

Institut für Informationssicherheit

Universität Stuttgart  
Universitätsstraße 38  
D-70569 Stuttgart

Bachelorarbeit

# **Umsetzung vorhandener Wahlssysteme mit einem E-Voting-System**

Jonas Kittelberger

**Studiengang:** Informatik

**Prüfer/in:** Prof. Dr. Ralf Küsters

**Betreuer/in:** Julian Liedtke, M.Sc.

**Beginn am:** 19. August 2019

**Beendet am:** 19. Februar 2020



## Kurzfassung

„Ordinos“ ist ein vielfältig einsetzbares E-Voting-System. Eine Besonderheit des Systems besteht in der Möglichkeit, die Gesamtzahl erhaltener Stimmen pro Kandidat geheim zu halten („Tally-Hiding“) und trotzdem verifizierbare Informationen über das Ergebnis, z.B. das Ranking der Kandidaten oder nur den Sieger der Wahl, preiszugeben.

Dazu werden Operatoren verwendet, die auf verschlüsselten Daten arbeiten und als modulare Bausteine genutzt werden können, um eine Vielzahl an Protokollen zur Auswertung des Wahlergebnisses zu ermöglichen.

Die Umsetzung bestehender Wahlsysteme mit Ordinos ist keine triviale Aufgabe und erfordert den systematischen Entwurf eines Protokolls, welches das Ergebnis effizient und korrekt berechnet und die geheimzuhaltenden Informationen nicht entschlüsselt.

Im Rahmen der Bachelorarbeit wird auf zwei bestehende Wahlsysteme näher eingegangen.

Dazu zählt die allgemeine Form der Parlamentswahl mit Erst- und Zweitstimme, wobei insbesondere auch die Wahl des deutschen Bundestages mit einbezogen wird.

Als weiteres Wahlsystem wird das „Instant-Runoff-Voting“ betrachtet, wozu der Wähler eine geordnete Präferenzliste der Kandidaten als Stimme abgibt. Beim Instant-Runoff-Voting ist, im Gegensatz zur üblichen Wahl mit nur einer Option, nicht nur die Erstpräferenz des Wählers ausschlaggebend für das Wahlergebnis.

Die entsprechenden Wahlsysteme werden in verschiedenen Varianten mit Ordinos umgesetzt. Dazu wird analysiert, wie sich die jeweilige Wahl mit Ordinos realisieren lässt und es wird ein Konzept entworfen, welches die Realisierung der Wahl detailliert veranschaulicht. Anhand des aufgestellten Konzepts werden die zur Durchführung der Wahl nötigen Erweiterungen des bereits existierenden Python-Codes implementiert.

Mithilfe des Konzepts und der Implementierung des Wahlsystems wird beurteilt, wie sich die Laufzeit für verschiedene Szenarien verhält und wie geeignet das jeweilige Verfahren zur Durchführung der Wahl ist. Desweiteren wird auf Vor- und Nachteile eingegangen, die sich bei anderen Umsetzungen bzw. ohne Verwendung von Ordinos ergeben. Außerdem wird analysiert, inwiefern die Umsetzung die geforderten Sicherheitskriterien und insbesondere die Geheimhaltung bestimmter Informationen erfüllt.



# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>11</b>
<b>2</b>	<b>Grundbegriffe</b>	<b>13</b>
2.1	Klassifikation von Wahlsystemen . . . . .	13
2.2	Teilnehmer einer mit Ordinos durchgeführten Wahl . . . . .	13
2.3	Definitionen . . . . .	14
<b>3</b>	<b>Allgemeine Parlamentswahl</b>	<b>19</b>
3.1	Einordnung . . . . .	19
3.2	Überblick über die Umsetzung . . . . .	20
3.3	Motivation zum Einsatz von Tally-Hiding . . . . .	21
3.4	Hare-Niemeyer-Verfahren . . . . .	23
3.5	Sitzberechnung mit verschlüsselten Daten . . . . .	24
3.6	Laufzeitanalyse . . . . .	27
3.7	Testergebnisse . . . . .	28
3.8	Sicherheit . . . . .	32
3.9	Geheime vs. Öffentliche Variante . . . . .	32
<b>4</b>	<b>Instant-Runoff-Voting</b>	<b>35</b>
4.1	Beschreibung des Verfahrens . . . . .	35
4.2	Verfahren mit öffentlicher Elimination . . . . .	37
4.3	Verfahren mit geheimer Elimination . . . . .	39
4.4	Beispielhafte Durchführung der Verfahren . . . . .	41
4.5	Vorgehen bei Stimmgleichheit . . . . .	46
4.6	Alternatives Verfahren: Stimme mit $n!$ Optionen . . . . .	47
4.7	Laufzeitanalyse . . . . .	49
4.8	Testergebnisse . . . . .	51
4.9	Sicherheit . . . . .	54
<b>5</b>	<b>Zusammenfassung und Ausblick</b>	<b>55</b>
	<b>Literaturverzeichnis</b>	<b>57</b>



# Abbildungsverzeichnis

2.1	Überblick über die Teilnehmer einer Wahl mit Ordinos . . . . .	14
3.1	Öffentliche Variante, 32 bit Integer . . . . .	29
3.2	Geheime Variante, 32 bit Integer . . . . .	29
3.3	Öffentliche Variante, 64 bit Integer . . . . .	30
3.4	Geheime Variante, 64 bit Integer . . . . .	30
3.5	Andere Darstellung der Szenarien aus Abbildung 3.3b . . . . .	31
3.6	Andere Darstellung der Szenarien aus Abbildung 3.4b . . . . .	31
4.1	Variierende Anzahl an Wählern, 64-bit-Schlüssel . . . . .	51
4.2	Variierende Anzahl an Kandidaten, 64-bit-Schlüssel . . . . .	51
4.3	Variierende Anzahl an Wählern, 2048-bit-Schlüssel . . . . .	52
4.4	Öffentliches alternatives Verfahren im Vergleich, 2048-bit-Schlüssel . . . . .	53
4.5	Geheimes alternatives Verfahren im Vergleich, 2048-bit-Schlüssel . . . . .	53
4.6	Variierende Anzahl an Wählern, 2048-bit-Schlüssel, inkl. alternative Verfahren . . . . .	54





## Verzeichnis der Algorithmen

2.1	Summation der Stimmen . . . . .	16
2.2	Matrix-zu-Ranking . . . . .	17
4.1	Hochrutschen . . . . .	37
4.2	Berechnung des Zählers . . . . .	38
4.3	Berechnung der neuen Erstpräferenz mithilfe des Zählers . . . . .	39
4.4	Teilschritt 2 (Variante mit Matrixmanipulation, geheime Elimination) . . . . .	40
4.5	Berechnung des Eliminationsvektors . . . . .	41
4.6	Instant-Runoff Voting (IRV) in $O(n!)$ . . . . .	48



# 1 Einleitung

„E-Voting“, d.h. die elektronische Stimmabgabe bei einer Wahl oder Abstimmung, wird seit Ende der 90er Jahre als Alternative zum klassischen „Urnengang“ diskutiert [VK06]. Während anfangs der Einsatz elektronischer Wahlgeräte in den Wahllokalen im Mittelpunkt stand, geht es heutzutage (so auch in dieser Arbeit) um die Möglichkeit der Online-Wahl, bei welcher der Wähler seine Stimme über das Internet zum Server überträgt.

Die wesentlichen Forderungen an ein E-Voting-System sind die Geheimhaltung der persönlichen Stimme, sowie die Verifizierbarkeit, d.h. dass das berechnete Gesamtergebnis der Summe der abgegebenen Stimmen entspricht. Eine Vielzahl an modernen E-Voting-Systemen erfüllen diese Forderungen auch im Falle korrupter Parteien und Angreifer auf das System.

Allerdings ergeben sich durch „E-Voting“ auch neue Möglichkeiten, die bei einem klassischen Urnengang bisher unmöglich waren. Wir nehmen eine Klassensprecherwahl an, bei welcher mehrere Kandidaten gegeneinander antreten. Ohne Einsatz elektronischer Hilfsmittel, d.h. wenn jeder Schüler seine Stimme auf einem gefalteten Papier notiert und abgibt, gibt es zwei Möglichkeiten:

Einerseits die geläufige Variante, dass die Zettel vor der Klasse ausgezählt werden. Wenn dabei ein Schüler gar keine oder nur seine eigene Stimme erhält, dann führt dies zu einer Blamage des Schülers. Obwohl nur die beiden Personen mit den meisten Stimmen gesucht waren, werden somit nebenbei auch die Verlierer mit den wenigsten Stimmen ermittelt.

Um dies zu verhindern, wäre die zweite Möglichkeit die unbeobachtete Auswertung der beiden Klassensprecher vom Lehrer oder einer Gruppe von Lehrern. Allerdings kann nun kein Schüler die Korrektheit des Ergebnisses überprüfen. Lehrer könnten beschuldigt werden, ihre beiden Favoriten ausgewählt zu haben. Sicherlich würde eine Vergrößerung der auswertenden Gruppe und eine Hinzunahme von verschiedenen Eltern das Vertrauen der Schüler in das Ergebnis erhöhen. Allerdings reicht ein Elternteil, welches das genaue Resultat der Wahl nicht für sich behalten kann, und das ganze Verfahren war nutzlos. Desweiteren könnten falsche Gerüchte über die Verlierer der Wahl entstehen, die keiner der Schüler widerlegen kann.

Diese Probleme können durch Verwendung des E-Voting-Systems „Ordinos“ [KLM+19] gelöst werden. Das System beruht auf der zweiten Möglichkeit der Klassensprecherwahl, denn auch hier führt eine heterogene Gruppe, sogenannte „Trustees“, die Auswertung durch. Die Rolle einer Trustee kann von einer beliebigen Person, inklusive eines an der Wahl teilnehmenden Schülers, übernommen werden. Der wesentliche Unterschied bei Verwendung von Ordinos besteht darin, dass bei der Auswertung keine Partei, inklusive der Trustees, das genaue Ergebnis erfährt. Trotzdem kann jede Person den Vorgang der Auswertung verifizieren und jeder Wähler kann sicherstellen, dass seine Stimme korrekt im Ergebnis repräsentiert wird.

Allerdings muss bei Verwendung von Ordinos auch beachtet werden, dass ein Bündnis aller Trustees unbeschränkte Möglichkeiten zur Entschlüsselung einzelner Wählerstimmen und Zuordnung der Stimmen zu den Wählern bietet. „Ordinos“ beruht daher auf der Annahme, dass mindestens eine der

Trustees „ehrlich“ ist und nicht die Entschlüsselung geheimzuhaltender Informationen unterstützt. Zur Anwendung von „Ordinos“ ist deshalb eine gezielte Auswahl verteilter Trustees nötig, sodass jeder Wähler in die Ehrlichkeit mindestens einer Trustee vertraut.

Dieser scheinbare „Vorteil“ der nicht-elektronischen und geheimen Auszählung wird jedoch dadurch revidiert, dass mit einem Bündnis aller auswertenden Personen eine beliebige Manipulation des Wahlergebnisses möglich wird.

Die Arbeit ist in folgender Weise gegliedert:

In Kapitel 2 werden die in dieser Arbeit verwendeten Grundbegriffe definiert und beschrieben. In den beiden anschließenden Kapiteln werden zwei ausgewählte Wahlsysteme jeweils detailliert und in verschiedenen Varianten geschildert.

Dazu zählt eine verallgemeinerte Form der Parlamentswahl in Kapitel 3, mit der sich eine Vielzahl derzeit abgehaltener Wahlen, wie z.B. die Bundestagswahl, durchführen lassen. Im Fokus steht der Einsatz von Tally-Hiding, wobei die Sitzverteilung des Parlaments errechnet werden soll, ohne dass dabei die exakte Stimmanzahl pro Partei bekannt gegeben wird.

Es folgt das „Instant-Runoff“-Wahlsystem in Kapitel 4. Die Besonderheit des Systems besteht darin, dass der Wähler ein Ranking unter allen Kandidaten erstellt. Daher kann, im Gegensatz zu Wahlsystemen mit einer Option pro Wähler, auch die Platzierung nicht favorisierter Kandidaten entscheidenden Einfluss auf das Resultat haben. Das Instant-Runoff-Voting wird vor allem bei Wahlen in kleineren Gruppen, z.B. bei Präsidentschaftswahlen durch die Abgeordneten, durchgeführt. Mithilfe von Tally-Hiding soll auch bei diesem Wahlsystem die genaue Anzahl an Stimmen pro Kandidat für jedermann geheim gehalten werden.

Für beide Wahlsysteme werden die Vor- und Nachteile der Anwendung von „Tally-Hiding“ diskutiert und abgewägt. Die Wahlsysteme wurden in verschiedenen Varianten implementiert und auf ihre Effizienz hin untersucht. Die jeweiligen Umsetzungen werden mit Erklärungen und Pseudocode veranschaulicht. Außerdem werden die Laufzeiten analysiert und die Testergebnisse werden abgebildet und diskutiert.

Schließlich fasst Kapitel 5 die Ergebnisse der Arbeit zusammen und stellt Anknüpfungspunkte vor.

## 2 Grundbegriffe

In diesem Kapitel werden die Grundbegriffe definiert und die Grundlagen erklärt, welche zum Verständnis der Wahlsysteme und deren Umsetzung mit Ordinos benötigt werden.

### 2.1 Klassifikation von Wahlsystemen

Die Einteilung in Mehrheits- und Verhältniswahl ist eine zentrale Vorgehensweise zur Klassifizierung von Wahlen. Bei der Verhältniswahl bilden die gewählten Kandidaten ein Abbild der Wählerschaft. Bei der Mehrheitswahl setzen sich überlegene Kandidaten durch („Die Mehrheit siegt“).

In der Realität existieren Wahlsysteme mit Mehrheits- und Verhältniswahlcharakter, wobei häufig eine Kombination dieser beiden grundsätzlich gegensätzlicher Ansätze zu beobachten ist. Beispielsweise repräsentiert die Anzahl der Sitze pro Partei im Bundestag das Verhältnis der Zweitstimmen (mit Einschränkung der Sperrklausel), allerdings wird in jedem Wahlkreis ein Abgeordneter per Mehrheitswahl über die Erststimme bestimmt.

Bei Entscheidungen mit Mehrheitswahl ist eine Gliederung in „absolute“ und „relative“ Mehrheitswahl üblich. Bei der relativen Mehrheitswahl, z.B. bei Wahl des Abgeordneten mit der Erststimme, gewinnt der Kandidat mit den meisten Stimmen. Dagegen erfordert eine absolute Mehrheitswahl das Erreichen von mindestens der Hälfte der Stimmen. Dazu werden Stichwahlen oder andere Maßnahmen benötigt, z.B. das in Kapitel 4 umgesetzte „Instant-Runoff“-Wahlverfahren.

### 2.2 Teilnehmer einer mit Ordinos durchgeführten Wahl

Analog zur nicht elektronisch durchgeführten Wahl werden auch für eine Durchführung mit Ordinos neben dem Wähler einige Teilnehmer benötigt, um einen geordneten Ablauf zu gewährleisten. Abbildung 2.1 stellt die Teilnehmer einer mit Ordinos durchgeführten Wahl und ihre wesentlichen Aufgaben dar. Für weitere Details und die genaue Umsetzung einzelner Bestandteile sei auf [KLM+19] verwiesen.

Insbesondere ist auf eine einfache Bedienbarkeit zu achten, sodass auch technisch unerfahrene Wähler problemlos ihre Stimme abgeben können. Außerdem ist es nötig, die Identität des Wählers zu prüfen. Dazu könnten über ein einfaches Passwort hinausgehende Maßnahmen, wie z.B. die Gesichtserkennung oder das Prüfen des Fingerabdrucks, sinnvoll sein.

Im weiteren Verlauf dieser Arbeit wird der Fokus auf die Berechnung des Resultats gelegt. Wir nehmen an, dass die Trustees eine Liste valider, verschlüsselter Wählerstimmen erhalten. Auf den verschlüsselten Daten soll gezielt eine Folge von Operationen durchgeführt werden, um den

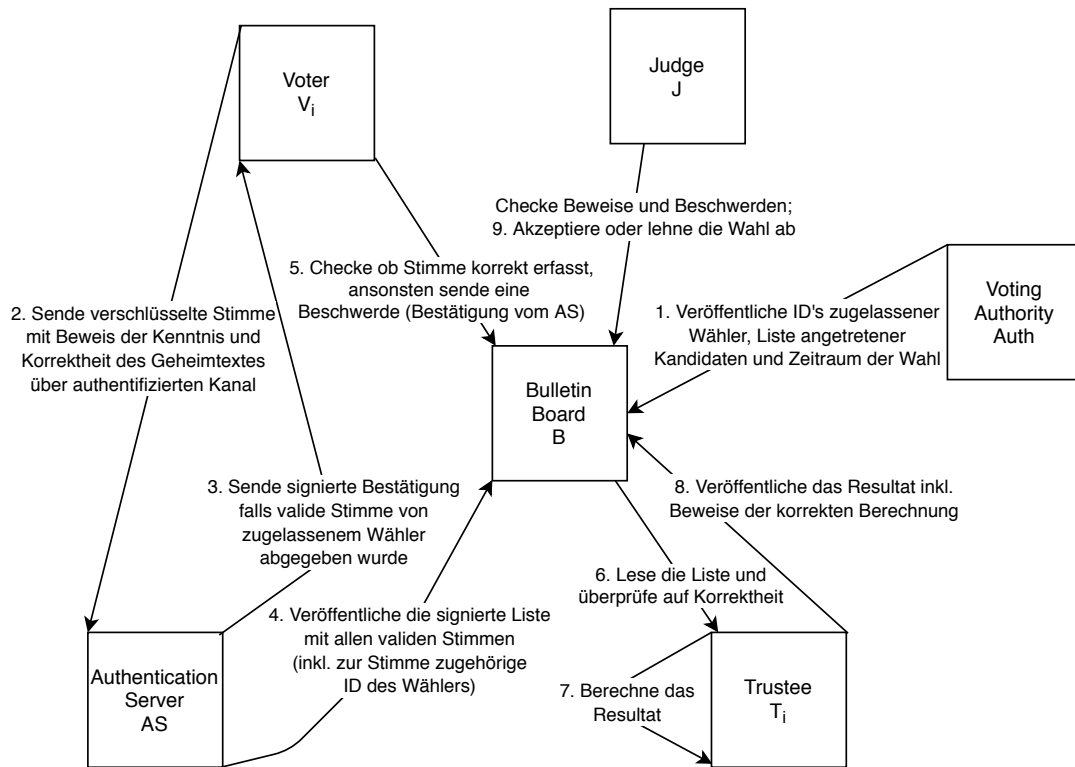


Abbildung 2.1: Überblick über die Teilnehmer einer Wahl mit Ordinos

öffentlichen Teil der Informationen über das Wahlergebnis zu berechnen. Dabei soll das Herleiten weiterer Informationen über einzelne Stimmen und den Ausgang der Wahl auch für die Trustees nicht möglich sein.

## 2.3 Definitionen

Zur Ver- und Entschlüsselung von Daten bei einer mit Ordinos durchgeführten Wahl wird ein „threshold public-key encryption scheme“ verwendet. Für Details sei auf [KLM+19] verwiesen.

### Definition 1 (Verschlüsselung)

Mit  $enc(x, r)$  bezeichnen wir die Verschlüsselung eines Klartexts  $x$  mit einem Wert  $r$  und nennen  $y = enc(x, r)$  einen „Geheimtext“. Wir verzichten auf die Angabe des Schlüssels, da Verschlüsselungen stets mit einem festen und öffentlich bekannten Schlüssel durchgeführt werden.

Bei der Verschlüsselung von Wählerstimmen muss  $r$  stets zufällig gewählt werden, damit der Klartext  $x$  geheim gehalten wird. Wir bezeichnen diesen Vorgang als „probabilistische“ Verschlüsselung. Für Verschlüsselungen, die bei Berechnung des Resultats durchgeführt werden, muss  $r$  unter allen Trustees einheitlich gewählt werden. Allerdings ist es für den weiteren Verlauf irrelevant, welches  $r$  ausgewählt wurde. Zur Vereinfachung notieren wir deshalb  $Enc(x)$  für einen Geheimtext  $y$ , falls gilt:  $\exists r : y = enc(x, r)$ . Wir bezeichnen  $Enc(x)$  auch als „verschlüsselten Wert von  $x$ “ bzw. „verschlüsseltes  $x$ “.

Zu beachten: Wir bezeichnen zwei Geheimtexte immer dann als „gleich“, wenn sie den gleichen Wert verschlüsseln. D.h. für Geheimtexte  $y_1, y_2$  sei  $y_1 = y_2 \Leftrightarrow \exists x, r_1, r_2 : y_1 = enc(x, r_1) \wedge y_2 = enc(x, r_2)$ . Trotzdem darf es nicht möglich sein, allein anhand des öffentlichen Schlüssels auf Gleich- oder Ungleichheit zweier Geheimtexte  $y_1$  und  $y_2$  zu schließen, falls  $y_1$  oder  $y_2$  probabilistisch verschlüsselt wurde. Andernfalls wäre die Geheimhaltung der persönlichen Wählerstimme nicht gewährleistet.

### Definition 2 (Entschlüsselung)

Mit *Dec* bezeichnen wir den deterministischen Algorithmus, der für einen Geheimtext  $y$  den Klartext  $x$  mit  $Enc(x) = y$  zurückgibt.

Zur Entschlüsselung von Geheimtexten werden mehrere Teile des privaten Schlüssels benötigt. Der Besitzer eines privaten Schlüsselteils wird „Trustee“ genannt. Zur Entschlüsselung ist mindestens ein vom Kryptosystem vorgegebener Schwellwert „threshold“ an kooperierenden Trustees nötig.

### Definition 3 (homomorphe Operatoren)

Für Ordinos benötigen wir ein „homomorphes“ threshold public-key encryption scheme, d.h. es gilt für bestimmte Operatoren  $\otimes$ :  $Enc(x \otimes y) = Enc(x) \odot Enc(y)$  mit öffentlich berechenbarer Operation  $\odot$ . Wir fordern insbesondere Homomorphie unter den Operatoren  $\otimes \in \{+, -\}$ .

Zur Vereinfachung bezeichnen wir im folgenden den Operator auf den Geheimtexten  $\odot$  gleich wie die dadurch durchgeführte Operation auf den Klartexten  $\otimes$ . Formal:  $Enc(x) \otimes Enc(y) := Enc(x \otimes y)$

Damit lässt sich aus zwei Geheimtexten die verschlüsselte Summe und Differenz deren Klartexte berechnen, ohne dass dafür die Kenntnis von Teilen des privaten Schlüssels nötig ist. In unserer Implementierung wird das „Paillier-Kryptosystem“ verwendet. Eine Multiplikation zweier Geheimtexte ist dabei ausreichend, um die verschlüsselte Summe deren Klartexte zu berechnen. Daraus folgt, dass sich eine Multiplikation eines Geheimtextes mit einer Konstanten durch eine Exponentiation realisieren lässt.

### Definition 4 (Berechnungsprotokolle)

Für zwei Geheimtexte  $c_1, c_2$  benötigen wir Protokolle zur Umsetzung der Vergleichsoperatoren  $\otimes \in \{\leq, =\}$ .

Das berechnete Resultat ist 
$$\begin{cases} Enc(1) & \text{für } c_1 \otimes c_2 \\ Enc(0) & \text{sonst} \end{cases}.$$

Erläuterungen zur Umsetzung dieser Operationen finden sich in [LT13]. Im Gegensatz zu den Operatoren aus Definition 3 werden die Trustees zur Durchführung benötigt, da das zugrundeliegende Berechnungsprotokoll die Entschlüsselungsoperation (*Dec*) verwendet.

Das gilt auch für die Anwendung des Multiplikationsoperators  $\otimes \in \{\cdot\}$  auf  $c_1$  und  $c_2$ .

Es kann unter bestimmten Voraussetzungen gezeigt werden, dass bei Ablauf der zugrundeliegenden Berechnungsprotokolle keinerlei Informationen über die Klartexte  $Dec(c_1)$  und  $Dec(c_2)$  bekannt werden.

Die Verwendung dieser Berechnungsprotokolle unterscheidet Ordinos von vielen anderen E-Voting Systemen, sie sind der Grundbaustein zur Umsetzung von „Tally-Hiding“.

### Definition 5 (Kommunikation der Trustees)

Bei der Berechnung des Resultats der Wahl durch die Trustees unterscheiden wir zwischen „lokaler“ und „globaler“ Phase.

In der lokalen Phase werden Berechnungen mit homomorphen Operatoren, nicht probabilistisch durchgeführte Verschlüsselungen, oder andere Anweisungen von den Trustees durchgeführt. Entscheidend ist, dass der Teil des privaten Schlüssels für keine der durchgeführten Anweisungen

benötigt wird. Insbesondere erfordert diese Phase keine Kommunikation zwischen den Trustees und kann von jeder Trustee (oder auch „Nicht-Trustee“) isoliert durchgeführt werden.

In der globalen Phase müssen die Trustees kommunizieren, um Dec-Operationen durchzuführen. Bei einer Dec-Operation müssen die Trustees den mit ihrem Teil des privaten Schlüssels entschlüsselten Geheimtext („Dec-Share“) und einen zugehörigen Beweis zur korrekten Berechnung („Zero-Knowledge-Proof“) an die anderen Trustees übermitteln. Eine Trustee kann erst dann das Ergebnis der Dec-Operation berechnen, wenn sie die durch den threshold vorgegebene Anzahl verschiedener Dec-Shares von anderen Trustees erhalten hat.

Für die Laufzeit ist insbesondere die globale Phase entscheidend. Daher wird im weiteren Verlauf bei einer „Kostenabschätzung“ im Wesentlichen die Anzahl durchzuführender  $\leq / = /$  Multiplikationsoperationen zwischen Geheimtexten berücksichtigt.

**Definition 6 (Summierbare Stimme)**

Sei  $n$  die Anzahl an Kandidaten, die an der Wahl teilnehmen, wobei jedem Kandidaten eine eigene ID  $i$  mit  $0 < i \leq n$  zugeordnet ist. Wir bezeichnen ein  $n$ -Tupel mit  $n$  verschlüsselten Einträgen als eine „summierbare Stimme“. Der  $i$ -te Eintrag des Tupels verschlüsselt die Anzahl an Stimmen, die dem Kandidaten mit ID  $i$  zugeordnet werden.

Mit „Zero-Knowledge-Proofs“ kann nachgewiesen werden, dass das Tupel eine valide Stimme darstellt und z.B. genau eine vorgegebene Anzahl an Stimmen auf die Kandidaten verteilt wurde [KLM+19]

Für den Spezialfall, dass für genau einen Kandidaten gestimmt werden darf, sprechen wir von „Summierbare Stimme  $s$  mit  $n$  Optionen“ mit Kurzschreibweise  $s \in \{1, \dots, n\}$  für die ID des „gewählten“ Kandidaten.

Für eine Menge  $S$  an summierbaren Stimmen lässt sich die verschlüsselte Summe an Stimmen pro Kandidat (ohne Kommunikation der Trustees!) berechnen, siehe Algorithmus 2.1.

---

**Algorithmus 2.1** Summation der Stimmen

---

```

function COMPUTE_SUM(Set S)
  votes[i]  $\leftarrow$  Enc(0) for all  $i \in \{1, \dots, n\}$ 
  for all  $s \in S$  do
    for all  $i \in \{1, \dots, n\}$  do
      votes[i]  $\leftarrow$  votes[i] +  $s_i$            //  $s_i$  ist  $i$ -ter verschlüsselter Eintrag der Stimme  $s$ 
    end for
  end for
  return votes
end function

```

---

**Definition 7 (Rangliste)**

Eine Rangliste ist eine Stimme, welche eine Sortierung der  $n$  Kandidaten darstellt, d.h. jedem der Kandidaten wird eine paarweise verschiedene „Platzierung“ aus der Menge  $\{1, \dots, n\}$  zugewiesen. Für eine Rangliste  $r$  und einen Kandidaten mit ID  $i$  notieren wir die Platzierung des Kandidaten mittels  $p(r, i)$ .

Um die Rangliste darzustellen, werden verschiedene Varianten verwendet:

- Jede Rangliste wird als eine mögliche Stimme betrachtet, es ergibt sich also eine summierbare Stimme mit  $n!$  Optionen



- „Order“:  $n$ -Tupel, wobei der  $i$ -te Eintrag die ID des auf Platz  $i$  platzierten Kandidaten in der Rangliste verschlüsselt
- „Ranking“:  $n$ -Tupel, wobei der  $i$ -te Eintrag die Platzierung des Kandidaten  $i$  in der Rangliste verschlüsselt
- $n \times n$  – Matrix mit Eintrag  $\begin{cases} Enc(1) & \text{falls Kandidat } j \text{ hat Platzierung } i \text{ in der Rangliste} \\ Enc(0) & \text{sonst} \end{cases}$   
in Spalte  $j$  und Zeile  $i$ .

Beispiel: 3 Kandidaten mit Rangliste  $K_3, K_1, K_2$ , d.h. Kandidat  $K_3$  ist die erste Wahl, dann  $K_1$  und zuletzt  $K_2$ . Die Indizes der  $K_i$  referenzieren die ID der Kandidaten.

Es ergibt sich im Klartext die Matrix  $\begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}$ , die Order  $(3,1,2)$  und das Ranking  $(2,3,1)$ .

Die Darstellung der Rangliste als Ranking und Order wird im weiteren Verlauf benötigt. Bei der Stimmabgabe wird trotzdem stets die Matrixdarstellung gewählt, denn daraus lassen sich Ranking und Order effizient berechnen, insbesondere ohne dass Kommunikation der Trustees benötigt wird, vgl. Algorithmus 2.2. Dabei wird jeder Wert der Matrix mit der Platzierung, gegeben durch den Index der Zeile, multipliziert. Da sich in einer Spalte nur genau eine verschlüsselte Eins befand, ergibt die Summe der Einträge einer Spalte dann die jeweilige Platzierung des Kandidaten.

Wenn stattdessen mit dem Index der Spalte multipliziert wird und die Einträge pro Zeile summiert werden, ergibt sich der Matrix-zu-Order Algorithmus. Denn dadurch wird mit der ID des jeweiligen Kandidaten multipliziert. Demnach ergibt sich durch die Summe in jeder Zeile die ID des Kandidaten mit der zu dieser Zeile gehörigen Platzierung.

---

#### Algorithmus 2.2 Matrix-zu-Ranking

---

```

function COMPUTE_RANKING(Matrix m)
  ranking[i]  $\leftarrow$  Enc(0) for all  $i \in \{0, \dots, n - 1\}$ 
  for all row in range(num_rows(m)) do
    for all column in range(num_columns(m)) do
      m[row, column]  $\leftarrow$  m[row, column]  $\cdot$  (row + 1) // Multiplikation mit Konstante
      ranking[column]  $\leftarrow$  ranking[column] + m[row, column]
    end for
  end for
  return ranking
end function

```

---

#### Definition 8 (Berechenbare Informationen über das Wahlergebnis)

Aus der verschlüsselten Summe an Stimmen pro Kandidat, die mit Algorithmus 2.1 berechnet wurde, lassen sich folgende Informationen über das Wahlergebnis berechnen (zur genauen Durchführung und für Testergebnisse sei auf [KLM+19] verwiesen):

1. Die Kandidaten auf den ersten oder letzten  $k$  Plätzen. Für  $k=1$  wird also die Kandidaten-ID des Siegers/Verlierers berechnet. Wir bezeichnen diese Operation mit „get\_last\_candidate“ bzw. „get\_first\_candidate“

2. *Dazu lässt sich falls gewünscht unter diesen  $k$  Kandidaten ein Ranking berechnen oder die genaue Anzahl an Stimmen entschlüsseln*
3. *Grenzwerttests, ob ein Kandidat mehr als eine bestimmte Anzahl an Stimmen erreicht hat, z.B. ob eine absolute Mehrheit erreicht wurde.*

Bei Stimmgleichheit auf Platz  $k$  werden mehr als  $k$  Kandidaten zurückgegeben. Mit einer kleinen Veränderung des Verfahrens kann dies vermieden werden:

In der „fist phase“ in [KLM+19] wird der '='-Operator nicht angewandt. Dann wird Stimmgleichheit durch den ' $\leq$ '-Operator zwischen den Kandidaten stets als Sieg für einen der Kandidaten gewertet, da der andere Kandidat bei Stimmgleichheit den invertierten Eintrag des ' $\leq$ '-Operators, also eine verschlüsselte Null, erhält.

Wenn vor Durchführung des Verfahrens eine eindeutige Reihenfolge der „Bevorzugung“ festgelegt ist, wird Stimmgleichheit als „Sieg“ für den bevorzugten Kandidaten gewertet. Damit ergibt sich eine eindeutige Platzierung für jeden Kandidaten und es werden nie mehr als  $k$  Kandidaten zurückgegeben. Durch die fehlenden '='-Operatoren ergibt sich eine Verbesserung der Laufzeit.

Die Verfahren zur Berechnung sind in der Laufzeit unabhängig von der Anzahl an Wählern. Entscheidend sind die  $O(n^2)$  verwendeten Vergleichsoperatoren. Bis zu 40 Kandidaten ist je nach Netzwerkverbindung zwischen den Trustees die Durchführung in wenigen Stunden möglich, bei weniger als 10 Kandidaten werden maximal ein paar Minuten benötigt. Demnach sind die Verfahren auch kombinierbar und können als Bausteine zur Auswertung eines komplexeren Wahlsystems verwendet werden.

Grenzwerttests sind effizienter durchführbar und erfordern nur  $n$  Vergleiche (mit dem Grenzwert) und anschließende Entschlüsselungen der verschlüsselten Vergleichsresultate. Auch ein Ranking unter allen Kandidaten ist effizienter (in  $O(n \cdot \log(n))$ ) berechenbar, wenn ein geeigneter Sortieralgorithmus gewählt wird [KLM+19].

## 3 Allgemeine Parlamentswahl

Im folgenden Teil wird der Ablauf einer Parlamentswahl in verschiedenen Varianten erklärt. Im Anschluss daran werden die Einsatzmöglichkeiten von „Tally-Hiding“ diskutiert und auf die Parlamentswahl angewendet, indem die Umsetzung auf verschlüsselte Operatoren erweitert wird. Die beschriebenen Algorithmen werden daraufhin in ihrer Laufzeit analysiert, getestet und bewertet.

### 3.1 Einordnung

In Deutschland und vielen weiteren Staaten ist die Wahl des Parlaments durch das Volk ein unentbehrlicher Bestandteil der Demokratie. In der Umsetzung gibt es verschiedene Varianten, um Mehrheitswahl und Verhältniswahl bei der Wahl des Parlaments einzubinden und zu gewichten [Noh07]:

- Das Grabenwahlsystem beruht auf einer in zwei Säulen getrennten Sitzzuteilung. In der ersten Säule wird ein vorgegebenes Kontingent an Sitzen in den Wahlkreisen nach Mehrheitsregel vergeben. In der zweiten Säule wird ein weiteres Kontingent an Sitzen mit Verhältniswahlcharakter (Proporz) an die Parteien vergeben. Ein prominentes Beispiel für ein Grabenwahlsystem (auch: segmentiertes Wahlsystem) ist Russland.
- Das kompensatorische System versucht, die Effekte der Mehrheitswahl durch die Verhältniswahl auszugleichen. Stimmen/ Sitze für in den Wahlkreisen erfolgreiche Kandidaten werden bei Auswertung der zweiten Säule für die Parteien (teilweise) abgezogen. Mit intensiverem Ausgleich der Stimmen steigt der Verhältniswahlcharakter des Systems gegenüber dem Mehrheitswahlcharakter.
- Die Bundestagswahl in Deutschland wird als „Personalisierte Verhältniswahl“ bezeichnet. Das System kann auch als Spezialfall des kompensatorischen Systems angesehen werden, wobei ein vollständiger Ausgleich der durch Erststimmen gewonnener Mandate erfolgt. Dadurch spiegelt die Sitzverteilung der Parteien exakt die Verhältnisse erhaltener Zweitstimmen wider (mit Ausnahme der Sperrklausel, siehe später).

Bei hinreichend vielen mit Mehrheitswahl zugeteilten Sitzen ist es möglich, dass ein hoher Stimmausgleich nicht ohne weiteres durchgeführt werden kann. Dieser Fall tritt auf, wenn die Anzahl gewonnener Direktmandate einer Partei höher ist als die Anzahl zustehender Sitze nach Stimmausgleich. Die Anzahl tatsächlicher Sitze im Parlament erhöht sich durch diese „Überhangmandate“. Bei der personalisierten Verhältniswahl entspricht dann die Sitzverteilung nicht mehr dem Verhältnis der Zweitstimmen. Eine Gegenmaßnahme gegen Überhangmandate sind sogenannte „Ausgleichsmandate“. Diese Mandate werden für Parteien vergeben, die durch Überhangmandate anderer Parteien weniger Sitze haben, als ihnen nach dem Verhältnis der Zweitstimmen eigentlich zustehen.

Ausgleichsmandate wurden 2013 bei der Bundestagswahl eingeführt. Allerdings wird das System mit Ausgleichsmandaten kritisiert, da die Anzahl an tatsächlichen Bundestagssitzen deutlich über den regulären 598 Sitzen liegt.

Auch bei der Stimmabgabe gibt es verschiedene Varianten. In einem Ein-Stimmen-System, so wie derzeit in Italien vorhanden, kann der Bürger seine Stimme entweder für die Liste einer Partei oder für einen Direktmandaten abgeben [Ros]. Eine Stimme für den Direktmandaten kommt auch den Parteien zugute, die den Kandidaten unterstützen. Analog unterstützt die Stimme für eine Partei die von dieser Partei im Wahlkreis befürworteten Kandidaten.

Das „Zwei-Stimmen-System“ der Bundestagswahl gewährt die voneinander unabhängige Parteien- und Personenwahl. Im weiteren Verlauf der Arbeit wird ausschließlich das Zwei-Stimmen-System betrachtet. Es ergibt sich eine analoge Möglichkeit zur Auswertung von Ein-Stimmen-Systemen, wenn die abgegebenen Stimmen in zwei unabhängige Stimmen für Partei und Kandidaten umrechenbar sind. Auf die Details wird hierzu nicht näher eingegangen.

Ein wichtiger Bestandteil vieler Parlamente ist die „Sperrklausel“. Die Sperrklausel ist der Anteil an Zweitstimmen, den eine Partei benötigt, um die durch Zweitstimmen zustehenden Sitze im Parlament zu erhalten. Abgegebene Stimmen für Parteien unter der Sperrklausel werden bei Berechnung der Sitzverteilung nicht berücksichtigt. Dies soll eine „Zersplitterung“ des Parlaments in Kleinparteien verhindern und damit die Bildung stabiler Mehrheiten ermöglichen. Die fehlende Sperrklausel wird häufig als wesentliche Ursache des Scheiterns der Weimarer Republik angesehen [Noh07]. Bei der Bundestagswahl liegt die Sperrklausel bei 5 %. Allerdings ist zu beachten, dass durch Erststimmen siegreiche Kandidaten ihren Sitz auch dann erhalten, wenn ihre Partei an der Sperrklausel gescheitert ist. Außerdem kann eine Partei die Sperrklausel auch durch eine Mindestanzahl gewonnener Direktmandate überwinden. Bei der Bundestagswahl ist die Sperrklausel bei drei Direktmandaten überwunden und die unter 5 % liegende Partei erhält dann trotzdem weitere Sitze, die ihr durch den Zweitstimmenanteil zustehen.

## 3.2 Überblick über die Umsetzung

Im folgenden Abschnitt wird ein Entwurf zur Umsetzung mit Ordinos dargestellt. Ziel des Entwurfs ist ein intaktes Verfahren für eine hohe Anzahl an „Szenarien“, d.h. für unterschiedliche Sperrklauseln, verschiedene Verrechnungsarten zwischen Mehrheits- und Verhältniswahl, abweichender Größe der Parlamente usw. Auf der anderen Seite umfasst das Konzept keine Einzelheiten der in der Realität existierenden Verfahren. Das liegt an den unzähligen weiteren Details vorhandener Wahlsysteme, dazu zählen z.B. verschiedene Systeme in unterschiedlichen Teilgebieten der Staaten oder andere Rundungsarten als beim in Abschnitt 3.4 vorgestellten Verfahren.

Das Hauptaugenmerk liegt auf einem Entwurf, der mit den in Kap. 3.3 erwähnten Bestandteilen des „Tally-Hidings“ vereinbar ist. Der folgende Ablauf stellt den Grobentwurf der Umsetzung dar:

1. Bestimmung der Direktmandate

Für jeden Wahlkreis gibt es eine feste Anzahl an Direktmandaten (bei der Bundestagswahl ein Kandidat pro Wahlkreis).

Aus den Erststimmen werden die Kandidaten mit den meisten Stimmen im Wahlkreis bestimmt. Diese werden bekannt gegeben und erhalten ein Direktmandat.

## 2. Bestimmung der Sitzverteilung der Parteien

Die Anzahl zu verteilender Sitze und die Sperrklausel werden festgelegt. Parteien mit einer vorher festgelegten Anzahl erreichter Direktmandate werden bestimmt und sind von der Sperrklausel befreit.

Aus den Zweitstimmen wird die noch nicht ausgeglichene Anzahl an Sitzen pro Partei berechnet und bekannt gegeben.

## 3. Ausgleich der Sitzverteilung mit den Direktmandaten

Ein Faktor  $f \in [0, 1]$  wird festgelegt. Je höher der Faktor, desto ausgeprägter ist der Ausgleich.  $f = 0$  bedeutet dass kein Ausgleich durchgeführt wird (Grabenwahlsystem).  $f = 1$  entspricht dem vollständigen Ausgleich bei der personalisierten Verhältniswahl. Außerdem ist festzulegen, ob Ausgleichsmandate bei Auftreten von Überhangmandaten vergeben werden sollen oder nicht.

Aus den Direktmandaten und der noch nicht ausgeglichenen Sitzanzahl der Partei wird die ausgeglichene Anzahl an Sitzen berechnet. Jede Partei erhält im Parlament die ausgeglichene Anzahl an Sitzen und die durch Direktmandate gewonnenen Sitze.

Die ausgeglichene Sitzanzahl  $s_{new}$  aus (3.) wird für jede Partei ohne verschlüsselte Operationen berechnet. Eingabe ist der Faktor  $f$ , die unausgeglichene Sitzzahl  $s_{old}$  und die Anzahl gewonnener Direktmandate  $n_{first}$ :

- Ohne Vergabe von Ausgleichsmandaten:  $s_{new} = \max\{ \text{round}(s_{old} - f \cdot n_{first}), 0 \}$
- Falls Ausgleichsmandate vergeben werden sollen, wird für jede Partei die Anzahl an Überhangmandaten berechnet:  $n_{overhang} = \min\{ (-1) \cdot \text{round}(s_{old} - f \cdot n_{first}), 0 \}$ . Daraus wird der „Überhangfaktor“  $f_{overhang}$  der Partei berechnet. Er gibt den noch nicht ausgeglichenen Anteil an Sitzen an:  $f_{overhang} = \frac{n_{overhang}}{s_{old}}$ . Aus dem unter den Parteien maximalen Überhangfaktor  $max_{overhang}$  ergibt sich die ausgeglichene Sitzanzahl jeder Partei:  $s_{new} = \text{round}(s_{old} - f \cdot n_{first}) + \text{round}(max_{overhang} \cdot s_{old})$

In der Praxis ist es kritisch, Überhangmandate für Parteien mit sehr geringem Zweitstimmenanteil mit diesem Verfahren vollständig auszugleichen. Für eine Partei mit drei unausgeglichenen Sitzen und sechs gewonnenen Direktmandaten ergibt sich mit  $f = 1$  der Überhangfaktor  $f_{overhang} = 1$ . Die anderen Parteien würden also die doppelte Anzahl ihrer Sitze erhalten. Hier müssen weitere Gegenmaßnahmen getroffen werden, z.B. dass Überhangmandate erst ab einem festgelegten Anteil an Zweitstimmen ausgeglichen werden. Allerdings wäre das genannte Beispiel bei der Bundestagswahl unrealistisch, da Direktmandate durchwegs von den „größeren“ Parteien gewonnen werden.

### 3.3 Motivation zum Einsatz von Tally-Hiding

Bei der Bundestagswahl werden alle „Zwischenergebnisse“, unabhängig von deren Relevanz für die endgültige Sitzverteilung, bekannt gegeben. Dazu zählt z.B. das Ergebnis von nicht siegreichen Direktkandidaten oder das Ergebnis von an der Sperrklausel gescheiterten Parteien. Im folgenden Abschnitt wird ein System entworfen, welches für die Sitzverteilung irrelevante Informationen geheim hält. Ziel des Entwurfs ist es, dass nach der Evaluation nur die gewonnenen Direktmandate und die Anzahl an unausgeglichenen Sitzen pro Partei bekannt ist. Daraus lässt sich dann mit dem bereits entworfenen Ausgleichsverfahren die endgültige Sitzverteilung berechnen.

Die Evaluation der Direktmandate lässt sich direkt mit der ersten Operation aus Definition 8 durchführen. In jedem Wahlkreis gibt die Operation für eine festgelegte Anzahl an Direktmandaten diejenigen Kandidaten zurück, die ein Direktmandat erhalten. Um bei Stimmgleichheit nicht mehr Direktmandate zu vergeben, kann zuvor eine Reihenfolge der „Bevorzugung“ ausgelost werden. Dann wird mit der in Def. 8 vorgestellten Variante zur Auflösung von Stimmgleichheit sichergestellt, dass in keinem Wahlkreis mehr als die vorgesehene Anzahl an Direktmandaten vergeben wird.

Die Evaluation der unausgeglichenen Sitzverteilung ist gravierend komplexer und bildet daher das Kernthema im verbleibenden Teil des Kapitels. Ab sofort bezeichnet der Begriff „Sitzverteilung“ die unausgeglichene Sitzverteilung.

Zu Beginn werden diejenigen Parteien ausgewählt, die an der Sperrklausel scheitern. Das kann direkt mit der dritten Operation aus Def. 8 herausgefunden werden. Mit der Operation werden alle Parteien getestet, die noch nicht durch das Erreichen ausreichender Direktmandate von der Sperrklausel befreit sind. Die Operation gibt für jede getestete Partei bekannt, ob sie an der Sperrklausel gescheitert ist oder nicht.

Eine sinnvolle Sperrklausel fordert mehr als die Anzahl an Zweitstimmen, die für das Erreichen eines einzigen Sitzes nötig ist, ansonsten ist die Sperrklausel überflüssig. Durch den Test werden also nur Informationen bekannt gegeben, die bei der späteren Sitzverteilung ohnehin erkennbar sind. Denn eine Partei erreicht offensichtlich genau dann keinen einzigen Sitz durch Zweitstimmen, wenn sie an einer sinnvollen Sperrklausel gescheitert ist.

Die an der Sperrklausel gescheiterten Parteien bleiben im weiteren Verlauf unberücksichtigt. Wir bezeichnen Stimmen für gescheiterte Parteien als „verfallende Stimmen“. Der Begriff vernachlässigt allerdings die Relevanz verfallender Stimmen für die Praxis. Bei der Bundestagswahl ergibt sich z.B. der Umfang der Parteienfinanzierung aus der Anzahl erreichter Zweitstimmen. Das gilt auch für an der Sperrklausel gescheiterte Parteien.

Unter den nicht gescheiterten Parteien könnte nun die Anzahl erreichter Stimmen entschlüsselt werden. Im Anschluss kann die Sitzverteilung mit einem beliebigen derzeit durchgeführten Verfahren bestimmt werden. Die Berechnung kann vollständig analog zur klassischen Urnenwahl erfolgen, nachdem die Stimmen ausgezählt wurden. Allerdings werden durch die einfache Entschlüsselung Informationen bekannt, die sich eigentlich nicht aus der Sitzverteilung ergeben:

- Die Sitzverteilung erlaubt keine Schlussfolgerungen auf den genauen Anteil an Stimmen. Ob eine Partei den nächsten Sitz nur knapp verfehlt hat oder die Sitze nur knapp erreicht hat, ist an der Sitzverteilung nicht erkennbar.
- Die Sitzverteilung erlaubt auch keine Schlussfolgerungen auf die grobe Anzahl an Stimmen: Die Anzahl an Sitzen einer Partei entspricht etwa dem Anteil an der Summe nicht-verfallender Stimmen. Zur Berechnung der ungefähren Anzahl an Zweitstimmen einer Partei im Parlament müsste die Summe verfallender Stimmen oder die Summe nicht-verfallender Stimmen bekannt sein.
- Aus der Anzahl erreichter Stimmen für verbleibende Parteien wäre die Anzahl verfallender Stimmen berechenbar. Wenn bei einer Wahl nur eine Partei an der Sperrklausel scheitern würde, so wäre die genaue Anzahl an Stimmen für die gescheiterte Partei berechenbar. Das folgende Beispiel soll dieses Szenario illustrieren.

In Mikroland leben 100 Leute. Seit vielen Jahren regieren die beiden großen Parteien „Rot“ und „Blau“. Es gibt eine Sperrklausel von 10 %. Da bekanntlich alle Mikroländer zur Wahl gehen, sind 10 Stimmen zum Einzug ins Parlament nötig. Für die Roten und Blauen stellt das Überwinden der Sperrklausel kein Problem dar.

Seit kurzer Zeit fühlt sich der Mikroländer Max in seinen Anliegen nicht mehr ernst genommen und möchte deshalb seine eigene Partei gründen. Max weiß, dass er bei der ersten Wahl höchstwahrscheinlich an der Sperrklausel scheitern wird. Allerdings hat er keine Ahnung, ob er zumindest von seinen Freunden gewählt wird. Er hat große Angst, dass im Wahlergebnis erkennbar ist, dass er nur seine eigene Stimme erhalten hat.

Nach Auszählung der Stimmen ergibt sich folgendes Wahlergebnis:

Blau	Rot	Max
60	39	1

Im Parlament gibt es 10 Sitze. Mit diesem Ergebnis erhalten die Blauen 6 Sitze und die Roten 4 Sitze. Genaueres zur Umrechnung der Stimmen in Sitze findet sich im nächsten Kapitel. Max' Befürchtungen werden wahr und er gibt nach der Wahl seinen Rückzug aus der Politik bekannt.

Max' Rückzug wäre mit einem System verhindert worden, welches nur das Scheitern an der Sperrklausel und nicht die genaue Anzahl erreichter Stimmen verrät. Mit dem System wäre veröffentlicht worden, dass die Blauen 6 Sitze und die Roten 4 Sitze erhalten. Allerdings hätte kein Mikroländer erfahren, ob Max nur seine eigene Stimme erhalten hat oder nicht. Z.B. wären die folgenden neun (und viele weitere) Ergebnisse mit gleicher Sitzverteilung genauso möglich gewesen:

	Erg. 1	Erg. 2	Erg. 3	Erg. 4	Erg. 5	Erg. 6	Erg. 7	Erg. 8	Erg. 9
Blau	60	59	59	58	58	57	57	56	56
Rot	40	39	38	38	37	37	36	36	35
Max	0	2	3	4	5	6	7	8	9

Im folgenden Abschnitt wird ein mögliches Konzept zur Umrechnung von Stimmen in Sitze umgesetzt, wobei keine der Operationen auf verschlüsselten Daten durchgeführt wird. Erst im Anschluss daran wird das Konzept auf verschlüsselte Daten erweitert, womit die Geheimhaltung der Stimmen pro Partei und der Anzahl verfallender Stimmen gewährleistet werden soll.

### 3.4 Hare-Niemeyer-Verfahren

Es existieren eine Vielzahl an Verfahren zur Umrechnung von Stimmen in Sitze. Auch die Geschichte der Bundestagswahl umfasst mehrere Verfahren. Die Methode von d'Hondt wurde 1985 durch das System von Hare/Niemeyer ersetzt [Noh07]. Mittlerweile ist das System von Hare/Niemeyer auf Bundesebene wieder abgelöst, aber es wird in zahlreichen anderen Staaten und bei Landtagswahlen in Deutschland weiterhin verwendet.

Sei  $v_i$  die Anzahl erreichter Stimmen einer nicht gescheiterten Partei  $i$ ,  $s_{seats}$  die Anzahl zu vergebender Sitze und  $s_{votes}$  die Summe nicht verfallender Stimmen. Ziel des Hare/Niemeyer-Verfahrens ist ein ziemlich exakter Proporz, d.h. der Partei  $i$  sollten  $x_i$  Sitze vergeben werden sodass:

$$\frac{v_i}{s_{votes}} = \frac{x_i}{s_{seats}} \implies x_i = \frac{v_i \cdot s_{seats}}{s_{votes}}$$

Offensichtlicherweise kann  $x_i$  in der Regel nicht als Sitzanzahl übernommen werden, da ein nicht-ganzzahliges  $x_i$  möglich ist. Deshalb werden die Sitze in zwei Phasen vergeben: In der ersten Phase bekommt jede Partei  $i$   $\lfloor x_i \rfloor$  Sitze. In der zweiten Phase werden die noch nicht verteilten „Restsitze“ vergeben. Bei  $n$  Parteien kann die Anzahl verbleibender Restsitze zwischen 0 und  $n - 1$  variieren.

Die gebräuchlichste Variante zur Verteilung der Restsitze ist die Methode des größten Überrestes. Das bedeutet, dass die Restsitze nach Höhe der Zahlenbruchteile hinter dem Komma von  $x_i$  vergeben werden [Noh07]. Die Parteien mit den größten Nachkommaanteilen erhalten jeweils einen Restsitz. Folgendes Beispiel, welches aus [Noh07] übernommen wurde, illustriert das Verfahren:

Zu vergebende Sitze: 21

Partei	Stimmen	$x_i$	Sitze nach P1	Nachkommaest	Divisionsrest	Restsitz
A	10 000	8,40	8	0,40	10 000	-
B	8 000	6,72	6	0,72	18 000	✓
C	4 000	3,36	3	0,36	9000	-
D	3 000	2,52	2	0,52	13 000	✓
Summe	25 000	21	19	2	50 000	2

Die Spalte „Divisionsrest“ wird bei Anwendung verschlüsselter Operationen relevant, da der Divisionsrest im Gegensatz zum Nachkommaest ganzzahlig ist und keine Division zur Berechnung nötig ist. Bei der Restsitzvergabe ist die Auswahl nach dem größten Divisionsrest äquivalent zur Vergabe nach größtem Nachkommaest. Das wird bei Betrachtung des folgenden Zusammenhangs klar:

$$\begin{aligned}
 \text{Nachkommaest} &= \frac{v_i \cdot s_{seats}}{s_{votes}} - \left\lfloor \frac{v_i \cdot s_{seats}}{s_{votes}} \right\rfloor = x_i - \lfloor x_i \rfloor \\
 \text{Divisionsrest} &:= \text{Nachkommaest} \cdot s_{votes} \\
 &= v_i \cdot s_{seats} - \left\lfloor \frac{v_i \cdot s_{seats}}{s_{votes}} \right\rfloor \cdot s_{votes} \\
 &= v_i \cdot s_{seats} - \lfloor x_i \rfloor \cdot s_{votes}
 \end{aligned}$$

### 3.5 Sitzberechnung mit verschlüsselten Daten

Das Hare-Niemeyer-Verfahren soll nun auf verschlüsselte Daten erweitert werden. Die Anzahl erreichter Stimmen pro Partei ist verschlüsselt ( $Enc(v_i)$ ) und darf während des Verfahrens nicht entschlüsselt werden. Auch die Anzahl nicht verfallender Stimmen  $s_{votes}$  ist und bleibt geheim. Der verschlüsselte Wert  $Enc(s_{votes})$  lässt sich einfach und ohne Trustee-Kommunikation berechnen:  $Enc(s_{votes}) = \sum_i Enc(v_i)$  wobei der Index  $i$  über alle nicht gescheiterten Parteien iteriere. Die Anzahl an Sitzen im Parlament  $s_{seats}$  ist nach wie vor im Klartext bekannt.

Es werden zwei Varianten zur Berechnung der Sitze vorgeschellt. Bei der „Öffentlichen Variante“ wird die Anzahl an Sitzen pro Partei nach der ersten Phase berechnet und veröffentlicht. Im Anschluss wird die zweite Phase durchgeführt und dabei wird für jede Partei ausgegeben, ob sie einen Restsitz erhält oder nicht. Weitere Informationen sollen bei Durchführung des Verfahrens nicht bekannt werden.



Bei der „Geheimen Variante“ soll die Anzahl an Sitzen nach der ersten Phase auch geheim bleiben und es soll nur die endgültige Anzahl zugeteilter Sitze ausgegeben werden. Im Vergleich zur ersten Variante bleibt geheim, ob eine Partei einen Restsitz erhalten hat oder nicht. Im späteren Verlauf wird auf Vor- und Nachteile beider Varianten eingegangen.

### 3.5.1 Öffentliche Variante

Es sei nach wie vor  $x_i = \frac{v_i \cdot s_{seats}}{s_{votes}}$  für jede nicht gescheiterte Partei  $i$

#### Berechnung von $\lfloor x_i \rfloor$

Zu Beginn wird  $Enc(z) := Enc(v_i \cdot s_{seats})$  ohne Trustee-Kommunikation berechnet.

$\lfloor x_i \rfloor$  ist das maximale  $k \in \mathbb{N}$ , für das  $Enc(k \cdot s_{votes}) \leq Enc(z)$  gilt. Der Wert  $k$  muss im Intervall  $[0, s_{seats}]$  gesucht werden. Um die Anzahl an „ $\leq$ “-Operationen auf verschlüsselten Daten zu verringern, wird eine binäre Suche im Intervall durchgeführt:

In einer Iteration wird das „mittlere Element“ des Intervalls als zu testender Wert  $k$  gewählt. Ohne Trustee-Kommunikation kann das Produkt  $Enc(k \cdot s_{votes})$  berechnet werden. Im Anschluss führen die Trustees den „ $\leq$ “-Vergleich des Produkts mit  $Enc(z)$  durch. Das Resultat des Vergleichs wird entschlüsselt. Wenn  $Enc(k \cdot s_{votes}) \leq Enc(z)$  gilt, dann ist  $k \leq \lfloor x_i \rfloor$  und es wird im „rechten“ Teil des Intervalls weitergesucht. Ansonsten wird im linken Teil weitergesucht.

Die entschlüsselten Resultate der Vergleichsoperatoren bei der binären Suche liefern nur die Information, ob  $\lfloor x_i \rfloor$  kleiner oder größer als der getestete Wert ist. Da  $\lfloor x_i \rfloor$  bei der öffentlichen Variante ohnehin bekannt gegeben wird, ergeben sich mit den entschlüsselten Zwischenergebnissen keine zusätzlichen Informationen.

#### Berechnung der Restsitze

In dieser Phase müssen  $k := s_{seats} - \sum_i \lfloor x_i \rfloor$  Restsitze verteilt werden. Im Fall  $k = 0$  ist nichts mehr zu tun. Ansonsten wird für jede nicht gescheiterte Partei deren verschlüsselter Divisionsrest  $Enc(div_i)$  berechnet:

$$Enc(div_i) = Enc(v_i) \cdot s_{seats} - Enc(s_{votes}) \cdot \lfloor x_i \rfloor$$

Die Berechnung erfordert keine Trustee-Kommunikation. Mithilfe des Protokolls aus Def. 8 werden nun die  $k$  Parteien mit den größten Divisionsresten bestimmt. Die ausgewählten Parteien erhalten je einen Restsitz.

Zu beachten: Falls mehrere Parteien den  $k$ -größten Divisionsrest haben, dann werden insgesamt mehr als  $s_{seats}$  Sitze vergeben. Falls immer genau  $s_{seats}$  Sitze vergeben werden sollen, dann ist die in Def. 8 erwähnte Variante zur Auflösung von Stimmgleichheit durchzuführen. Zuvor muss eine Reihenfolge der „Bevorzugung“ ausgelost werden. Bei zwei Parteien mit gleichem Divisionsrest wird die Partei höher platziert, die in dieser Reihenfolge weiter oben steht.

Im folgenden Abschnitt soll die Häufigkeit dafür abgeschätzt werden, dass mehrere Parteien den gleichen Divisionsrest haben.

Zum Erreichen von  $k \in \mathbb{N}$  Parlamentssitzen in der ersten Phase sind  $\left\lceil \frac{k \cdot s_{votes}}{s_{seats}} \right\rceil$  Stimmen nötig. Es gilt :

$$\left\lceil \frac{k \cdot s_{votes}}{s_{seats}} \right\rceil - \left\lceil \frac{(k-1) \cdot s_{votes}}{s_{seats}} \right\rceil \geq \left\lceil \frac{k \cdot s_{votes}}{s_{seats}} \right\rceil - \left\lfloor \frac{k \cdot s_{votes}}{s_{seats}} \right\rfloor + \left\lfloor \frac{s_{votes}}{s_{seats}} \right\rfloor$$

Demnach gibt es mindestens  $\lfloor \frac{s_{votes}}{s_{seats}} \rfloor$  mögliche Werte erhaltener Stimmen für jedes  $k$ . Der Divisionsrest verändert sich durch eine einzige Stimme. Dementsprechend gibt es mindestens  $\lfloor \frac{s_{votes}}{s_{seats}} \rfloor$  verschiedene Divisionsreste für ein beliebiges  $k$ .

Wenn  $s_{seats}$  ein Teiler von  $s_{votes}$  ist, dann gibt es genau die Divisionsreste  $j \cdot s_{seats}$  mit  $j \in \{0, \dots, \frac{s_{votes}}{s_{seats}} - 1\}$ . Wenn es kein Teiler ist, dann gibt es deutlich mehr Divisionsreste. Das liegt daran, dass sich die Divisionsreste von  $k - 1$  nicht für  $k$  wiederholen.

### 3.5.2 Geheime Variante

#### Berechnung von $Enc(\lfloor x_i \rfloor)$

Analog zur öffentlichen Variante wird  $Enc(z) := Enc(v_i \cdot s_{seats})$  berechnet. Im Gegensatz zur öffentlichen Variante muss jedes  $k \in [1, s_{seats}]$  getestet werden. Das Resultat des „ $\leq$ “-Vergleichs darf nicht entschlüsselt werden.

Seien  $[r_1, \dots, r_{s_{seats}}]$  die verschlüsselten Resultate der durchgeführten „ $\leq$ “-Vergleiche:

$$r_k = \begin{cases} Enc(1) & \text{falls } Enc(k \cdot s_{votes}) \leq Enc(z) \\ Enc(0) & \text{sonst} \end{cases}$$

Wir setzen dementsprechend  $r_0 := Enc(1)$  und  $r_{s_{seats}+1} := Enc(0)$ .

Es ist  $[r_0, \dots, r_{s_{seats}+1}] = [Enc(1), \dots, Enc(1), Enc(0), \dots, Enc(0)]$  aufgrund der Monotonie des „ $\leq$ “-Operators.

Nun werden neue Werte  $[\tilde{r}_0, \dots, \tilde{r}_{s_{seats}}]$  berechnet, wobei

$$\tilde{r}_k = \begin{cases} Enc(1) & \text{falls } r_k = Enc(1) \wedge r_{k+1} = Enc(0) \\ Enc(0) & \text{sonst} \end{cases} = r_k - r_{k+1}$$

Die Berechnung der  $\tilde{r}_k$  klappt mit der Subtraktion ohne Trustee-Kommunikation. Es gibt genau ein  $\tilde{r}_k$  mit  $\tilde{r}_k = Enc(1)$ .

$\lfloor x_i \rfloor$  ist der Index des  $\tilde{r}_k$  mit  $\tilde{r}_k = Enc(1)$ . Wir berechnen  $Enc(\lfloor x_i \rfloor) = \sum_k \tilde{r}_k \cdot k$  ohne weitere Trustee-Kommunikation.

#### Berechnung der Restsitze

Im Gegensatz zur öffentlichen Variante bleibt die Anzahl zu vergebender Restsitze unbekannt. Daher muss die Berechnung der Restsitze auch durchgeführt werden, wenn kein Restsitz vergeben wird. Die Berechnung mit dem Protokoll aus Def. 8 funktioniert identisch mit einer verschlüsselten Anzahl zu vergebender Restsitze  $Enc(k)$ . Analog zur öffentlichen Variante kann  $Enc(k)$  nun auf Geheimtexten berechnet werden:

$Enc(k) = Enc(s_{seats}) - \sum_i Enc(\lfloor x_i \rfloor)$ . Bei Berechnung des Divisionsrests wird Trustee-Kommunikation benötigt, da pro Partei eine Multiplikation von Geheimtexten durchgeführt wird:

$$Enc(div_i) = Enc(v_i) \cdot s_{seats} - Enc(s_{votes}) \cdot Enc(\lfloor x_i \rfloor)$$

Das Protokoll aus Def. 8 liefert für jede Partei  $i$  ein verschlüsseltes Bit  $Enc(b_i)$  mit

$$Enc(b_i) = \begin{cases} Enc(1) & \text{falls Partei } i \text{ erhält Restsitz} \\ Enc(0) & \text{sonst} \end{cases}$$

Bei der öffentlichen Variante werden die  $Enc(b_i)$  entschlüsselt und es wird veröffentlicht,

welche Parteien einen Restsitz erhalten. Bei der geheimen Variante wird pro Partei  $Enc(\lfloor x_i \rfloor) + Enc(b_i)$  berechnet und dann entschlüsselt. Damit wird nur die endgültige Sitzverteilung nach Durchführung beider Phasen bekannt.

### 3.6 Laufzeitanalyse

Bei beiden Varianten (öffentlich und geheim) wird Trustee-Kommunikation für folgende Operationen benötigt:

- **Bestimmung gewonnener Direktmandate:**  
Für jeden Wahlkreis wird ein Aufruf des Protokolls aus Definition 8 durchgeführt. Die Laufzeit hängt von der Anzahl antretender Kandidaten ab und ergibt sich direkt aus Analyse und Ergebnissen in [KLM+19].
- **Bestimmung gescheiterter Parteien:**  
Für jede Partei wird geprüft, ob sie an der Sperrklausel gescheitert ist. Das erfordert einen „ $\leq$ “-Vergleich pro angetretener Partei mit anschließender *Dec*-Operation zur Entschlüsselung des Resultats.

Sei  $n$  die Anzahl nicht gescheiterter Parteien und  $s_{seats}$  die Anzahl zu verteilender Sitze. Bei der öffentlichen Variante sind zusätzlich die folgenden Operationen für die Laufzeit relevant:

- **Berechnung von  $\lfloor x_i \rfloor$ :**  
Bei einer Binären Suche werden ca.  $\lceil \log_2(s_{seats}) \rceil$  Vergleiche durchgeführt. Das Resultat eines Vergleichs wird direkt entschlüsselt. Insgesamt werden also ungefähr  $n \cdot \lceil \log_2(s_{seats}) \rceil$  *Dec*-Operationen und „ $\leq$ “-Vergleiche durchgeführt.
- **Berechnung der Restsitze:**  
Der Fall, dass  $k = 0$  Restsitze zu vergeben sind, ergibt sich nur genau dann wenn  $\lfloor x_i \rfloor = x_i$  für alle Parteien  $i$  gilt. Dazu müssen alle Divisionsreste null sein. Es wurde bereits festgestellt, dass gleiche Divisionsreste mit steigendem Verhältnis  $\frac{s_{votes}}{s_{seats}}$  seltener auftreten. Wir nehmen deshalb ab sofort den (i.A. deutlich häufigeren) Fall  $k \neq 0$  an.

Zur Verteilung der Restsitze ist ein Aufruf des Protokolls aus Def. 8 nötig. Die Laufzeit des Protokolls ist abhängig von  $n$ .

Bei der geheimen Variante kommen folgende Operationen hinzu:

- **Berechnung von  $\lfloor x_i \rfloor$ :**  
Für jedes  $k \in \{1, \dots, s_{seats}\}$  ist ein „ $\leq$ “-Vergleich durchzuführen. Das ergibt insgesamt  $n \cdot s_{seats}$  Vergleiche. Die deutlich höhere Anzahl an Vergleichen bei der geheimen Variante ist ausschlaggebend für die signifikante Erhöhung der Laufzeit im Vergleich zur öffentlichen Variante (vgl. Abschnitt 3.7).
- **Berechnung der Restsitze:**  
Die Ausführung des Protokolls aus Def. 8 verläuft in der geheimen Variante identisch und es gibt keine Unterschiede in der Laufzeit.  
Hinzu kommen  $n$  Multiplikationen von Geheimtexten zur Berechnung der Divisionsreste. Multiplikationen sind deutlich effizienter durchführbar als „ $=$ “-Vergleiche und „ $\leq$ “-Vergleiche sind effizienter durchführbar als „ $\leq$ “-Vergleiche.

Die Multiplikationen sind deshalb nicht ausschlaggebend für die Laufzeit der geheimen Variante. Es ergeben sich keine relevanten Laufzeitunterschiede zur öffentlichen Variante bei der Restsitzberechnung.

In Abschnitt 4.8 werden die Laufzeiten von Multiplikation und „=-Vergleich verglichen. Bei Ausführung eines „≤“-Vergleichs werden mehrere „=-Vergleiche durchgeführt. Die Anzahl durchgeführter „=-Vergleiche liegt dabei in  $O(\log(l))$  für  $l$ -bit Integer-Werte. Die Ausführungszeit eines „=-Vergleichs ist in  $O(l)$  [LT13].

Um die korrekte Durchführung eines Vergleichs sicherzustellen, muss  $l$  hinreichend hoch gewählt werden. Allerdings ist während des Vergleichs verschlüsselter Werte eventuell nicht bekannt, mit wie vielen Bits sich beide Werte im Klartext darstellen lassen. Deshalb ist bei der Wahl von  $l$  vom „worst-case“ auszugehen:

Bei Berechnung von  $Enc(\lfloor x_i \rfloor)$  (öffentlich und geheim) wird stets ein Geheimtext mit dem Geheimtext  $Enc(z) = Enc(v_i \cdot s_{seats})$  verglichen. Damit  $l$  hinreichend hoch gewählt wird, wird davon ausgegangen, dass jede Partei alle abgegebenen Stimmen erhalten haben könnte. Sei  $s_{voter}$  die Summe aller abgegebenen Stimmen. Für  $l$  wird nun die minimale Anzahl an Bits, um den Integerwert  $s_{voter} \cdot s_{seats}$  darzustellen, gewählt. Die verglichenen Werte sind niemals größer als  $s_{voter} \cdot s_{seats}$ , weshalb das gewählte  $l$  bei allen Vergleichen hinreichend groß ist.

Für die Restsitzberechnung gilt:  $\text{Nachkommarest} < 1 \Rightarrow \text{Divisionsrest} < s_{votes}$ , wobei  $s_{votes}$  weiterhin die Anzahl nicht verfallender Stimmen sei. Demnach würde die Bitanzahl von  $s_{voter}$  als  $l$  für die Restsitzberechnung ausreichen. Aus Gründen der Einheitlichkeit wird bei der Implementierung trotzdem das  $l$  aus der ersten Phase verwendet. Die negativen Auswirkungen auf die Laufzeit sind gering, da sie ohnehin nicht entscheidend von der Restsitzberechnung abhängt (vgl. Abschnitt 3.7).

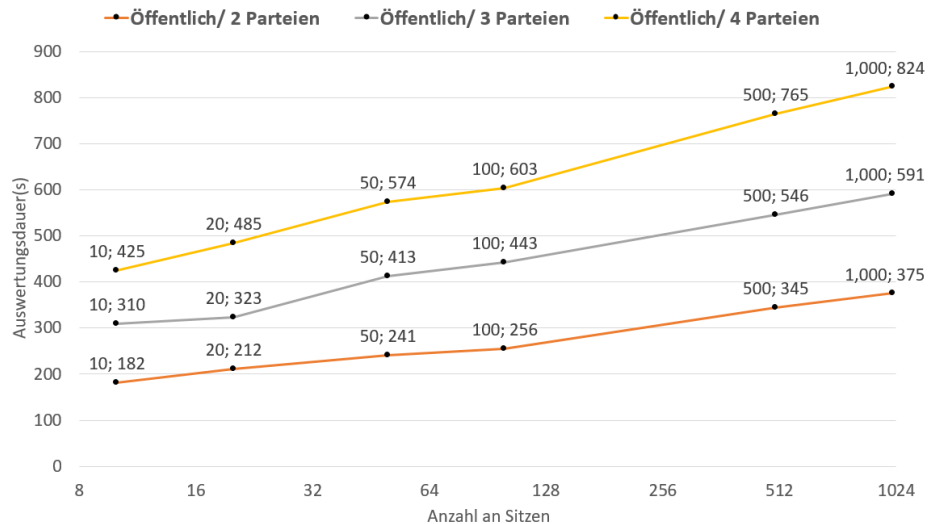
$l = 32$  bit reicht bei vielen realistischen Szenarien aus, z.B. für 3 Millionen Wähler und 500 zu vergebenden Sitzen. Allerdings muss bei Wahlen mit sehr vielen Wählern, z.B. bei 50 Millionen Wählern und 500 Parlamentssitzen, zu  $l = 64$  bit gewechselt werden. Die Verwendung von 64 bit reicht für alle in der Praxis denkbaren Szenarien aus.

### 3.7 Testergebnisse

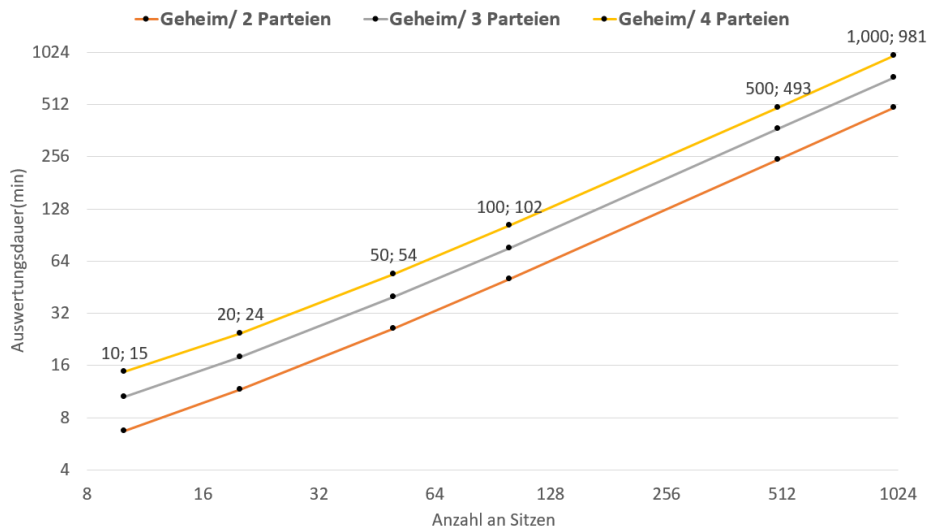
Um die Laufzeit zu messen und zu vergleichen, wurden 2048 bit als Schlüssellänge gewählt. Die Programme wurden auf einer Maschine mit vier Kernen und drei Trustees ausgeführt. Die Messung umfasst folgende Operationen:

1. Ein Vergleich jeder Partei mit der Sperrklausel. Allerdings wird stets eine Sperrklausel von null Stimmen verwendet, sodass keine Partei scheitert und alle Parteien bei Berechnung der Sitze berücksichtigt werden.
2. Die erste Phase der Sitzberechnung in der öffentlichen bzw. geheimen Variante.
3. Die zweite Phase der Sitzberechnung in der öffentlichen bzw. geheimen Variante. Bei allen durchgeführten Tests mit öffentlicher Variante mussten Restsitze vergeben werden, d.h. die zweite Phase wurde niemals übersprungen.

Für den ersten Test erhält jede Partei eine zufällige Anzahl an Stimmen zwischen 0 und 300. Die Integer-Bitlänge für Vergleichsoperatoren wurde auf 32 bit festgesetzt. Diese Länge ist ausreichend groß für alle bei diesem Test durchlaufenen Testfälle.



**Abbildung 3.1:** Öffentliche Variante, 32 bit Integer



**Abbildung 3.2:** Geheime Variante, 32 bit Integer

Bei der öffentlichen Variante wird die Ausführungszeit in Sekunden dargestellt (Abbildung 3.1). Das Diagramm zeigt die logarithmische Abhängigkeit der Laufzeit in der Anzahl der Sitze, da die horizontale Achse logarithmisch skaliert ist. Deshalb ist die öffentliche Variante auch für sehr viele zu vergebende Parlamentssitze äußerst effizient durchführbar. Der nicht ganz geradlinige Verlauf der Kurven begründet sich mit dem variierenden Verlauf einer binären Suche: In einem

### 3 Allgemeine Parlamentswahl

Rekursionsschritt ist, je nach Resultat des Vergleichs, eine Aufteilung in eine um ein Element kleinere bzw. größere Teilliste möglich. Die exakte Anzahl durchgeführter Vergleiche hängt deshalb auch von der genauen Anzahl erhaltener Stimmen pro Partei ab.

Bei der geheimen Variante ist die Ausführungszeit in Minuten dargestellt (Abbildung 3.2). Beide Achsen sind logarithmisch skaliert, damit zeigt sich die lineare Abhängigkeit in der Anzahl der zu vergebenden Sitze. Für eine steigende Anzahl an Sitzen ergeben sich zunehmend deutlichere Unterschiede in der Laufzeit beider Varianten. Bei 1000 Sitzen und 4 Parteien werden über 16 Stunden für die geheime Variante benötigt. Die Ausführungszeit bei der öffentlichen Variante beträgt weniger als 15 Minuten für das gleiche Szenario.

Bei den im weiteren Verlauf dargestellten Tests wird die Integerlänge auf 64 bit erhöht. Mit dieser Länge ist eine höhere Anzahl teilnehmender Wähler möglich. Einmal wurden genau die Szenarien aus den Abbildungen 3.1 und 3.2 mit erhöhter Bitlänge getestet. Dabei erhält jede Partei weiterhin zwischen 0 und 300 Stimmen.

Außerdem wurden die gleichen Szenarien mit erhöhter Bitlänge und Wählerzahl getestet. Jede Partei erhält eine zufällige Anzahl an Stimmen zwischen  $10^5$  und  $10^9$ . Die folgenden Abbildungen zeigen eindeutig, dass die Ausführungszeit für eine feste Bitlänge unabhängig von der Anzahl abgegebener Stimmen ist.

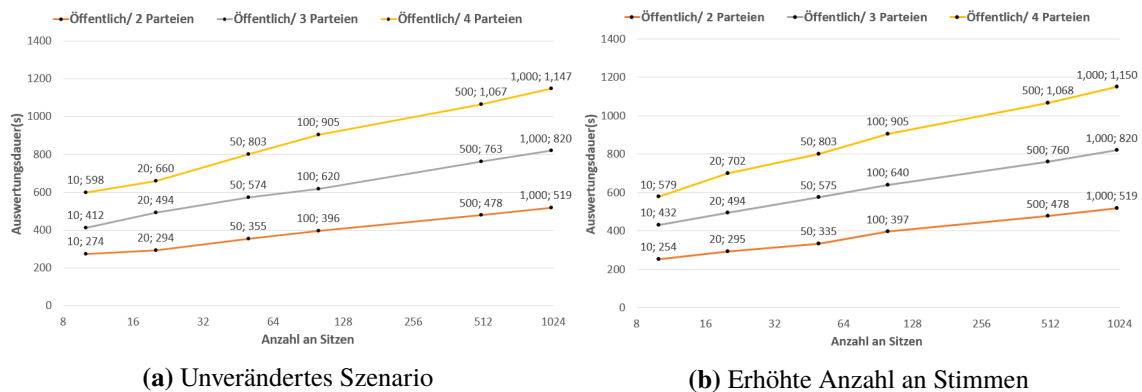


Abbildung 3.3: Öffentliche Variante, 64 bit Integer

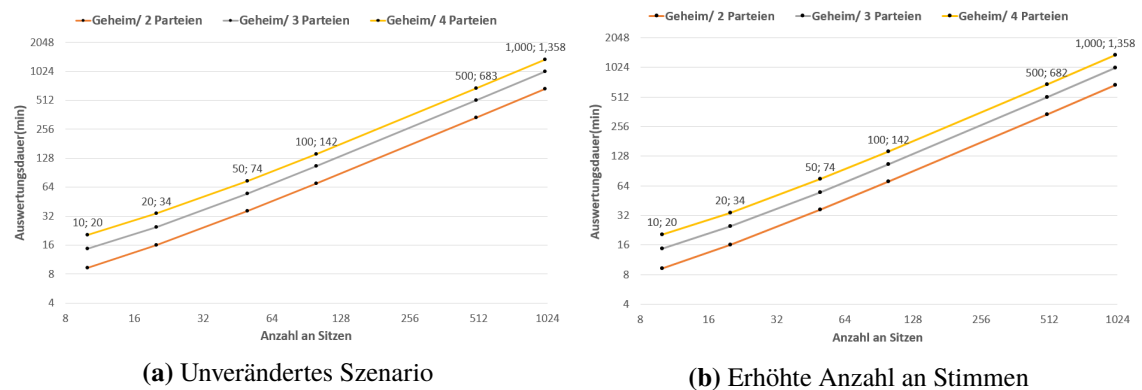


Abbildung 3.4: Geheime Variante, 64 bit Integer

Es ist offensichtlich, dass sich die Ausführungszeit durch Verwendung der 64-bit-Integerlänge ebenfalls erhöht. Allerdings gilt für alle durchgeführten Tests mit geheimer Variante, dass sich die Ausführungszeit um weniger als Faktor 1.5 erhöht hat. Bei der öffentlichen Variante ist der Faktor teilweise geringfügig höher, was jedoch dem etwas unregelmäßigen Verlauf der Kurven geschuldet ist. Die erhöhte Ausführungszeit bei 64-bit-Integerlängen ist demnach gut kalkulierbar und sorgt für keine drastischen Einschränkungen in der praktischen Nutzbarkeit.

Die Tests wurden nur für 2-4 Parteien durchgeführt. Die Ausführungszeit zur Berechnung der Sitze in der ersten Phase ist linear in der Anzahl an Parteien. Sie lässt sich deshalb auch für eine höhere Anzahl an Parteien genau kalkulieren. Hinzu kommt quadratische Komplexität für die Restsitzberechnung. Allerdings ist die Restsitzberechnung nicht entscheidend, da die in Def. 8 erwähnten Ausführungszeiten des Protokolls vergleichsweise gering sind. Bei der Bundestagswahl wären mehr als 10 nicht scheidende Parteien derzeit unrealistisch, weshalb die Restsitzberechnung im Bereich weniger Minuten liegt und vernachlässigbar ist.

Die folgenden Darstellungen veranschaulichen den linearen Verlauf bei variierender Zahl teilnehmender Parteien.

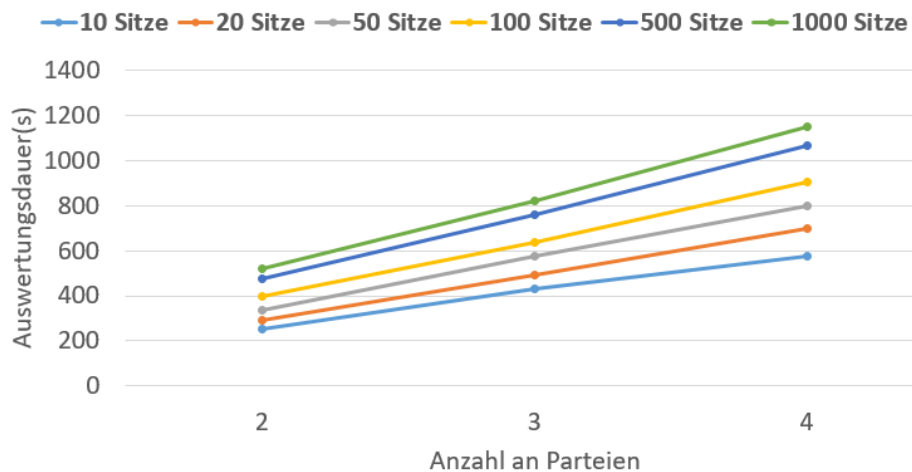


Abbildung 3.5: Andere Darstellung der Szenarien aus Abbildung 3.3b

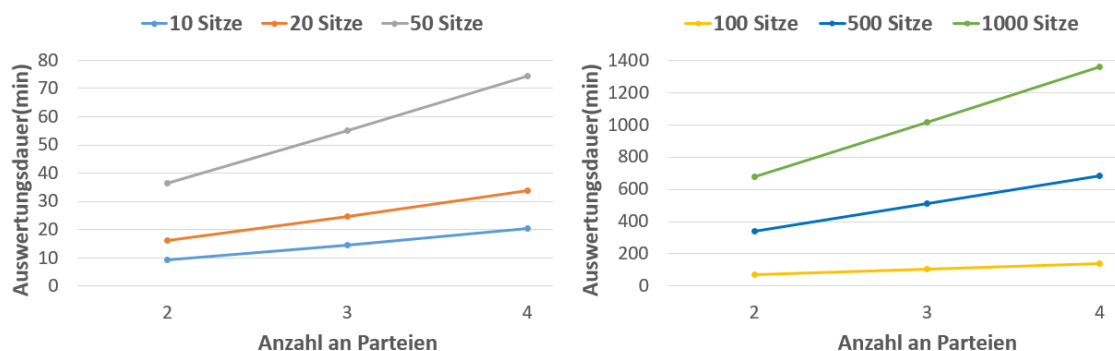


Abbildung 3.6: Andere Darstellung der Szenarien aus Abbildung 3.4b

### 3.8 Sicherheit

Die Sicherheit der Umsetzung basiert auf der Sicherheitsanalyse in [KLM+19], es werden ausschließlich Protokolle aus [KLM+19] auf verschlüsselten Daten durchgeführt.

Bei der öffentlichen Variante werden die Resultate der Vergleiche während der binären Suche entschlüsselt. Die entschlüsselten Werte liefern allerdings nur Informationen, die sich ohnehin aus den bekannt zu gebenden Ergebnissen nach der ersten Phase ergeben.

Am Ende der zweiten Phase werden die Resultate der Restsitzberechnung entschlüsselt. Dabei ergeben sich zwei mögliche Werte pro Partei („Restsitz oder kein Restsitz erhalten“). Weitere Entschlüsselungen werden nur bei Durchführung der verwendeten Protokolle (Def. 4) durchgeführt.

Bei der geheimen Variante bleiben die errechneten Sitze nach Phase 1 verschlüsselt. Das gilt auch für die Resultate der Restsitzberechnung. Es wird, abgesehen von den Protokollen aus Def. 4, nur die endgültige Anzahl an Sitzen pro Partei entschlüsselt.

Somit werden bei Berechnung der Sitzverteilung keine geheimen Informationen preisgegeben. Das gilt natürlich nur bei hinreichend hoher Sicherheit des Kryptosystems und der verwendeten Protokolle.

### 3.9 Geheime vs. Öffentliche Variante

Bei der öffentlichen Variante wird für jede Partei veröffentlicht, ob sie einen Restsitz erhalten hat oder nicht. Im folgenden Abschnitt soll erörtert werden, inwieweit sich dadurch weitere Informationen über das Wahlergebnis ableiten lassen. Ziel ist es, dass für eine durchzuführende Wahl bewertet werden kann, ob die zusätzlichen Informationen der öffentlichen Variante preisgegeben werden dürfen oder der zusätzliche Aufwand der geheimen Variante in Kauf genommen werden soll. Insbesondere soll die Anzahl verfallender bzw. nicht verfallender Stimmen auf keinen Fall aus den veröffentlichten Informationen berechenbar sein.

Zuerst wird der (Sonder-)Fall betrachtet, dass kein Restsitz vergeben wird.

Da die Summe der Nachkommaresten genau der Anzahl an vergebenen Restsitzen entspricht, müssen alle Nachkommaresten null sein. Bei jeder nicht gescheiterten Partei  $i$  ist  $\frac{v_i \cdot s_{seats}}{s_{votes}} = x_i$  ganzzahlig. Es gilt:  $\frac{v_i}{s_{votes}} = \frac{x_i}{s_{seats}}$ , also lässt sich von jeder nicht gescheiterten Partei der exakte erhaltene Anteil an nicht verfallenden Stimmen berechnen.

Trotzdem kann gezeigt werden, dass auch andere Szenarien mit gleichem Resultat möglich wären. Es gilt:

$$\begin{aligned} x_i &= \frac{v_i}{s_{votes}} \cdot s_{seats} = \frac{1 + \frac{s_{seats}}{s_{votes}}}{1 + \frac{s_{seats}}{s_{votes}}} \cdot \frac{v_i}{s_{votes}} \cdot s_{seats} \\ &= \frac{v_i + \frac{v_i}{s_{votes}} \cdot s_{seats}}{s_{votes} + s_{seats}} \cdot s_{seats} = \frac{v_i + x_i}{s_{votes} + s_{seats}} \cdot s_{seats} \end{aligned}$$

Die Gleichungen zeigen, dass sich ein Szenario mit identischem Resultat ergibt, falls  $s_{votes} + s_{seats}$  (statt  $s_{votes}$ ) Stimmen nicht verfallen. Das Resultat kommt dann zustande, wenn jede nicht gescheiterte Partei  $v_i + x_i$  (statt  $v_i$ ) Stimmen erhält.



Falls die Anzahl verfallender Stimmen auf  $\pm s_{seats}$  Stimmen eingeschränkt werden kann, sind die Überlegungen zwecklos: Beim Beispiel aus Abschnitt 3.3 wären  $s_{seats} = 10$  verfallende Stimmen nicht möglich, da 10 Stimmen zu Max' Einzug in das Parlament geführt hätten. Nach Durchführung der öffentlichen Variante ohne Vergabe von Restsitzen kann deshalb nicht sichergestellt werden, dass mehrere Szenarien für ein identisches Resultat möglich sind. Je nach veröffentlichter Sitzverteilung wäre es möglich, dass sich Max' exakte Stimmanzahl herleiten lässt. Bei der Wahl aus Abschnitt 3.3 sollte deshalb auf jeden Fall die geheime Variante verwendet werden, die bei nur 10 Sitzen und drei Parteien ohnehin effizient durchführbar ist.

Bei Wahlen mit hinreichend vielen Wählern, wie z.B. die Bundestagswahl, ist  $s_{seats}$  nur ein ziemlich kleiner Teil der verfallenden Stimmen. Eine Folgerung auf die exakte Anzahl verfallender Stimmen wäre nicht möglich und eine Eingrenzung auf  $\pm x_i$  Stimmen für nicht gescheiterte Parteien wäre mindestens genauso schwer. Die Folgen der zusätzlich preisgegebenen Informationen wären deshalb weitaus weniger gravierend.

Falls Restsitze vergeben werden, gibt es deutlich mehr mögliche Szenarien, die mit den bekannt gegebenen Informationen bei der öffentlichen Variante infrage kommen. Trotzdem lassen sich aus dem veröffentlichten Resultat einige Informationen über das Wahlergebnis ableiten:

Beim Hare-Niemeyer-Verfahren wird versucht, dass jede nicht gescheiterte Partei den Anteil  $a_i = \frac{v_i}{s_{votes}} = \frac{x_i}{s_{seats}}$  aller Parlamentssitze bekommt. Mithilfe des veröffentlichten Werts  $\lfloor x_i \rfloor$  aus Phase 1 kann  $a_i$  abgeschätzt werden:  $\frac{\lfloor x_i \rfloor}{s_{seats}} \leq a_i < \frac{\lfloor x_i \rfloor + 1}{s_{seats}}$ . Die Gleichheit der ersten Ungleichung gilt genau dann, wenn  $x_i$  eine ganze Zahl ist.

Außerdem gilt:  $\frac{Nachkommarest}{s_{seats}} = \frac{x_i}{s_{seats}} - \frac{\lfloor x_i \rfloor}{s_{seats}} = a_i - \frac{\lfloor x_i \rfloor}{s_{seats}}$ . Bei einer Partei, die einen Restsitz erhält, hat  $a_i$  also bei obiger Abschätzung einen „größeren Abstand“ zur unteren Grenze  $\frac{\lfloor x_i \rfloor}{s_{seats}}$  als bei einer Partei ohne erhaltenen Restsitz. Die Bekanntgabe von Restsitzen erlaubt demnach eine genauere Einschätzung der Stimmanteile  $a_i$ .

Eine exakte und formale Aussage über die Anzahl möglicher Szenarien ist nicht einfach und kann hier nicht angegeben werden. Allerdings zeigen sich mithilfe obiger Abschätzung von  $a_i$  zwei grundlegende Tendenzen:

Mit einer steigenden Anzahl an Wählern sind mehr Werte für  $a_i = \frac{v_i}{s_{votes}}$  in einem gegebenen Intervall möglich, deshalb steigt auch die Anzahl möglicher Szenarien für ein veröffentlichtes Resultat.

Mit weniger zu verteilenden Sitzen wird das Intervall  $\left[ \frac{\lfloor x_i \rfloor}{s_{seats}} ; \frac{\lfloor x_i \rfloor + 1}{s_{seats}} \right)$  größer und die Anzahl möglicher Szenarien steigt demzufolge ebenfalls. Bei weniger vergebenen Sitzen zeigt sich durch das größer werdende Intervall auch die problematische und ohnehin offensichtliche Tatsache, dass eine größere Differenz zwischen Stimmanteil und Sitzanteil möglich wird.

Bei der geheimen Variante ist mithilfe der veröffentlichten Sitzverteilung ohne Weiteres lediglich eine ungenauere Aussage über die  $a_i$  möglich:

Sei  $s_i$  die veröffentlichte Anzahl an Sitzen einer nicht gescheiterten Partei. Es ist entweder  $s_i = \lfloor x_i \rfloor$  oder  $s_i = \lfloor x_i \rfloor + 1$ . Der zweite Fall tritt genau dann ein, wenn Partei  $i$  einen Restsitz erhalten hat, was jedoch unbekannt bleibt. Insbesondere ist  $x_i$  im zweiten Fall niemals ganzzahlig.

In obige Abschätzung von  $a_i$  eingesetzt ergibt sich:  $\frac{s_i - 1}{s_{seats}} < a_i < \frac{s_i + 1}{s_{seats}}$ .



## 4 Instant-Runoff-Voting

In diesem Kapitel wird das Instant-Runoff-Voting (IRV) mit Ordinos umgesetzt. Nach Beschreibung des Wahlsystems werden vier entworfene Verfahren zur Auswertung präsentiert und beispielhaft durchgeführt. Im Anschluss daran werden zwei alternative Verfahren präsentiert, die auf einer anderen Form der Wählerstimme basieren. Im weiteren Verlauf wird die Laufzeit aller Varianten analysiert und getestet.

### 4.1 Beschreibung des Verfahrens

Zuerst soll das IRV-Wahlsystem charakterisiert werden und es wird erläutert, wie sich das Wahlergebnis berechnen lässt. Daraufhin werden die Möglichkeiten vorgestellt, die sich bei der Umsetzung mit Ordinos und der Anwendung von Tally-Hiding ergeben.

#### 4.1.1 Einordnung

Das IRV ist ein Wahlverfahren, bei dem der Wähler seine Präferenz als Rangliste von Kandidaten abgibt. In die Typologie von Nohlen lässt sich das Verfahren der „Übertragbaren Einzelstimmgebung“ zuordnen [Noh07]. Bei der übertragbaren Einzelstimmgebung werden Wählerstimmen bereits gewählter oder eliminierter Kandidaten auf den nächsten Kandidaten der persönlichen Rangliste übertragen.

Das IRV lässt sich hierbei als Spezialfall einordnen, wobei nur ein Sieger zu ermitteln ist. Nohlen ordnet das auch als „alternative vote“ bezeichnete Verfahren jedoch eigenständig als eines von neun Wahlsystemtypen [Noh07] ein. Grund hierfür ist die Tatsache, dass sich eine IRV-Wahl, im Gegensatz zur übertragbaren Einzelabstimmung mit mehreren zu bestimmenden Wahlsiegern, eindeutig als absolute Mehrheitswahl (2.1) bezeichnen lässt. Die Partizipation wird gestärkt, da Wählerstimmen für schwächere Kandidaten nicht verloren gehen [Cat09]. Gegenüber Wahlsystemen mit mehreren Runden besteht der Vorteil, dass eine einmalig abgegebene Stimme pro Wähler ausreicht. Das System wird für zahlreiche derzeitig durchgeführte Wahlen, z.B. zur Wahl des Präsidenten in Indien, eingesetzt.

#### 4.1.2 Ablauf der Stimmauswertung

Die Auswertung der Stimmzettel erfolgt in mehreren Iterationen [Jer06]. In einer Iteration zählt jeder Stimmzettel als eine Stimme für den Kandidaten, der an erster Position der Rangliste platziert ist. Der Kandidat, der in Summe die wenigsten Stimmen erhält, scheidet aus („Elimination“). Bei Stimmgleichheit werden entweder mehrere Kandidaten eliminiert oder es muss ein vorher

festgelegter „Tie-Breaker“ angewendet werden. Im Folgenden wird zur Vereinfachung der Verfahren bei Stimmgleichheit jeweils der Kandidat mit kleinster ID eliminiert. Für eine genauere Analyse zur Mehrfachelimination sei auf Kapitel 4.5 verwiesen.

Nach jeder Elimination rutschen bei jedem Stimmzettel alle hinter dem eliminierten Kandidaten Platzierten um einen Platz nach vorne („Hochrutschen“).

Elimination und Hochrutschen werden nacheinander durchgeführt, bis nur noch ein Kandidat, der Sieger der Wahl, übrig bleibt. Es kann auch bereits abgebrochen werden, falls ein Kandidat eine absolute Mehrheit an Stimmen erhält. Andererseits werden durch die Überprüfung der absoluten Mehrheit weitere Informationen über das Stimmergebnis bekannt gegeben. In unserer Implementierung wird pro Elimination nur der eliminierte Kandidat bestimmt und keine weitere Information preisgegeben. Deshalb kann das Verfahren nicht vorzeitig abgebrochen werden.

### 4.1.3 Motivation zur Umsetzung mit Ordinos

Die iterative und komplexe Stimmauswertung des IRV-Systems erschwert die Umsetzung als E-Voting-System. Eine Entschlüsselung des Stimmzettels oder von Teilen davon verstößt gegen das Prinzip der Geheimhaltung und darf deshalb nicht durchgeführt werden.

Eine Möglichkeit zur Umsetzung ist das Verfahren „sElect“ [KMST16]. Mit diesem Verfahren lassen sich die Stimmzettel mischen und anschließend wieder entschlüsseln. Die entschlüsselten Stimmzettel können im Anschluss öffentlich ausgewertet werden. Das funktioniert sehr effizient und entspricht dem gängigen Vorgehen bei der Auswertung von nicht elektronisch durchgeführten Wahlen. Allerdings ist dieses Vorgehen insbesondere bei Wahlen mit wenigen Wählern äußerst kritisch, da ein bestimmter Wahlzettel bereits aufgrund einiger bekannter persönlicher Präferenzen mit einem Wähler in Verbindung gebracht werden könnte.

Um dies zu verhindern, eignet sich die Umsetzung mit Ordinos. Die Fähigkeit der verschlüsselten Durchführung von Vergleichen erlaubt es nicht nur, Teile des Ergebnisses geheim zu halten („Tally-Hiding“). Es lässt sich auch die Auswertung des IRV ohne Entschlüsselung der Stimmzettel durchführen.

### 4.1.4 Überblick über verschiedene Verfahren

Für die Auswertung des IRV werden mehrere Verfahren präsentiert. Sie unterscheiden sich in den während der Auswertung preisgegebenen Informationen:

1. Bei jeder Elimination wird der eliminierte Kandidat öffentlich bekannt gegeben. Das hat den Nachteil, dass zu Beginn eliminierte Kandidaten eventuell blamiert werden und als Verlierer der Wahl angesehen werden, obwohl nur der Gewinner gesucht war. Der Vorteil ist, dass dadurch eine einfachere Implementierung mit erheblichen Laufzeit-Vorteilen ermöglicht wird. Mögliche Umsetzungen mit Ordinos werden in Abschnitt 4.2 beschrieben.
2. Die Reihenfolge der Eliminationen bleibt geheim, d.h. es wird nur der Sieger bekannt gegeben. Die Verfahren mit öffentlicher Elimination müssen verändert werden, denn sie setzen die Entschlüsselung des eliminierten Kandidaten während jeder Iteration voraus. Mögliche Umsetzungen werden in Abschnitt 4.3 beschrieben.

## 4.2 Verfahren mit öffentlicher Elimination

In diesem Teil werden die ersten zwei Varianten zur Berechnung des Wahlergebnisses präsentiert. Bei beiden Varianten wird eine öffentliche Elimination durchgeführt, d.h. in jeder Iteration wird der eliminierte Kandidat bekannt gegeben.

### 4.2.1 Variante mit Matrixmanipulation

Diese Variante beruht auf drei Teilschritten, die iterativ ausgeführt werden, bis alle  $n - 1$  eliminierten Kandidaten feststehen. Jeder Wähler gibt seine Rangliste verschlüsselt in der Matrixdarstellung ab (Def. 7). Die folgenden Teilschritte 2 und 3 werden dabei für jede Wählermatrix durchgeführt.

1. Aufruf von *get\_last\_candidate* (siehe Def. 8)
2. Streichen der Matrixspalte des eliminierten Kandidaten
3. Hochrutschen

In Teilschritt 1 bildet die erste Zeile (Erstpräferenz) jeder Matrix eine summierbare Stimme und mit Algorithmus 2.1 wird die verschlüsselte Anzahl an Stimmen pro Kandidat berechnet. Durch Ausführung von *get\_last\_candidate* kann der Kandidat mit der geringsten Anzahl an Stimmen bestimmt werden.

In Teilschritt 2 wird in jeder Matrix die Spalte gelöscht, welche die Platzierung des eliminierten Kandidaten angegeben hat. Es entsteht eine Matrix mit einer verschlüsselten Nullzeile (genau die Zeile, welche in der gelöschten Spalte eine verschlüsselte 1 enthielt). Allerdings bleibt bei jeder Matrix unbekannt, welche Zeile die Nullzeile ist. Ansonsten wäre ersichtlich, auf welchem Platz der Wähler den eliminierten Kandidaten platziert hatte.

Teilschritt 3 („Hochrutschen“) bildet den zeitaufwändigsten Teil dieses Verfahrens. Ziel von Teilschritt 3 ist es, die Nullzeile aus jeder Matrix zu löschen:

---

#### Algorithmus 4.1 Hochrutschen

---

```

1: procedure HOCHRUTSCHEN(Matrix m)
2:   for all row in range(num_rows(m) - 1) do
3:     sum_row  $\leftarrow$  Enc(0)
4:     for all column in range(num_columns(m)) do
5:       sum_row  $\leftarrow$  sum_row + m[row, column]
6:     end for
7:     sum_inverted  $\leftarrow$  Enc(1) - sum_row
8:     for all column in range(num_columns(m)) do
9:       move_up  $\leftarrow$  sum_inverted · m[row + 1, column]
10:      m[row, column]  $\leftarrow$  m[row, column] + move_up
11:      m[row + 1, column]  $\leftarrow$  m[row + 1, column] - move_up
12:    end for
13:  end for
14: end procedure

```

---

Die nach Ausführung von Algorithmus 4.1 entstandene Matrix stellt die Rangliste unter den noch nicht eliminierten Kandidaten dar. Die letzte Zeile bildet die Nullzeile und muss im weiteren Verlauf nicht mehr berücksichtigt werden, sie kann deshalb aus der Matrix „gestrichen“ werden.

Für die Laufzeit ist Zeile 9 entscheidend, denn hier wird eine Multiplikation zweier Geheimtexte durchgeführt (benötigt Trustee-Kommunikation). Nur wenn die Summe der Zeile eine verschlüsselte 0 ist und der Eintrag darunter eine verschlüsselte 1, dann ist *move\_up* eine verschlüsselte 1 und damit „rutscht der Eintrag darunter hoch“.

#### 4.2.2 Variante mit Zähler

Um eine Multiplikation pro Matrixeintrag bei jedem Hochrutschen zu vermeiden, wurde eine weitere Variante implementiert. Die Teilschritte 1 und 2 des bisherigen Verfahrens werden weiterhin unverändert durchgeführt. In Teilschritt 3 wird die Nullzeile nicht mehr eliminiert, sondern ein verschlüsselter Zähler berechnet, der auf die oberste Nicht-Null-Zeile zeigt:

Wir bezeichnen die verschlüsselte Summe einer Zeile *row* mit  $sum\_row(row)$  und die Invertierung des Summen-Bits mit  $sum\_row\_inv(row)$ . Die Berechnung der Summe bzw. invertierten Summe wird wie in Algorithmus 4.1 Zeile 4 bis 7 durchgeführt.

Desweiteren bezeichnen wir die Anzahl bisher eliminierter Kandidaten mit  $num\_elim$  (d.h.  $num\_elim = 1$  beim ersten Aufruf von Algorithmus 4.2).

---

#### Algorithmus 4.2 Berechnung des Zählers

---

```

1: function COMPUTE_COUNTER()
2:   counter ←  $sum\_row\_inv(0)$ 
3:   b ← counter
4:   for all row in range(1, num_elim) do
5:     b ← b ·  $sum\_row\_inv(row)$ 
6:     counter ← counter + b
7:   end for
8:   counter ← counter +  $Enc(1)$ 
9:   return counter
10: end function

```

---

Nach Ausführung von Zeile 5 für Matrix-Zeile *row* gilt für das Bit  $b = \begin{cases} Enc(1) & \text{falls } row < i \\ Enc(0) & \text{sonst} \end{cases}$  wobei *i* der Index der ersten Nicht-Null-Zeile sei. Der Zähler wird inkrementiert solange  $b = Enc(1)$ .

Im Anschluss kann mithilfe des Zählers die neue Erstpräferenz als summierbare Stimme eines Wählers ermittelt werden (Algorithmus 4.3).

Dabei wird der Ranking-Vektor *rank* verwendet, der vor erstmaliger Ausführung des ersten Teilschrittes für jede Stimme aus der Matrixdarstellung der abgegebenen Wählerstimme berechnet wird, vgl. Algorithmus 2.2. Bei Entfernen der Matrixspalte in Teilschritt 2 wird stets zugleich der Eintrag des eliminierten Kandidaten im Ranking-Vektor *rank* entfernt.

**Algorithmus 4.3** Berechnung der neuen Erstpräferenz mithilfe des Zählers

---

```

1: function COMPUTE_VOTE(rank, counter)
2:   vote ← []
3:   for all entry in range(length(rank)) do
4:     eq ← (rank[entry] = counter)    // Vergleichsoperator mit Trustee-Kommunikation
5:     vote.append(eq)
6:   end for
7:   return vote
8: end function

```

---

Ab der zweiten Iteration werden die mit dieser Funktion berechneten Erstpräferenzen jedes Wählers summiert und zur Bestimmung des nächsten zu eliminierenden Kandidaten eingesetzt. Nach der ersten Iteration kann dazu nämlich nicht mehr die erste Zeile der Matrix genutzt werden, da sie eine Nullzeile sein könnte und bei der Variante mit Zähler kein Hochrutschen von Matrixeinträgen durchgeführt wird.

## 4.3 Verfahren mit geheimer Elimination

Nun werden zwei weitere Varianten zur Auswertung des IRV vorgestellt. Sie basieren auf einer Veränderung und Erweiterung der beiden Varianten mit öffentlicher Elimination, sodass die Reihenfolge der Eliminationen nun geheim bleibt und lediglich der endgültige Wahlsieger veröffentlicht wird.

### 4.3.1 Variante mit Matrixmanipulation

Dieses Verfahren knüpft an die Variante mit Matrixmanipulation und öffentlicher Elimination an. Der grundsätzliche Aufbau mit den drei Teilschritten bleibt erhalten. *get\_last\_candidate* berechnet einen verschlüsselten Bitvektor mit Einträgen  $Enc(0)$  oder  $Enc(1)$  für jeden Kandidaten, je nachdem ob dieser Kandidat die wenigsten Stimmen erreicht hat und eliminiert wird (verschlüsselte 1) oder nicht (verschlüsselte 0).

Bei den Varianten mit öffentlicher Elimination werden diese Einträge entschlüsselt, bis der erste Kandidat mit den wenigsten Stimmen gefunden wurde.

Für die geheime Elimination entschlüsseln wir den Bitvektor nicht. Wir nehmen an, dass nur genau einer der Kandidaten die wenigsten Stimmen hat und den Eintrag  $Enc(1)$  erhält. Für Details zur Auflösung von Stimmgleichheit sei auf Abschnitt 4.5 verwiesen.

Teilschritt 2 wird nun komplexer, da das Löschen von Spalten nicht mehr möglich ist. Ziel dieses Teilschrittes ist es, bei jeder Wählerstimme die verschlüsselte Eins für die Platzierung des eliminierten Kandidaten auf eine verschlüsselte 0 zu setzen. Es entsteht, analog zum Löschen einer Spalte bei der öffentlichen Variante, genau eine Nullzeile.

Sei  $b$  der verschlüsselte Bitvektor, der von *get\_last\_candidate* berechnet wurde. Der  $i$ -te Eintrag in  $b$  referenziert den Kandidaten mit ID  $i$ . Teilschritt 2 wird nun, wie durch folgenden Algorithmus beschrieben, durchgeführt.

**Algorithmus 4.4** Teilschritt 2 (Variante mit Matrixmanipulation, geheime Elimination)

---

```

1: function COMPUTE_ELIMINATED_MATRIX(Matrix m)
2:   for all row in range(num_rows(m)) do
3:     for all column in range(num_columns(m)) do
4:        $m[\text{row}, \text{column}] \leftarrow m[\text{row}, \text{column}] - m[\text{row}, \text{column}] \cdot b[\text{column}]$ 
5:     end for
6:   end for
7:   return m
8: end function

```

---

Mit der erhaltenen Matrix kann das Hochrutschen (Teilschritt 3), genau wie mit Algorithmus 4.1 beschrieben, durchgeführt werden. Zu beachten ist, dass die Matrix  $m$  stets unverändert in der Spaltenanzahl bleibt, da kein Entfernen von Spalten in Teilschritt 2 möglich ist. Allerdings kann auch hier nach jedem Hochrutschen die letzte Zeile der Matrix gelöscht werden, da diese wiederum eine Nullzeile ist.

Vor erneutem Aufruf von *get\_last\_candidate* muss noch beachtet werden, dass durch das fehlende Löschen von Spalten bereits eliminierte Kandidaten wieder eliminiert werden würden. Dies wird wie folgt verhindert: Nach jeder Ausführung von *get\_last\_candidate* wird der ausgegebene verschlüsselte Bitvektor eintragsweise mit der Konstanten „Anzahl an Wählern + 1“ multipliziert. Wenn die durch diese Operation erhaltene Vektoren unterschiedlicher Iterationen eintragsweise summiert werden, entsteht ein neuer Vektor  $v$  mit Einträgen  $v_1, \dots, v_n$  und

$$v_i = \begin{cases} \text{Enc}(\text{Anzahl an Wählern} + 1) & \text{falls Kandidat mit ID } i \text{ bereits eliminiert wurde} \\ \text{Enc}(0) & \text{sonst} \end{cases}$$

Vor Aufruf von *get\_last\_candidate* wird  $v_i$  für jeden Kandidaten auf die Summe dessen Erstpräferenzen addiert, sodass alle eliminierte Kandidaten mehr Stimmen als noch nicht eliminierte Kandidaten haben und von *get\_last\_candidate* nicht erneut zur Elimination ausgewählt werden.

Die Einträge  $v_i$  können außerdem nach allen  $n - 1$  Eliminationen entschlüsselt werden, um den Sieger zu bestimmen (für den Sieger ergibt sich eine null, alle anderen Kandidaten haben den entschlüsselten Eintrag „Anzahl an Wählern + 1“).

### 4.3.2 Variante mit Zähler

Auch mit geheimer Elimination lässt sich ein Zähler berechnen. Eine Möglichkeit wäre es, Algorithmus 4.4 durchzuführen und auf der resultierenden Matrix den Zähler analog zur Variante mit öffentlicher Elimination zu berechnen. Allerdings wäre dieses Verfahren ineffizienter als das vorherige Verfahren aus Abschnitt 4.3.1, denn aufgrund des fehlenden Hochrutschens wäre ein Streichen der letzten Zeile pro Iteration nicht mehr möglich. Algorithmus 4.4 müsste also immer für alle Zeilen der ursprünglichen Matrix durchgeführt werden, anstatt die Algorithmen 4.4 und 4.1 jeweils „im Durchschnitt“ auf der Hälfte aller ursprünglichen Zeilen.

Der Zähler würde demnach ausschließlich zusätzlichen Aufwand bedeuten.

Eine andere Lösung basiert auf der Berechnung des Zählers ohne vorherige Durchführung von Algorithmus 4.4. Teilschritt 1 wird unverändert durchgeführt und liefert den Bitvektor  $b$ . Mit  $b$  lässt sich ohne Kommunikation der Trustees die verschlüsselte ID des eliminierten Kandidaten



berechnen. Dazu wird jeder Eintrag in  $b$  mit der ID des referenzierenden Kandidaten multipliziert und die Summe über alle erhaltenen Produkte berechnet. Wir bezeichnen die verschlüsselte ID des eliminierten Kandidaten mit  $Enc(ID)$ . Außerdem sei  $order$  die Order-Darstellung der Wählerstimme (siehe Def. 7). Sie wird für jeden Wähler anfangs einmalig aus der Matrix-Darstellung berechnet, ohne dass dazu Kommunikation der Trustees nötig ist.

Nun wird für jeden Wähler mit Algorithmus 4.5 der „Eliminationsvektor“  $e$  berechnet.

---

**Algorithmus 4.5** Berechnung des Eliminationsvektors
 

---

```

1: function COMPUTE_ELIMINATION_VECTOR(Order order)
2:    $e \leftarrow []$ 
3:   for all  $entry$  in  $range(length(order))$  do
4:      $eq \leftarrow (order[entry] = Enc(ID))$  // Vergleichsoperator mit Trustee-Kommunikation
5:      $e.append(eq)$ 
6:   end for
7:   return  $e$ 
8: end function

```

---

Der Eliminationsvektor  $e$  eines Wählers hat den  $i$ -ten Eintrag  $Enc(1)$ , falls der auf Platz  $i$  gewählte Kandidat zuletzt eliminiert wurde. Ansonsten hat der Eintrag den Wert  $Enc(0)$ .

Der nun erhaltene Eliminationsvektor wird auf die Eliminationsvektoren des Wählers aus früheren Iterationen addiert. Dadurch ergibt sich der „summierte Eliminationsvektor“  $sum\_elim$  mit dem Wert  $Enc(1)$  auf allen Plätzen, deren Kandidat bereits eliminiert wurde.

Betrachte nun die Variante mit Zähler bei der öffentlichen Elimination. Der Zähler kann mit Algorithmus 4.2 berechnet werden, wenn die Summe einer Zeile der Matrixdarstellung bekannt ist. Ersetze nun jeweils  $sum\_row\_inv(row)$  durch  $sum\_elim[row]$  im Algorithmus zur Berechnung des Zählers. Dieses Vorgehen ist korrekt, da die invertierte Summe der  $i$ -ten Zeile genau dann eine verschlüsselte Eins ist, wenn der auf Platz  $i$  platzierte Kandidat bereits eliminiert wurde.

Im weiteren Verlauf einer Iteration kann nahezu analog zur Variante der öffentlichen Elimination mit Zähler vorgegangen werden. Der Unterschied ist, dass beim Ranking-Vektor  $rank$  keine Einträge gelöscht werden können, da der eliminierte Kandidat unbekannt bleibt. Entsprechend umfasst die durch Algorithmus 4.3 berechnete neue Erstpräferenz auch bereits eliminierte Kandidaten. Um zu verhindern, dass diese nochmals eliminiert werden, müssen Stimmen für bereits eliminierte Kandidaten addiert werden (analog zur Variante aus Abschnitt 4.3.1).

## 4.4 Beispielhafte Durchführung der Verfahren

Wir betrachten eine Wahl mit 4 Kandidaten  $K_1, K_2, K_3, K_4$ , wobei der Index jeweils der ID des Kandidaten entspricht. Angenommen, bei der Auswertung wird zuerst  $K_3$ , dann  $K_2$  und dann  $K_4$  eliminiert,  $K_1$  ist also der Sieger. Max gibt die folgende Matrix als Wählerstimme ab, wobei er die

angegebenen Einträge vor der Wahl natürlich noch probabilistisch verschlüsselt hat:

$$\begin{pmatrix} 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Also ist der Kandidat  $K_3$  Max' erste Wahl, dann  $K_1$ , dann  $K_2$  und zuletzt  $K_4$ . Wir schauen nun, was mit Max' Wählerstimme während der Auswertung passiert. Dabei werden stets die entschlüsselten Matrixeinträge angegeben, auch wenn eigentlich keiner von Max' Matrixeinträgen während der Auswertung der Wahl von den Trustees entschlüsselt werden darf.

#### 4.4.1 Öffentliche Elimination: Variante mit Matrixmanipulation

1. Aufruf von *get\_last\_candidate*: Dazu werden die ersten Zeilen der Matrix summiert. Für Max fließt der Vektor  $(Enc(0), Enc(0), Enc(1), Enc(0))$  in die Summe ein. Doch Max' bevorzugter Kandidat  $K_3$  erhält leider trotzdem in Summe die wenigsten Stimmen und wird deshalb von *get\_last\_candidate* als zu eliminierender Kandidat bekannt gegeben.

2. Streichen der Matrixspalte des eliminierten Kandidaten liefert die Matrix 
$$\begin{pmatrix} 0 & 0 & 0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Hochrutschen liefert die Matrix 
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

Bei Max ergibt sich die letzte Zeile als Nullzeile und kann, wie auch bei jeder anderen Wählerstimme, gestrichen werden.

4. Aufruf von *get\_last\_candidate*: Für Max fließt der Vektor  $(Enc(1), Enc(0), Enc(0))$  in die neue Summe ein. Zu beachten ist, dass der dritte zu summierende Eintrag jetzt für  $K_4$  steht. Nun bekommt  $K_2$  die wenigsten Wählerstimmen und wird deshalb eliminiert.

5. Streichen der Matrixspalte des eliminierten Kandidaten liefert die Matrix 
$$\begin{pmatrix} 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

6. Hochrutschen und Streichen der letzten Zeile liefert die Matrix 
$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

7. Aufruf von *get\_last\_candidate*: Für Max fließt der Vektor  $(Enc(1), Enc(0))$  in die neue Summe ein. Der erste Eintrag steht für  $K_1$  und der zweite Eintrag für  $K_4$ .  $K_4$  erhält nun am wenigsten Stimmen und wird eliminiert. Max' zweite Präferenz  $K_1$  gewinnt somit die Wahl.

Hinweis: An diesem Beispiel unschwer zu erkennen ist, dass die letzte Zeile einer Matrix niemals zur Erstpräferenz während der Auswertung hochrutschen kann. Daraus ergibt sich die erste der in 4.7.3 vorgestellten Verbesserungen. Aus Gründen der besseren Verständlichkeit werden die in 4.7.3 vorgestellten Verbesserungen bei der Erklärung der Varianten nicht umgesetzt.

#### 4.4.2 Öffentliche Elimination: Variante mit Zähler

Bei der Variante mit Zähler wird am Anfang für jede Wählerstimme der Ranking-Vektor aus der Matrixdarstellung berechnet, ohne dass dabei die Trustees kommunizieren müssen. Für Max ergibt sich der Vektor  $(Enc(2), Enc(3), Enc(1), Enc(4))$ , der für jeden Kandidaten dessen Platzierung verschlüsselt.

1. Aufruf von *get\_last\_candidate* exakt gleich wie zuvor
2. Streichen der Matrixspalte exakt gleich wie zuvor. Zusätzlich wird auch der Eintrag des eliminierten Kandidaten im Ranking-Vektor gestrichen. Wir erhalten den neuen Ranking-Vektor  $(Enc(2), Enc(3), Enc(4))$
3. Berechnung des Zählers: Der Zähler ergibt sich in der ersten Iteration aus der invertierten Summe der ersten Zeile der Matrix.  
Wir erhalten also  $Enc(1) - (Enc(0) + Enc(0) + Enc(0)) = Enc(1)$ , da Max' Erstpräferenz bereits eliminiert wurde. Außerdem wird der Zähler in Zeile 8 von Algorithmus 4.2 noch inkrementiert, sodass er den Wert  $Enc(2)$  annimmt.
4. Berechnung der neuen Erstpräferenz: Für jeden Eintrag im Ranking-Vektor wird ein verschlüsselter '='-Vergleich mit dem Zähler durchgeführt. Da der erste Eintrag den gleichen Wert wie der Zähler verschlüsselt, ergibt sich der neue Vektor  $(Enc(1), Enc(0), Enc(0))$ .
5. Aufruf von *get\_last\_candidate*: Für Max fließt der zuvor berechnete Vektor  $(Enc(1), Enc(0), Enc(0))$  in die neue Summe ein. Dieser entspricht dem Vektor bei der Variante mit Matrixmanipulation (was natürlich auch zwingend so sein sollte). Nun bekommt  $K_2$  die wenigsten Wählerstimmen und wird deshalb eliminiert.
6. Streichen der Matrixspalte: Da kein Hochrutschen bei der Matrix stattfindet, ergibt sich:

$$\begin{pmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{pmatrix}$$

Außerdem erhalten wir den aktualisierten Ranking-Vektor  $(Enc(2), Enc(4))$ .

7. Berechnung des Zählers: Wie zuvor ergibt sich  $Enc(1)$  für die invertierte Summe der ersten Zeile der Matrix. Nun wird in Alg. 4.2 ein Durchlauf der For-Schleife durchgeführt.  
 $b$  hat in Zeile 5 zwar den Wert  $Enc(1)$ , aber die invertierte Summe der zweiten Matrixzeile hat den Wert  $Enc(0)$ . Deshalb wird der Zähler in der Schleife nicht weiter erhöht. Nach Zeile 8 hat der Zähler wiederum den Wert  $Enc(2)$ .
8. Berechnung der neuen Erstpräferenz: Nach den '='-Vergleichen mit dem Ranking-Vektor ergibt sich der neue Vektor  $(Enc(1), Enc(0))$ , also wurde Max' neue Erstpräferenz für den letzten Aufruf von *get\_last\_candidate* wiederum erfolgreich berechnet.

### 4.4.3 Geheime Elimination: Variante mit Matrixmanipulation

1. Aufruf von *get\_last\_candidate*: Dazu werden die ersten Zeilen der Matrix summiert. Für Max fließt der Vektor  $(Enc(0), Enc(0), Enc(1), Enc(0))$  in die Summe ein. Bei dieser Variante gibt *get\_last\_candidate* den verschlüsselten Bitvektor  $b = (Enc(0), Enc(0), Enc(1), Enc(0))$  zurück. Max wird also niemals erfahren, dass seine Erstpräferenz  $K_3$  bereits in der ersten Iteration eliminiert wurde.
2. Teilschritt 2: Das Löschen einer Matrixspalte ist nicht möglich, da der eliminierte Kandidat unbekannt bleibt. Die Trustees führen also Algorithmus 4.4 aus. Das Resultat ist die Matrix

$$\begin{pmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

3. Hochrutschen liefert die Matrix  $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$

Bei Max ist die letzte Zeile nun eine Nullzeile und kann, wie auch in jeder anderen Wählerstimme, gestrichen werden.

4. Aufruf von *get\_last\_candidate*: Für Max fließt der Vektor  $(Enc(1), Enc(0), Enc(0), Enc(0))$  in die neue Summe ein. Zu beachten ist, dass der dritte Eintrag für den bereits eliminierten Kandidaten  $K_3$  bei jedem Wähler den Wert  $Enc(0)$  hat. Allerdings wird der in Schritt 1 erhaltene Bitvektor  $b$  mit einer großen Konstante  $k$  multipliziert und auf die Summe der Stimmen addiert.

Für  $k$  wird die „Anzahl an Wählern + 1“ gewählt. Kandidat  $K_3$  erhält dadurch so viele zusätzliche Stimmen, dass er nicht erneut eliminiert wird. Stattdessen liefert *get\_last\_candidate* nun den neuen Bitvektor  $b = (Enc(0), Enc(1), Enc(0), Enc(0))$ , der die Elimination von  $K_2$  verschlüsselt.

5. Teilschritt 2 liefert die Matrix  $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

6. Hochrutschen und Streichen der letzten Zeile liefert die Matrix  $\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

7. Aufruf von *get\_last\_candidate*: Für Max fließt der Vektor  $(Enc(1), Enc(0), Enc(0), Enc(0))$  in die neue Summe ein. Allerdings wird die Summe beider mit  $k$  multiplizierter Bitvektoren  $b$ , also  $(Enc(0), Enc(k), Enc(k), Enc(0))$ , eintragsweise auf die Summe der Stimmen addiert. Deshalb können nur  $K_1$  oder  $K_4$  die wenigsten Stimmen erhalten.  $K_4$  erhält die wenigsten Stimmen, d.h. *get\_last\_candidate* liefert den Bitvektor  $b = (Enc(0), Enc(0), Enc(0), Enc(1))$ .

8. Nun sind alle drei Eliminationen durchgeführt. Zuletzt wird noch die Summe der mit  $k$  multiplizierten Bitvektoren entschlüsselt und veröffentlicht, es ergibt sich der Vektor  $(0, k, k, k)$ . Dieser verrät uns nur, dass  $K_1$  die Wahl gewonnen hat.

#### 4.4.4 Geheime Elimination: Variante mit Zähler

Analog zur Variante mit öffentlicher Elimination wird am Anfang für jede Wählerstimme der Ranking-Vektor aus der Matrixdarstellung berechnet. Für Max ergibt sich der Vektor  $(Enc(2), Enc(3), Enc(1), Enc(4))$ , der für jeden Kandidaten dessen Platzierung verschlüsselt.

Bei geheimer Elimination wird außerdem noch der Order-Vektor gebraucht, der für jeden Platz die ID des Kandidaten verschlüsselt. Für Max ergibt sich der Vektor  $(Enc(3), Enc(1), Enc(2), Enc(4))$ . Genau wie beim Ranking-Vektor ist zur Berechnung keine Trustee-Kommunikation nötig.

1. Der Aufruf von *get\_last\_candidate* verläuft exakt gleich wie bei der geheimen Variante mit Matrixmanipulation. Mit dem resultierenden Bitvektor  $(Enc(0), Enc(0), Enc(1), Enc(0))$  wird nun die verschlüsselte ID des eliminierten Kandidaten berechnet:  $1 \cdot Enc(0) + 2 \cdot Enc(0) + 3 \cdot Enc(1) + 4 \cdot Enc(0) = Enc(3)$
2. Berechnung des Eliminationsvektors: Nun werden für jeden Wähler '='-Vergleiche der Einträge des Order-Vektors mit der verschlüsselten ID des eliminierten Kandidaten durchgeführt. Für Max ergibt sich dadurch der Eliminationsvektor mit Einträgen  $(Enc(1), Enc(0), Enc(0), Enc(0))$ .  
Dieser Vektor ist zugleich der „summierte Eliminationsvektor“ von Max, da noch keine Eliminationsvektoren zuvor berechnet wurden. Er verschlüsselt die Information, dass Max' ursprüngliche Erstpräferenz bereits eliminiert wurde.
3. Berechnung des Zählers: Der einzige Unterschied zur öffentlichen Elimination mit Zähler ist, dass statt der invertierten Summe der ersten Zeile der Matrix nun der erste Eintrag des summierten Eliminationsvektors verwendet wird. Der entsprechende Eintrag des summierten Eliminationsvektors stimmt stets mit der invertierten Summe bei der öffentlichen Variante überein. Also nimmt der Zähler wieder den Wert  $Enc(2)$  an.
4. Berechnung der neuen Erstpräferenz: Auch das klappt fast gleich wie bei der öffentlichen Variante mit Zähler. Beim Ranking-Vektor konnte kein Eintrag gelöscht werden, da der eliminierte Kandidat unbekannt ist. Entsprechend tauchen auch bereits eliminierte Kandidaten wieder in der neuen Erstpräferenz auf:  $(Enc(1), Enc(0), Enc(0), Enc(0))$
5. Aufruf von *get\_last\_candidate* wieder exakt gleich wie bei geheimer Variante mit Matrixmanipulation. Wir erhalten den resultierenden Bitvektor  $(Enc(0), Enc(1), Enc(0), Enc(0))$ . Anschließend die Berechnung der verschlüsselten ID des eliminierten Kandidaten:  $1 \cdot Enc(0) + 2 \cdot Enc(1) + 3 \cdot Enc(0) + 4 \cdot Enc(0) = Enc(2)$
6. Berechnung des Eliminationsvektors: Für Max ergibt sich der Eliminationsvektor mit Einträgen  $(Enc(0), Enc(0), Enc(1), Enc(0))$ . Die Summe beider bisher berechneter Eliminationsvektoren bilden Max' neuen „summierten Eliminationsvektor“  $(Enc(1), Enc(0), Enc(1), Enc(0))$ . Der summierte Eliminationsvektor verschlüsselt die Information, dass der erstplatzierte und der drittplatzierte Kandidat von Max bereits eliminiert wurden.
7. Berechnung des Zählers: Betrachte die Matrix aus Teilschritt 6 der öffentlichen Variante mit Zähler. Die invertierten Summen der Zeilen stimmen mit den Einträgen des gerade berechneten summierten Eliminationsvektors überein. Es ergibt sich also wieder der Zähler mit dem Wert  $Enc(2)$ .

8. Berechnung der neuen Erstpräferenz: '='-Vergleiche des Zählers mit dem Ranking-Vektor liefern die neue Erstpräferenz ( $Enc(1), Enc(0), Enc(0), Enc(0)$ ). Diese würde mit der Erstpräferenz der öffentlichen Variante übereinstimmen, wenn vor der Berechnung die Einträge des Ranking-Vektors für eliminierte Kandidaten gelöscht wären.

Da bei geheimer Elimination kein Löschen möglich ist, enthält die neue Erstpräferenz auch eine verschlüsselte Null für bereits eliminierte Kandidaten. Allerdings wird beim abschließenden Aufruf von `get_last_candidate` analog zur geheimen Elimination mit Matrixmanipulation vorgegangen.

Dadurch wird wieder sichergestellt, dass die bereits eliminierten Kandidaten  $K_2$  und  $K_3$  nicht erneut eliminiert werden.

### 4.5 Vorgehen bei Stimmgleichheit

Bisher sind wir davon ausgegangen, dass genau ein Kandidat pro Iteration die wenigsten Stimmen hat und eliminiert wird. Diese Annahme ist offensichtlich im Allgemeinen nicht möglich.

`get_last_candidate` berechnet einen verschlüsselten Bitvektor  $b$  mit Einträgen  $Enc(0)$  oder  $Enc(1)$  für jeden Kandidaten, je nachdem ob dieser Kandidat die wenigsten Stimmen erreicht hat (verschlüsselte 1) oder nicht (verschlüsselte 0).

Anders als bei der Variante mit geheimer Elimination angenommen, sind mehrere Einträge mit Wert  $Enc(1)$  möglich.

Im folgenden wird diskutiert, wie die bisher vorgestellten Verfahren trotzdem korrekt durchgeführt werden können.

#### 4.5.1 Stimmgleichheit bei öffentlicher Elimination

Mit öffentlicher Elimination gibt es kein Problem bei Stimmgleichheit. Es gibt die folgenden beiden Möglichkeiten:

1. Die Einträge von  $b$  werden entschlüsselt, bis „die erste Eins gefunden wird“. Der entsprechende Kandidat wird eliminiert. Es bleibt daher unbekannt, ob sich weitere Kandidaten stimmgleich auf dem letzten Platz befanden. Die stimmgleichen Kandidaten werden in dieser Iteration nicht eliminiert.
2. Genauso lassen sich alle Einträge von  $b$  entschlüsseln und falls nötig mehrere Kandidaten eliminieren. Diese Mehrfachelimination wird als Folge einzelner Eliminationen durchgeführt.

Falls nur ein Kandidat eliminiert werden soll, lässt sich auch die in Def. 8 erwähnte Variante zur Auflösung von Stimmgleichheit anwenden.

Zuvor ist eine Reihenfolge der „Bevorzugung“ festzulegen oder auszulosen, sodass Stimmgleichheit als Sieg für den bevorzugten Kandidaten gewertet wird. Durch fehlende '='-Operatoren ergibt sich eine Laufzeitverbesserung gegenüber der ersten Variante ohne Mehrfachelimination.

### 4.5.2 Stimmgleichheit bei geheimer Elimination

Da die Einträge des Bitvektors  $b$  nicht entschlüsselt werden, ist die Auflösung von Stimmgleichheit aus Def. 8 nötig, um (ohne Mehrfachelimination) die korrekte Durchführung des Verfahrens zu gewährleisten.

Ohne erhebliche Änderungen lässt sich die Mehrfachelimination mit den vorgestellten Verfahren zur geheimen Elimination nur durchführen, wenn die Anzahl zu eliminierender Kandidaten bekannt gegeben wird, d.h. die Summe der Einträge in  $b$  entschlüsselt wird. Vor der Wahl muss beurteilt werden, ob diese Information bekannt gegeben werden darf.

Falls ja, dann lässt sich mit einem Verfahren ähnlich zur Berechnung des Zählers ein reduzierter Bitvektor  $b_{act}$  berechnen, der nur den Eintrag  $Enc(1)$  bei einem zu eliminierenden Kandidaten besitzt. Falls nach Durchführung der Elimination ein weiterer Kandidat zu eliminieren ist, wird  $b_{act}$  eintragsweise von  $b$  abgezogen und damit kann ein neuer Vektor  $b_{act}$  für den nächsten zu eliminierenden Kandidaten berechnet werden.

## 4.6 Alternatives Verfahren: Stimme mit $n!$ Optionen

Im kommenden Abschnitt wird ein weiteres Verfahren mit öffentlicher Elimination vorgestellt. Anschließend werden Veränderungen einzelner Teile des Verfahrens geschildert, woraus sich ein weiteres Verfahren mit geheimer Elimination ergibt. Das Ziel dieser beiden „alternativen“ Verfahren ist das Erreichen einer von der Anzahl teilnehmender Wähler unabhängigen Laufzeit.

### 4.6.1 Alternatives Verfahren: Öffentliche Elimination

Jeder Wähler wählt eine summierbare Stimme  $s \in \{1, \dots, n!\}$ , wobei jede Stimme  $s$  genau eine der  $n!$  möglichen Ranglisten identifiziert, die wir mit  $r(s, n)$  bezeichnen. Nach Elimination von  $t$  Kandidaten kennzeichne jeder Wert  $s \in \{1, \dots, (n-t)!\}$  genau eine der möglichen Ranglisten  $r(s, n-t)$  unter allen  $n-t$  noch nicht eliminierten Kandidaten.

Algorithmus 4.6 zeigt die Elimination- und Hochrutschen-Prozedur, wobei die Trustees nur bei den  $n-1$  Durchführungen von *get\_last\_candidate* kommunizieren. Insbesondere wird keine Trustee-Kommunikation für die Hochrutschen-Prozedur benötigt und die Anzahl durchzuführender Operationen mit Trustee-Kommunikation ist unabhängig von der Anzahl an Wählern. Allerdings ist zu beachten, dass der Wähler eine Stimme mit  $n!$  verschlüsselten Einträgen abgibt und die Trustees einen Algorithmus mit  $\Theta(n!)$ -Komplexität (genauerer siehe Kap. 4.7) durchführen.

Das Verfahren beruht auf der Idee, bei jeder Elimination durch alle möglichen Ranglisten zu iterieren: Für jeden Kandidaten wird die Summe an Stimmen von den Ranglisten berechnet, bei denen der jeweilige Kandidat die Erstpräferenz bildet. Beim Hochrutschen wird wieder über alle Ranglisten iteriert und die Stimmen jeder ursprünglichen Rangliste werden zur „aktualisierten“ Rangliste addiert, bei der der eliminierte Kandidat aus der ursprünglichen Rangliste entfernt wurde.

**Algorithmus 4.6** IRV in  $O(n!)$ 

$S$  sei die Menge der verschlüsselten Stimmen  $s$   
 $n\_act$  zählt die Anzahl an Kandidaten, die noch nicht eliminiert wurden. Für  $n\_act = 2$  ist also noch einmal die eliminiere-Funktion durchzuführen und anschließend ist das Gesamtergebnis bekannt.

```

1: initialize:  $n\_act \leftarrow n$ 
2: initialize:  $votes \leftarrow []$ 
3:
4: function ELIMINIERE()
5:   if  $n\_act == n$  then
6:      $votes \leftarrow compute\_Sum(S)$  // Summation der Stimmen [2.1]
7:   end if
8:    $votes\_per\_candidate \leftarrow []$ 
9:    $votes\_per\_candidate[j] \leftarrow Enc(0)$  for all  $j \in [\{1, \dots, n\} \setminus \{j \mid k_j \text{ bereits eliminiert}\}]$ 
10:  for all  $i \in \{1, \dots, n\_act!\}$  do // Alle möglichen Ranglisten von nicht eliminierten Kand.
11:     $first\_cand \leftarrow j \Leftrightarrow p(r(i, n\_act), j) = 1$  // Notation aus Def. 7
12:     $votes\_per\_candidate[first\_cand] \leftarrow votes\_per\_candidate[first\_cand] + votes[i]$ 
13:  end for
14:   $loser \leftarrow GET\_LAST\_CANDIDATE(votes\_per\_candidate)$  // Def 8
15:  return  $loser$ 
16: end function
17:
18: procedure HOCHRUTSCHEN( $loser$ )
19:   $new\_votes \leftarrow []$ 
20:   $new\_votes[i] \leftarrow Enc(0)$  for all  $i \in \{1, \dots, (n\_act - 1)!\}$ 
21:  for all  $old\_vote \in \{1, \dots, n\_act!\}$  do
22:     $correspond\_vote \leftarrow i \Leftrightarrow r(i, n\_act - 1) = r(old\_vote, n\_act)$  nach Streichen von  $loser$ 
23:     $new\_votes[correspond\_vote] \leftarrow new\_votes[correspond\_vote] + votes[old\_vote]$ 
24:  end for
25:   $votes \leftarrow new\_votes$ 
26:   $n\_act \leftarrow n\_act - 1$ 
27: end procedure

```

**4.6.2 Alternatives Verfahren: Geheime Elimination**

Für die geheime Elimination ist eine Anpassung von Algorithmus 4.6 denkbar. Die Eliminationsfunktion lässt sich analog durchführen, wobei sie pro Kandidat  $k$  ein verschlüsseltes Bit  $[k]$  zurückgibt, welches genau dann im Klartext 1 ist, wenn der Kandidat eliminiert wurde. Da der eliminierte Kandidat unbekannt bleibt, müssen Zeile 22 und 23 für jeden Kandidaten  $k$  (statt nur für  $loser$ ) durchgeführt werden.

Dabei erfordert Zeile 23 eine Multiplikation von Geheimtexten:

$$new\_votes[correspond\_vote] \leftarrow new\_votes[correspond\_vote] + votes[old\_vote] \cdot [k]$$



Desweiteren ist zu beachten: Nach dem  $t$ -ten Hochrutschen beträgt die Anzahl an möglichen Ranglisten statt  $(n-t)!$  nun  $n \cdot (n-1) \cdot \dots \cdot (1+t)$ , was der Anzahl an  $(n-t)$ -Permutationen aus der Menge aller Kandidaten-IDs  $\{1, \dots, n\}$  entspricht.

Außerdem muss nun verhindert werden, dass eliminierte Kandidaten nochmal eliminiert werden. Dies kann durch Addition einer hohen Anzahl an Stimmen für eliminierte Kandidaten realisiert werden, so wie es auch in Abschnitt 4.3 durchgeführt wurde.

Der Gewinner kann nach  $n-1$  Eliminationen identifiziert werden, indem für jeden Kandidaten die nach den Eliminationen erhaltenen verschlüsselten Bits  $[k]$  summiert werden und im Anschluss die Summen entschlüsselt werden.

Das soeben beschriebene Verfahren mit geheimer Elimination erfordert Kommunikation der Trustees in der Hochrutschen-Prozedur. In einer Iteration ist eine Multiplikation in einer Schleife über alle Kandidaten und über alle noch möglichen Ranglisten durchzuführen. Daher ist das alternative Verfahren mit geheimer Elimination für eine steigende Anzahl an Kandidaten zunehmend ineffizienter als die Version mit öffentlicher Elimination (vgl. Kap 4.8).

## 4.7 Laufzeitanalyse

Bei den Varianten mit Matrixmanipulation und Zähler ist stets zu beachten, dass die angegebenen Operationen auf jeder Wählerstimme durchzuführen sind. Die Operationen für *get\_last\_candidate* und die gesamten alternativen Verfahren sind hingegen einmalig und nicht pro Wählerstimme durchzuführen.

### 4.7.1 Öffentliche Elimination

Bei allen Varianten mit öffentlicher Elimination muss *get\_last\_candidate* nur zwischen den noch nicht eliminierten Kandidaten auswerten. *get\_last\_candidate* ist also mit  $n, n-1, \dots, 2$  Kandidaten durchzuführen.

Für das alternative Verfahren 4.6 sind keine weiteren Operationen mit Kommunikation der Trustees durchzuführen. Bei jeder Iteration werden alle möglichen Ranglisten unter noch nicht eliminierten Kandidaten in einer lokalen Phase durchlaufen. Es lässt sich zeigen, dass die Anzahl durchzuführender Operationen in  $O(n!)$  liegt:

$$\sum_{i=1}^n i! = n! + \sum_{i=1}^{n-1} i! \leq n! + n \cdot (n-1)! = 2 \cdot n! \in O(n!)$$

Der Algorithmus führt also  $\Theta(n!)$  Operationen in der lokalen Phase durch.

Bei der Variante mit Matrixmanipulation wird weitere Kommunikation für das „Hochrutschen“ benötigt. Bei jeder Elimination wird eine Spalte gestrichen, nach jedem Hochrutschen wird die letzte Zeile gestrichen. Die Anzahl durchzuführender Multiplikationen beträgt:  $(n-1)^2$  in der ersten Iteration,  $(n-2)^2$  in der zweiten Iteration usw. Damit ergeben sich  $\Theta(n^3)$  Multiplikationen.

Bei der Variante mit Zähler werden Multiplikationen zur Berechnung des Zählers durchgeführt, wobei pro Iteration eine weitere Multiplikation durchzuführen ist. Es ergeben sich  $0 + 1 + \dots + (n-3)$  Multiplikationen. Außerdem werden  $(n-1) + (n-2) + \dots + 2$  ‚=‘-Vergleiche des Zählers mit dem

Ranking-Vektor durchgeführt.

Im Vergleich zur Matrixmanipulation wird demnach die Anzahl durchzuführender Operationen mit Trustee-Kommunikation pro Wählerstimme zu  $\Theta(n^2)$  reduziert. Allerdings werden jeweils zusätzlich  $\Theta(n^2)$  '='-Vergleiche benötigt.

### 4.7.2 Geheime Elimination

Bei den Varianten mit geheimer Elimination wird *get\_last\_candidate*  $n - 1$  Mal mit je  $n$  Kandidaten durchgeführt.

Für das alternative Verfahren wird Trustee-Kommunikation beim Hochrutschen benötigt. Es wird eine Multiplikation in einer Schleife über alle Kandidaten und über alle noch möglichen Ranglisten durchgeführt, insgesamt ergeben sich  $n \cdot \sum_{i=2}^n n \cdot (n - 1) \cdot \dots \cdot (n - i)$  Multiplikationen. Bereits in der ersten und zweiten Iteration werden jeweils  $n! \cdot n$  Multiplikationen durchgeführt.

Bei der Variante mit Matrixmanipulation mit geheimer Elimination ist kein Streichen von Spalten möglich. Pro Iteration kann weiterhin eine Zeile gestrichen werden. Daher sind insgesamt  $n^2 + n \cdot (n - 1) + n \cdot (n - 2) + \dots + n \cdot 3 \approx \frac{n^3}{2}$  Multiplikationen für Teilschritt 2 durchzuführen. Für das Hochrutschen wird nochmals annähernd die gleiche Anzahl an Multiplikationen benötigt.

Bei der Variante mit Zähler bleiben die Multiplikationen zur Berechnung des Zählers unverändert zur öffentlichen Variante. Es werden  $n - 2$  Mal je  $n$  '='-Vergleiche mit dem Ranking-Vektor durchgeführt, da der Ranking-Vektor unverändert in der Anzahl an Einträgen bleibt. Außerdem kommen  $n - 2$  Mal je  $n$  '='-Vergleiche zur Berechnung des Eliminationsvektors hinzu, die bei der öffentlichen Variante nicht benötigt wurden.

### 4.7.3 Weitere Verbesserungen der Laufzeit

Die Auswertung bei öffentlicher und geheimer Variante mit Matrixmanipulation kann effizienter durchgeführt werden, indem anfangs die letzte Zeile einer Wählermatrix gelöscht wird. Dies ist möglich, da die letzte Zeile niemals zur Erstpräferenz hochrutschen kann. Bei Elimination und Hochrutschen kann man sich also stets den Durchlauf der letzten Matrixzeile sparen.

Die geheime Elimination mit Zähler lässt sich ebenfalls effizienter durchführen, da die letzten beiden Durchläufe der For-Schleife in Algorithmus 4.5 überflüssig sind. Denn die Einträge der letzten beiden Durchläufe verschlüsseln die Information, ob die Kandidaten auf dem letzten und vorletzten Platz eliminiert wurden. Aber bei anschließender Berechnung des Zählers werden nur die Plätze 1 bis  $n - 2$  benötigt. Das wird auch dadurch offensichtlich, dass der Zähler niemals auf die letzte Zeile der Matrix zeigen kann und bei jeder Durchführung von Alg. 4.2 nur Matrixzeilen vor der letzten möglichen Position des Zählers betrachtet werden.

## 4.8 Testergebnisse

Um die Laufzeit der vier Verfahren (öffentlich/geheim mit Matrixmanipulation/Zähler) zu vergleichen, wurden 16 bit als Integerlänge für die Vergleichsoperatoren und 64 bit als Schlüssellänge verwendet. Diese Schlüssellänge ist für die Praxis untauglich, führt jedoch zu einer Reduktion der Laufzeit, wodurch sich die vier Verfahren auch für eine höhere Anzahl an Kandidaten und Wählern effizient testen lassen.

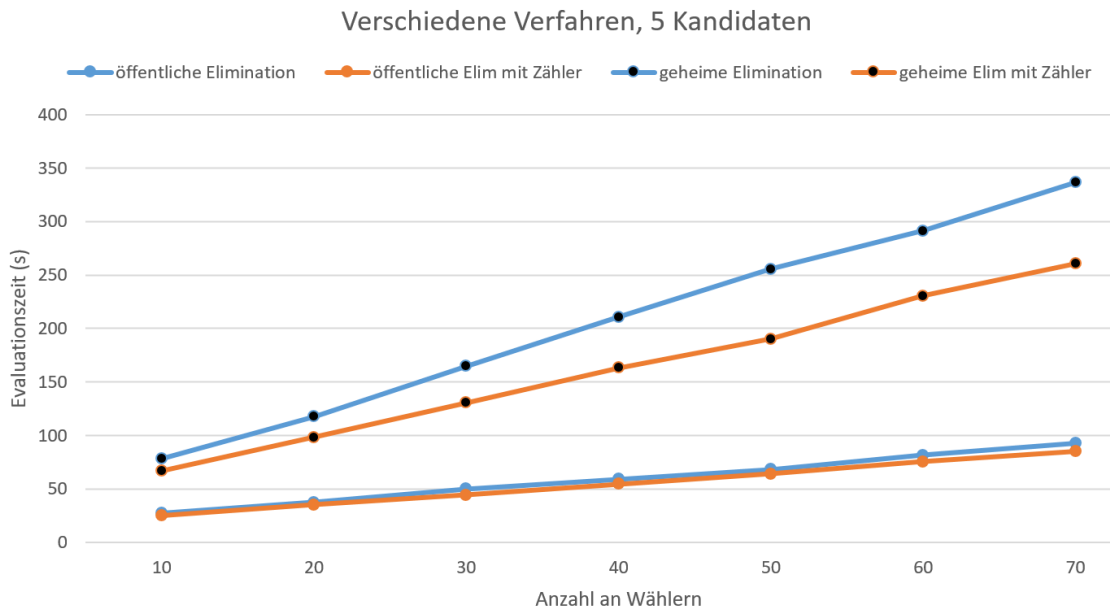


Abbildung 4.1: Variierende Anzahl an Wählern, 64-bit-Schlüssel

Abbildung 4.1 zeigt die Laufzeit der Verfahren für 5 Kandidaten. Dabei zeigt sich der lineare Verlauf in der Anzahl an Wählern. Die beiden Verfahren mit Zähler (orangene Kurven) haben eine geringere Laufzeit als die Verfahren mit Matrixmanipulation (blaue Kurven).

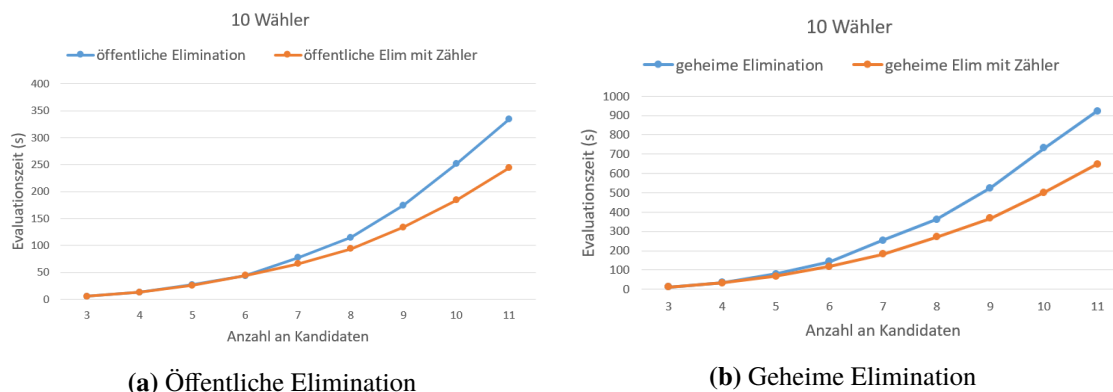


Abbildung 4.2: Variierende Anzahl an Kandidaten, 64-bit-Schlüssel

Abbildung 4.2 variiert die Anzahl an Kandidaten, wobei die Wählerzahl auf 10 Wähler festgesetzt wird. Dabei deutet sich der erwartete quadratische Verlauf der orangenen Kurve und der kubische Verlauf der blauen Kurve an. Bei Betrachtung der exakten Zahlenwerte sind für drei und vier Kandidaten noch keine Unterschiede in der Laufzeit auszumachen. Erst bei einer höheren Anzahl an Kandidaten zeigt sich der Vorteil des Zählers.

Eine für die Praxis derzeit realistische Schlüssellänge von 2048 bit führt zu erheblichen Änderungen der Laufzeiten. Die Ausführungszeit des '='-Operators stieg von ca. 0.1 auf 11 Sekunden an, wohingegen eine Multiplikation von Geheimtexten nur um den Faktor 70 auf ca. 1.5 Sekunden anstieg. Die Messungen wurden auf einer Maschine mit vier Kernen und drei Trustees durchgeführt. Die deutlich erhöhte Ausführungszeit des '='-Operators führt dazu, dass die Variante mit Zähler nun für drei und vier Kandidaten deutlich ineffizienter als die Variante mit Matrixmanipulation ist (siehe Abbildung 4.3). Die Komplexität in  $O$ -Notation ist in der Praxis nicht aussagekräftig, da die vier Verfahren in realistischer Zeit (maximal 6-10 Stunden) nur mit einer geringen Anzahl an Kandidaten durchführbar sind.

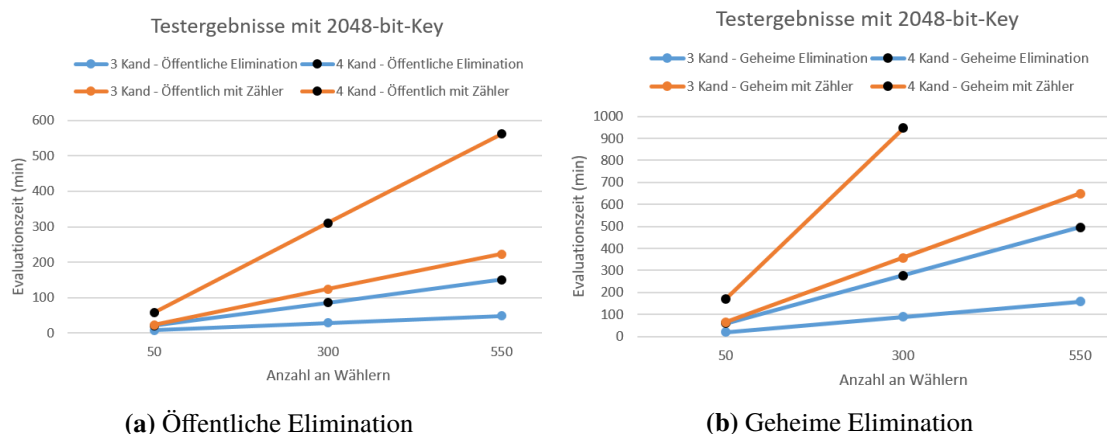


Abbildung 4.3: Variierende Anzahl an Wählern, 2048-bit-Schlüssel

Trotz  $O(n!)$ -Komplexität bieten die alternativen Verfahren aus Abschnitt 4.6 für die meisten Szenarien geringere Ausführungszeiten. Der entscheidende Vorteil ist, dass die Anzahl an Operationen mit Trustee-Kommunikation nicht von der Anzahl an Wählern abhängig ist.

Die Summation der Stimmen aus Algorithmus 4.6 in Zeile 6 wurde getrennt als „Vorbereitungszeit“ gezählt. Bei einer Million Wähler und drei Kandidaten werden eine Million „Dictionaries“ mit je  $3! = 6$  Einträgen gelesen und pro Eintrag eine verschlüsselte Addition von Geheimtexten durchgeführt. Die gemessene Vorbereitungszeit betrug 10 Minuten. Für drei Kandidaten und 50 bzw. 500 Wähler lag die Vorbereitungszeit im vernachlässigbaren Bereich weniger Sekunden. Die Vorbereitungszeit steigt auch mit der Kandidatenzahl aufgrund der steigenden Zahl verschlüsselter Einträge pro Wählerstimme.

Außerdem ist zu beachten, dass jeder Wähler  $n!$  Verschlüsselungen durchführen muss, was bereits bei 6 Kandidaten je nach Rechner des Wählers einige Minuten in Anspruch nehmen kann.

Die gemessene Evaluationszeit (ohne Vorbereitungszeit!) der alternativen Verfahren blieb auch für eine Million Wähler unverändert zu den Zeiten mit 50 oder 500 Wählern. Die Evaluationszeit ist in den Diagrammen 4.4 und 4.5 dargestellt (schwarze Kurve). Zum Vergleich sind die Verfahren mit Matrixmanipulation und Zähler für je 5 und 10 Wähler dargestellt.

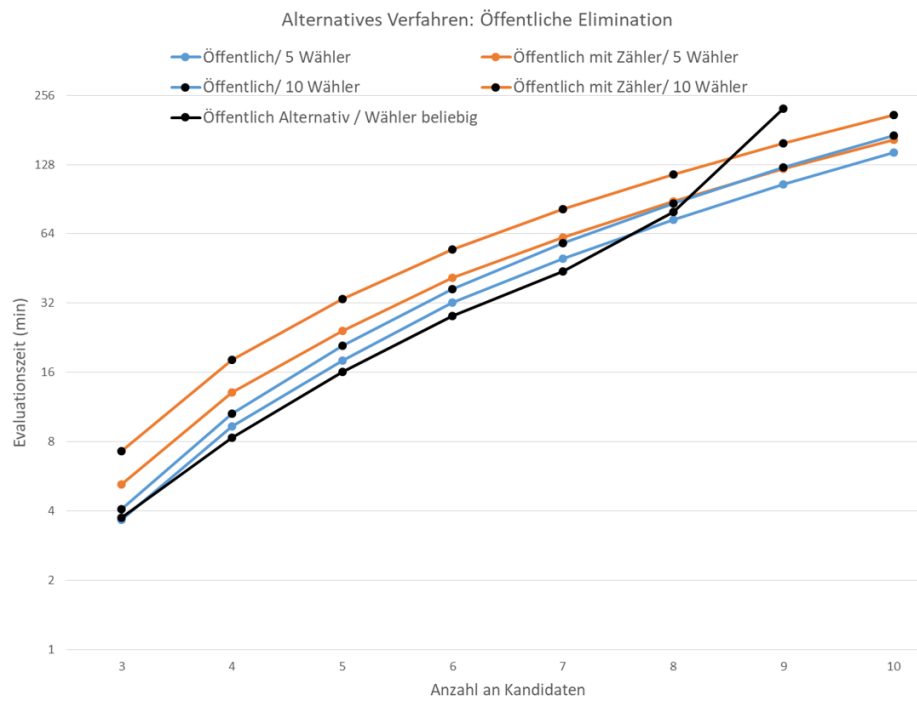


Abbildung 4.4: Öffentliches alternatives Verfahren im Vergleich, 2048-bit-Schlüssel

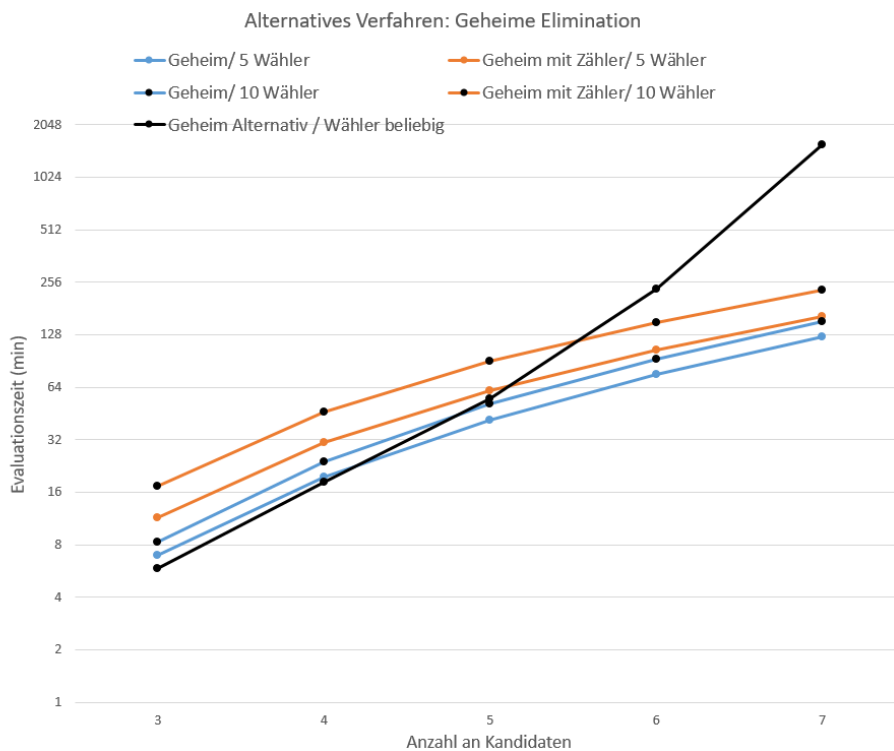


Abbildung 4.5: Geheimes alternatives Verfahren im Vergleich, 2048-bit-Schlüssel

Es ist erkennbar, dass die Verfahren mit Matrixmanipulation und Zähler in der Evaluationszeit konkurrenzfähig sind, wenn sehr wenige Wähler an der Wahl teilnehmen. Mit öffentlicher Elimination ist der Vorteil des alternativen Verfahrens stärker ausgeprägt, da keine Kommunikation der Trustees beim Hochrutschen benötigt wird.

Bei den alternativen Verfahren ist eine eindeutige obere Grenze an Kandidaten sichtbar, bis zu welcher die Verfahren - unabhängig von der Anzahl teilnehmender Wähler - in der Praxis durchführbar sind. An der logarithmischen Skalierung ist die  $O(n!)$ -Komplexität sehr gut zu erkennen, denn es ist  $\log(n!) \in \Theta(n \cdot \log(n))$

In den Abbildungen 4.6 werden die Ergebnisse aus den Abbildungen 4.3 mit der Ausführungszeit der alternativen Verfahren verglichen. Der Vergleich zeigt, dass die alternativen Verfahren bereits bei 50 Wählern überlegen sind. Bei weiterer Erhöhung der Wählerzahl werden die Unterschiede markant. Zu beachten ist die Darstellung mit logarithmischer Skala.

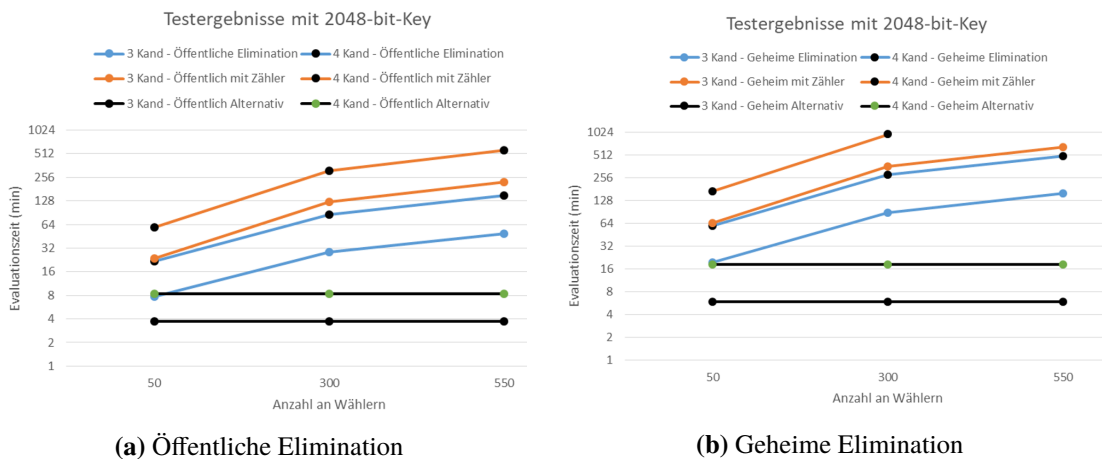


Abbildung 4.6: Variierende Anzahl an Wählern, 2048-bit-Schlüssel, inkl. alternative Verfahren

## 4.9 Sicherheit

Die Verfahren nutzen ausschließlich die in [KLM+19] verwendeten Protokolle. Die Sicherheit der vorgestellten Verfahren beruht daher auf den in [KLM+19] analysierten Sicherheitskriterien. Die Verfahren entschlüsseln keine Wählerstimmen bzw. keine Einträge der aus mehreren Geheimtexten zusammengesetzten Stimmen.

Bei den Verfahren mit öffentlicher Elimination werden nach Durchführung von *get\_last\_candidate* Entschlüsselungen durchgeführt. Dabei ergibt sich aus den entschlüsselten Werten ausschließlich die Information, ob der Kandidat in dieser Iteration eliminiert wurde oder nicht, d.h. pro Kandidat gibt es zwei mögliche Werte. Weitere Entschlüsselungen werden nur von den verwendeten Protokollen aus Def. 4 durchgeführt.

Bei den Verfahren mit geheimer Elimination werden Entschlüsselungen, abgesehen von den Protokollen aus Def. 4, ausschließlich nach der letzten Elimination durchgeführt. Pro Kandidat ergeben sich nur zwei mögliche Werte im Klartext („Wahlsieger“ oder „Wahl nicht gewonnen“).

## 5 Zusammenfassung und Ausblick

In dieser Arbeit wurde das E-Voting-System „Ordinos“ auf konkrete Wahlsysteme erweitert. Im Mittelpunkt stand der sinnvolle Einsatz von „Tally-Hiding“, sodass ein Teil des Ergebnisses veröffentlicht wird und der andere Teil der Informationen für alle Teilnehmer verborgen bleibt.

Als erstes Wahlsystem wurde eine allgemeine Form der Parlamentswahl betrachtet und es wurde analysiert, inwiefern sich „Tally-Hiding“ einsetzen lässt. Bei Auswertung der Erststimmen werden nur die gewonnenen Direktmandate (ohne exaktes Ergebnis) veröffentlicht. Im Großen und Ganzen ist dies zur Auswertung einer Mehrheitswahl ausreichend, da für jeden Kandidaten eine binäre Entscheidung „gewonnen/verloren“ zugrunde liegt.

Die Anwendung von „Tally-Hiding“ bei den Zweitstimmen gestaltete sich komplexer. Im Fokus stand die Geheimhaltung des genauen Ergebnisses und der Anzahl an Stimmen für Parteien unter der Sperrklausel. Die Ergebnisse für Parteien über der Sperrklausel wurden ebenfalls nicht entschlüsselt. Durch gezielte Anwendung der auf Geheimtexten möglichen Operationen konnte die Sitzverteilung nach dem „Hare-Niemeyer-Verfahren“ bestimmt werden. Dazu wurden zwei Varianten präsentiert: Bei der „Öffentlichen Variante“ wird für jede Partei veröffentlicht, wie viele Sitze sie nach der ersten Phase erhält. Im Anschluss wird für jede Partei berechnet und bekannt gegeben, ob sie einen „Restsitz“ erhält oder nicht. Bei der „Geheimen Variante“ wird nur die endgültige Sitzverteilung publik. Die Information, welche Partei einen Restsitz erhalten hat, wird deshalb auch geheim gehalten. Allerdings führt die geheime Variante zu einer erheblichen Erhöhung der Laufzeit, die im Wesentlichen von der Anzahl an Sitzen im Parlament abhängig ist.

Bei einer „klassischen“ absoluten Mehrheitswahl ist eine weitere Stichwahl notwendig, falls keine absolute Mehrheit erreicht wurde. Eine Alternative mit einmaliger Stimmabgabe bietet das „Instant-Runoff Voting“ (IRV), wobei jeder Wähler eine sortierte Liste an Präferenzen abgibt. Die Auswertung erfolgt in mehreren Iterationen, wobei in jeder Iteration der stimmungsschwächste Kandidat (bzgl. den Erstpräferenzen) eliminiert wird. Bei jeder Präferenzliste rutschen die hinter ihm platzierten Kandidaten um einen Platz nach vorne. Mit „Tally-Hiding“ ergibt sich die Möglichkeit, die genaue Anzahl an Stimmen pro Kandidat in jeder Iteration geheim zu halten. Es wurden wiederum zwei Varianten entworfen: Bei der „Öffentlichen Variante“ wird am Ende jeder Iteration der eliminierte Kandidat bekannt gegeben. Bei der „Geheimen Variante“ wird auch diese Information geheim gehalten und es wird nur der Sieger der Wahl, d.h. der letzte nicht eliminierte Kandidat, bekannt gegeben. Die Algorithmen der öffentlichen Variante sind wiederum effizienter durchführbar.

Die komplexe Auswertung beim IRV hat erhebliche Auswirkungen auf die Laufzeit. In jeder Iteration werden Operationen auf allen verschlüsselten Wählerstimmen durchgeführt. Jede Wählerstimme besteht aus  $n^2$  verschlüsselten Einträgen ( $n$  = Anzahl Kandidaten). Die Laufzeit der Algorithmen ist polynomiell in der Anzahl an Kandidaten und linear in der Anzahl an Wählern. In der Praxis ist das Verfahren auf eine geringe Anzahl an Kandidaten und Wähler beschränkt.

Deshalb wurde ein weiteres Verfahren entworfen, das in der Laufzeit unabhängig von der Anzahl an Wählern ist. Das Verfahren beruht auf einer „detaillierteren“ Form der Wählerstimme mit  $n!$

verschlüsselten Einträgen. Zwar arbeitet der Algorithmus in  $O(n!)$ , doch das alternative Verfahren ist trotzdem effizienter für eine geringe Anzahl an Kandidaten (max. 7 bis 9). Insbesondere bei vielen teilnehmenden Wählern zeigt sich der enorme zeitliche Vorteil des alternativen Verfahrens.

Bei beiden Wahlsystemen wird deutlich, dass „Tally-Hiding“ sehr viele Einsatzmöglichkeiten bietet. Diese Arbeit beschränkt sich auf das „Paillier-Kryptosystem“ mit den bereits konzipierten Operationen „+, −, ·, =, ≤, *Enc*, *Dec*“. Tally-Hiding lässt sich anwenden, wenn ein Berechnungsprotokoll gefunden werden kann, welches mit Geheimtexten ausschließlich auf natürlichen Zahlen arbeitet und auf die genannten Operationen beschränkt ist.

Allerdings ist die Laufzeit entscheidend und sorgt für erhebliche Einschränkungen der Verfahren. Die Effizienz der Algorithmen ist außerordentlich wichtig, im Mittelpunkt steht die Reduktion der beim „Tally-Hiding“ relevanten Operatoren „≤“ und „=“. Entsprechend führt i.d.R. die verstärkte Geheimhaltung von Informationen auch zu einer höheren Laufzeit. Für die Praxis ist genau zu überlegen, ob bestimmte Informationen öffentlich werden dürfen oder nicht. Auf die hier entworfenen Wahlsysteme bezogen: „Darf veröffentlicht werden, ob die Parteien Restsitze erhalten?“/ „Darf die Reihenfolge der eliminierten Kandidaten bekannt gegeben werden?“

Ein möglicher und sinnvoller Anknüpfungspunkt an diese Arbeit wäre ein Vergleich mit Umsetzungen unter Verwendung anderer homomorpher Verschlüsselungssysteme. Des Weiteren existieren hier nicht verwendete Operationen, welche sich auf Geheimtexten anwenden lassen (z.B. Mixnets, vgl. [KMST16]). Allerdings sind bei Verwendung weiterer Operationen eventuell stärkere Annahmen über die Trustees nötig, damit eine gleichrangige Laufzeit, Sicherheit und Verifizierbarkeit des Ergebnisses gewährleistet werden kann.

Außerdem gibt es unzählige andere Möglichkeiten zur Umsetzung der betrachteten Wahlsysteme, die ausschließlich mit den in dieser Arbeit verwendeten Operatoren auskommen. Selbstverständlich wäre es möglich, dass sich Verbesserungen mithilfe von Änderungen und Optimierungen der entworfenen Konzepte ergeben, sodass die Auswertung der Wahlen auch für mehr Kandidaten oder Wähler in realistischer Zeit durchgeführt werden kann.

Darüber hinaus bieten die umgesetzten Wahlsysteme verschiedene Möglichkeiten zur Erweiterung: Es gibt neben dem Hare-Niemeyer-Verfahren eine Vielzahl an Alternativen zur Errechnung der Sitzverteilung, z.B. das D'Hondt-Verfahren und das bei der Bundestagswahl verwendete Sainte-Laguë-Verfahren. Allerdings basieren beide Verfahren auf einer vollkommen anderen Methode zur Berechnung der Sitze, wobei sich keine banale Erweiterung auf verschlüsselte Operatoren ergibt.

Das IRV-Verfahren dient ausschließlich der Bestimmung eines Siegers bzw. einer absoluten Mehrheit. Zudem existieren verwandte Ranglistenverfahren, welche z.B. die Verteilung mehrerer Sitze vorsehen, die im Kollektiv durch den Begriff „Übertragbare Einzelstimmgebung“ charakterisiert werden. Die Erweiterung auf weitere Ranglistenverfahren wäre ein sinnvoller Anknüpfungspunkt an das in dieser Arbeit umgesetzte IRV-Wahlsystem.



# Literaturverzeichnis

- [Cat09] M. Catón. „Wahlssysteme und Parteiensysteme im Kontext: Vergleichende Analyse der Wirkung von Wahlssystemen unter verschiedenen Kontextbedingungen“. In: 2009. URL: [http://archiv.ub.uni-heidelberg.de/volltextserver/9054/1/wahlssystem\\_kontext.pdf](http://archiv.ub.uni-heidelberg.de/volltextserver/9054/1/wahlssystem_kontext.pdf) (zitiert auf S. 35).
- [Jer06] C. Jerdonek. „Bringing the Election to the Voters with Instant Runoff Voting“. In: 2006. DOI: [10.1002/ncr](https://doi.org/10.1002/ncr). URL: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/ncr.158> (zitiert auf S. 35).
- [KLM+19] R. Küsters, J. Liedtke, J. Müller, D. Rausch, A. Vogt. *Ordinos: A Verifiable Tally-Hiding Remote E-Voting System*. Techn. Ber. 2019. URL: <https://publ.sec.uni-stuttgart.de/kuestersliedtkemuellerrauschvogt-ordinos-tr-2019.pdf> (zitiert auf S. 11, 13, 14, 16–18, 27, 32, 54).
- [KMST16] R. Küsters, J. Müller, E. Scapin, T. Truderung. „sElect: A Lightweight Verifiable Remote Voting System“. In: *IEEE 29th Computer Security Foundations Symposium (CSF 2016)*. IEEE Computer Society, 2016, S. 341–354. URL: <https://publ.sec.uni-stuttgart.de/kuestersmuellerscapintruderung-csf-2016.pdf> (zitiert auf S. 36, 56).
- [LT13] H. Lipmaa, T. Toft. „Secure Equality and Greater-Than Tests with Sublinear Online Complexity“. In: Hrsg. von F. V. Fomin, M. Kwiatkowska, D. Peleg. Juli 2013. DOI: [10.1007/978-3-642-39212-2\\_56](https://doi.org/10.1007/978-3-642-39212-2_56) (zitiert auf S. 15, 28).
- [Noh07] D. Nohlen. *Wahlrecht und Parteiensystem*. 5, überarbeitete und erweiterte Auflage. Verlag Barbara Budrich, 2007. ISBN: 978-3-86649-969-0 (zitiert auf S. 19, 20, 23, 24, 35).
- [Ros] URL: <https://www.bundestag.de/resource/blob/538898/4dc1460ec7b2bf159982652edbb24448/WD-3-214-17-pdf-data.pdf> (zitiert auf S. 20).
- [VK06] M. Volkamer, R. Krimmer. „Ver-/Misstrauen Schaffende Massnahmen beim e-Voting“. In: *GI Jahrestagung* (2006), S. 418–425 (zitiert auf S. 11).

Alle URLs wurden zuletzt am 10.02.2020 geprüft.



### **Erklärung**

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

---

Ort, Datum, Unterschrift