänge: 1
Zertifikat[1]:
Eigner: CN=Schluessel Urne, OU=Polyas, O=Polyas GmbH, L=Kassel, ST=Hessen, C=DE
Aussteller: CN=Schluessel Urne, OU=Polyas, O=Polyas GmbH, L=Kassel, ST=Hessen, C=DE
Seriennummer: 509106bb
Gültig von: Wed Oct 31 12:08:43 CET 2012 bis: Sat Oct 29 13:08:43 CEST 2022
Digitaler Fingerabdruck des Zertifikats:
     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
     Unterschrift-Algorithmusname: SHA256withRSA
     Version: 3
```

3. Wahldaten

Die Wahldaten des Wählerverzeichnisses und der Urne werden als XML-Datei geliefert. Die Datei enthält die Wahldaten des jeweiligen Teilsystems zum Zeitpunkt der Abschaltung der Wahl (Wahlstopp). Die Dateien müssen in der UTF-8-Kodierung vorliegen.

Im Folgenden werden die erwarteteten Formate und Inhalte der XML-Dateien im Einzelnen beschrieben und anhand von Beispielen illustriert. Für die Beispiele ist zu beachten:

1. Die in die Beispiele eingefügte Zeichenfolge "`{JSON}`" weist darauf hin, dass im tatsächlichen XML-Dokument an dieser Stelle Daten im JSON-Format zu erwarten sind.

2. Die in die Beispiele eingefügte Zeichenfolge "{x|y|z}" weist darauf hin, dass im tatsächlichen XML-Dokument an dieser Stelle entweder der Wert x oder y oder z zu erwarten ist, wobei x, y, z hier Platzhalter für die eigentlichen Werte sind.

3. Die in die Beispiele eingefügte Zeichenfolge "..." weist darauf hin, dass im tatsächlichen XML-Dokument weitere Zeichen zu erwarten sind. Das Datenformat der zu erwarteten Zeichenkette ist in der jeweiligen Beschreibung der Elemente/Attribute unten angegeben.

3.1. Datenformat der Wahldaten des Wählerverzeichnisses

Beispiel einer XML-Datei mit Wahldaten aus dem Wählerverzeichnis

```
<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>
```

Im Folgenden wird die Struktur des XML tabellarisch beschrieben.

- 1. Spalte: Überschrift enthält den Namen des Elementes, welches in der Tabelle beschrieben wird. Die Werte listen die Elemente und Attribute auf.
- 2. Spalte: Typ. Hier wird im Falle eines Attributs der Typ der Daten angegeben. Im Falle eines Elementes bleibt diese Splate leer.
- 3. Spalte: M. Gibt die Multiplizität des Elementes oder Attributes an.
- 4. Spalte: Beschreibung.

Wahlergebnis Wählerverzeichnis

registry-result	Typ	M	Beschreibung
date	String	1	Zeitstempel der Auszählung; im xsd:date Format (ISO 8601)

registry-result	Typ	M	Beschreibung
num-voters	Long	1	Gesamtanzahl der registrierten Wähler
has-voted	Long	1	Anzahl der Wähler, die Ihre Stimme abgegeben haben
voters	_	1	Container für Liste der Wählerrepräsentationen
ballot-templates	_	1	Container für Liste der Stimmzettel-Vorlagen
checksums	_	1	Container für Liste der Hash-Summen

Wählerrepräsentation

voters	Typ	M	Beschreibung
voter	_	0+	Wählerrepräsentationen

voter	Typ	M	Beschreibung
pin	String	1	PIN
key-present	Boolean	1	Vermerk, ob der angemeldete Wähler seine Stimme nicht abgegeben hat: true == Wähler angemeldet, aber keine Stimme abgegeben; benötigt für Statistik in PDF
has-voted	Integer	1	Kennzeichnung der Stimmabgabe; mögliche Werte: 0 == nicht abgegeben, 1 == nicht erfolgreich abgegeben, 2 == erfolgreich abgegeben
ballots	String	1	Liste der Stimmzettel-IDs, getrennt durch "|"-Zeichen
sign-registry	String	1	TAN-Signatur des Wählerverzeichnisses ($Sig_{SK_{registry}}(pin, saltedtan)$) als Base64 Codierung der SHA256_RSA Signatur, d.h. pin und tan als String konkateniert und die Bytes der Ergebnis-Zeichenkette werden signiert.
sign-validator	String	1	Signatur des Validators der TAN-Signatur des Wählerverzeichnisses: $Sig_{SK_{validator}}(Sig_{SK_{registry}}(pin, saltedtan))$ als Base64 Codierung der SHA256_RSA Signatur
tan	String	0/1	Salted TAN zu der PIN
data	JSON	0/1	Informationen über Wähler im JSON-Format, wie Name, Vorname, Straße, etc.

Stimmzettel-Vorlagen eines Wahlkreises

ballot-templates	Typ	M	Beschreibung
ballot-template	_	1+	Stimmzettel-Vorlage

ballot-template	Typ	M	Beschreibung
id	Integer	1	ID der Stimmzettel-Vorlage
name	String	1	Überschrift dieses Stimmzettels
data	JSON	1	Wahlregeln und andere Informationen über Stimmzettel im JSON-Format, Inhalt ist für die Verifizierung der Daten nicht relevant.

Blockprüfsummen der Urne

checksums	Typ	M	Beschreibung
checksum	_	0+	Element für Prüfsumme der Urne aus der Datenbank des Wählerverzeichnisses

checksum	Typ	M	Beschreibung
—	CDATA	1	Prüfsumme der Urne aus der Datenbank des Wählerverzeichnisses, übermittelt bei jedem vollständigen 30-er Block aus der Urne, muss mit den Prüfsummen aus der Urnendatei übereinstimmen.

3.2. Datenformat der Wahldaten der Urne

Beispiel einer XML-Datei mit Wahldaten aus der Urne

```
<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>
```

Wahlergebnis der Urne (vote-result)

Das Top-Element der XML-Datei der Wahldaten der Wahlurne lautet **vote-result**.

vote-result	Typ	M	Beschreibung
date	String	1	Zeitstempel der Auszählung; im xsd:date Format (ISO 8601)
ballot-templates	_	1	Liste der Stimmzettel-Vorlagen
block	_	0+	Stimmblöcke
checksums	_	1	Prüfsummen

Stimmzettel-Vorlagen (ballot-templates)

ballot-templates	Typ	M	Beschreibung
ballot-template	1	0+	Definition einer Stimmzettel-Vorlage

ballot-template	Typ	M	Beschreibung
id	Integer	1	ID
method	String	1	nicht in Gebrauch
data	JSON	1	Wahlregeln und andere Informationen über Stimmzettel im JSON-Format. Siehe Beschreibung des data-Attributs weiter unten. Siehe ballot-template
sheet-template	_	1+	

Das data-Attribut (JSON-String) des Elementes *ballot-template*

Schlüsselwort	Typ	M	Beschreibung
name	String	1	Name dieses Stimmzettels

Schlüsselwort	Typ	M	Beschreibung
sheets	String-Liste	0+	Liste von Sheet IDs, welche als Sheets für diesen Stimmzettel (Ballot) gelten

Beispiel:

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

Der JSON-String im Attribut data kann darüber hinaus weitere Einträge enthalten, welche aber sämtlich nicht für die Auszählung relevant sind.

sheet-template	Typ	M	Beschreibung
id	Integer	1	ID
method	String	1	Bezeichnung der Wahlmethode / des Wahlmodus (einfache Wahl, ja/nein-Abstimmung, Listenwahl etc.), vgl. Abschnitt Wahlregeln
data	JSON	1	Wahlregeln und andere Informationen über Stimmzettel im JSON-Format. Siehe Beschreibung des data-Attributs weiter unten. sheet-template
candidate-template	_	1+	

Das data-Attribut (JSON-String) des Elementes *sheet-template*

Schlüsselwort	Typ	M	Beschreibung
max-votes	Integer	1	Anzahl der maximal zu vergebenden Stimmen; muss eine positive ganze Zahl sein.
columns	String	n	Spaltenbeschriftung der Kandidatenspalten

Beispiel:

```
{"max-votes":8,
 "columns":["Vorname","Name"],
 ...
}
```

Der JSON-String im Attribut data kann darüber hinaus weitere Einträge enthalten, welche aber sämtlich nicht für die Auszählung relevant sind.

candidate-template	Typ	M	Beschreibung
id	Integer	1	ID
data	JSON	1	JSON Repräsentation des Kanditaten, Schlüssel `column1`, `column2`, ... in der Reihenfolge der Spaltenbeschriftung aus dem übergeordneten `columns`-Element des sheet-templates, Wert ist der jeweilige Inhalt.

Beispiel:

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

30-er Stimmenblöcke (block)

block	Typ	M	Beschreibung
id	Integer	1	ID
vote	_	1-30	Stimme im Stimmenblock

Stimmen (vote)

vote	Typ	M	Beschreibung
key	String	1	Eindeutige Bezeichnung der Stimme aus der Datenbank. Zufallswert nach aktiver Löschung des Wahltokens.
commited	Integer	1	Kennzeichnung der Stimmabgabe; mögliche Werte: [0,1,2,3] wobei nur 3 == erfolgreich und vollständig abgegeben
signed-object	String	1	Repräsentation der verschlüsselten Stimme
ballot	_	1+	Votum der Wählers

Ausgefüllte Stimmzettel (ballot)

ballot	Typ	M	Beschreibung
reference	Integer	1	Verweis auf ID der Stimmzettel-Vorlage
valid	Boolean	1	Vermerk, ob der Stimmzettel gemäß den Wahlregeln gültig ist
abstain	Boolean	1	Vermerk, ob Enthaltung vorliegt
intentionally-invalid	Boolean	1	Vermerk, ob der Wähler den Stimmzettel explizit (per Button) ungültig gemacht hat
sheet	_	1+	Ausgefüllte Kandidatenliste

Ausgefüllte Kandidatenlisten (sheet)

sheet	Typ	M	Beschreibung
reference	Integer	1	Referenz auf ID der Kandidatenlisten-Vorlage
valid	Boolean	1	Vermerk, ob Stimmen in Kandidatenliste gültig abgegeben sind
candidate	_	1+	Votum für einen Kanditaten auf der Kandidatenliste

Kandidaten mit konkreten Stimmenzahlen (candidate)

candidate	Typ	M	Beschreibung
reference	Integer	1	Referenz auf ID der Kandidatenlisten-Vorlage
count-yes	Integer	1	Anzahl der abgegeben Stimmen
count-no	Integer	0-1	Wird nicht genutzt, immer 0

Blockprüfsummen der Urne

checksums	Typ	M	Beschreibung
checksum	_	1+	Liste der Prüfsummen der Urne aus der Datenbank der Urne

checksum	Typ	M	Beschreibung
—	CDATA	1	Prüfsumme eines 30-er Blocks der Urne aus der Datenbank der Urne

4. Verarbeitungsschritte

4.1. Einlesen der Wahldaten

Als Erstes müssen die in den XML-Repräsentationen der Wahldaten enthaltenen Daten gemäß den im vorherigen Abschnitt (siehe Abschnitt 3) beschriebenen Datenformaten aus den entsprechenden Dateien eingelesen (Hinweis: Das Encoding sollte UTF-8 sein) und in geeignete Repräsentationen der eingesetzten Programmiersprache überführt werden. Im Folgenden wird davon ausgegangen, dass für die Implementierung eines Verifikationstools die Sprache Java verwendet werden soll. Somit könnten die XML-Elemente beispielsweise durch Objekte der Klassen Candidate (Kandidat), HashcodeSequence (Block-Prüfsummen-Folge), Voter (Wahlberechtigter/Wähler) und Votum (abgegebener Stimmzettel / abgegebene Stimme) abgebildet werden.

4.2. Entschlüsselung und Auszählung der Stimmen

Wichtigster Bestandteil der Verifikation ist die Auszählung der abgegebenen Stimmen und der Abgleich des Ergebnisses mit dem veröffentlichten Wahlergebnis. Hierzu müssen die im Attribut signed-object des Elements vote noch verschlüsselt vorliegenden Stimmen entschlüsselt werden und mit den bereits im Klartext im XML vorhandenen Stimmauszählungen verglichen werden. Die so abgeglichenen Daten werden schließlich ausgezählt. Sind die im Element vote enthaltenen Daten korrekt und vollständig eingelesen worden, enthält die Repräsentation eines jeden Stimmen-Eintrags, das Attribut commited sowie das Attribut signed-object. Ein Eintrag in der Urne gilt zunächst nur dann als gültig abgegebene Stimme, wenn das Attribut committed den Wert 3 hat.

Bei der in signed-object gespeicherten Zeichenkette handelt es sich um eine Zeichenkette, die die Stimme als Object (serialisiertes Java Object, gezipped) repräsentiert, welches mit dem öffentlichen Schlüssel mit dem Alias 'voteCount' verschlüsselt, mit dem privaten (Signatur-)Schlüssel der Urne, Alias 'vote', signiert und schließlich Base64 kodiert wurde. Hierbei ist zu beachten, dass die Signatur über ein Objekt vom Typ Object[] (Java Object-Array) der Länge zwei durchgeführt wurde, wobei der erste Eintrag den mit dem privaten Schlüssel per RSA verschlüsselten AES-Schlüssel (symmetrische Verschlüsselung des Datenstroms) enthält und der zweite Eintrag den tatsächlichen Inhalt, also das serialisierte Object mit den Stimmeninformationen enthält.

Zur Entschlüsselung der Stimme sind demnach folgende Schritte notwendig:

1. Dekodierung der Zeichenkette gemäß Base64-Kodierung,
2. Deserialisierung der Bytes in einen Instanz der Klasse `java.security.SignedObject`. Die Instanz des SignedObject enthält ein Java-Array vom Typ Object der Länge 2. Der Eintrag mit dem Index 0 enthält den verschlüsselten AES-Schlüssel (vgl. Schritt 4). Der Eintrag mit dem Index 1 enthält die AES-verschlüsselten gezippten Daten der Stimmen
3. Überprüfung der Signatur der dekodierten Zeichenkette mit dem öffentlichen Schlüssel der Urne (Alias vote im Keystore der Urne) gemäß SHA-256 mit RSA ("SHA256withRSA"),
4. Entschlüsselung des *256 Bit* AES-Schlüssels mittels RSA mit dem privaten Schlüssel, Alias 'voteCount' und schließlich
5. Entschlüsselung des eigentlichen Inhalts über AES mit dem AES-Schlüssel.
6. Unzippen und deserialisieren des enthaltenen Java-Objektes vom Typ `java.util.Map<Integer, Object[]>`

Dieser Prozess wird durch den Code in Abschrit 4.2.3 illustiert.

Als Ergebnis dieser Prozedur enthält man ein Java Objekt vom Typ `java.util.Map<Integer, Object[]>`, siehe JSON-Beispiel. Die `Map` enthält eine Abbildung von Stimmzettelnummern — die Stimmzettel, auf denen der Wähler abgestimmt hat — auf eine Stimmzettel-Struktur vom Typ `Object[]`, welches schließlich folgenden Aufbau hat:

- Index 0: ein Objekt vom Typ `java.util.Collection`, welches die Stringrepräsentation der Kandidaten enthält, wie unter Kandidaten-Repräsentation beschrieben, für die der Wähler gewählt hat. Bei mehreren Stimmen für einen Kandidaten wird dieser gemäß der Anzahl der Stimmen wiederholt.

- Index 1: ein boolean, welches anzeigt, ob die Stimmzettel-Struktur ungültig abgestimmt wurde, sowie
- Index 2: ein Objekt vom Typ `java.util.Map` Attribut-Name auf Attribut-Typ, dies kann zukünftig für zusätzliche Informationen genutzt werden.

Hinweis: Abhängig von den Wahlregeln muss evtl. eine Entscheidung getroffen werden, ob ein Ballot ungültig wird, wenn einer von mehreren Stimmzetteln innerhalb des Ballots ungültig ist.

Kandidaten-Repräsentation

Ein Kandidat wird innerhalb eines Stimmzettels als eine Zeichenkette (String) dargestellt. Diese Zeichenkette enthält 3 Bestandteile, die durch das Pipe-Symbol (vertikaler Strich) voneinander getrennt sind. Diese Teile entsprechen der Datenbank-Repräsentation eines Kandidaten.

1. Kandidaten-Id, die eindeutige Nummer dieses Kandidaten
2. Sheet-Id, die eindeutige Nummer des den Kandidaten enthaltenen Sheets, muss mit dem Schlüssel der Map übereinstimmen.
3. JSON-Struktur mit den Attributen des Kandidaten, z. B. der angezeigten Spalten-Werte `column1`, ...

Beispiel

Beispiel einer decodierten Stimme in einer Javascript Repräsentation:

```
{
  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,
      {},
    ]
}
```

Dies Votum repräsentiert eine Stimmabgabe für 2 Stimmzettel, sheet-template id 1 und 2. Das erste sheet hat den Wahlmodus `CHECKBOX\_YES\_NO`, das zweite `CHECKBOX\_SINGLE\_YES`, s.u.

4.2.1 Wahlregeln / Wahlmodi

Polyas ermöglicht es für ganze Stimmzettel (Ballots) sowie auch für einzelne Kandidatenlisten innerhalb eines Stimmzetttels (Sheets) jeweils einen Wahlmodus festzulegen. Die folgenden Wahlmodi werden unterstützt:

Schlüsselwort	Beschreibung
CHECKBOX_SINGLE_YES	Checkboxen für einzelne Kandidaten, d.h. man kann für jeden Kandidaten eine oder mehrere Stimmen vergeben.
CHECKBOX_YES_NO	Je eine YES- und NO-Checkbox für jeden einzelnen Kandidaten. Wie CHECKBOX_SINGLE_YES, nur, dass man hier auch "negative" Stimmen (also 'Ablehnung') vergeben kann.

Im Falle der Methode CHECKBOX_YES_NO werden für jeden Kandidat 2 Kandidateneinträge im `sheet-template` generiert. Eine für die Ja-Option und einen für die Nein-Option. Zu erkennen ist die Zuordnung an der Kandidaten ID, welche mindestenz 2-stellig ist. Die rechte Ziffer ist kodiert wie folgt

rechte Ziffer	Beschreibung
0	Wahlregel CHECKBOX_SINGLE_YES
1	Wahlregel CHECKBOX_YES_NO, 'JA'-Stimme
2	Wahlregel CHECKBOX_YES_NO, 'NEIN'-Stimme

Der Rest der Kandidaten ID beschreibt eindeutig den Kandidaten.

Der jeweils gültige Wahlmodus für einen Stimmzettel bzw. eine Liste an Kandidaten kann dem Attribut method der Vorlagen der Stimmzettel bzw. Kandidatenlisten (siehe folgenden Abschnitt) entnommen werden. Die weiteren Wahlregeln (z.B. die maximale Anzahl an Stimmen pro Stimmzettel) finden sich im Attribut data der Vorlagen der Stimmzettel bzw. Kandidatenlisten (ebenfalls im folgenden Abschnitt beschrieben).

4.2.2. Auszählung

Für eine Auszählung der Stimmen gelten diejenigen Stimmen-Einträge der Urne als ungültige Stimmen, die, wie soeben beschrieben,

- als ungültig markiert (Ballot) sind,
- die mehr als die maximal erlaubte Anzahl zu vergebener Stimmen enthalten (vgl. die beschriebene Liste der gewählten Kandidaten — oben: die `java.util.Collection`).

Einträge, die, wie in Abschnitt 4.2 beschrieben, nicht endgültig abgegebene Stimmzettel darstellen und solche deren Attribut commited ungleich 3 ist, und somit ebenfalls als nicht endgültig abgegeben zu betrachten sind, dürfen in der Auszählung keine Berücksichtigung finden.

Stimmzettel mit der Wahlmethode `CHECKBOX_YES_NO`, sind als ungültig anzusehen, wenn für mindestens ein Kandidat sowohl eine 'NEIN'-Stimme als auch ein 'JA'-Stimme abgegeben wurde.

Die Auszählung der Stimmen für die aufgestellten Kandidaten darf also nur solche Einträge der Urne einbeziehen, für die keine der genannten Eigenschaften zutreffen. Um einen korrekten Abgleich der in einem abgegebenen Stimmzettel enthaltenen Kandidaten mit den tatsächlich aufgestellten Kandidaten (vgl Abschnitt 4.2.1) zu gewährleisten, muss die Repräsentation der Kandidaten auf dem abgegebenen Stimmzettel bekannt sein (siehe Abschnitt 3) und ein String-Vergleich (zeichenweiser Vergleich) durchgeführt werden.

Es kann sein, dass die vom Wähler als 'ungültig' markierten Stimmzettel von denen die auf Grund der Auszählungsregeln ungültig sind, getrennt auszuweisen sind.

4.2.3. Code-Beispiele

Der folgende Code illustriert, wie die Zeichenkette im Attribute `signed-object` geprüft und die darin enthaltene Stimmzettelinformation dekodiert werden kann.

```
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();
```

```java
/**
 * 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. Berechnung und Vergleich der Block-Prüfsummen

Um die Korrektheit der während der Wahl über die Stimmen gebildeten Block-Prüfsummen zu verifizieren, müssen zum einen die Prüfsummen der beiden XML-Repräsentationen (Wählerverzeichnis und Urne, Element `checksums`) verglichen werden. Sie müssen in Reihenfolge und Wert übereinstimmen. Zum anderen sollten die Blockprüfsummen auch aus den Stimmen in der Urne (signed-object) neu errechnet werden und ebenfalls anschließend paarweise mit denen in den XML-Repräsentationen aufgeführten verglichen werden. Zur Ermittlung der Block-Prüfsummen aus den Stimmen der Urne, geht man wie folgt vor:

1. man sortiere alle (endgültig) abgegebenen Stimmzettel in aufsteigender Reihenfolge nach der id des Blocks, in dem sie sich befinden (strukturell innerhalb eines block-Elements) und innerhalb eines Blocks in aufsteigender Reihenfolge alphanumerisch nach dem key (G-...),
2. man bilde den Hashwert mittels SHA-256 für die Zeichenfolge "(C) *Micromata Polyas*" (die Anführungszeichen sind nicht Teil der Zeichenfolge) und merke sich diesen als initialen Hashwert,
3. man bilde in aufsteigender Reihenfolge der Blocknummern für jeden Stimmenblock den Hashwert wie folgt:
    1. man nehme den Hashwert des vorherigen Blocks — für den Block mit der Nummer 1 nehme man den initialen Hashwert — als Zeichenkette und füge an diese Zeichenkette nacheinander für jeden Stimmzettel, die in signed-object enthaltene Zeichenkette (ohne Leerzeichen oder dergleichen) an,
    2. man bilde den Hashwert über diese Zeichenkette mittels SHA-256,
    3. man merke sich den Hashwert als Prüfsumme für den aktuellen Block.

5. Verifikationsschritte

Zur Prüfung der Integrität der gegebenen Daten und erneuten Auszählung sind die folgenden Schritte notwendig.

Datenlieferung

1. Prüfung der Datenlieferung auf Vollständigkeit, siehe Abschnitt 2.2.
2. Prüfung der Hashsummen der enthaltenen Dateien, siehe Abschnitt 2.2.
3. Prüfung der Verfügbarkeit der benötigten Schlüssel im Keystore, siehe Abschnitt 2.2 and 2.4.

Wählerverzeichnis

1. Prüfung aller Signaturen `sign-registry` mit den öffentlichen Schlüssel 'registry', siehe Abschnitt 3.1.
2. Prüfung aller Signaturen `sign-validator` mit den öffentlichen Schlüssel 'validator', siehe Abschnitt 3.1.
3. Prüfung der Übereinstimmung der `checksum` Elemente auf Reihenfolge und Gleichheit der Werte mit den Einträgen im Urnen XML.

Urne

1. Prüfung der hash chain im Element `checksums` durch Neuberechung mit dem im XML gegebenen Daten. Es ist zu beachten, dass der letzte, eventuell angebrochene Block über keine eigene checksumme verfügt, siehe Abschnitt 4.3.
2. Prüfung der Signatur der verschlüsselten Stimme `signed-object`, siehe Abschnitt 4.2 und 4.2.3.
3. Entschlüsselung der im `signed-object` enthaltenen Stimme, siehe Abschnitt 4.2 und 4.2.3.
4. Prüfung der Übereinstimmung der so entschlüsselten Stimme mit den zugehörigen Angaben im XML. Insbesondere sollte hier auch die Zuordnung der Kandidaten, sheets und ballots auf ihre Referenznummern im Element `ballot-template`, `sheet-template` and `candidate-template` geprüft werden, siehe Abschnitt 3.1, 3.2 und 4.2.
5. Durchführung einer Auszählung der Stimmen.

6. Glossar

Prüfsumme

Als Prüfsumme wird eine Zeichenkette bezeichnet, die für eine Datei oder einen Datensatz erstellt wird und diesen eindeutig kennzeichnet. Unterschiedliche Dateien bzw. Datensätze erhalten

somit stets unterschiedliche Prüfsummen. POLYAS verwendet derzeit Prüfsummen der SHA-2-Familie (Secure Hash Algorithm), genauer: SHA-256. Die erzeugten Prüfsummen bestehen in ihrer Hexadezimal-Darstellung aus genau 64 Zeichen. Ein Beispiel für eine mit SHA-256 generierte Prüfsumme: 0739fdaee043869d6c481922bd780cd2cbb563c3cc6a1b15bd347820dd37dfd0

Block-Prüfsumme

Die Block-Prüfsumme ist eine Prüfsumme (siehe *Prüfsumme*), die über einen Block von abgegebenen Stimmzetteln und der Block-Prüfsumme des vorhergehenden Blocks gebildet wird. Stimmzettel innerhalb eines Blocks werden hierzu eindeutig sortiert. Diese Sortierung erfolgt jedoch nach Ihnen bei der Stimmabgabe zugewiesenen Zufallswerten. Die Zufallswerte stellen sicher, dass die Reihenfolge, in denen die Stimmzettel bei der Wahl abgegeben wurden und eine Zuordnung zu den Wählern, die sie abgegeben haben, unmöglich ist.

Keystore

Ein Keystore ist eine Datei, welche private und/oder öffentliche digitale Schlüssel (vgl. *Schlüssel, privater*) für eine (asymmetrische) Verschlüsselung von Daten enthält und mit einem Passwort gesichert ist.

Schlüssel, digitaler

Ein *privater* digitaler Schlüssel dient dazu sensitive Daten so zu verschlüsseln, dass nur derjenige, der die Daten mit diesem Schlüssel verschlüsselt hat, sie auch wieder entschlüsseln kann. Ein privater Schlüssel kann zudem zur Erzeugung einer eindeutigen digitalen Signatur verwendet werden. Anhand einer solchen Signatur können Empfänger derart signierter Daten die korrekte Identität des Absenders/Urhebers verifizieren. Um den Mißbrauch eines solchen privaten Schlüssels durch Dritte zu verhindern, wird dieser mit einem nur dem Besitzer des Schlüssels bekannten Passwort versehen. Ein *öffentlicher* digitaler Schlüssel wird aus einem privaten Schlüssel generiert und kann an beliebige Dritte ausgegeben werden, welche damit dann Daten so verschlüsseln können, dass nur der Besitzer des entsprechenden privaten Schlüssels diese wieder entschlüsseln kann.