Hochschule für Technik Rapperswil

Vote Generator

Wahlgenerator für UniVote

Gian Poltéra

8. September 2014

Advisor: Prof. Dr. Andreas Steffen

Hochschule für Technik Rapperswil (HSR), Schweiz

Abstract

Das Thema E-Voting wird aktuell rege diskutiert und insbesondere die Sicherheit bemängelt. Mit dem UniVote Projekt wird derzeit ein alternatives E-Voting-System entwickelt. Diese Arbeit hat zum Ziel einen VoteGenerator für UniVote zu entwickeln, der Wahldaten generieren kann, in die auch gezielt Fehler eingebaut werden können. Die generierte Wahl kann anschliessend mit dem VoteVerifier überprüft werden und es soll ermittelt werden können, ob er sämtliche Fehler erkennen kann. Der Fokus der gesamten Arbeit lag dabei auf den kryptografischen Feinheiten, so wurde ein eigener Primzahlentest entwickelt, der um 100% schneller als der Java-BigInteger-Test arbeitet. Die vorliegende Arbeit entspricht dem zweiten Teil des Projekts und ist die finale Version.

Inhaltsverzeichnis

1 Einführung

1.1 E-Voting

Als E-Voting wird die elektronische Abwicklung einer Wahl verstanden. Aktuell werden insbesondere die E-Voting Projekte auf Bundesebene, d.h. für eidgenössische Wahlen, rege diskutiert. Zurzeit laufen verschiedene Systeme in den Pilotkantonen Genf, Neuenburg und Zürich. Für Aufsehen sorgte in der Vergangenheit insbesondere das Genfer E-Voting-System. *Andrivet* [**1**] demonstrierte in seiner Präsentation wie sich Stimmen durch Malware ändern lassen ohne dass es vom Abstimmenden bemerkt werden kann. Dieses Problematik wurde bereits im Jahre 2002 in einer Publikation von *Oppliger* [**2**] festgestellt. Neben der fehlenden Verifizierbarkeit ist der nicht öffentliche Wahlprozess einer der Hauptkritikpunkte der aktuellen Systeme. So schrieb *Kerckhoff* [**3**] bereits 1883, dass die Sicherheit eines Verschlüsselungsverfahrens auf der Geheimhaltung des Schlüssels beruht und nicht auf der Geheimhaltung des Verschlüsselungsverfahrens. Diese Aussage lässt in abgeänderter Form auch auf den Wahlprozess einer E-Voting-Wahl anwenden. Diese Unsicherheiten führten *Zanetti* und *Guyer* [**4**] dazu, eine parlamentarische Initiative zu lancieren, die die Abschaffung des E-Votings im Kanton Zürich bezweckt. Der *Chaos Computer Club Zürich* (CCC) [**5**] unterstützt diese Initiative und sieht eine Gefahr für die Demokratie in der Schweiz. Der *Bundesrat* [**6**] hat im Dezember 2013 entschieden, dass erst nach Umsetzung der erhöhten Sicherheitsanforderungen die Anzahl Stimmberechtigten erhöht werden soll. Wie die Zukunft für E-Voting-Systeme aussieht ist daher noch ungewiss.

1.2 UniVote

UniVote ist ein Projekt der Berner Fachhochschule, das zum Ziel hat, ein E-Voting-System für Hochschulen zu entwickeln, welches auf dem neuesten Stand der kryptografischen Möglichkeiten basiert. Das System soll vor allem eine Offenlegung des Wahlprozesses und ein öffentliches ElectionBoard bieten.

Damit soll das Vertrauen für E-Voting-Systeme in der Schweiz gestärkt werden. Die komplette Spezifikation wird in der *UniVote System Specification* [**7**] beschrieben. UniVote wurde bereits für einige Wahlen bei grösseren Universitäten und Fachhochschulen verwendet. Die Grössenordnung der Wähler entspricht dabei einer Wahl in einem kleineren Schweizer Kanton. Das Projekt wird laufend weiterentwickelt und es ist eine zweite Version auf Ende 2014 geplant.

1.3 VoteVerifier

Der VoteVerifier wurde als Bachelor-Arbeit von zwei Studenten der Berner Fachhochschule erstellt [**8**]. Es ist ein unabhängiges Projekt, das die Resultate einer UniVote Wahl überprüfen kann und dabei Verfälschungen im gesamten Wahlprozess aufzeigen soll. Durch das öffentliche ElectionBoard von UniVote ist es jedermann möglich die Resultate und den Wahlprozess einer Wahl zu überprüfen. Abbildung 1 zeigt die VoteVerifier Applikation bei der Überprüfung einer Wahl. Korrekte Werte werden dabei mit einem grünen und fehlerhafte mit einem roten Symbol markiert und durch eine kurze Fehlerbeschreibung ergänzt. Da noch nicht alle Schritte in UniVote und dem VoteVerifier implementiert sind, werden einige Schritte mit einem orangen Symbol und der Bemerkung *nicht implementiert* angezeigt.

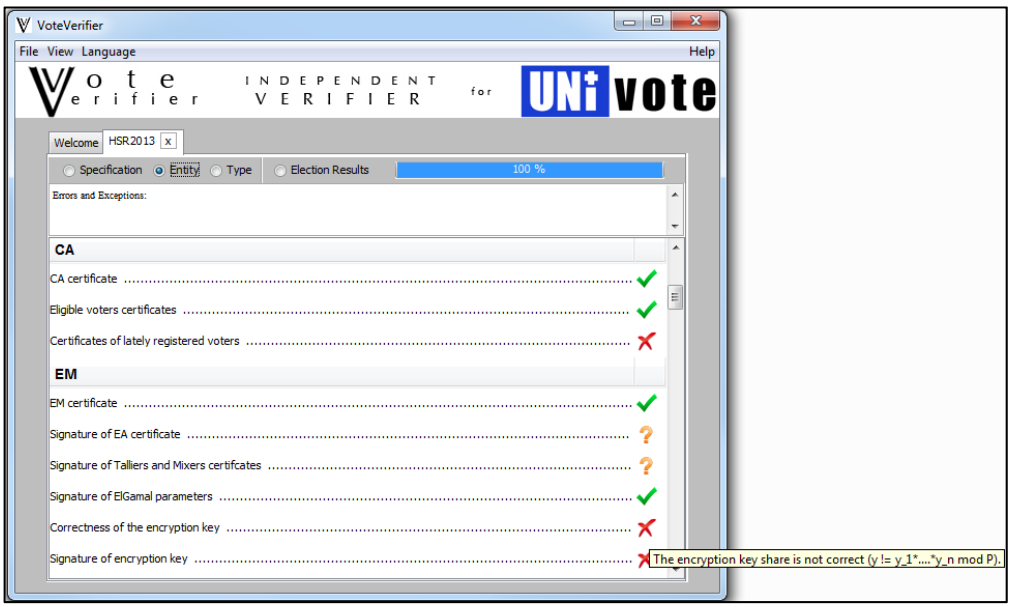


Abbildung 1: VoteVerifier Demo

1.4 Ziel der Arbeit

Ziel dieser Arbeit war es, einen VoteGenerator zu erstellen, der verschiedene Wahlen generieren kann und die Möglichkeit bietet, die Wahl zu verfälschen. Die Schnittstelle und die übergebenen Parameter sollten dabei die gleichen wie beim UniVote Projekt sein. Der Hauptfokus der gesamten Arbeit lag dabei auf den kryptografischen Spezialitäten und Funktionen.

1.5 Bemerkungen

Das komplette Projekt besteht aus zwei Teilen. Das vorliegende Dokument beschreibt den finalen Stand des zweiten Teils des Projekts.

1.6 Aufbau

In Kapitel 2 werden die kryptografischen Grundlagen, die für das weitere Verständnis der Arbeit benötigt werden, beschrieben. Kapitel 3 beschreibt das UniVote-System, welche Parteien beteiligt sind, was ihre Aufgabe sind und den groben Ablauf einer Wahl. Die Umsetzung des VoteGenerator‘s wird in den

Kapiteln 0 bis 8 erläutert. Im Kapitel 9 werden die erzielten Resultate aufgezeigt und in Kapitel 10 werden sie diskutiert und interpretiert. In Kapitel 11 wird das Fazit gezogen und der weitere Ausblick für das Projekt gegeben. Um den Lesefluss nicht unnötig zu stören wird in dieser Arbeit nur das Wesentlichste beschrieben.

2 Kryptographische Grundlagen

Nachfolgend werden die verschiedenen kryptografischen Verfahren, die von UniVote eingesetzt werden, beschrieben. Sie werden auch für die Entwicklung des VoteGenerators benötigt.

2.1 ElGamal Kryptosystem

Das ElGamal Kryptosystem basiert auf dem schwer zu brechenden, diskreten Logarithmus Problem. Es wurde von *Gamal* [**9**] bereits im Jahre 1985 entwickelt. ElGamal wird in UniVote zur Verschlüsselung der Stimmen verwendet und wurde ausgewählt, da es die Möglichkeit bietet, den Schlüssel zu teilen und zudem können für die Entschlüsselung, diese Teilschlüssel einfach miteinander multipliziert werden.

Öffentliche Parameter

Als Gruppenordung q wird eine Primzahl gewählt. Die Primzahl p kann anschliessend durch $p = q \cdot 2 + 1$ berechnet werden und muss ebenfalls wieder eine Primzahl sein. Der Wert für q sollte dabei so gewählt werden, dass p eine genügend grosse Länge hat, um möglichen Angriffen stand zu halten. Der Generator g muss folgende Bedingungen erfüllen $g^2 \neq 1 \bmod p$ und $g^q \neq 1 \bmod p$. Die Werte (p, q, g) bilden die öffentlichen Parameter.

Privater und öffentlicher Schlüssel

Der private Schlüssel x wird zufällig gewählt, dabei gilt $1 < x < q$. Der öffentliche Schlüssel y kann nun mittels $y = g^x \bmod p$ berechnet werden.

Verschlüsselung

Sei m die zu verschlüssende Nachricht und r eine zufällig gewählte, durch den Verschlüssler geheim gehaltene Zahl, so dass gilt $1 < r < q$. Berechnet man nun:

$$a = g^r \mod p \tag{1}$$
$$b = m \cdot y^r \mod p \tag{2}$$

Die Werte (a, b) bilden zusammen die verschlüsselte Nachricht. Diese ist ohne Kenntnis des privaten Schlüssels x oder Zufallszahl r nur sehr schwer zu entschlüsseln.

Entschlüsselung

Die ursprüngliche Nachricht m kann mittels des privaten Schlüssels und der öffentlichen Parameter entschlüsselt werden:

$$m = a^{p-1-x} \cdot b \mod p \tag{3}$$

2.2 Schnorr Signatur

Die Schnorr Signatur ist ähnlich aufgebaut wie das ElGamal Kryptosystem und die Sicherheit beruht ebenfalls auf der Komplexität des diskreten Logarithmus. Entwickelt wurde es von *Schnorr* [**10**] im Jahre 1991. Die Schnorr Signatur wird in UniVote zum signieren der Stimmen verwendet und wurde ausgewählt, da es mehr Freiheiten als z.B. das RSA-System bietet, wo der Ablauf fest vorgeben ist.

Öffentliche Parameter

Die öffentlichen Parameter können gleich gebildet werden wie beim ElGamal Kryptosystem und bestehen aus der Primzahl p, der Gruppenordnung q sowie dem Generator g.

Signierungs- und Verifizierungsschlüssel

Der Signierungsschlüssel sk wird zufällig gewählt, dabei gilt $0 < sk < q$. Der Verifizierungsschlüssel vk kann nun mittels $vk = g^{sk} \mod p$ berechnet werden.

Signieren

Sei m die zu signierende Nachricht und r eine zufällig gewählte Zahl, so dass gilt $0 < r < q$. Berechnet man nun:

$$Sign_{sk}(m, r) = (a, r - a \cdot sk) \in \mathbb{Z}_q \times \mathbb{Z}_q, wobei\ a = H(m \parallel g^r) \tag{4}$$

Zum einfacheren Verständnis wird das Signatur-Verfahren in vier Schritte aufgeteilt:

1. $gr = g^r \mod p$
2. $mgr = m \parallel gr$ [1]
3. $a = SHA256(mgr)$
4. $b = r - a \cdot sk \mod q$

Die Signatur S setzt sich aus dem Wertepaar (a, b) zusammen.

Verifizieren

$$Verify_{vk}(m, S) = \begin{cases} akzeptiere, \text{wenn } a = H(m \parallel g^b \cdot vk^a), \\ verwerfe, \quad \text{sonst.} \end{cases} \tag{5}$$

Zum einfacheren Verständnis dieser Formel wird das Verifizierungsverfahren in vier Schritten aufgeteilt:

1. $rv = g^b \cdot vk^a \mod p$
2. $mrv = m \parallel rv$
3. $av = SHA256(mrv)$
4. Falls av identisch zu a ist gib $true$, andernfalls $false$ zurück.

[1] mit $\parallel$ ist die Konkatenation der beiden Mengen auf Byte-Ebene gemeint.

2.3 RSA Signatur

Das RSA-Kryptosystem wurde von *Rivest*, *Shamir* und *Adleman* [**11**] entwickelt und ist ein asymmetrisches kryptografisches Verfahren. Es kann sowohl für die Verschlüsselung als auch für das Signieren von Daten verwendet werden. UniVote verwendet ausschliesslich die Signatur Funktion von RSA für das signieren der einzelnen Wahl-Schritte. Auf die Beschreibung des Verfahrens zur Schlüsselgenerierung wird an dieser Stelle verzichtet, da für die Umsetzung auf die vorhandene Java Security Library zurückgegriffen wurde.

Signieren

Sei m die zu signierende Nachricht und sk der private Signatur Schlüssel. Berechnet man nun:

$$Sign_{sk}(m) = (b^{sk}) \in \mathbb{Z}_p \times \mathbb{Z}_p, wobei\ b = H(m) \tag{6}$$

Zum einfacheren Verständnis wird das Signatur-Verfahren in zwei Schritte aufgeteilt:

1. $b = SHA256(m)$
2. $a = b^{sk} \bmod p$

Die Signatur S besteht aus dem Wert a.

Verifizieren

$$Verify_{vk}(m, S) = \begin{cases} akzeptiere, \text{wenn } S^{vk} = H(m), \\ verwerfe, \qquad \text{sonst.} \end{cases} \tag{7}$$

Zum einfacheren Verständnis dieser Formel wird das Verifizierungsverfahren in drei Schritten aufgeteilt:

1. $a = SHA256(m)$
2. $av = S^{vk} \bmod p$
3. Falls av identisch zu a ist gib $true$, andernfalls $false$ zurück.

2.4 Diskretes Logarithmus-Problem

Das diskrete Logarithmus-Problem auf dem das ElGamal- und Schnorr-Verfahren beruhen, lässt sich vereinfacht folgendermassen darstellen:

Es ist sehr schwer den diskreten Logarithmus zu finden jedoch sehr einfach die diskrete Exponentialfunktion, bei gegebenem Schlüssel, zu berechnen. Eine Methode den diskreten Logarithmus einer Zahl n bezüglich einer Basis b zu berechnen ist, die Variable z auf 1 zu setzen und solange $z = z \cdot b$ durchzuführen bis $z = n$ ist. Die Anzahl Multiplikationen die man durchgeführt hat, entsprechen dabei dem diskreten Logarithmus. Bei dieser Methode werden alle Exponenten durchgetestet. Der Aufwand ist entsprechend hoch. Natürlich gibt es noch bessere Verfahren, doch im Allgemeinen kann gesagt werden, dass der Aufwand für die Exponentiation linear $\big(\mathrm{O(n)}\big)$und für die Logarithmierung exponential $\big(\mathrm{O}(2^{\mathrm{n}})\big)$ ist. [**12**]

2.5 Sichere Zufallszahlen

Die grösste Schwachstelle bei aktuellen Kryptosystemen die auf dem diskreten Logarithmus Problem basieren, ist das finden von sicheren Zufallszahlen. Für deren Generierung gibt es verschiedene Ansätze wie die Generierung per Hardwarekomponenten, Umgebungsgeräuschen, der Bewegung der Maus und viele weitere. Einige Beispiele sind:

- Stanford Javascript Crypto Library
- Hardware Zufallsgenerator der Ivy-Bridge Intel-Prozessoren
- Java SecureRandom
- BigInteger Library
- Leemon Baird Random Generator

Laut verschiedenen unbestätigten Meinungen, soll es der NSA möglich sein, die Generatoren zu manipulieren. Daher ist eine 100% sichere Generierung nicht gegeben. Der VoteGenerator verwendet die Java SecureRandom Funktion, welche „zuverlässige“ Pseudo-Zufallszahlen liefert.

2.6 Primzahlengenerierung

Die Schwierigkeit bei der Generierung von Primzahlen ist die Tatsache, dass nicht mit vollständiger Sicherheit gesagt werden kann, ob eine Zahl eine Primzahl ist. Es kann jedoch nach einem Widerspruch gesucht werden und so mit Sicherheit gesagt werden, dass diese Zahl keine Primzahl ist. Für den VoteGenerator wird der von *Miller* und *Rabin* [**13**] [**14**] entwickelte Miller-Rabin-Test angewendet um zu überprüfen, ob eine Zahl eine Primzahl ist. Dieser funktioniert vereinfacht für die Zahl n folgendermassen:

1. Zufällige Wahl von a aus den Zahlen 2,3, ... , $n-1$
2. $n-1 = d \cdot 2^j$ mit ungeradem d
   Falls n eine Primzahl ist, gilt:
   $a^d \equiv 1 \mod n$
   oder
   $a^{d \cdot 2^r} \equiv -1 \mod n$

Die Sicherheit mit der gesagt werden kann, ob die Zahl prim ist, steigt mit der Anzahl Durchgänge r mit verschiedenen Werten für a und wird mit $1 - \frac{1}{4^r}$ ermittelt.

2.7 Digitale Zertifikate

Als digitale Zertifikate kommen in UniVote X.509 Zertifikate zum Einsatz. Der VoteGenerator erstellt diese mittels der BouncyCastle Library. Die Beschreibung beschränkt sich deshalb auf die Zusammensetzung eines Zertifikates in UniVote.

Das Zertifikat wie in Formel (8) abgebildet stellt sich aus der *id* des Halters, seinem öffentlichen Schlüssel k, einem Zeitstempel t und der Signatur des Herausgebers, der Certificate Authority CA, zusammen.

$$Z_{id} = Certify_{sk_{CA}}(id, k, t) = (id, k, t, CA, S_{id}) \tag{8}$$

2.8 Zero-Knowledge Proofs of Knowledge

Ein Zero-Knowledge Proof of Knowledge (ZKP) oder auf Deutsch kenntnisfreier Beweis ist ein kryptografisches Protokoll. Partei A möchte dabei Partei B davon überzeugen ein gewisses Geheimnis zu kennen ohne dabei Informationen über das Geheimnis selbst preis zu geben. Beim Non-Interactive Zero Knowledge Proof (NIZKP) findet dabei keine Interaktion zwischen Partei A und B statt. In UniVote wird der NIZKP zum Beweisen der Kenntnis über einen Teil des diskreten Logarithmus verwendet.[2]

2.9 Threshold Cryptosystem

Das ElGamal Kryptosystem bietet die Möglichkeit den privaten Entschlüsselungsschlüssel x auf n verschiedene Parteien zu verteilen. Ist eine Entschlüsselung durch $t \leq n$ Parteien möglich, so spricht man von einem threshold Kryptosystem. Dies lässt sich beispielsweis mit dem Shamir Secret Sharing Schema [**15**] [**16**] realisieren.

In der aktuellen Form von UniVote werden zur Entschlüsselung jedoch sämtliche Parteien benötigt $t = n$. Dies wird als verteiltes Kryptosystem [**17**] bezeichnet. Dabei kann der Verschlüsselungsschlüssel y durch die Formel (9) berechnet werden:

$$y = \prod_i y_i = g^{\Sigma_i x_i} = g^x \tag{9}$$

Der Entschlüsselungsschlüssel x entspricht dabei der Summe aller privaten Schlüssel aller Parteien x_i:

$$x = \sum_i x_i \tag{10}$$

Da die einzelnen Schlüsselhalter ihren privaten Schlüssel nicht preisgeben sollten, kann eine mit dem öffentlichen Schlüssel y verschlüsselte Nachricht $E = (a, b) = Enc_y(m, r)$, durch jede Partei n teilentschlüsselt werden. Dies lässt sich mittels zwei verschiedenen Varianten durchführen:

[2] Weitere Infos über den NIZKP in UniVote sind in der Spezifikation [**7**] zu finden.

Variante 1

1. $a_i = a^{-x_i}$.
2. $a^{-x} = \prod_i a_i$
3. $m = Dec_x(E) = a^{-x} \cdot b$

Variante 2

1. $m_i = Dec_{x_i}(E) = a^{-x_i} \cdot b$
2. $m = b^{1-n} \cdot \prod_i m_i$

Die Entschlüsselung lässt sich auch sequentiell durchführen, was als Partielle-Entschlüsselungsfunktion bezeichnet wird.

$$Dec'_{x_i}(E) = (a, a^{-x_i} \cdot b) \tag{11}$$

Dabei entfernt jeder Schlüsselhalter nach und nach den öffentlichen Schlüssel y_i aus E. Dies wird durch Transformation in eine neue Verschlüsselung $E' = Dec'_{x_i}(E)$ mit dem neuen öffentlichen Schlüssel $y \cdot y_i^{-1}$ erreicht. Sobald alle Schlüssel in willkürlicher Form entfernt sind, erhält man (a, m) und kann daraus m extrahieren.

2.10 Verifiable Mix-Nets

Die verifizierbaren Mix-Netze sind in der aktuellen Version von UniVote noch nicht implementiert. In der Spezifikation [**7**] ist dieser Abschnitt ebenfalls noch nicht definiert. Einige Eindrücke wie sich ein solches System umsetzten lässt, liefern die folgenden Arbeiten:

- *„Telefon-MIXe: Schutz der Vermittlungsdaten für zwei 64-kbit/s-Duplexkanäle über den (2• 64+ 16)-kbit/s-Teilnehmeranschluß“* von *Pfitzmann et al.* 1989 [**18**]
- *„Receipt-free mix-type voting scheme“* von *Sako et al.* 1995 [**19**],
- *„Some remarks on a receipt-free and universally verifiable mix-type voting scheme“* von *Michels* und *Horster* 1996 [**20**],
- *„Anonyme Signalisierung in Kommunikationsnetzen“* von *Müller* 1997 [**21**]
- *„Universally verifiable mix-net with verification work independent of the number of mix-servers“* von *Masayuki* 1998 [**22**],
- *„Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking“* von *Jakobsson et al.* 2002 [**23**],
- *„End-to-End verifizierbare Wahlverfahren in Hinblick auf den Grundsatz der Öffentlichkeit der Wahl“* von *Hupf* und *Meletiadou* 2009 [**24**].

3 UniVote

3.1 Beteiligte Parteien

Die beteiligten Parteien in UniVote werden nachfolgend kurz beschrieben. Jede Partei hat dabei seine eigene Funktion:

Die *Root Certificate Authority RA* ist die höchste Stelle und kann eine externe Institution sein. Jeder Teilnehmer muss der *RA* vertrauen, da ihr Zertifikat selbstsigniert ist. In der aktuellen Version wird auf eine *RA* verzichtet und stattdessen die *Certificate Authority CA* mit einem selbstsignierten Zertifikat versehen. Alle Parteien müssen daher der *CA* vertrauen. Der *Election Manager EM* verwaltet die Wahl und überprüft die kryptografischen Aussagen der anderen Parteien. Das *Election Board EB* ist das öffentliche Board und alle Wahlschritte werden darauf publiziert. Der *Election Administration EA* definiert die Parameter für eine Wahl, er erstellt die Parteien- und Kandidatenlisten und definiert wer die Mixer und Tallier sind. Die *Tallier* T_j generieren jeweils einen Teil des Entschlüsselungsschlüssels, nur durch alle Talliers kann so eine Stimme wieder entschlüsselt werden. Die Mixer M_k sind für das mischen der verschlüsselten Stimmen zuständig. So ist keine Verbindung zwischen einem Wähler und seiner Stimme möglich. Die Voter V_i schliesslich sind die Wähler die an einer Wahl teilnehmen. In Abbildung 2 ist der Signierungsprozess für das Erstellen der verschiedenen Zertifikate für alle Parteien ersichtlich.

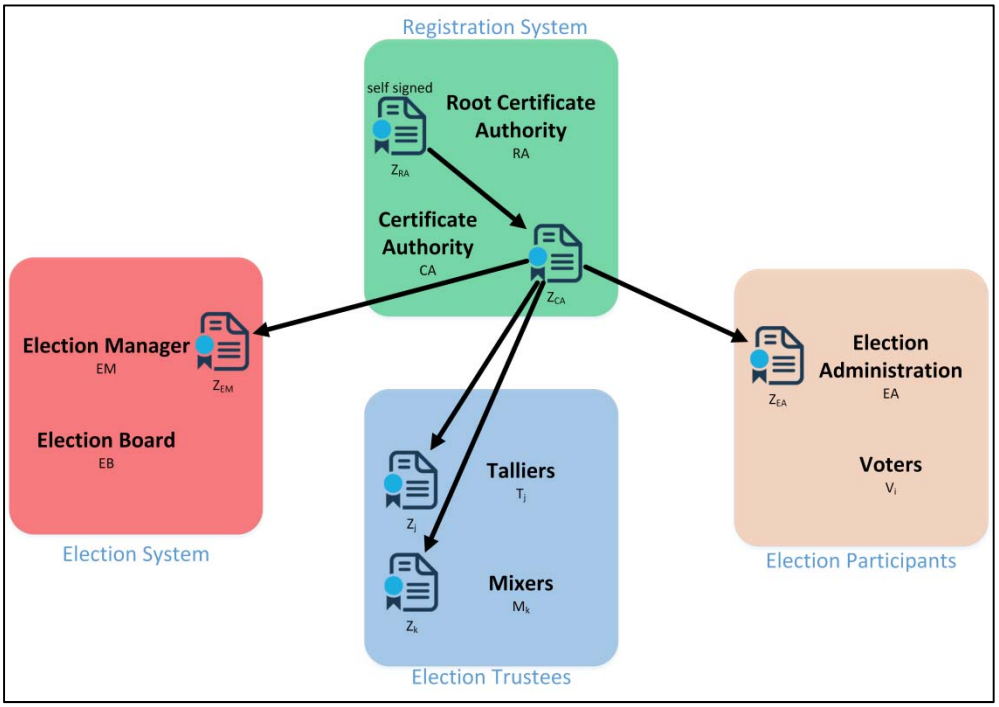


Abbildung 2: Zertifikatsverteilung

3.2 Wahlablauf

Der Wahlablauf von UniVote ist in sieben Phasen aufgeteilt. Auf eine detailliertere Beschreibung der einzelnen Phasen wird an dieser Stelle verzichtet und auf das *UniVote System Specification* [7] Dokument verwiesen.

Phase	Beschreibung
1. Public Parameters	Die erste Phase beinhaltet die Definition der öffentlichen Parameter. Diese sind fest vorgegeben. Dazu gehören die Schnorr Signatur-Parameter p, q, g, sowie die Hash Funktion $H(x) = SHA256(x) \mod q$.

2. Public Identifiers and Keys a. Registration System b. Election System c. Election Trustees d. Election Participants	In der zweiten Phase werden die Zertifikate und Schlüsselpaare für die Parteien Root Certificate Authority, Certificate Authority, Election Manager und Election Trustees erstellt.
3. Registration a. First-Time Registration b. Registration Renewal c. Late Registration	Während dieser Phase können sich die Wähler für die Wahl registrieren. Es werden die Zertifikate und Schlüsselpaare für die Wähler erstellt.
4. Election Setup a. Initialization b. Election Definition c. Parameter Generation d. Distributed Key Generation e. Constructing the Election Generator	Die vierte Phase beinhaltet die Definition der wahlspezifischen Parameter wie die ElectionId, Beschreibung der Wahl, Festlegung der Sicherheitsparameter, Definition der Mixer und Tallier, Festlegung der ElGamal Parameter und Generierung des Verteilten-Schlüssels sowie des ElectionGenerators.
5. Election Preparation a. Definition of Election Options b. Publication of Election Data c. Electoral Roll Prepartion d. Mixing the Public Verification Keys	Diese Phase wird kurz vor dem Wahlbeginn durchgeführt und beinhaltet die Festlegung der Wahloptionen wie Parteilisten und Kandidaten.
6. Election Period a. Late Registration b. Late Renewal of Registration c. Vote Creation and Casting d. Closing the Electronic Urn	Während der eigentlichen Wahlphase werden die Stimmen generiert und es können sich noch zusätzliche Wähler registrieren.

7. Mixing and Tallying a. Mixing the Encryptions b. Decrypting the Votes	In der letzten Phase werden die verschlüsselten Stimmen gemixt und durch die Tallier entschlüsselt und anschliessend die Resultate publiziert.

Tabelle 1: Wahlablauf UniVote

4 VoteGenerator Grundlagen

4.1 Anforderungen

Der VoteGenerator in der ersten Version sollte folgende Funktionen bieten:

1. Erstellung einer Wahl mit den Grundfunktionen
2. Implementierung erster Fehler
3. Nach Möglichkeit Implementierung eigener Kryptofunktionen
4. Konfiguration der Wahl und der Fehler per Konfigurationsfile
5. Gleiches Web-Service-Interface wie das UniVote System

In seiner finalen Version:

1. Erstellung einer kompletten Wahl
2. Speicherung der generierten Wahlen in einer Datenbank
3. Konfiguration mittels Grafischem User Interface (GUI)
4. Implementierung von Fehlern während der Wahl

4.2 Eingesetzte Technologien

Nachfolgend werden die Technologien, die zur Entwicklung des VoteGenerators eingesetzt wurden, beschrieben.

4.2.1 Java

Java ist eine Plattformunabhängige und objektorientierte Programmiersprache. Sie wird bereits von VoteVerifier und UniVote verwendet. Die Programmiersprache wurde von der Projektleitung vorgegeben.

4.2.2 NetBeans IDE

Die Entwicklungsumgebung NetBeans bietet eine ideale Integration von Maven und GitHub und eignet sich gut für die Entwicklung in Java. Die Vorgabe für die Verwendung von NetBeans wurde von der Projektleitung vorgeben.

4.2.3 Maven

Apache Maven ist ein Software-Projekt-Management-Tool. Es ermöglicht eine standardisierte Erstellung und Verwaltung von Java-Programmen. Maven basiert dabei auf dem Konzept eines Projektobjektmodell (POM) und wurde ebenfalls von der Projektleitung vorgeben.

4.2.4 GitHub

GitHub ist eine webbasierte Versionsverwaltungssoftware auf Basis von Git. Durch GitHub war es möglich die verschiedenen Versionen des VoteGenerators zu verwalten und zwischen verschiedenen Geräten zu verteilen. Die aktuellste Version des VoteGenerators ist öffentlich unter folgender Adresse abrufbar: https://github.com/gpoltera/VoteGen.git.

4.2.5 WSDL

Die Web Service Description Language (WSDL) ist eine auf dem XML-Format basierende Beschreibungssprache für Netzwerkdienste. WSDL ist unabhängig von der Plattform, der Programmiersprachen und der Protokolle. Ziel ist es die von aussen zugänglichen Operationen zu definieren und deren Parameter und Rückgabewerte festzulegen. Um einen Endpunkt zu definieren, werden die Operationen und Nachrichten dabei abstrakt beschrieben und an ein konkretes Netzwerkprotokoll und Nachrichtenformat gebunden. [**25**]

WSDL wird für den Austausch der Daten zwischen dem UniVote- und VoteVerifier-System verwendet und wird deshalb auch vom VoteGenerator eingesetzt.

4.2.6 db4o

Die database for objects (db4o) ist eine Objektdatenbank und gehört zu den NoSQL-Datenbanken. Durch die komplexen Konstrukte in UniVote ist es sehr aufwendig, eine herkömmliche relationale Datenbank zu verwendet. Db4o

ermöglicht es hier einfach die Objekte als Ganzes in der Datenbank abzuspeichern.

4.3 Architektur

Auf eine Beschreibung sämtlicher Klassen wird an dieser Stelle verzichtet, es werden lediglich die essentiellen Klassen genauer beschrieben.

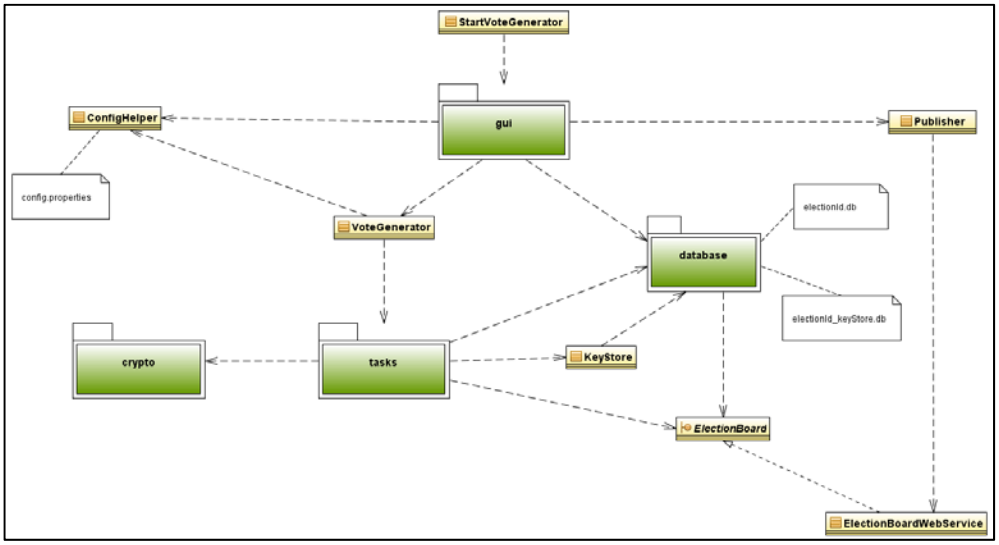


Abbildung 3: VoteGenerator Architektur

Der VoteGenerator ist einfach aufgebaut. In Abbildung 3 ist die Architektur stark vereinfacht abgebildet. Die Startklasse *StartVoteGenerator* startet das GUI. Vom GUI aus kann mittels der *VoteGenerator* Klasse eine neue Wahl generiert werden oder eine bereits generierte Wahl aus der Datenbank geladen werden. Die *VoteGenerator* Klasse ruft die verschiedenen Wahlschritte aus dem Package *tasks* auf. Für jeden Wahlschritt existiert dabei eine eigene Klasse, diese benutzen dabei die verschiedenen Kryptoklassen aus dem Package *crypto*. Ist eine Wahl generiert, kann dieses vom GUI aus mithilfe der *Publisher* und *ElectionBoardWebService* Klasse publiziert werden.

4.4 Abbildung der Wahlschritte in Klassen

Das Package *tasks* Enthält die Klassen für die jeweiligen Wahlschritte, diese sind am Wahlablauf von UniVote aus Tabelle 1 orientiert.

Phase	**Abgebildet in Klasse**
1. Public Parameters	*SignatureParametersTask*
2. Public Identifiers and Keys	*ElectionSystemInfoTask*
3. Registration	*VoterCertsTask*
4. Election Setup	*ElectionDefinitionTask*
	EncryptionParametersTask
	EncryptionKeyShareTask
	EncryptionKeyTask
	BlindedGeneratorTask
	ElectionGeneratorTask
5. Election Preparation	*ElectionOptionsTask*
	ElectionDataTask
	ElectoralRollTask
	MixedVerificationKeysByTask
	MixedVerificationKeysTask
6. Election Period	*LatelyRegistredVoterCertsTask*
	LatelyMixedVerificationKeysByTask
	LatelyMixedVerificationKeysTask
	BallotsTask
7. Mixing and Tallying	*MixedEncryptedVotesByTask*
	EncryptedVotesTask
	PartiallyDecryptedVotesTask
	DecryptedVotesTask
	DecodedVotesTask

Tabelle 2: Wahlablauf und die Abbildung in den Klassen

5 VoteGenerator Wahlablauf

Nachfolgend wird die Implementation des Wahlablaufs im VoteGenerator beschrieben.

5.1 Mögliche Abläufe

Für den Wahlablauf sind drei verschiedene Varianten vorgesehen.

Variante 1

1. Erstellung einer neuen Konfiguration
2. Generierung der Wahl
3. Publizierung des ElectionBoard und Start des Webservices

Variante 2

1. Laden einer gespeicherten Konfiguration
2. Generierung der Wahl
3. Publizierung des ElectionBoard und Start des Webservices

Variante 3

1. Laden einer bereits generierten Wahl
2. Publizierung des ElectionBoard und Start des Webservices

5.2 Konfiguration

Sämtliche Anpassungen können im GUI vorgenommen werden. Es werden jeweils die Werte der letzten Konfiguration voreingestellt. Die Konfiguration einer zu generierenden Wahl ist in sechs Schritte aufgeteilt:

1. **Grundkonfiguration**

In der Grundkonfiguration können die Basisparameter einer Wahl angepasst werden. Einstellen lässt sich die Wahl ID und deren Beschreibung, die Anzahl Wähler, die sich während und vor Beginn der Wahl registrieren, die Anzahl Mixer und Tallier, die Anzahl Kandidaten und Listen, die maximale Anzahl wählbarer Kandidaten pro Wähler sowie die maximal abzugebenden Stimmen.

2. **Administratoren Namen Konfiguration**

Im zweiten Schritt können die Namen der verschiedenen Administratoren angepasst werden. Dies sind die Zertifizierungsstelle, der Wahl Manager, der Wahl Administrator und die Mixer und Tallier.

3. **Kandidaten Konfiguration**

In der Kandidaten Konfiguration können die wählbaren Kandidaten in einer Tabelle angepasst werden. Es lässt sich der Name und Vorname, das Geschlecht und der Jahrgang sowie die Liste, zu welcher der Kandidat gehört, anpassen.

4. **Krypto Konfiguration**

Die Krypto Einstellungen sind in drei Teile aufgeteilt: die Verschlüsselungsparameter, die Signaturparameter und die Hashparamater. Da die meisten Werte von UniVote vorgegeben sind, lassen sich nur die Schlüssellänge für die Verschlüsselung und die Signatur anpassen. Zu beachten gilt, dass insbesondere die Schlüssellänge für die Verschlüsselung einen grossen Einfluss auf die beanspruchende Zeit zur Generierung einer Wahl hat. Da für die Generierung der Parameter ein SafePrime gesucht werden muss.

5. Fehler Konfiguration

Im fünften Schritt lassen sich die Fehler für die zu generierende Wahl auswählen. Bei einigen kann zusätzlich der entsprechende Mixer, der Tallier oder der Wähler ausgewählt werden, für den der Fehler implementiert werden soll.

6. Konfiguration abgeschlossen

Im letzten Schritt ist die eigentliche Konfiguration abgeschlossen. Hier können der Port und die IP-Adresse für die Veröffentlichung des ElectionBoards angepasst werden und die soeben konfigurierte Wahl generiert werden.

5.3 Wahlgenerierung

Die Wahl wird in einem separaten Thread generiert um auf Benutzereingaben im GUI weiterhin reagieren zu können. Ist ein Schritt abgeschlossen wird dies im GUI jeweils angezeigt und bei Fehlern wird der entsprechende Fehler ausgegeben. Zur Vorbereitung wird ein neues ElectionBoard erstellt und die Konfiguration geladen. Die eigentliche Wahl-Generierung ist in neun Schritte unterteilt und wurde in einigen Punkten im Vergleich zur UniVote System Specification vereinfacht bzw. ergänzt:

1. Public Parameters

Im ersten Schritt werden die Signatur Parameter, die später für die Schnorr Signatur benötigt werden, gesetzt. Diese sind durch die UniVote System Specification vorgegeben und bestehen aus den Werten Prime p, der Group Order q und dem Generator g. Als Letztes werden die SignatureParamaters an das ElectionBoard EB übermittelt.

2. Public Identifiers and Keys

In diesem Schritt werden die Zertifikate für die Wahl-Administratoren erstellt. Dazu wird zuerst jeweils ein RSA KeyPair (sk, vk) erstellt und, basierend darauf, ein Zertifikat generiert. Die entsprechenden Schlüssel werden im

KeyStore KS abgelegt, da diese im weiteren Verlauf der Wahl noch benötigt werden. Als Letztes werden die Zertifikate durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

Name	**Identifier**	**Zertifikat**	**Signiert**	**Private Key**	**Public Key**
Root Certificate Authority	RA	Z_{RA}	Selbst	sk_{RA}	vk_{RA}
Certificate Authority	CA	Z_{CA}	RA	sk_{CA}	vk_{CA}
Election Manager	EM	Z_{EM}	CA	sk_{EM}	vk_{EM}
Election Administrator	EA	Z_{EA}	CA	sk_{EA}	vk_{EA}
Talliers	T_j	Z_j	CA	sk_j	vk_j
Mixers	M_k	Z_k	CA	sk_k	vk_k

Tabelle 3: Public Identifiers and Keys Zertifikate

3. Registration

Im dritten Schritt werden die Zertifikate für die Wähler erstellt. Dazu wird zuerst jeweils ein Schnorr KeyPair (sk_i, vk_i) erstellt und, basierend darauf, ein Zertifikat Z_i generiert. Die entsprechenden Schlüssel werden im KeyStore KS abgelegt, da diese im weiteren Verlauf der Wahl noch benötigt werden. Als Letztes werden die Zertifikate Z_i durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

Name	Identifier	Zertifikat	Signiert	Private Key	Public Key
Voters	V_i	Z_i	CA	sk_i	vk_i

Tabelle 4: Registration Zertifikate

4. Election Setup

Zuerst wird die aktuelle ElectionId aus der Konfiguration gelesen und an das ElectionBoard EB übermittelt.

Als Zweites wird die ElectionDefinition erstellt. Diese besteht aus verschiedenen Werten, die vorher in der Konfiguration definiert wurden. Unter anderen sind dies: die ElectionId id, der ElectionTitle $descr$, der Beginn und das Ende der Wahlphase [3], die Mixer- und TallierIds und die Schlüssellänge ℓ. Die ElectionDefinition wird durch den ElectionAdministrator EA signiert und anschliessend an das ElectionBoard EB übermittelt.

Als Drittes werden die EncryptionParameters erstellt. Diese werden später für die ElGamal Verschlüsslung benötigt und bestehen aus den Werten Prime P, der Group Order Q und dem Generator G. Die Grösse dieser Parameter ist abhängig von der in der Konfiguration definierten Schlüsselgrösse für die Verschlüsselung. Die Generierung der Group Order Q kann bei einer Schlüsselgrösse >2048 Bit sehr lange dauern, da dafür eine SafePrime gefunden werden muss. Die EncryptionParameters werden im Anschluss durch den Election Manager EM signiert und an das ElectionBoard EB übermittelt.

Als Nächstes wird der EncryptionKey y erstellt. Dazu wird zuerst für jeden Tallier T_j ein ElGamal KeyPair (x_j, y_j) generiert und im KeyStore KS abgelegt. Der PublicKey y_j wird zum EncryptionKeyShare hinzugefügt und ein NIZKP berechnet sowie durch den Tallier T_j signiert. Haben alle Tallier T_j diesen Schritt beendet, werden diese an das ElectionBoard EB übermittelt.

[3] Der Beginn und das Ende der Wahlphase wurden in der Konfiguration nicht berücksichtigt, da sie in der generierten Wahl nicht weiter von Bedeutung sind.

Anschliessend können die PublicKeys y_j miteinander multipliziert werden. Daraus entsteht der EncryptionKey y, welcher durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt wird.

$$y = \prod_j y_j \ mod \ P \tag{12}$$

Als Letztes wird der ElectionGenerator $\hat{g}$ erstellt. Dazu wird als Erstes für jeden Mixer M_k ein Schnorr KeyPair (α_k, β_k) generiert und im KeyStore KS abgelegt. Anschliessend wird der BlindedGenerator $g_k = g_{k-1}^{\alpha_k}$ berechnet. Der erste Tallier T_1 nimmt dazu den Generator g der Schnorr Signatur Parameter und die folgenden Tallier T nehmen jeweils den BlindedGenerator g_k des Vorgängers. Jeder Tallier berechnet einen NIZKP und signiert den BlindedGenerator g_k. Der BlindedGenerator g_k des letzten Mixers M entspricht dem ElectionGenerator $\hat{g}$. Dieser wird durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

$$g_k = g_{k-1}^{\alpha_k} \ mod \ p \tag{13}$$

5. Election Preparation

Zuerst werden die ElectionOptions mittels der in der Konfiguration ausgewählten Einstellungen erstellt. Die ElectionOptions enthalten die Auswahlmöglichkeiten an Kandidaten und Listen sowie die Regeln, die für diese Wahl gelten. Die Kandidaten und Listen sind in den Choices C und die Regeln in den Rules R enthalten. An dieser Stelle wird auf eine weitere Beschreibung, des komplexen Aufbaus der Choices C und Rules R verzichtet und auf die UniVote System Specification [**7**] verwiesen. Die ElectionOptions werden durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

Als Zweites wird die ElectionData erstellt. Diese enthält die Werte, welche in den vorherigen Schritten erstellt wurden, den ElectionGenerator $\hat{g}$, den EncryptionKey y, den Generator G, die GroupOrder Q, die Prime P, den Titel der Wahl, die Choices C und die Rules R. Die ElectionData wird durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

Als Drittes wird die ElectoralRoll erstellt. Diese enthält eine Liste aller wahlberechtigten Wähler V. Dazu wird für jeden Wähler ein Hash über die VoterID berechnet und diesen zu der ElectoralRoll hinzugefügt. Der ElectionAdministrator EA signiert diese und übermittelt die ElectoralRoll an das ElectionBoard EB.

Als Letztes werden die VerificationKeys der Voter vk_i durch die Mixer M_k gemixt. Dazu nimmt der Erste Mixer M_1 die Liste der VerificationKeys der Voter vk_i und mischt die Liste zufällig[4]. Anschliessend berechnet er für jeden VerificationKey vk_i einen Blinded Verification Key $vk'_i = vk_i^{\alpha_k}$. Diese Blinded Verification Keys vk'_i fügt er zu den MixedVerificationKeys VK_k hinzu, berechnet einen NIZKP und signiert sie. Die folgenden Mixer M führen diese Schritte analog aus, nehmen jedoch die MixedVerificationKeys VK_{k-1} des Vorgängers anstelle der originalen VerificationKeys vk_i . Die MixedVerificationKeys VK_k des letzten Mixers entsprechen den VerificationKeys VK. Diese werden durch den ElectionManager EM signiert und an das ElectionBoard EB übermittelt.

$$vk'_i = vk_i^{\alpha_k} \ mod \ p \tag{14}$$

6. Election Period

Als Erstes werden die Zertifikate für die Wähler, die sich erst während der Wahlphase registrieren, erstellt. Dazu wird für die in der Konfiguration ausgewählte Anzahl spät registrierende Wähler, Analog Schritt 3, zuerst ein Schnorr KeyPair $(\overline{sk}_i, \overline{vk}_i)$ und, basierend darauf, ein Zertifikat $\bar{Z}_i$ generiert. Die entsprechenden Schlüssel werden im KeyStore KS abgelegt, da diese im weiteren Verlauf der Wahl noch benötigt werden. Anschliessend werden die Zertifikate $\bar{Z}_i$ an das ElectionBoard EB übermittelt.

[4] Im VoteGenerator wird das Mischen durch `Collections.shuffle(Liste, new SecureRandom())` durchgeführt.

Name	**Identifier**	**Zertifikat**	**Signiert**	**Private Key**	**Public Key**
Lately Registred Voters	$\bar{V}_i$	$\bar{Z}_i$	CA	$\overline{sk}_i$	$\overline{vk}_i$

Tabelle 5: Election Period Zertifikate

Als Zweites werden die VerificationKeys der Lately Registred Voters $\overline{vk}_i$ durch die Mixer M_k anonymisiert. Dazu nimmt der Erste Mixer M_1 die Liste der VerificationKeys der Lately Registred Voters $\overline{vk}_i$ und berechnet für jeden VerificationKey $\overline{vk}_i$ den MixedVerificationKey $\overline{vk}_{i,k} = \overline{vk}_i^{\alpha_k}$. Für jeden MixedVerificationKey $\overline{vk}_{i,k}$ berechnet er einen NIZKP und signiert ihn. Die folgenden Mixer M führen diese Schritte analog aus, nehmen aber jeweils die MixedVerificationKeys des Vorgängers $\overline{vk}_{i,k-1}$ anstelle der originalen VerificationKeys $\overline{vk}_i$. Die MixedVerificationKeys $\overline{vk}_{i,k}$ des letzten Mixers entsprechen den VerificationKeys $\overline{vk}'_i$. Der ElectionManager EM berechnet für jeden VerificationKeys $\overline{vk}'_i$ einen NIZKP, signiert ihn und übermittelt sie anschliessend an das ElectionBoard EB.

$$\overline{vk}_{i,k} = \overline{vk}_{i,k-1}^{\alpha_k} \; mod \; p \tag{15}$$

Als Letztes werden die Ballots B, sprich die Wahlzettel der Voter V_i, generiert. Der VoteGenerator erstellt dabei für jeden Voter V_i ein Ballot B_i. Für die Generierung eines Ballots B_i wird zuerst eine Vote v_i erstellt. Diese enthält die ausgewählten Choices C. Anschliessend wird die Vote v_i in einen BitString umgewandelt und ins dezimale System konvertiert. Das Ergebnis ist eine codierte Vote m'_i.[5] Die codierte Vote m'_i wird nun in $m = G(m')$ gemappt und anschliessend wird m in $E_i = (a_i, b_i)$ mittels des EncryptionKey y verschlüsselt. Jeder Voter V_i berechnet einen NIZKP und signiert das Ballot B_i. Dabei wird zur Anonymisierung der VerificationKey vk_i durch einen anonymisierten VerificationKey $vk'_j = \hat{g}^{sk_i}$ ersetzt. Wenn alle Ballots B erstellt sind, signiert der ElectionManager EM diese und übermittelt sie an das ElectionBoard EB.

[5] Die Stimmzettelgenerierung wird in Abschnitt 5.5 genauer beschrieben

$$m_i = G(m_i') = \begin{cases} m_i' + 1, & falls\ (m_i' + 1)^Q = 1, \\ P - (m_i' + 1), & sonst. \end{cases} \tag{16}$$

$$vk_j' = \hat{g}^{sk_i}\ mod\ p \tag{17}$$

7. Mixing and Tallying

Als Erstes werden die Ballots der Voter b_i durch die Mixer M_k gemixt. Dazu nimmt der Erste Mixer M_1 die Ballots B und erstellt daraus eine Liste von EncryptedVotes E. Diese Liste mischt er zufällig[6]. Anschliessend wird jede EncryptedVote E_i erneut verschlüsselt ohne dabei den Inhalt der EncryptedVote E_i zu kennen oder den Inhalt m zu verändern. Dies wird durch die Multiplikation von E_i mit der Verschlüsselung des neutralen Elements 1 erreicht. Die berechnete ReEncryptedVote E_i' fügt der Mixer M_1 zu den MixedEncryptedVotes ε_k hinzu, berechnet einen NIZKP und signiert sie. Die folgenden Mixer M führen diese Schritte analog aus, nehmen aber jeweils die MixedEncryptedVotes des Vorgängers ε_{k-1} anstelle der originalen EncryptedVotes E. Die MixedEncryptedVotes ε_k des letzten Mixers entsprechen den EncryptedVotes ε'. Diese werden durch den ElectionAdministrator EA signiert und an das ElectionBoard EB übermittelt.

$$E_i' = E_i \cdot Enc_y(1) = \big((a_i \cdot a_{neutr}\ mod\ P), (\ b_i \cdot b_{neutr}\ mod\ P)\big) \tag{18}$$

Als Nächstes werden die EncryptedVotes ε' jeweils durch die Tallier T_j partiell entschlüsselt. Das heisst, jeder Tallier T_j entfernt seinen Teil des EncryptionKey y aus den EncryptedVotes ε'. Dazu nimmt jeder Tallier T_j die EncryptedVotes ε' und berechnet für jede ReEncryptedVote $E_i' = (a_i, b_i)$ die darin enthalten ist $a_{ij} = {a_i}^{-x_j}$. Diese PartiallyDecryptedVote a_{ij} fügt er zu den PartiallyDecryptedVotes $\bar{a}_j$ hinzu, berechnet einen NIZKP und signiert sie. Der ElectionAdministrator EA nimmt aus jeder ReEncryptedVote $E_i' = (a_i, b_i)$ den Wert b_i und multipliziert ihn mit dem Produkt der a_{ij} Werte der PartiallyDecryptedVotes $\bar{a}_j$ aller Tallier T_j und erhält so m_i . Durch die

[6] Im VoteGenerator wird das Mischen durch `Collections.shuffle(Liste, new SecureRandom())` durchgeführt.

Rekonstruktion von $m'_i = G^{-1}(m_i)$ wird die unverschlüsselte codierte Vote m'_i wiederhergestellt. Den Wert m'_i fügt er zur Liste der DecryptedVotes $\overline{m}'$ hinzu und sendet sie an das ElectionBoard EB.

$$a_{ij} = {a_i}^{-x_j} \ mod \ P \tag{19}$$

$$m_i = b_i \cdot \prod_j a_{ij} \ mod \ P \tag{20}$$

$${m_i}' = G^{-1}(m_i) = \begin{cases} m_i - 1, & falls \ (m \leq Q), \\ (P - m_i) - 1, & sonst. \end{cases} \tag{21}$$

Als Letztes werden die DecryptedVotes $\overline{m}'$ durch den ElectionAdministrator EA decodiert [7] und zur Liste der DecodedVotes $\mathcal{V}$ hinzugefügt. Der ElectionManager EM signiert die DecodedVotes $\mathcal{V}$ und übermittelt diese an das ElectionBoard EB.

8. Fault Implementation

Die in der Konfiguration ausgewählten Fehler werden geladen und der Reihe nach implementiert. Die Fehlerimplementation wird in Abschnitt 0 genauer beschrieben.

9. Store in DB

Als letzten Schritt wird die generierte Wahl in der db4o-Datenbank gespeichert. Die generierte Wahl besteht aus dem ElectionBoard EB und dem KeyStore KS. Für das spätere Laden einer generierten Wahl wird nur noch das ElectionBoard EB benötigt. Der vollständigkeitshalber und für Überprüfungszwecke wird der KeyStore KS jedoch ebenfalls gespeichert.

[7] Der Stimmzettelaufbau wird in Abschnitt 5.5 genauer beschrieben

5.4 Publizierung des ElectionBoards

Das ElectionBoard EB wird an den ElectionBoardWebService EBWS gesendet und auf der IP-Adresse und den Port, der in der Konfiguration angegeben wurde, publiziert. Bevor der Webservice gestartet wird, führt der Publisher eine Überprüfung durch, ob die Adresse und der Port frei sind. Ist der Webservice gestartet, kann eine Überprüfung der Wahl mittels des VoteVerifiers durchgeführt werden. Dieser muss gegebenenfalls noch angepasst werden, damit er sich mit der richtigen Adresse und Port verbindet. Im VoteGenerator ist eine angepasste Version des VoteVerifiers enthalten, der auf der Adresse „localhost“ und dem Port „8080“, die der Standard Konfiguration entspricht, eingestellt ist. Diese lässt sich im Anschluss an den Start des Webservices direkt von der GUI aus starten.

5.5 Stimmzettelgenerierung und Aufbau

Die Stimmen werden im VoteGenerator zufällig generiert. Jedem Kandidaten wird dabei eine zufällige Anzahl Stimmen zwischen 0 und der maximalen Anzahl Stimmen, die pro Wählenden abgegeben werden dürfen, zugeteilt. Der Aufbau eines Stimmzettels ist dabei folgendermassen aufgebaut:

Zuerst die politische Liste 1, dann alle Kandidaten dieser Liste. Anschliessend die nächste politische Liste 2 und wiederum alle Kandidaten dieser Liste. Die Reihenfolge ist jedoch invertiert, d.h. man muss mit dem letzten Element beginnen. Jeder politischen Liste kann der Binärwert 0 (keine Stimme) oder 1 (eine Stimme) zugewiesen werden. Die Bitgrösse der Kandidaten hängt von der maximalen Anzahl möglicher Stimmen pro Wähler ab. Zu beachten ist, dass bereits bei der maximalen Anzahl von zwei Stimmen zwei Bits benötigt werden, da die Werte 00 (keine Stimme), 01 (eine Stimme) und 10 (zwei Stimmen) benötigt werden. Führende Nullen müssen jeweils aufgefüllt werden, da sonst der falsche Dezimalwert entstehen kann. Als Letztes wird der Binärwert in eine dezimale Zahl konvertiert und als Stimmzettel abgespeichert.

Beispiel:

Die Wahl besteht aus zwei politischen-Listen (L_1, L_2) und vier Kandidaten $(C_1 - C_4)$, wobei nur der vierte Kandidat (C_4) auf der politischen Liste (L_2) steht. Die Maximale Anzahl Stimmen pro Wähler ist auf zwei festgelegt (2 Bits). Die Erstellung eines Stimmzettels mit dem Inhalt, eine Stimme für Kandidat2 (C_2) sieht folgendermassen aus:

ChoiceId	6	5	4	3	2	1
Choice	C_4	L_2	C_3	C_2	C_1	L_1
Bits	00	0	00	01	00	1

Tabelle 6: Beispiel Stimmzettelgenerierung

Der Stimmzettel hätte also den binären Wert 0000001001 was im dezimalen System der Zahl 9 entspricht.

Die Ausgabe einer simulierten Wahl im VoteVerifier ist in Abbildung 4 abgebildet.

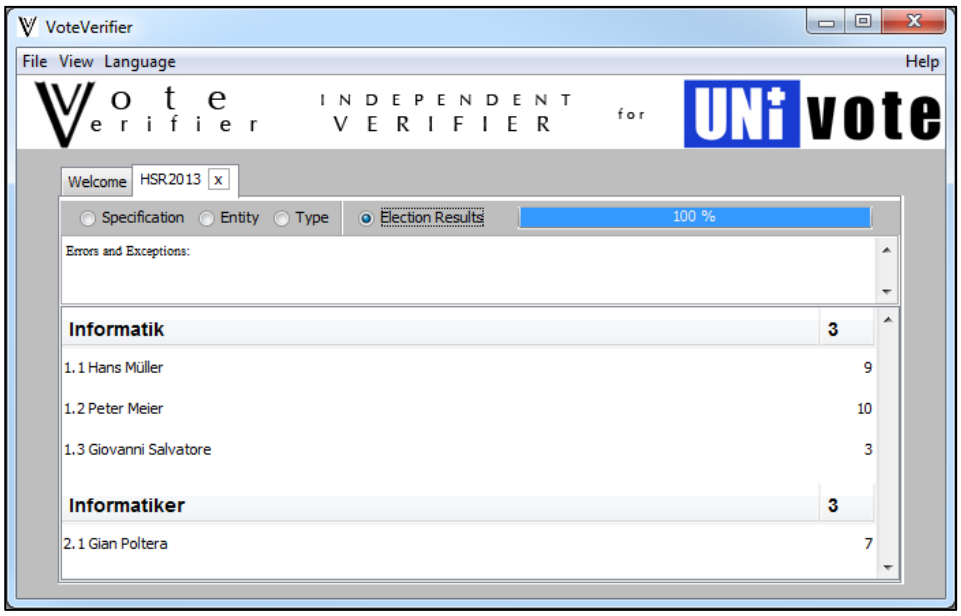


Abbildung 4: Stimmzettelgenerierung und Ausgabe im VoteVerifier

5.6 Fehlerimplementation

Die zu implementierenden Fehler können in der Konfiguration ausgewählt werden. Die zur Verfügung stehenden Fehler sind:

Nr	Fehler	Nr	Fehler
1.	Schnorr p ist Prime	17.	EA Zertifikat
2.	Schnorr q ist Prime	18.	Tallier Zertifikat*
3.	Schnorr g ist Generator	19.	Mixer Zertifikat*
4.	Schnorr p ist SafePrime	20.	Voter Zertifikat*
5.	Schnorr Parameter Länge	21.	Signatur des EA Zertifikats
6.	ElGamal P ist Prime	22.	Signatur der ElectionBasicParameters
7.	ElGamal Q ist Prime	23.	Signatur der Tallier und Mixer Zert.
8.	ElGamal G ist Generator	24.	Signatur der ElGamal Parameter
9.	ElGamal P ist SafePrime	25.	Signatur des EncryptionKey*
10.	ElGamal Parameter Länge	26.	Signatur der EncryptionKeys
11.	EncryptionKey y	27.	Signatur des BlindedGenerators*
12.	ElectionGenerator $\hat{g}$	28.	Signatur des ElectionGenerators
13.	VerificationKeys vk'	29.	Signatur der ElectionOptions
14.	Ballots B	30.	Signatur der ElectionData
15.	CA Zertifikat	31.	NIZKP des EncryptionKeyShare*
16.	EM Zertifikat	32.	NIZKP des BlindedGenerators*

Tabelle 7: Mögliche Fehler des VoteGenerators

Die Wahl wird zuerst korrekt generiert und anschliessend durch die ausgewählten Fehler verfälscht. Für die Fehler, die in Tabelle 7 mit einem * markiert sind, steht eine Auswahl des entsprechenden Mixers, Talliers oder Voters, zur Verfügung.

5.6.1 Parameter

Bei den Fehlern 1, 2 und 6 bis 8 wird der gültige Schnorr und ElGamal Parameter durch eine Zahl, die keine Primzahl ist, ersetzt. Dies wird durch die Addition von 1 zum aktuellen Wert erreicht. Da diese Zahl gerade ist, ist sie sicher keine Primzahl mehr. Die Fehler 3 und 8 werden durch die Subtraktion von 1 des aktuellen Werts erzielt. Da bei der Generierung der Wahl jeweils der Kleinste Generator gewählt wird, ist der neue Wert sicher kein Generator mehr.

In Fehler 4 und 9 wird die aktuelle SafePrime durch eine Primzahl der gleichen Grösse ersetzt. Der neue Wert ist mit sehr hoher Wahrscheinlichkeit kein SafePrime mehr. Durch die Division mit 10 können die Fehler 5 und 10 erreicht werden. Beim Fehler 11 wird der EncryptionKey um 1 erhöht. Der Fehler 11 wird durch das Ersetzen des ElectionGenerators durch den BlindedGenerator des ersten statt des letzten Mixers erreicht. Durch das Hinzufügen eines VerificationKey zur Menge der VerificationKeys wird der Fehler 13 erzeugt. In Fehler 14 wird ein zusätzliches Ballot der Menge der Ballots hinzugefügt.

5.6.2 Zertifikate

Bei den Fehlern 15 bis 20 wird das aktuelle Zertifikat durch ein identisches, jedoch mit einem falschen SignatureKey versehenem Zertifikat, ersetzt. In Fehler 18 bis 20 wird nur das Zertifikat des jeweiligen Mixers, Talliers oder Voters, das in der Konfiguration ausgewählt wurde, ersetzt.

5.6.3 Signaturen

Bei den Fehlern 21 bis 30 wird die aktuelle Signatur durch eine identische, jedoch mit einem falschen SignatureKey errechnete Signatur, ersetzt. In Fehler 25 und 27 wird nur die Signatur des jeweiligen Mixers oder Talliers, die in der Konfiguration ausgewählt wurde, ersetzt.

5.6.4 NIZKP

Bei den Fehlern 31 und 32 wird der NIZKP des in der Konfiguration ausgewählten Mixers oder Talliers durch den NIZKP eines anderen Mixers, bzw. Talliers ersetzt.

6 VoteGenerator Kryptographie

Nachfolgend wird die Implementation der verschiedenen kryptographischen Komponenten des VoterGenerators beschrieben.

6.1 RSA Implementierung

6.1.1 RSA KeyPair Generierung

Zuerst wird die Schlüssellänge von der Konfiguration geladen und anschliessend mittels des KeyPairGenerator[8] ein neues Schlüsselpaar generiert. Die Rückgabe ist ein RSA KeyPair (sk, vk).

6.1.2 RSA Signatur

Als Input wird ein Value val und ein RSA PrivateKey sk benötigt. Über den Wert val wird ein Hash H berechnet. Anschliessend wird für diesen Hash H eine Signatur $S = H^{sk}$ berechnet. Die notwendige Prime p' wird dabei aus dem RSA PrivateKey sk ausgelesen. Die berechnete Signatur S wird zurückgegeben.

$$S = H(val)^{sk} \bmod p' \tag{22}$$

6.1.3 RSA Verifizierung

Die RSA Verifizierung wird im VoteGenerator eigentlich nicht benötigt, wurde jedoch trotzdem implementiert, um die korrekte Funktionsweise der RSA Implementierung zu testen.

Als Input wird ein Value val, seine Signatur S und ein PublicKey vk benötigt. Über den Wert val wird ein Hash H' berechnet. Als Zweites wird die Verifikation S' durch $S' = S^{vk}$ berechnet. Die notwendige Prime p' wird dabei

[8] Ist nicht durch den VoteGenerator implementiert, es wird die vorhandene Klasse `java.security.KeyPairGenerator` verwendet.

aus dem RSA PrivateKey vk ausgelesen. Als Letztes werden der Hash H' und die Verifikation S' miteinander verglichen. Stimmen die Werte überein, wird $true$, andernfalls $false$, zurückgegeben.

$$S' = S^{vk} \bmod p' \tag{23}$$

6.2 Schnorr Implementierung

6.2.1 Schnorr KeyPair Generierung

Als Input werden die Schnorr SignatureParamater (p, q, g) benötigt. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von q ist, gesucht. Diese Primzahl ist der geheime Schlüssel sk. Der öffentliche Schlüssel vk wird nun durch $vk = g^{sk}$ berechnet. Die Rückgabe ist ein Schnorr KeyPair (sk, vk).

$$vk = g^{sk} \bmod p \tag{24}$$

6.2.2 Schnorr Signatur

Als Input wird ein Wert m, der ElectionGenerator $\hat{g}$ und ein Schnorr PrivateKey sk benötigt. Die Werte p, q können aus dem Schnorr PrivateKey sk ausgelesen werden. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von q, ist gesucht. Diese Primzahl ist der RandomValue r. Nun wird der Wert $gr = g^r$ berechnet. Der erste Wert der Schnorr Signatur a wird durch die Berechnung eines Hashs H über die Konkatenation von m und gr berechnet. Der zweite Wert der Schnorr Signatur b wird nun durch $b = r - sk \cdot a$ berechnet. Die Schnorr Signatur $S = (a, b)$ wird zurückgegeben.

$$gr = \hat{g}^r \bmod p \tag{25}$$

$$a = H(m \parallel gr) \tag{26}$$

$$b = r - sk \cdot a \bmod q \tag{27}$$

6.2.3 Schnorr Verifizierung

Die Schnorr Verifizierung wird im VoteGenerator eigentlich nicht benötigt, wurde jedoch trotzdem implementiert, um die korrekte Funktionsweise der Schnorr Implementierung zu testen.

Als Input wird ein Wert m , die Schnorr Signatur $S = (a, b)$, der ElectionGenerator $\hat{g}$ und ein Schnorr PublicKey vk benötigt. Der Wert p kann aus dem Schnorr PublicKey vk ausgelesen werden. Als Erstes wird der Wert rv durch $rv = \hat{g}^b \cdot vk^a$ berechnet. Anschliessend wird durch die Berechnung eines Hashs H über die Konkatenation von m und rv der Wert av generiert. Nun wird überprüft ob der erste Wert der Schnorr Signatur a identisch zu dem berechneten Wert av ist. Trifft dies zu, wird $true$, andernfalls $false$ zurückgegeben.

$$rv = \hat{g}^b \cdot vk^a \bmod p \tag{28}$$

$$av = H(m \parallel rv) \tag{29}$$

$$Verify_{vk}(m, S) = \begin{cases} true, \text{wenn } a = av, \\ false, \quad \text{sonst.} \end{cases} \tag{30}$$

6.3 ElGamal Implementierung

6.3.1 ElGamal PublicParameters Generierung

Als Input wird die gewünschte Schlüssellänge ℓ benötigt. Basierend darauf wird mittels des PrimeGenerators eine neue SafePrime generiert und als GroupOrder Q verwendet. Dies kann bei einem Wert $\ell > 2048$ sehr lange dauern. Als Nächstes wird die Prime $P = Q \cdot 2 + 1$ berechnet. Zuletzt muss noch ein geeigneter Generator G gefunden werden. Dazu wird der Generator G auf den Wert 2 gesetzt und solange um 1 erhöht bis die Bedingung $G^Q \not\equiv 1$ erfüllt ist. Die EncryptionParameter (P, Q, G) werden zurückgegeben.

$$Q = SafePrime^{\ell} \tag{31}$$
$$P = Q \cdot 2 + 1 \tag{32}$$
$$G^{Q} \not\equiv 1 \bmod P \tag{33}$$

6.3.2 ElGamal KeyPair Generierung

Als Input werden die EncryptionParamter (P, Q, G) benötigt. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von Q ist, gesucht. Diese Primzahl ist der geheime Schlüssel x. Der öffentliche Schlüssel y wird nun durch $y = G^x$ berechnet. Die Rückgabe ist ein ElGamal KeyPair (x, y).

$$x = Prime^{\ell_Q} \tag{34}$$
$$y = G^x \bmod P \tag{35}$$

6.3.3 ElGamal Encryption

Als Input wird ein Wert m , ein öffentlicher Schlüssel y und die EncryptionParameters (P, Q, G) benötigt. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von Q ist, gesucht. Diese Primzahl ist der RandomValue r. Der erste Wert der Verschlüsselung a wird nun durch $a = G^r$ berechnet. Der Zweite Wert b wird mittels $b = m \cdot y^r$ berechnet. Die ElGamal Verschlüsselung (a, b) wird zurückgegeben.

$$a = G^r \bmod P \tag{36}$$
$$b = m \cdot y^r \bmod P \tag{37}$$

6.3.4 ElGamal Decryption

Die ElGamal Decryption wird im VoteGenerator eigentlich nicht benötigt. Sie wurde trotzdem implementiert, um die korrekte Funktionsweise der ElGamal Implementierung zu testen.

Als Input wird eine ElGamal Verschlüsselung (a, b) sowie einen privaten ElGamal Schlüssel x benötigt. Der entschlüsselte Wert m wird durch $m = a^{-x} \cdot b$ berechnet. Die notwendige Prime P wird dabei aus dem privaten ElGamal Schlüssel x ausgelesen. Die Rückgabe ist der entschlüsselte Wert m.

$$m = a^{-x} \cdot b \bmod P \tag{38}$$

6.4 Zertifikat Implementierung

Als Input wird ein CName *cname*, ein PrivateKey *sk* und ein PublicKey *vk* benötigt. Zuerst wird der Signatur Algorithmus *sigalg* von der Konfiguration geladen. Anschliessend werden die verschiedenen Parameter für die Zertifikatsgenerierung gesetzt und mittels des X509V1CertificateGenerator[9] und des PrivateKey *sk* ein neues X509Certificate erstellt.

Parameter	**Wert**
SerialNumber	Aktueller timestamp
IssuerDN	X500Principal(*cname*)[10]
NotBefore	Aktuelles Datum und Zeit
NotAfter	Aktuelles Datum und Zeit + 1000 Tage
SubjectDN	X500Principal(*cname*)[11]
PublicKey	*vk*
SignatureAlgorithm	*sigalg*

Tabelle 8: Aufbau Zertifikat

Dieses X509Certificate wird zuletzt durch den PEMWriter[12] in einen PEM String umgewandelt. Der PEM String entspricht dem Certificate Z und wird zurückgegeben.

[9] Vorhandene Klasse `org.bouncycastle.x509.X509V1CertificateGenerator` verwendet.
[10] Vorhandene Klasse `javax.security.auth.x500.X500Principal` verwendet.
[11] Vorhandene Klasse `javax.security.auth.x500.X500Principal` verwendet.
[12] Vorhandene Klasse `org.bouncycastle.openssl.PEMWriter` verwendet.

6.5 Signatur Implementierung

Es gibt zwei verschiedene Typen von Signaturen, die RSA Signatur und die Schnorr Signatur. Wie diese Signaturen generiert werden, wurde bereits in Abschnitt 6.1.2 und 6.2.2 aufgezeigt. Die jeweiligen Werte, die signiert werden sollen, müssen vorher noch in die richtige Form gebracht werden.

Als Input wird der PrivateKey sk des Signierers, das Object, das signiert werden soll und, je nachdem, noch der Identifier des Signierers benötigt. Das Object wird mithilfe des StringConcatenator[13] in die richtige Form gebracht und anschliessend durch RSA bzw. Schnorr signiert. Der Rückgabewert ist bei RSA die Signatur S und bei Schnorr die Signatur $S = (a, b)$.

6.6 RSA Signatur

Object	**Form**
Ballots	$\big(id\big\vert\big((b_1)\vert(\dots)\vert(b_n)\big)\big)\,\vert timestamp$ $b_i = id\vert vk'_1\vert(E_i)\big\vert\big(\pi_{r_i}\big)\big\vert(S_i)$ $E_i = a\vert b$ $S_i = a\vert b$ $\pi_{r_i} = t\vert s$
BlindedGenerator	$\big(id\vert g_k\vert\big((t_1\vert\dots\vert t_n)\vert(s_1\vert\dots s_n)\big)\big)\,\vert timestamp$
DecodedVotes	$\big(id\vert(v_1\vert\dots\vert v_n)\big)\vert timestamp$
ElectionData	$\big(id\vert EA\vert descr\vert P\vert Q\vert G\vert y\vert\hat{g}\vert(C)\vert(R)\big)\vert timestamp$ $C = c_1\vert\dots\vert c_n$ $R = r_1\vert\dots\vert r_n$
ElectionDefinition	$\big(id\vert descr\vert\ell\vert(T_1\vert\dots\vert T_n)\vert(M_1\vert\dots\vert M_n)\big)\vert timestamp$
ElectionGenerator	$(id\vert\hat{g})\vert timestamp$
ElectionOptions	$\big(id\vert(c_1\vert\dots\vert c_n)\vert(r_1\vert\dots\vert r_n)\big)\vert timestamp$
ElectoralRoll	$\big(id\vert(H_1\vert\dots\vert H_n)\big)\vert timestamp$
ElectionSystemInfo	$\Big(id\big\vert\big(Z_{T_1}\vert\dots\vert Z_{T_n}\big)\big\vert\big(Z_{M_1}\vert\dots\vert Z_{M_n}\big)\Big)\,\vert timestamp$
EncryptedVotes	$\big(id\vert\big((E_1)\vert\dots\vert(E_n)\big)\big)\,\vert timestamp$
EncryptionKey	$(id\vert y)\vert timestamp$

[13] Wurde vom VoteVerifier übernommen, Autor Scalzi Giuseppe

EncryptionKeyShare	$\big(id\|y_j\|\big((t_1\|\ldots\|t_n)\|(s_1\|\ldots\|s_n)\big)\big)\,\|timestamp$
EncryptionParameters	$(id\|P\|Q\|G)\|timestamp$
MixedEncryptedVotes	$\big(id\|\big((a_1\|b_1)\|\ldots\|(a_n\|b_n)\big)\|\big(( )\|( )\big)\big)\,\|timestamp$
MixedVerificationKey	$(id\|vk)\|timestamp$
MixedVerificationKey	$\big(id\|vk\|\big(( )\|( )\big)\big)\,\|timestamp$
MixedVerificationKeys	$\big(id\|(vk_1\|\ldots\|vk_n)\|\big(( )\|( )\big)\big)\,\|timestamp$
PartiallyDecryptedVotes	$\big(id\|(a_1\|\ldots\|a_n)\|\big((t_1\|\ldots\|t_n)\|(s_1\|\ldots\|s_n)\big)\big)\,\|timestamp$
VerificationKeys	$\big(id\|(vk_1\|\ldots\|vk_n)\big)\|timestamp$
VoterCertificates	$\big(id\|(Z_1\|\ldots\|Z_n)\big)\|timestamp$

Tabelle 9: Signatur Form, RSA

Die Form der Choices C, Rules R und Votes $\mathcal{V}$ wurde jeweils vereinfacht dargestellt. Weitere Informationen sind in der UniVote System Specification [**7**] Abschnitt 1.5.1 verfügbar. Bei den MixedEncryptedVotes, dem MixedVerificationKey und den MixedVerificationKeys fehlt der jeweilige NIZKP, welcher jedoch gemäss der UniVote System Specification [**7**] noch nicht implementiert ist.

6.7 Schnorr Signatur

Object	**Form**
Ballot	$\big(id\|(a\|b)\|\big((t)\|(s)\big)\big)$

Tabelle 10: Signatur Form, Schnorr

Bei der Schnorr Signatur eines Ballots ist zu beachten, dass der ElectionGenerator $\hat{g}$ zur Signierung verwendet wird.

6.8 NIZKP Implementierung

Die NIZKP sind sehr verschieden aufgebaut, daher variieren die Input Werte. Zur Berechnung des Hashs müssen die Werte konkateniert werden, dies wird mithilfe des StringConcatenator[14] durchgeführt. Als Rückgabewert wird jeweils ein Proof π, bestehend aus einem oder mehreren Commitments t und einem Response s, zurückgegeben.

[14] Wurde vom VoteVerifier übernommen, Autor Scalzi Giuseppe

6.8.1 EncryptionKeyShare Proof

Als Input wird der Tallier Identifier T_j, der PrivateKey x_j und der PublicKey y_j benötigt. Aus dem PublicKey y_j können die EncryptionParamater (P, Q, G) ausgelesen werden. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von Q ist, gesucht. Diese Primzahl ist der RandomValue w. Anschliessend wird das Commitment $t = G^w$ berechnet. Der Wert c wird durch die Berechnung des Hashs H über die Werte (y_j, t, T_j) generiert. Als Letztes wird der Response $s = c \cdot x_j + w$ berechnet. Der Rückgabewert ist der NIZKP Proof $\pi_{x_j} = (t, s)$.

$$w = Prime^{\ell_Q} \tag{39}$$

$$t = G^w \bmod P \tag{40}$$

$$c = H(y_j \parallel t \parallel T_j) \bmod Q \tag{41}$$

$$s = c \cdot x_j + w \bmod Q \tag{42}$$

6.8.2 BlindedGenerator Proof

Als Input wird der Mixer Identifier M_k, der BlindedGenerator g_k, der BlindedGenerator des Vorgängers g_{k-1} und der PrivateKey α_k benötigt. Aus dem PrivateKey α_k, können die SignatureParamaters (p, q, g) ausgelesen werden. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von q ist, gesucht. Diese Primzahl ist der RandomValue w. Anschliessend wird das Commitment $t = {g_{k-1}}^w$ berechnet. Der Wert c wird durch die Berechnung des Hashs H über die Werte (g_k, t, M_k) generiert. Als Letztes wird der Response $s = c \cdot \alpha_k + w$ berechnet. Der Rückgabewert ist der NIZKP Proof $\pi_{\alpha_k} = (t, s)$.

$$w = Prime^{\ell_q} \tag{43}$$

$$t = {g_{k-1}}^w \bmod p \tag{44}$$

$$c = H(g_k \parallel t \parallel M_k) \bmod q \tag{45}$$

$$s = c \cdot \alpha_k + w \bmod q \tag{46}$$

6.8.3 MixedVerificationKeys Proof

Der MixedVerificationKeys Proof ist, gemäss der UniVote System Specification [**7**], noch nicht implementiert. Der VoteGenerator liefert daher einen leeren Proof zurück.

6.8.4 LatelyMixedVerificationKey Proof

Als Input wird der Mixer Identifier M_k, der Verification Key des Vorgängers $\overline{vk}_{i,k-1}$, der neue Verification Key $\overline{vk}_{i,k}$, der BlindedGenerator des Vorgängers g_{k-1}, der PrivateKey α_k sowie die SignatureParameters (p, q, g) benötigt. Aus dem PrivateKey α_k, kann der BlindedGenerator g_k ausgelesen werden. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von q ist, gesucht. Diese Primzahl ist der RandomValue w. Anschliessend wird das erste Commitment $t_1 = g_{k-1}^w$ und das zweite Commitment $t_2 = \overline{vk}_{i,k-1}$ berechnet. Der Wert c wird durch die Berechnung des Hashs H über die Werte $(g_k, \overline{vk}_{i,k}, t_1, t_2, M_k)$ generiert. Als Letztes wird der Response $s = c \cdot \alpha_k + w$ berechnet. Der Rückgabewert ist der NIZKP Proof $\pi_{\overline{vk}_{i,k}} = \big((t_1, t_2), s\big)$.

$$w = Prime^{\ell_q} \tag{47}$$

$$t_1 = g_{k-1}^w \mod p \tag{48}$$

$$t_2 = \overline{vk}_{i,k-1} \mod p \tag{49}$$

$$c = H(g_k \parallel \overline{vk}_{i,k} \parallel t_1 \parallel t_2 \parallel M_k) \mod q \tag{50}$$

$$s = c \cdot \alpha_k + w \mod q \tag{51}$$

6.8.5 Ballot Proof

Als Input wird der anonymisierte Voter Identifier vk'_j, der erste Wert der Verschlüsselung a_i, der RandomValue r_i, der zur Verschlüsselung verwendet wurde und die EncryptionParameters (P, Q, G) benötigt. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von Q, ist gesucht. Diese Primzahl ist der RandomValue w. Anschliessend wird das Commitment $t = G^w$ berechnet. Der Wert c wird durch die Berechnung des

Hashs H über die Werte (a_i, t, vk'_j) generiert. Als Letztes wird der Response $s = c \cdot r_i + w$ berechnet. Der Rückgabewert ist der NIZKP Proof $\pi_{r_i} = (t, s)$.

$$w = Prime^{\ell_Q} \tag{52}$$
$$t = G^w \bmod P \tag{53}$$
$$c = H(a_i \parallel t \parallel vk'_j) \bmod Q \tag{54}$$
$$s = s = c \cdot r_i + w \bmod Q \tag{55}$$

6.8.6 MixedEncryptedVotes Proof

Der MixedEncryptedVotes Proof ist, gemäss der UniVote System Specification [**7**], noch nicht implementiert. Der VoteGenerator liefert daher einen leeren Proof zurück.

6.8.7 PartiallyDecryptedVotes Proof

Als Input wird der Tallier Identifier T_j , die EncryptedVotes ε' , die PartiallyDecryptedVotes $\bar{a}_j$, der PrivateKey x_j und der PublicKey y_j benötigt. Aus dem PublicKey y_j können die EncryptionParamater (P, Q, G) ausgelesen werden. Zuerst wird mittels des PrimeGenerators eine Primzahl, die kleiner als die Länge von Q ist, gesucht. Diese Primzahl ist der RandomValue w . Anschliessend wird das erste Commitment $t_0 = G^w$ berechnet. Nun wird für jede EncryptedVote, die in ε' enthalten ist, ein weiteres Commitment $t_1, ..., t_n = a_i^{-w}$ berechnet. Der Wert c wird durch die Berechnung des Hashs H über die Werte $(y_j, \bar{a}, \bar{t}, T_j)$ generiert. Als Letztes wird der Response $s = c \cdot x_j + w$ berechnet. Der Rückgabewert ist der NIZKP Proof $\pi'_{x_j} = (\bar{t}, s)$.

$$w = Prime^{\ell_Q} \tag{56}$$
$$t_1 = G^w \bmod P \tag{57}$$
$$\begin{gathered} t_2 = a_1^{-w} \bmod P \\ \cdots \\ t_n = a_n^{-w} \bmod P \end{gathered} \tag{58}$$
$$c = H(y_j \parallel \bar{a}_j \parallel t_1 \parallel \cdots \parallel t_n \parallel T_j) \bmod Q \tag{59}$$
$$s = c \cdot x_j + w \bmod Q \tag{60}$$

6.9 Hash Implementierung

Als Input werden ein String Wert, der HashAlgorithmus und die Zeichencodierung benötigt. Zuerst wird der String Wert in ein Byte-Array konvertiert und anschliessend mithilfe von MessageDigest [15] ein Hash-Wert berechnet. Zur korrekten Zeichencodierung wird der Charset[16] angegeben. Der Rückgabewert ist der Hash H.

6.10 Implementierung der Zufallszahlen

6.10.1 Zufallszahl

Als Input wird die Länge ℓ benötigt. Zuerst wird eine neue Zufallszahl n der Bitlänge ℓ mithilfe von SecureRandom[17] generiert. Anschliessend werden die beiden ersten Bits auf 1 gesetzt. Durch die Generierung von Zufallszahlen mit SecureRandom kann es vorkommen, dass die Länge um einige wenige Bits abweicht. Daher wird die Länge überprüft und, falls notwendig, von vorne begonnen. Der Rückgabewert ist eine Zufallszahl n der Länge ℓ mit zwei 1er Bits an der ersten Position.

6.10.2 Prime

Als Input wird die Länge ℓ benötigt. Als Erstes wird eine Zufallszahl n, wie in Abschnitt 6.10.1 beschrieben, generiert. Anschliessend wird die Zufallszahl n mittels des MillerRabin Tests auf prime überprüft. Schlägt dieser Test fehl, wird solange mithilfe von BigInteger.nextProbablePrime [18] die nächstmögliche Primzahl generiert bis der MillerRabin Test positiv ist. Der Rückgabewert ist eine mögliche Primzahl p der Länge ℓ.

[15] Vorhandene Klasse `java.security.MessageDigest` verwendet.
[16] Vorhandene Klasse `java.nio.charset.Charset` verwendet.
[17] Vorhandene Klasse `java.security.SecureRandom` verwendet.
[18] Vorhandene Klasse `java.math.BigInteger` verwendet.

6.10.3 SafePrime

Als Input wird die Länge ℓ benötigt. Als Erstes wird eine Primzahl p, wie in Abschnitt 6.10.2 beschrieben, generiert. Anschliessend wird die Berechnung $p \cdot 2 + 1$ durchgeführt und mittels des MillerRabin Test überprüft, ob das Ergebnis wiederum eine Primzahl ist. Schlägt dieser Test fehl, wird solange mithilfe von BigInteger.nextProbablePrime[19] die nächste mögliche SafePrime generiert, bis das Ergebnis der Berechnung $p \cdot 2 + 1$ ebenfalls eine Primzahl ist. Der Rückgabewert ist eine mögliche SafePrime sp der Länge ℓ.

6.10.4 MillerRabin

Eine zufällige Zahl n wird mittels des Miller-Rabin-Tests auf prime überprüft. Durch den zusätzlichen Wert s kann die Anzahl an Durchläufen und damit die Wahrscheinlichkeit für eine korrekte Ermittlung einer Primzahl angegeben werden. Der Miller-Rabin-Test wurde in einigen Punkten erweitert um eine bessere Performance zu erzielen, so ist gemäss *Pomerance et al.* [**26**] und *Jaeschke* [**27**] ausreichend bis zu einer gewissen Grösse einer Zahl nur vorgegebene Werte für a zu verwenden. Dies wurde im nachfolgenden Algorithmus unter Punkt 4 berücksichtigt.

Der Algorithmus sieht wie folgt aus und benötigt als Input eine zu testende Zahl n und die Anzahl an Durchgängen s:

n muss eine natürliche, ungerade Zahl grösser als 2 sein (wird überprüft)

1. Überprüfung ob $n = 2$, $n = 3$ oder $n = 7$ falls ja gib *true* zurück
2. Durchführung von Basis Tests:

Ist n grösser als 2?	$n > 2$
Ist n nicht durch zwei teilbar?	$n \neq 0 \mod 2$
Ist n nicht durch drei teilbar?	$n \neq 0 \mod 3$
Ist n nicht durch fünf teilbar?	$n \neq 0 \mod 5$
Ist n nicht durch sieben teilbar?	$n \neq 0 \mod 7$

   Falls eine der Fragen mit nein beantwortet wird, gib *false* zurück

[19] Vorhandene Klasse `java.math.BigInteger` verwendet.

3. a) Wähle die Werte für a für kleine Zahlen:
   Ist n kleiner als 1373653 wähle die Werte (2,3)
   Ist n kleiner als 9080191 wähle die Werte (31,73)
   Ist n kleiner als 4759123141 wähle die Werte (2,7,61)
   Ist n kleiner als 2152302898747 wähle die Werte (2,3,5,7,11)
   Ist n kleiner als 3474749660383 wähle die Werte (2,3,5,7,11,13)
   Ist n kleiner als 341550071728321 wähle die Werte (2,3,5,7,11,13,17)
   b) Wähle die Werte für a für grosse Zahlen, mit s Wiederholungen:
   Generiere ein Seed mittels SecureRandom
   Generiere eine neuen Zahl R der Bit-Länge $\log(n) - 2$ aus dem Seed
   Falls $R < n$, füge R zu a hinzu.
4. Für jeden Wert a wird nun der eigentliche Miller-Rabin-Test durchgeführt:
   a) Setze $j = 0$ und erhöhe j solange um 1, bis $\frac{n-1}{2^j} = 0 \mod 2$ erfüllt ist
   b) Berechne $d = \frac{n-1}{2^j}$
   c) Falls $a^d = 0 \mod n$, gib *true* zurück
   d) Falls $a^d = n - 1 \mod n$, gib *true* zurück
   e) Setze $x = a^d \mod n$
   f) Führe $j - 1$ mal, $x = x^2 \mod n$ durch und überprüfe jedes Mal ob $x = n - 1 \mod n$ erfüllt ist, falls ja gib *true* zurück
   g) Sonst gib *false* zurück

Der Rückgabewert ist ein Boolean der das Ergebnis des Tests beschreibt.

6.11 JUnit Tests

Um das korrekte arbeiten der Kryptofunktionen des VoteGenerators zu überprüfen, werden vier JUnit-Tests durchgeführt:

Als Erstes wird ein RSA KeyPair generiert, ein Wert mittels RSA signiert und anschliessend verifiziert. Falls die Verifizierung fehlschlägt, schlägt auch der Test fehl.

Als Zweites wird wird ein Schnorr KeyPair generiert, ein Wert mittels Schnorr signiert und anschliessend verifiziert. Falls die Verifizierung fehlschlägt, schlägt auch der Test fehl.

Als Drittes werden ElGamal Paramater und, basierend darauf, ein ElGamal KeyPair generiert. Ein Wert mittels ElGamal ver- und wieder entschlüsselt. Falls der entschlüsselte Wert nicht dem Wert vor der Entschlüsselung entspricht, schlägt der Test fehl.

Als Viertes wird überprüft ob der Miller-Rabin-Test alle Primzahlen zwischen 1 und 10‘000‘000 sowie zwischen 10^{308} und $10^{308} + 10'000$ findet. Werden nicht alle Primzahlen gefunden, schlägt der Test fehl.

7 VoteGenerator Data Management

7.1 ElectionBoard

Das ElectionBoard ist das zentrale Element bei der Generierung und Publizierung einer Wahl. Im Anschluss an sämtliche Wahl-Generierungsschritte wird das Ergebnis an das ElectionBoard übermittelt und, falls notwendig, werden die Werte wieder geladen. Der ElectionBoardWebService greift ebenfalls auf das ElectionBoard zu. Für die Vereinfachung dieser Aufgaben verfügt das ElectionBoard für jeden Variablen Typ über eine get und set Methode. Die Variabeln in einer Map können zusätzlich durch Angabe des Mixer oder Tallier Identifiers direkt abgerufen werden. Der Aufbau eines Variablentyps wurde mittels der, vom UniVote Team zur Verfügung gestellten Common.xsd Files, generiert.

Variablen Typ	**Enthält**
Ballots	Ballots B
Certificate	RootCertificate RA_Z
DecodedVotes	Decodierten Votes $\mathcal{V}$
DecryptedVotes	Entschlüsselte Votes m'_i
ElectionSystemInfo	Zertifikate $\left(Z_{EM}, Z_{EA}, Z_{M_k}, Z_{T_j}\right)$
ElectionDefinition	ElectionId id, Wahlbeschreibung $descr$, Schlüssellänge ℓ, Mixer Identifier M_k, Tallier Identifier T_j
ElectionGenerator	ElectionGenerator $\hat{g}$
ElectionOptions	Choices C, Rules R
ElectionData	ElectionId id, Wahlbeschreibung $descr$, ElectionGenerator $\hat{g}$, EncryptionKey y, ElGamal Parameter (P, Q, G)
ElectoralRoll	Hashs der Voter H_i

EncryptedVotes	Verschlüsselte Votes $\mathcal{E}$
EncryptionParameters	ElGamal Parameter (P, Q, G)
EncryptionKey	EncryptionKey y
KnownElectionIds	ElectionId id
List<Certificate>	Lately Registred Voter Zertifikate $\hat{Z}_i$
List<MixedVerificationKey>	VerificationKeys $\hat{v}k'_i$
Map<String, BlindedGenerator>	Mixer M_k, BlindedGenerator g_k
Map<String, EncryptionKeyShare>	Tallier T_j, EncryptionKey y_j
Map<String, List>	Mixer M_k, VerificationKeys $\hat{v}k_{i,k}$
Map<String, MixedEncryptedVotes>	Mixer M_k, EncryptedVotes $\mathcal{E}_k$
Map<String, MixedVerificationKeys>	Mixer M_k, VerificationKeys vk_k
Map<String, PartiallyDecryptedVotes>	Tallier T_j, PartiallyDecryptedVote $\bar{a}_j$
SignatureParameters	Schnorr Parameter (p, q, g)
VoterCertificates	Voter Zertifikate Z_i
VerificationKeys	VerificationKey vk_i

Tabelle 11: ElectionBoard Varianlen Typen

7.2 ElectionBoardWebService

Für die Erstellung des ElectionBoardWebService wurden vom UniVote Team die zwei Files ElectionBoardService.wsdl und ElectionBoardService.xsd zur Verfügung gestellt. Dies ermöglichte einen WebService mit den gleichen Parametern wie UniVote zu erstellen. Nach dem Start ist der WebService standardmässig unter folgender Adresse abrufbar:

http://localhost:8080/ElectionBoardService/ElectionBoardServiceImpl?wsdl

Der VoteVerifier ruft die Daten über diese Adresse ab. Je nach Bedarf können die IP-Adresse und der Port in der Konfiguration angepasst werden.

7.3 KeyStore

Damit die Wahl korrekt simuliert werden kann, muss der VoteGenerator auch im Besitzt der privaten Schlüssel der Wahlbeteiligten sein. Der KeyStore verwaltet alle Schlüssel während der Generierung einer Wahl. Dafür verfügt er für jeden Variablen Typ über eine get und set Methode. Die Methoden in einer

Liste können zusätzlich durch Angabe des Voter, Mixer oder Tallier Identifiers direkt abgerufen werden.

Variablen Typ	**Enthält**
RSAPrivateKey	RootCertificateAuthority RSA PrivateKey sk_{RA}
RSAPublicKey	RootCertificateAuthority RSA PublicKey vk_{RA}
RSAPrivateKey	CertificateAuthority RSA PrivateKey sk_{CA}
RSAPublicKey	CertificateAuthority RSA PublicKey vk_{CA}
RSAPrivateKey	ElectionManager RSA PrivateKey sk_{EM}
RSAPublicKey	ElectionManager RSA PublicKey vk_{EM}
RSAPrivateKey	ElectionAdministrator RSA PrivateKey sk_{EA}
RSAPublicKey	ElectionAdministrator RSA PublicKey vk_{EA}
List<RSAPrivateKey>	Mixers RSA PrivateKey sk_{M_k}
List<RSAPublicKey>	Mixers RSA PublicKey vk_{M_k}
List<RSAPrivateKey>	Talliers RSA PrivateKey sk_{T_j}
List<RSAPublicKey>	Talliers RSA PublicKey vk_{T_j}
List<DSAPrivateKey>	Voters Schnorr PrivateKey sk_{V_i}
List<DSAPublicKey>	Voters Schnorr Public Key vk_{V_i}
List<DSAPrivateKey>	Lately Registered Voters Schnorr PrivateKey $sk_{\bar{V}_i}$
List<DSAPublicKey>	Lately Registered Voters Schnorr PublicKey $vk_{\bar{V}_i}$
List<DSAPrivateKey>	Talliers ElGamal PrivateKey x_j
List<DSAPublicKey>	Talliers ElGamal PublicKey y_j
List<DSAPrivateKey>	Mixers Schnorr BlindedGenerator PrivateKey α_k

Tabelle 12: KeyStore Variablen Typen

7.4 ConfigHelper

Der ConfigHelper ladet bei der Instanzierung die Konfigurationswerte aus dem Konfigurationsfile. Die Werte können anschliessend durch das Konfigurationsmenü genändert werden. Nach Abschluss einer Konfiguration oder durch drücken auf „speichern“ werden die neuen Werte wieder in das Konfigurationsfile gespeichert und stehen für den nächsten Start des VoteGenerator zur Verfügung.

Es können sämtliche Werte abgerufen werden, dabei werden die Werte durch den ConfigHelper entsprechend geparst. Ist ein abgerufener

Konfigurationseintrag nicht vorhanden, wird ein PopUp Menü eingeblendet, das eine Beschreibung des fehlenden Wertes enthält. Dieser kann darin direkt eingegeben werden.

7.5 Datenbank Implementierung

Das komplette ElectionBoard wird nach der Generierung einer Wahl in ein Datenbankfile gespeichert. Dafür wurde eine db4o-Datenbank verwendet. Diese bietet, im Gegensatz zu herkömmlichen relationalen Datenbanken, die Möglichkeit, die komplexen UniVote Objekte direkt als solche abzuspeichern und wieder zu laden. Weiter sind keine Wartung und keine externen Datenbank-Server notwendig.

Neben dem ElectionBoard wird auch der KeyStore in einer separaten Datenbank abgelegt. Diese Datenbank wird nicht weiter benötigt. Sie wird nur zum Nachrechnen von bereits generierten Wahlen angelegt.

7.6 FileHandler

Der FileHandler speichert das aktuelle Config File, die ElectionBoard und KeyStore Datenbanken in ein externes *.vgc File ab. Dieses kann mittels des GUI in einem beliebigen Ordner abgelegt werden und auch wieder geladen werden. Da jede generierte Wahl in einer separaten Datenbank abgespeichert wird, können auch mehrere Wahlen in einem File abgelegt sein.

8 VoteGenerator User Interface

Die komplette GUI wurde möglichst einfach und intuitiv gestalten. Trotzdem ist es möglich auch recht komplexe Einstellungen vorzunehmen und die generierte Wahl nach Belieben zu gestalten. Die Implementierung wurde mittels Swing[20] und AWT[21] umgesetzt.

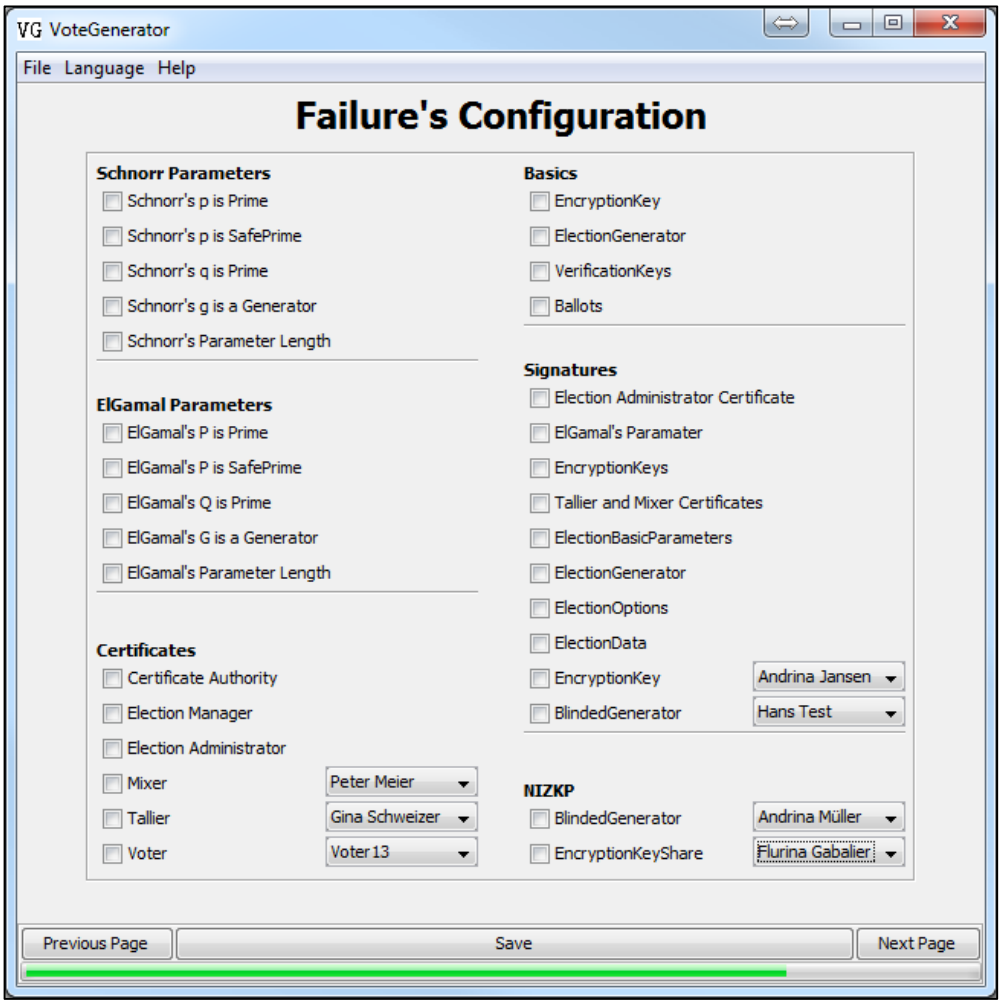


Abbildung 5: VoteGenerator GUI

[20] Vorhandene Klasse `javax.swing` verwendet

[21] Vorhandene Klasse `java.awt` verwendet

8.1 Aufbau

Das VoteGenerator GUI besteht aus einem *MainGUI*, einer *MenuBar* und einem *MiddlePanel*.

8.2 Menü

Die *MenuBar* bietet die Möglichkeit, die Einstellungen und generierten Wahlen zu speichern und zu öffnen, sowie den VoteGenerator zu beenden. Weiter kann die Sprache umgestellt werden und die Dokumentation abgerufen werden.

8.3 MiddlePanel

Das *MiddlePanel* ist die zentrale Komponente des VoteGenerators GUI. Es besteht aus dem *TitelPanel* und dem Anzeigebereich und, bei Bedarf, wird zusätzlich eine *StatusBar* eingeblendet. Wird der VoteGenerator gestartet wird als Erstes das *StartpagePanel* angezeigt, dieses bietet zwei Auswahlmöglichkeit für den weiteren Ablauf der Wahl.

8.4 Konfigurationspanel

Für die sechs Konfigurationsschritte wurde jeweils ein Panel erstellt. Diese sind das *InitialConfigurationPanel*, das *AdminsConfigurationPanel*, das *CandidatesConfigurationPanel*, das *CryptoConfigurationPanel*, das *FailureConfigurationPanel* und das *FinishConfigurationPanel*. Der Aufbau des jeweiligen Panels wird dynamisch durchgeführt. Sind bereits Werte aus einer früheren Konfiguration vorhanden, werden diese geladen. Ansonsten werden Standardwerte gesetzt. Die verschiedenen Panel werden durch den *ConfigurationPanelManager* geladen und wieder entfernt. Dadurch ist es möglich, Schritte vor und zurückzugehen.

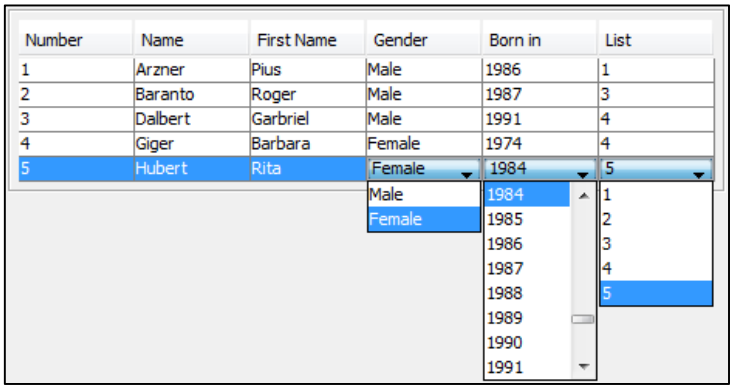


Abbildung 6: VoteGenerator GUI, CandidatesConfigurationPanel

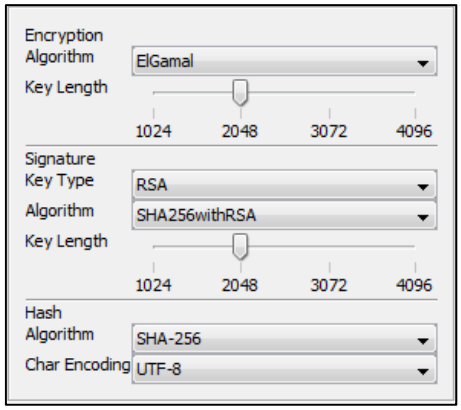


Abbildung 7: VoteGenerator GUI, CryptoConfigurationPanel

8.5 GenerateVotes Panel

Das *GenerateVotesPanel* zeigt die bereits erzeugten Wahlen in einer Liste an. Das gewünschte ElectionBoard kann selektiert und anschliessend gestartet werden. Für das Anzeigen der bereits generierten Wahlen wurde keine eigene Datenbank angelegt. Es werden die vorhandenen Datenbankfiles gezählt und aufgelistet.

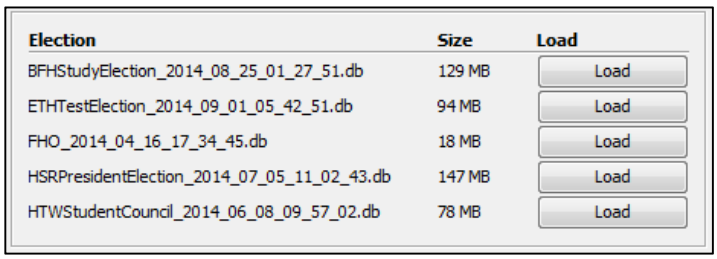


Abbildung 8: VoteGenerator GUI, GenerateVotesPanel

8.6 VoteGeneration Panel

Das *VoteGenerationPanel* zeigt den Verlauf der Generierung einer Wahl an. Es werden jeweils die einzelnen Schritte und Unterpunkte angezeigt. Ist eine Phase erfolgreich abgeschlossen oder ist ein Fehler aufgetreten, wird dies ausgegeben.

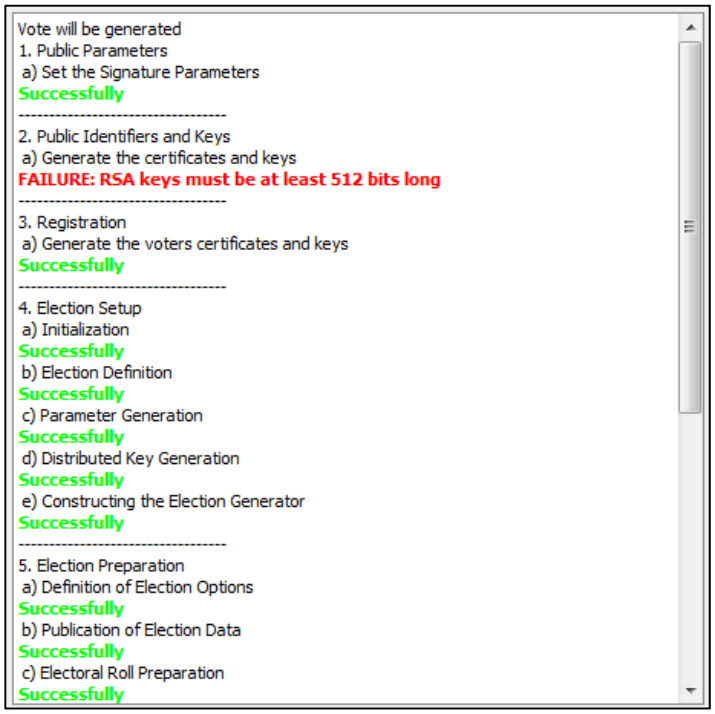


Abbildung 9: VoteGenerator GUI, VoteGenerationPanel

8.7 GeneratedVotePublish Panel

Mit dem *GeneratedVotePublishPanel* kann eine generierte Wahl Publiziert werden und anschliessend der VoteVerifier gestartet werden. Der Webservice lässt sich dabei starten und beenden.

8.8 Listener

Um Veränderungen von Paramatern durch einen Benutzer festzustellen und entsprechen reagieren zu können, wurden sechs eigene Listener erstellt. Je einen für CheckBoxen, ComboBoxen, Sliders, Spinners, Tables und TextFields. Ändert der Benutzer nun einen Wert, kann dieser entsprechend überprüft und anschliessend an die Konfiguration übermittelt werden.

8.9 Mehrsprachigkeit

Die komplette Benutzeroberfläche ist in Englisch und Deutsch verfügbar. Die Sprache lässt sich einfach über das Menü umstellen. Die Standardsprache ist Englisch. Einige Log Outputs bei der Wahlgenerierung sind nur in englischer Sprache verfügbar. Umgesetzt ist die Mehrsprachigkeit mittels ResourceBundle[22]. Dies ermöglicht eine einfache Änderung der verschiedenen Werte in einer Tabelle. Weitere Sprachen können ebenfalls einfach hinzugefügt werden.

[22] Vorhandene Klasse `java.util.ResourceBundle` verwendet.

9 Resultate

9.1 Erzielte Resultate

Mit dem VoteGenerator kann eine Wahl mit beliebiger Anzahl Mixer, Tallier und Wähler generiert werden. Die komplette Konfiguration kann komfortabel mittels des GUI durchgeführt werden. Die zentralen Kryptofunktionen wie beispielsweise die Primzahlenüberprüfung, die RSA Signatur, die Schnorr Signatur sowie alle ElGamal Funktionen konnten ohne fremde Libraries implementiert werden.

9.2 Primzahlentest

Folgende Primzahlentests wurden auf kleine Zahlenbereiche angewendet:[23]

Primzahlen bis	**Primzahlen**	**EMRT (in Sek.)**	**BIPP (in Sek)**
10	4	<0.01	<0.1
100	25	<0.01	<0.1
1‘000	168	<0.03	<0.1
10‘000	1‘229	<0.1	<0.2
100‘000	9‘592	0.1	0.2
1‘000‘000	78‘498	0.8	1.6
10‘000‘000	664‘579	7.2	15.9
100‘000‘000	5‘761‘455	87	164.9
1‘000‘000‘000	50‘847‘534	919.5	1805.5

Tabelle 13: Primzahlentest für kleine Zahlen

Der Vergleich mit der Referenztabelle [**28**] zeigt, dass die Anzahl korrekt gefundener Primzahlen bei 100% liegt.

[23] Getestet auf einem PC mit Windows 7, mit einem i7-2600K Prozessor und 16GB Ram

Für dem Miller-Rabin-Test wurde die Anzahl Werte für a auf 10 und für den BigInteger-Test auf 20 gesetzt: $1 - \frac{1}{4^{10}} = 1 - \frac{1}{2^{20}} \approx 99.9990\%$. Somit werden beide Tests mit gleichen Bedingungen durchgeführt.

Folgende Primzahlentests wurden auf grosse Zahlenbereiche angewendet:

Da nicht sämtliche Primzahlen in dieser Grössenordnung geprüft werden können, wurde nur ein kleiner Wertebereiche für 1024 und 2048-Bit grosse Zahlen gewählt:[24]

Primzahlen zwischen	**Primzahlen**	**EMRT (in Sek.)**	**BIPP (in Sek.)**
10^{308} und $10^{308} + 10'000$	8	8.9	18.3
10^{308} und $10^{308} + 100'000$	130	91.9	190.0
10^{616} und $10^{616} + 10'000$	7	50.1	106.3

Tabelle 14: Primzahlentest für grosse Zahlen

In der Praxis würde man nicht alle Zahlen in einem Bereich testen, sondern gewisse, wie gerade Zahlen, bereits im Vorfeld ausschliessen. So erzeugt man alle Primzahlen zwischen 10^{616} und $10^{616} + 10'000$ mit der BigInteger Funktion nextProbablePrime in 11 Sekunden.

9.3 Stimmzettelgenerierung

Die zufällige Verteilung der Stimmen führt bei der Konfiguration einer grossen Anzahl Wähler und einer kleinen Anzahl maximaler Stimmen zu einer annähernden Gleichverteilung.

In der Abbildung 10 wird das Wahlresultat für 1000 Wähler und maximal 15 Stimmen dargestellt.

[24] Getestet auf einem PC mit Windows 7, mit einem i7-2600K Prozessor und 16GB Ram.

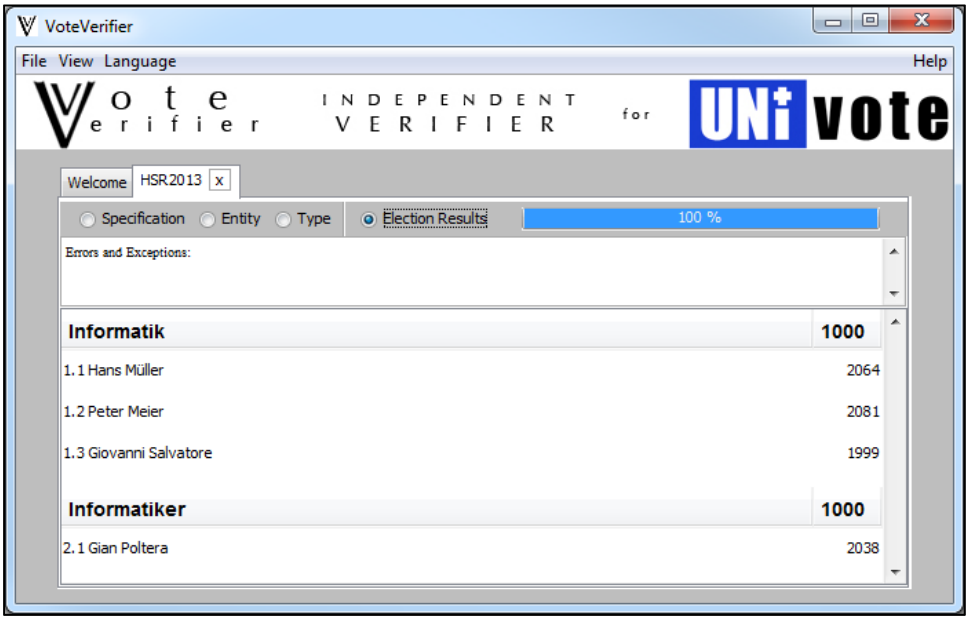


Abbildung 10: Stimmzettelgenerierung bei grosser Anzahl Wähler

9.4 Wahlgenerierung

Die Generierung einer kompletten Wahl dauert je nach Konfiguration unterschiedlich lange. Um die Dauer abschätzen zu können, wurden einige Messungen vorgenommen. In der Tabelle 15 sind die Resultate der Messungen dargestellt. Es wurde die Zeit für die Erstellung einer Wahl gemessen. Die Zeit für die Fehlerimplementation und der Speicherung in die Datenbank wurde dabei nicht berücksichtigt. Bei Generierungen, die weniger als 15 Minuten benötigten, wurden jeweils 5 Messungen durchgeführt und der Durchschnittswert ermittelt. Die Grösse des Speichers beschreibt nur das ElectionBoard, ohne den KeyStore.[25]

[25] Getestet auf einem PC mit Windows 7, mit einem i7-2600K Prozessor und 16GB Ram.

Key Grösse (in Bit)	Mixer	Tallier	Voter	Zeit (in Min.)	Speicher (in MB)
512	5	5	5	0.2	0.2
512	10	10	1‘000	5.0	12
512	5	5	10‘000	30.9	86
1024	5	5	5	0.5	0.2
1024	10	10	1‘000	48.1	16
1024	5	5	10‘000	213.4	103
2048	5	5	5	17.2	0.2
2048	10	10	1‘000		
4096	5	5	5	673.5	0.3

Tabelle 15: Wahlgenerierung Zeit und Speicher

9.5 Fehler Implementierung & Erkennung

Die Tabelle 16 bis Tabelle 19 zeigen die implementierten Fehler und die Erkennung des VoteVerifier‘s. Nicht implementiert (NI) bedeutet, dass der VoteVerifier die Überprüfung nicht durchführt oder nicht durchführen kann weil UniVote diese Parameter noch nicht liefert.

9.5.1 Parameter

Nr	Fehler	Erkannt	Bemerkung
1.	Schnorrs p ist keine Primzahl	Nein	VoteVerifier ladet die Werte aus eigenem Konfigurationsfile
2.	Schnorrs q ist keine Primzahl	Nein	
3.	Schnorrs g ist kein Generator	Nein	
4.	Schnorrs p ist kein SafePrime	Nein	
5.	Schnorrs Parameterlänge ist falsch	Nein	
6.	ElGamals p ist keine Primzahl	Ja	no prime
7.	ElGamals q ist keine Primzahl	Ja	no prime
8.	ElGamals g ist kein Generator	Ja	$g^q \neq 1 \bmod p$
9.	ElGamals p ist kein SafePrime	Ja	$p \neq k \cdot q + 1$
10.	ElGamals Parameterlänge ist falsch	Ja	$p, g \neq 1024\text{bit}$ $q \neq 256\text{bit}$
11.	Der *encryption key* ist falsch	Ja	$y \neq y_1 \cdot ... \cdot y_n \bmod p$

12.	Der *election generator* ist falsch	Ja	$\hat{g} \neq g_{lastMixer}$
13.	Zuviele *verification keys*	Ja	$vk' \neq vk_{lastMixer}$
14.	Zuviele *ballots*	Ja	Index out of bounds

Tabelle 16: Fehler Parameter

9.5.2 Zertifikate

Nr	Fehler	Erkannt	Bemerkung
15.	CA-Zertifikat falsch	Ja	Signature check failed
16.	EM-Zertifikat falsch	Ja	Signature check failed
17.	EA-Zertifikat falsch	Ja	Signature check failed
18.	Ein Tallier Zertifikat falsch	Ja	Signature check failed
19.	Ein Mixer Zertifikat falsch	Ja	Signature check failed
20.	Ein Wähler Zertifikat falsch	Ja	Signature check failed

Tabelle 17: Fehler Zertifikate

9.5.3 Signaturen

Nr	Fehler	Erkannt	Bemerkung
21.	Signierung des EA Zertifikates falsch	NI	
22.	Signierung der *election basic parameters* falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
23.	Signierung der Tallier und Mixer Zertifikate falsch	NI	
24.	Signierung der ElGamal Parameter falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
25.	Signierung des *encryption key* eines Talliers falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
26.	Signierung des *encryption keys* falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
27.	Signierung des *blinded generator* eines Mixers falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$

28.	Signierung des election generator falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
29.	Signierung der election options falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$
30.	Signierung der election data falsch	Ja	$signature^{pubExp}$ mod $modulus \neq sha256(data)$

Tabelle 18: Fehler Signaturen

9.5.4 NIZKP

Nr	Fehler	Erkannt	Bemerkung
31.	NIZKP eines Talliers für den *encryption key share* falsch	Ja	$v \neq w$
32.	NIZKP eines Mixers für den *blinded generator* falsch	Ja	$v \neq w$
X	Weitere NIZKP falsch	NI	Es werden nur die folgenden Fehler erkannt: • Size of the verification key set • Each verification key belongs G_q • There aren't duplicate keys in the set

Tabelle 19: Fehler NIZKP

Die Fehlererkennungsrate des VoteVerifiers liegt bei den aktuell implementierten Fehlern bei 84%.

10 Diskussion und Interpretation

10.1 Erzielte Resultate

Der Grossteil der gesetzten Ziele konnte erreicht werden. Eine Wahl kann mit der aktuellen Version des VoteGenerators komplett erstellt werden. Der Wahlablauf von UniVote wird dabei vollständig abgebildet. Die Konfiguration einer Wahl ist über das GUI möglich und die Resultate einer generierten Wahl werden in einer Datenbank abgelegt.

10.2 Primzahlentest

Der eigene Miller-Rabin-Test funktioniert einwandfrei. Der Geschwindigkeitsvorteil gegenüber dem BigInteger.isProbablePrime-Test ist überraschend gross. So benötigt der BIPP-Test rund doppelt so lange wie der eigene MR-Test. Die Annahme, dies sei auf die vorgegebenen Werte für a bei kleinen Zahlen, zurückzuführen, hat sich beim Testen von grossen Zahlen nicht bestätigt.

10.3 Stimmzettelgenerierung

Die zufällige Verteilung der Stimmen führt bei der Konfiguration einer grossen Anzahl Wähler und einer kleinen Anzahl maximaler Stimmen zu einer annähernden Gleichverteilung. Die Resultate einer Wahl sind somit nicht realitätsnah, da bei einer echten Wahl in der Regel einige Kandidaten erheblich mehr Stimmen als andere erhalten würden. Dies ist Rahmen dieser Arbeit jedoch nicht weiter von Bedeutung, da es keinen Einfluss auf die korrekte Generierung einer Wahl hat.

10.4 Wahlgenerierung

Mit dem VoteGenerator können theoretisch Wahlen beliebiger Grösse generiert werden. Da jedoch die komplette Berechnung, die normalerweise auf die verschiedenen Parteien aufgeteilt ist, von einer Maschine erledigt werden muss, dauert die Generierung bei einer grossen Anzahl Voter sehr lange. Ein weiteres Problem ist die Suche nach einem ElGamal SafePrime. Bei der Wahl der Schlüssellänge von lediglich 512 Bit kann eine komplette Wahl mit 10‘000 Wählern in nur 30 Minuten generiert werden. Die Schlüssellänge ist jedoch bereits heute nicht mehr als sicher anzusehen. Bei einer Schlüsselgrösse von 4096 Bit dauert alleine die Suche nach der ElGamal SafePrime ungefähr 10 Stunden. Da die Suche auf Zufall basiert, ist es jedoch möglich, bereits mit der ersten Primzahl eine SafePrime gefunden zu haben.

Die Wahl von Java als Programmiersprache ist für diese Rechenintensiven Aufgaben nicht ideal. Da der VoteGenerator in einer Sandbox läuft, können nicht alle vorhandenen Hardware Ressourcen genutzt werden. Für diese Aufgabe ist eine hardwarenahe Programmiersprache wie C++ besser geeignet.

10.5 Fehlerimplementierung und Erkennung

Es konnten einige Fehler erfolgreich implementiert werden. Die Fehlererkennungsrate des VoteVerifiers scheint mit 84% nicht sehr hoch zu sein. Dies lässt sich jedoch durch die Anpassung der Überprüfung der Schnorr Signatur-Parameter auf die effektiv übermittelten Werte und nicht durch die Vorgabe des eigenen Konfigurationsfile leicht ändern. Auf eine Implementierung von Fehlern während der Wahl wurde verzichtet, da bereits ein ungültiger Wert die komplette Wahl unbrauchbar macht. Dem UniVote Team war es wichtiger, dass die entsprechenden Tallier, Mixer oder Voter für die ein Fehler implementiert werden soll, ausgewählt werden können.

11 Fazit und Ausblick

Der Projektplan in Abbildung 11 (Anhang) konnte leider nicht vollständig eingehalten werden, da im August und September mehrere Wochenenden zusätzlich für die Projektarbeit aufgewendet werden mussten. Dies ist insbesondere auf den grösseren Zeitbedarf für die komplette Generierung des Wahlablaufs und die Erstellung des GUI zurückzuführen. Dafür musste bedeutend mehr Zeit als geplant investiert werden. Es konnten dadurch jedoch wichtige Erkenntnisse für zukünftige Projekte gesammelt werden.

Der VoteGenerator in seiner aktuellen Form ist für die Generierung von kompletten Wahlen geeignet. Da jedoch alle Berechnungen auf nur einer Maschine durchgeführt werden, dauert die Generierung von grossen Wahlen sehr lange. Das UniVote Team hat bereits die Entwicklung von UniVote 2 gestartet. Diese wird in einigen Punkten stark von der aktuellen Version abweichen. Daher wird es notwendig sein, auch den VoteGenerator entsprechend anzupassen.

12 Abkürzungsverzeichnis

AWT	Abstract Window Toolkit
BIPP	BigInteger isProbable Prime
CCC	Chaos Computer Club
EMRT	Eigener Miller-Rabin-Test
GUI	Graphical User Interface
IDE	Integrated Development Environment
NI	Nicht implementiert
NIZKP	Non-Interactive Zero-Knowledge Proof
NSA	National Security Agency
PEM	Privacy Enhanced Mail
RSA	Rivest, Shamir und Adleman
US	United States
WSDL	Web Services Description Language
XML	Extensible Markup Language

Tabelle 20: Abkürzungsverzeichnis

Wahlbeteiligte	
RA	Root Certificate Authority
CA	Certificate Authority
EM	Election Manager
EA	Election Administration
M_k	Mixer
T_j	Tallier
V_i	Voters
EB	Election Board
Diverses	
id	ElectionId
$descr$	ElectionDescription
C	Choices
R	Rules
H	Hash
NIZKP	
π	NIZKP Proof

t	Proof Commitment
s	Proof Response
Schnorr Signatur	
ℓ	Länge der Schnorr Keys
p	Schnorr Prime
q	Schnorr GroupOder
g	Schnorr Generator
sk	Signature Key
vk	Verification Key
$\hat{g}$	Election Generator
α_k	Zufallszahl des Mixers k für den BlindedGenerator
g_k	Blinded Generator des Mixers k
S	Signatur
a	Erster Wert der Signatur
b	Zweiter Wert der Signatur
ElGamal	
ℓ	Länge der ElGamal Keys
P	ElGamal Prime
Q	ElGamal GroupOrder
G	ElGamal Generator
x_j	Decryption Key des Talliers j
y_j	Decryption Key des Talliers j
y	Encryption Key
r_i	Zufallszahl für die Encryption des Voters i
E_i	Encryption des Voters i
a	Erster Wert der Encryption
b	Zweiter Wert der Encryption
$\mathcal{E}$	Mixed Encrypted Votes
RSA	
ℓ	Länge der RSA Keys
sk	Signature Key
vk	Verification Key

Tabelle 21: Abkürzungen Variablen

13 Abbildungsverzeichnis

14 Tabellenverzeichnis

15 Literaturverzeichnis

[1] S. Andrivet. (2013) Attaque de l'e-voting: Un cas concret. Video.

[2] R. Oppliger, "Addressing the Secure Platform Problem for Remote Internet Voting in Geneva," eSECURITY Technologies, 2002.

[3] A. Kerckhoffs, "La cryptographie militaire," *Journal des sciences militaires*, 1883.

[4] C. Zanetti and E. Guyer, "Parlamentarischen Initiative 258/2013 "Abschaffung der elektronischen Stimmabgabe"," 2013.

[5] Chaos Computer Club Zürich. (2013, Nov.) Offener Brief: Demokratie sichern – E-Voting-Einsatz beenden!. [Online]. https://www.ccczh.ch/News

[6] Schweizerische Bundeskanzlei. (2013,) Neue Bestimmungen für die elektronische Stimmabgabe. [Online]. http://www.bk.admin.ch/themen/pore/evoting/

[7] R. Haenni, "UniVote System Specification," Bern University of Applied Sciences, 2013.

[8] G. Scalzi and J. Springer, "VoteVerifier," Bern University of Applied Sciences, 2013.

[9] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *Information Theory, IEEE Transactions on*, 1985.

[10] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of cryptology*, pp. 161-174, 1991.

[11] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, pp. 120-126, 1978.

[12] D. Skoraszewsky, "Diskreter Logarithmus und das Diffie-Hellman-Verfahren," BTU Cottbus, 2001.

[13] M. O. Rabin, "Probabilistic algorithms," *Algorithms and Complexity 21*, 1976.

[14] M. O. Rabin, "Probabilistic algorithm for testing primality," *Journal of number theory*, pp. 128-138, 1980.

[15] T. P. Pedersen, "A threshold cryptosystem without a trusted party," *Advances in Cryptology—EUROCRYPT'91*, pp. 522-526, 1991.

[16] A. Shamir, "How to share a secret," *Communications of the ACM*, pp. 612-613, 1979.

[17] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Advances in Cryptology—EUROCRYPT'99*, pp. 295-310, 1999.

[18] A. Pfitzmann, B. Pfitzmann, and M. Waidner, "Schutz der Vermittlungsdaten für zwei 64-kbit/s-Duplexkanäle über den $(2 \bullet 64 + 16)$-kbit/s-Teilnehmeranschluß," *Datenschutz und Datensicherung*, 1989.

[19] K. Sako and J. Kilian, "Receipt-free mix-type voting scheme," *Advances in Cryptology—EUROCRYPT'95*, pp. 393-403, 1995.

[20] M. Michels and P. Horster, "Some remarks on a receipt-free and universally verifiable mix-type voting scheme," *Advances in Cryptology—ASIACRYPT'96*, pp. 125-132, 1996.

[21] J. Müller, "Anonyme Signalisierung in Kommunikationsnetzen," TU Dresden, Institut für Theoretische Informatik, 1997.

[22] A. Masayuki, "Universally verifiable mix-net with verification work independent of the number of mix-servers," *Advances in Cryptology—EUROCRYPT'98*, pp. 437-447, 1998.

[23] M. Jakobsson, A. Juels, and R. L. Rivest, "Making Mix Nets Robust For Electronic Voting By Randomized Partial Checking," 2002.

[24] K. Hupf and A. Meletiadou, "End-to-End verifizierbare Wahlverfahren in Hinblick auf den Grundsatz der Öffentlichkeit der Wahl," Universität Koblenz-Landau, 2009.

[25] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. (2001, Mar.) Web Services Description Language (WSDL) 1.1. [Online]. http://www.w3.org/TR/wsdl.html

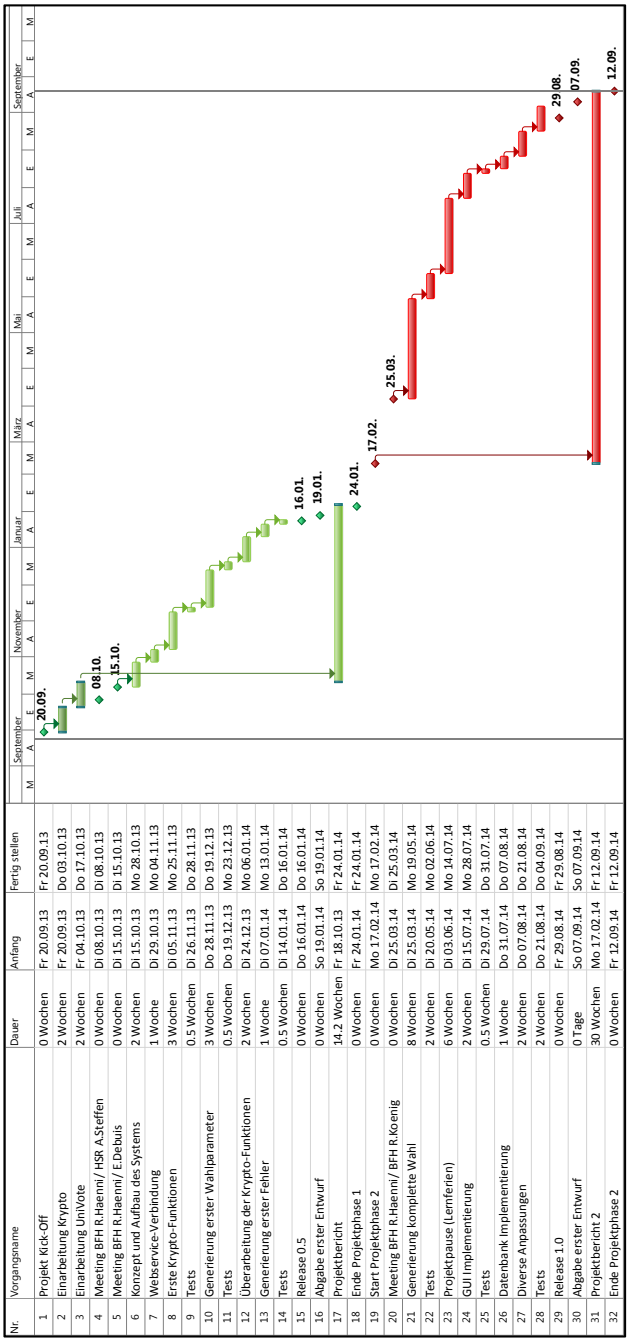
[26] C. Pomerance, J. L. Selfridge, and S. S. Wagstaff, "The pseudoprimes to $25 \cdot 10^9$," *Mathematics of Computation*, pp. 1003-1026, 1980.

[27] G. Jaeschke, "On strong pseudoprimes to several bases," *Mathematics of Computation*, pp. 915-926, 1993.

[28] J. Bernheiden, "Wie viele Primzahlen gibt es?".

ANHANG 1

Projektplan



Nr.	Vorgangsname	Dauer	Anfang	Fertig stellen
1	Projekt Kick-Off	0 Wochen	Fr 20.09.13	Fr 20.09.13
2	Einarbeitung Krypto	2 Wochen	Fr 20.09.13	Do 03.10.13
3	Einarbeitung UniVote	2 Wochen	Fr 04.10.13	Do 17.10.13
4	Meeting BFH R.Haenni/ HSR A.Steffen	0 Wochen	Di 08.10.13	Di 08.10.13
5	Meeting BFH R.Haenni/ E.Debuis	0 Wochen	Di 15.10.13	Di 15.10.13
6	Konzept und Aufbau des Systems	2 Wochen	Di 15.10.13	Mo 28.10.13
7	Webservice-Verbindung	1 Woche	Di 29.10.13	Mo 04.11.13
8	Erste Krypto-Funktionen	3 Wochen	Di 05.11.13	Mo 25.11.13
9	Tests	0.5 Wochen	Di 26.11.13	Do 28.11.13
10	Generierung erster Wahlparameter	3 Wochen	Do 28.11.13	Do 19.12.13
11	Tests	0.5 Wochen	Do 19.12.13	Mo 23.12.13
12	Überarbeitung der Krypto-Funktionen	2 Wochen	Di 24.12.13	Mo 06.01.14
13	Generierung erster Fehler	1 Woche	Di 07.01.14	Mo 13.01.14
14	Tests	0.5 Wochen	Di 14.01.14	Do 16.01.14
15	Release 0.5	0 Wochen	Do 16.01.14	Do 16.01.14
16	Abgabe erster Entwurf	0 Wochen	So 19.01.14	So 19.01.14
17	Projektbericht	14.2 Wochen	Fr 18.10.13	Fr 24.01.14
18	Ende Projektphase 1	0 Wochen	Fr 24.01.14	Fr 24.01.14
19	Start Projektphase 2	0 Wochen	Mo 17.02.14	Mo 17.02.14
20	Meeting BFH R.Haenni/ BFH R.Koenig	0 Wochen	Di 25.03.14	Di 25.03.14
21	Generierung komplette Wahl	8 Wochen	Di 25.03.14	Mo 19.05.14
22	Tests	2 Wochen	Di 20.05.14	Mo 02.06.14
23	Projektpause (Lernferien)	6 Wochen	Di 03.06.14	Mo 14.07.14
24	GUI Implementierung	2 Wochen	Di 15.07.14	Mo 28.07.14
25	Tests	0.5 Wochen	Di 29.07.14	Do 31.07.14
26	Datenbank Implementierung	1 Woche	Do 31.07.14	Do 07.08.14
27	Diverse Anpassungen	2 Wochen	Do 07.08.14	Do 21.08.14
28	Tests	2 Wochen	Do 21.08.14	Do 04.09.14
29	Release 1.0	0 Wochen	Fr 29.08.14	Fr 29.08.14
30	Abgabe erster Entwurf	0 Tage	So 07.09.14	So 07.09.14
31	Projektbericht 2	30 Wochen	Mo 17.02.14	Fr 12.09.14
32	Ende Projektphase 2	0 Wochen	Fr 12.09.14	Fr 12.09.14

Abbildung 11: Projektplan

ANHANG 2

Systemvoraussetzungen

Hardware	Je mehr Leistung, desto schneller werden die Wahlen generiert. Genügend Speicherplatz für die Generierung von grossen Wahlen.
Software	Java Version 7 oder höher

Der VoteGenerator wurde erfolgreich unter folgenden Umgebungen getestet:

- Windows 7 Professional, 64 Bit, ServicePack 1
- Windows 8.1 Pro, 64 Bit (2 Maschinen)

Bedienungsanleitung

1. Startbildschirm:
   a) Neue Wahl konfigurieren
   b) Neue Wahl mit vorhandener Konfiguration generieren
   c) Bereits generierte Wahl laden

a) Neue Wahl generieren:
   1. Basisparameter
   2. Mixer/Tallier
   3. Kandidaten
   4. Krypto
   5. Fehlereinbau
   6. Wahl generieren

b) Neue Wahl mit vorhandener Konfiguration generieren

c) Bereits generierte Wahl laden:

Wechsel der Sprache:

Speichern/Laden der generierten Wahlen und Einstellungen:

Richtzeiten zur Wahl der Schlüsselgrösse und der Anzahl Wähler

Je nach eingestellter Schlüsselgrösse kann das Finden einer SafePrime sehr lange dauern. Folgende Richttabelle kann als Herleitung verwendet werden:

Schlüsselgrösse:	**Benötigte Zeit:**
1024 Bit	20 Sekunden
2048 Bit	12 Minuten
4096 Bit	10 Stunden

Je nach Anzahl Wähler kann das Generieren der Wahl sehr lange dauern. Folgende Richttabelle kann als Herleitung verwendet werden:

Anzahl Wähler:	**Benötigte Zeit:**
5	Zeit Schlüsselgrösse x 1.2
1‘000	Zeit Schlüsselgrösse x 5
10‘000	Zeit Schlüsselgrösse x 100

Häufige Fehler

Port 8080 wird bereits verwendet.

Bitte überprüfen Sie, dass kein anderes Programm/Dienst auf Ihrem Computer diesen Port verwendet. Die Wahl kann zwar generiert werden, der Webservice wird jedoch nicht korrekt gestartet und es ist keine Überprüfung der Wahl mittels VoteVerifier möglich.