Institut für Informationssicherheit

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Zero Knowledge Beweise für E-Voting

Paul Mayer

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Ralf Küsters
Betreuer/in:	Julian Liedtke, M.Sc.
Beginn am:	14. Dezember 2020
Beendet am:	14. Juni 2021

Kurzfassung

Die Überprüfung von Wahlergebnissen ist eine der wichtigsten Bestandteile in E-Voting Systemen. Es dürfen nur gültige Stimmzettel mitgezählt werden. Die Stimmzettel sollen zur Überprüfung jedoch nicht geöffnet werden. Doch die Korrektheit soll trotzdem garantiert sein.

Um gültige Stimmen zu kontrollieren, benutzt man Zero-Knowledge Beweise. Diese Beweise zeigen, dass ein Statement wahr ist. Dabei wird aber keine zusätzliche Information, wie der Inhalt eines Stimmzettels, verraten.

Das Ziel dieser Arbeit ist es passende Beweise für das E-Voting System Ordinos zu erstellen. Ordinos legt den Fokus auch auf die Nachvollziehbarkeit der Ergebnisse. Auch von Außenstehenden soll die Wahl überprüfbar sein.

Verschiedene Wahlarten besitzen unterschiedliche gültige Stimmzettel. Deswegen wurden Beweise für mehrere Kategorien mit unterschiedlichen Wahlarten zusammengestellt. Der Fokus liegt hierbei auch bei der Erklärung der Beweisideen. Auch komplexere Wahlen mit mehreren Stimmen oder Ranglisten wurden verwirklicht. Diese Beweise sind vor allem für den/die WählerIn, also das Wahlgerät, effizient.

Inhaltsverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

Wahlen sind ein fundamentaler Bestandteil der Demokratie. Aber auch in anderen Bereichen, wie Vereinen und Unternehmen, werden Wahlen für wichtige Entscheidungen abgehalten. Meistens passiert dies durch Papierwahlen. Inzwischen wird jedoch fast alles über das Internet erledigt. So gibt es auch schon Wahlen, bei denen auch elektronisch abgestimmt wurde. Eine große Frage dabei ist natürlich die Sicherheit und das Vertrauen in das Wahlsystem. Mit E-Voting Systemen können jedoch auch Eigenschaften erreicht werden, die mit klassischen Wahlen nicht möglich sind.

Dabei ist es von großer Bedeutung nachvollziehbare Wahlen zu garantieren. Die Verifizierbarkeit, also die Überprüfung des Wahlergebnisses, ist dabei sehr wichtig. Sodass auch externe Beobachter überprüfen können, dass alle Stimmzettel korrekt gezählt wurden. Ein essenzieller Teil davon ist, dass nur gültige Stimmzettel mitgezählt werden. In normalen Wahlen geschieht dies durch WahlhelferInnen, die jeden Stimmzettel öffnen, überprüfen und zählen müssen. E-Voting hat den Vorteil, dass mithilfe von kryptografischen Protokollen, die verschlüsselten Stimmzettel gezählt und überprüft werden können. Die Stimmzettel werden also nicht geöffnet. Die Stimmen können aber nicht einfach ohne Kontrolle addiert werden, da zum Beispiel eine Vergabe von Minuspunkten die Wahl unfair beeinflussen könnte. Für die Gültigkeit der Stimmzettel werden *Zero-Knowledge* Beweise **ZKP** verwendet. Das sind Beweise, die ein Statement, wie zum Beispiel die Wohlgeformtheit eines Stimmzettels, zeigen können. Dabei wird jedoch keine weitere Information, wie der Inhalt des Stimmzettels, preisgegeben.

Ordinos[KLM+20] ist ein E-Voting System, dass den Fokus auch auf die Verifizierbarkeit legt. Im Rahmen dieser Forschungsarbeit werden verschiedene Wahlarten in Kategorien eingeteilt und passende ZKP dafür erstellt. Diese können im Ordinos System verwendet werden. Es ist Ziel dieser Arbeit effiziente Beweise für diese Kategorien zu erstellen und deren Funktionsweise zu erklären. Es gibt viele verschiedene Wahlarten, bei denen gültige Stimmzettel sich stark unterscheiden können. Deswegen werden auch unterschiedliche Beweise benötigt, die an die Wahlart angepasst sind.

In Kapitel 2 werden die Grundlagen erläutert. Der Fokus liegt hierbei bei den Definitionen von Zero-Knowledge Beweisen. Im Anschluss wird in Kapitel 3 die Ziele von E-Voting Systemen diskutiert. Zusätzlich wird Ordinos und dessen Anforderungen zusammengefasst. Die resultierenden ZKP werden in Kapitel 4 vorgestellt. Dabei werden auch die Beweisideen erläutert und erklärt. In Kapitel 5 wird die theoretische Performance der Beweise besprochen. Im letzten Kapitel 6 wird die Arbeit nochmal zusammengefasst. Außerdem wird ein Ausblick gegeben.

2 Grundlagen

Zuerst werden Grundlagen gebraucht, um zu definieren, was Zero-Knowledge Beweise formal sind. Diese können verschiedene Eigenschaften haben, die meistens in Abstufungen existierten. Außerdem werden für die Beweise ein Kryptosystem und ein Commitment Verfahren benötigt, die auch in diesem Kapitel erklärt und definiert werden.

2.1 Zero-Knowledge Proof

Ein Zero-Knowledge Proof eines Statement, ist ein Beweis, der nicht mehr Informationen, als die Korrektheit des Statements zeigt. Goldwasser, Micali und Rackoff [GMR89] hat diese Beweisart in 1989 eingeführt. Dieses System besteht aus einem Verifier V und einem Prover P. In diesem *Interactive Proof System* ist abwechselnd immer einer der beiden Beteiligten aktiv. Dabei versucht der Prover, den Verifier davon zu überzeugen, dass das Statement wahr ist. Ohne weitere Informationen preiszugeben. So ein Statement kann die Lösbarkeit eines Problems, wie zum Beispiel das Färbungsproblem für Graphen, sein. Der Prover muss dann, ohne die Lösung zu verraten, zeigen, dass das Problem lösbar ist. Die Lösung wird dabei *Witness* genannt. Am Ende eines Durchlaufs so eines Systems entscheidet der Verifier, ob er annimmt oder ablehnt. Gültige Statements x und die passende Witness w stehen dabei in einer Relation $(x, w) \in R$. Die Sprache der gültigen Statements ist dann $x \in L$, wenn $\exists w : (x, w) \in R$. Ein ZKP muss folgende drei Eigenschaften erfüllen.

- **Completeness:** Mit einem gültigen Statement und Witness kann der Prover den Verifier überzeugen.
- **Soundness:** Der Verifier kann nicht von einem falschen Statement überzeugt werden.
- **Zero-Knowledge:** Der Verifier lernt nichts, wenn das Statement gültig ist.

Diese Eigenschaften gibt es in drei Abstufungen. Von *computational* spricht man, wenn ein *probabilistic polynomial time* Angreifer angenommen wird. Es soll also zu schwierig sein diese Eigenschaft zu brechen. Wenn dieser jedoch uneingeschränkte Rechenleistung besitzt, ist die Eigenschaft *statistical*. Und die Wahrscheinlichkeit, dass die Eigenschaft gebrochen wird ist vernachlässigbar. Oft werden Protokolle mit diesen Eigenschaftsarten mehrfach ausgeführt, um die Wahrscheinlichkeiten weiter anzunähern. Wenn die Eigenschaft nicht gebrochen werden kann, ist sie *perfect* und niemals Brechbar.

Man unterscheidet ZKP zwischen Beweisen und Argumenten. So muss ein Argument nur gegen einen, in der Rechenleistung eingeschränkten, Angreifer die geforderten Sicherheitsversprechen einhalten. Es werden also nur *computational* Eigenschaften gefordert. Bei Beweisen hingegen ist der Angreifer nicht eingeschränkt *statistical* oder *perfect* Eignechaften werden benötigt. Argumente sind oft deutlich effizienter und somit eher in der Praxis einsatzfähig. In der Literatur werden die zwei Begriffe allerdings oft als Synonyme verwendet.

Auch zwischen *ZKP of Knowledge* und *ZKP of Membership* wird differenziert. So beweist der Prover bei Membership, dass ein Statement zu einer Sprache gehört $x \in L$. Bei Knowledge wird zusätzlich bewiesen, dass der Prover die Witness kennt, die in Relation mit dem Statement steht $(x, w) \in R$.

Des Weiteren kann man ZKP darin unterscheiden, ob jeder den Beweis verifizieren kann oder, ob nur ein bestimmter, vorgesehener Verifier die Korrektheit des Statements nachvollziehen kann. Anwendungen für letzteres wird in [JSI96] diskutiert.

Ein Beweis heißt *Public Coin*, wenn der Verifier nur uniform zufällige Nachrichten schickt und nicht auf die Nachrichten des Provers eingeht.

Ein *Range Proof* ist ein Beweis, dass eine Zahl in einem bestimmten Intervall liegt. Der Input ist dabei ein Commitment, das die Zahl enthält.

Ein *Argument of Shuffle* für zwei Listen A, B ist ein Beweis, dass A eine Permutation von B ist.

Zero-Knowledge Beweise sind in vielen Bereichen von Nutzen. Zum Beispiel können Berechnungen verifiziert werden, ohne private oder schützenswerte Details zu verraten. Auch in MPC-Protokollen kann die Korrektheit der einzelnen Berechnungen gezeigt werden [CDN01]. Für diese Arbeit am Interessantestem ist jedoch der Einsatz in E-Voting.

Oft sind die Beweissysteme allgemein für beliebige NP-Sprachen [GMW91], [GOS06]. Oder beschäftigen sich mit allgemeineren Problemen, wie das Erfüllbarkeitsproblem für arithmetische Schaltkreise [Gro09], bei denen das Statement ein Schaltkreis und die Witness eine Lösung dafür ist.

In dieser Arbeit steht die Box $\boxed{a}$ um a dafür, dass ein Zufallswert den ursprünglichen Wert a verschleiert. Dieses Prinzip der Blackbox wird immer wieder verwendet, damit keine Information preisgegeben wird, man jedoch die verschleierten Werte trotzdem zum Beweisen benutzen kann.

Die folgenden Definitionen sind nach [BCCG16].

2.1.1 Σ-Protokoll

Ein Σ-Protokoll ist ein *3-move interactive Proof* um einen Verifier von einem Statement zu überzeugen. Das Protokoll ist *Public Coin* und muss nur einmal ausgeführt werden. Der Prover sendet dabei eine initiale Nachricht. Der Verifier antwortet mit einer zufälligen Nachricht, der *Challenge*. Der Prover antwortet daraufhin nochmal und der Verifier entscheidet anhand dieser Kommunikation.

Definition 2.1.1 (Σ-Protokoll)
Ein Σ-Protokoll für eine Relation R ist ein Tupel (P,V) von ppT interaktiven Algorithmen.

- $a \leftarrow P(u, w)$*: mit Statement u und Witness w berechnet der Prover die initiale Nachricht.*
- $x \leftarrow S$*: Verifier wählt Challenge x aus der Menge S.*
- $z \leftarrow P(x)$*: mit der Challenge x berechnet der Prover die Antwort z.*
- $1/0 \leftarrow V(u, (a, x, z))$*: Der Verifier entscheidet aufgrund von* (a, x, z) *und akzeptiert oder lehnt ab.*

Das Protokoll ist Complete, special Sound und special Honest Verifier Zero-Knowledge, wie im Folgenden definiert.

Definition 2.1.2 (Completeness)
Für alle Angreifer $\mathcal{A}$ gilt:

$$Pr\left[(u,w) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow P(u,w); e \leftarrow S; z \leftarrow P(x) : V(u,(a,e,z)) = 1\right] \approx 1$$

Für computational Completeness wird angenommen, dass $\mathcal{A}$ ein ppT Angreifer ist. Wenn dieser jedoch uneingeschränkt ist, erhält man statistical Completeness. Durch das Ersetzen durch ein Gleichheitszeichen wird perfect Completeness erreicht.

Ein Σ-Protokoll ist ein *Proof of Knowledge*, da der Prover nur mit einer Witness eine gültige Antwort erstellen kann. Dies wird durch *Special Soundness* erreicht. Bei zwei Durchläufen mit derselben initialen Nachricht und unterschiedlichen Challenges ist es möglich die Witness zu berechnen. Dazu wird ein *Extractor* Algorithmus verwendet. Dieser kann den Prover als Orakel benutzen und versucht damit eine Witness zu berechnen.

Definition 2.1.3 (Special Soundness)
Es existiert ein effizienter Extractor Algorithmus $\mathcal{E}$, sodass für jeden Angreifer $\mathcal{A}$ gilt:

$$Pr\begin{bmatrix}(u,a,x,z,x',z') \leftarrow \mathcal{A}(1^\lambda); w \leftarrow \mathcal{E}(a,x,z,x',z') : \\ V(u,(a,x,z)) = 0 \vee V(u,(a,x',z')) = 0 \vee (u,w) \in R\end{bmatrix} \approx 1$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

Um Zero-Knowledge zu definieren, wird ein Simulator verwendet. Wenn ein akzeptierender Durchlauf simuliert werden kann, ohne die Witness zu kennen, dann verrät das Protokoll keine zusätzliche Information. Bei *Special Honest Verifier* Zero-Knowledge handelt es sich um ein *Public Coin* Verifier.

Definition 2.1.4 (Special Honest Verifier Zero-Knowledge)
Es existiert ppT Simulator S, sodass für alle Angreifer $\mathcal{A}$ gilt:

$$\begin{aligned}&Pr\left[(u,w,x) \leftarrow \mathcal{A}(1^\lambda); a \leftarrow P(x) : \mathcal{A}(a,x,z) = 1\right] \\ &\approx Pr\left[(u,w,x) \leftarrow \mathcal{A}(1^\lambda); (a,z) \leftarrow S(u,x) : \mathcal{A}(a,x,z) = 1\right]\end{aligned}$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

2.1.2 Non-Interactive Zero-Knowledge

Man spricht von *non-interactive* ZKP, wenn nur eine Nachricht von Prover zu Verifier geschickt wird. Diese Nachricht wird nun als Beweis bezeichnet. *Non-interactive* Zero-Knowledge NIZK Beweise wurden zuerst von Blum et al. [BFM19] definiert. Für einen Beweis ohne Interaktion wird ein Setup benötigt. Dieses kann durch einen *Common Reference String* CRS. Auf diesen haben sowohl der Prover als auch der Verifier Zugriff. Dieser CRS enthält Information, um einen gemeinsamen Startzustand zu finden. Zum Beispiel welche Relation gezeigt wird oder eine Beschreibung von kryptografischen Primitiven, die verwendet werden.

Definition 2.1.5 (Non-Interactive Zero-Knowledge)
Ein non-interactive Proof System für eine Relation R besteht aus drei ppT Algorithmen. Common Reference Generator oder Setup, Prover P und Verifier V

- $\sigma \leftarrow Gen(1^\lambda)$*: generiert mit dem Security Parameter* λ *ein CRS* σ.
- $\pi \leftarrow P(\sigma, x, w)$*: mit dem CRS* σ*, Statement* x *und Witness* w *wird der Beweis berechnet.*
- $1/0 \leftarrow V(\sigma, x, \pi)$*: Verifier entscheidet anhand von CRS* σ*, Statement* x *und Beweis* π.

Das System erfüllt Completeness, computational Soundness und Zero-Knowledge, wie im Folgenden definiert.

Definition 2.1.6 (Completeness)
Für alle $\lambda \in \mathbb{N}$ *und* $(u, w) \in R$ *gilt:*

$$Pr\left[\sigma \leftarrow Gen(1^\lambda); \pi \leftarrow P(\sigma, u, w) : V(\sigma, u, \pi) = 1\right] \approx 1$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

Definition 2.1.7 (Soundness)
Für alle $\lambda \in \mathbb{N}$ *und Angreifer* $\mathcal{A}$ *gilt:*

$$Pr\left[\sigma \leftarrow Gen(1^\lambda); (u, \pi) \leftarrow \mathcal{A}(\sigma, u) : u \notin L \wedge V(\sigma, u, \pi) = 1\right] \approx 0$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

Ein *Proof of Knowledge* entsteht, wenn es möglich ist, die Witness w vom Beweis wiederherzustellen. Es gibt also einen *Extractor* $E = (E_1, E_2)$, sodass E_1 einen gültigen CRS zusammen mit einem *Extraction Key* generiert. E_2 kann diesen *Key* benutzen, um eine gültige Witness zu berechnen.

Definition 2.1.8 (Knowledge Extraction)
Für alle $\lambda \in \mathbb{N}$ *und Angreifer* $\mathcal{A}$ *gilt:*

$$Pr\left[\sigma \leftarrow Gen(1^\lambda) : \mathcal{A} = 1\right] \approx Pr\left[(\sigma, \xi) \leftarrow E_1(1^\lambda) : \mathcal{A} = 1\right]$$

und

$$Pr\left[(\sigma, \xi) \leftarrow E_1(1^\lambda); (u, \pi) \leftarrow \mathcal{A}(\sigma); w \leftarrow E_2(\sigma, \xi, u, \pi) : (u, w) \in R \text{ wenn } V(\sigma, u, \pi) = 1\right] = 1$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

Perfect Knowledge Extraction impliziert *perfect Soundness*. Zero-Knowledge wird wieder mit einem Simulator definiert. Dieser braucht aber mehr als das Statement und den CRS, da sonst jeder einen gültigen Beweis simulieren und somit erstellen könnte. Dem Simulator wird dabei erlaubt einen eigenen CRS zu erstellen. Außerdem darf er zusätzliche Information, die *Simulation Trapdoor* τ benutzen. Wichtig ist das ein Angreifer diese Information nicht benutzen kann.

Definition 2.1.9 (Zero-Knowledge)
Es existiert ein Simulator $S = (S_1, S_2)$*, sodass für alle* $\lambda \in \mathbb{N}$ *und Angreifer* $\mathcal{A}$ *gilt:*

$$\begin{aligned} &Pr\left[\sigma \leftarrow Gen(1^\lambda); (u, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow P(\sigma, u, w) : \mathcal{A}(\sigma, \pi) = 0\right] \\ &\approx Pr\left[(\sigma, \tau) \leftarrow S_1(1^\lambda); (u, w) \leftarrow \mathcal{A}(\sigma); \pi \leftarrow S_2(\sigma, \tau, u) : \mathcal{A}(\sigma, \pi) = 0\right] \end{aligned}$$

Computational, statistical und perfect definiert wie zuvor in Definition 2.1.2

2.1.3 Fiat-Shamir

Die Fiat-Shamir Transformation [FS86] wird verwendet, um die Kommunikation in interaktiven Beweisen zu entfernen. Dabei werden *Public Coin ineractive* ZKP in *non-interactive* ZKP umgewandelt. Es können also auch Σ-Protokolle umgeformt werden. Dafür enthält der CRS die Beschreibung einer Hash-Funktion. Die Verifier Nachrichten werden nun durch einen Hash ersetzt. Σ-Protokolle besitzen nur *special Honest Verifier* Zero-Knowledge. Nach der Transformation hat es jedoch volles Zero-Knowledge. Protokolle mit Fiat-Shamir Transformation sind effizient, jedoch befindet man sich durch die Nutzung von Hash-Funktionen im *Random Oracle Model* ROM. Im ROM wird angenommen, dass die Hashwerte komplett zufällig sind. In der Praxis sind Hash-Funktionen jedoch deterministisch und es gibt Protokolle, die mit einer echten Hash-Funktion nicht mehr die *Soundness* Eigenschaft besitzen [CGH04].

2.2 Kryptosystem

Ein Kryptosystem wird verwendet um Nachrichten zu verschlüsseln.

Definition 2.2.1 (Kryptosystem)
Ein Kryptosystem is ein Tupel (P,C,K,E,D) mit folgenden Eigenschaften. P ist die Menge der Klartexte. C ist die Menge der Ciphertexte. K ist die Menge aller Schlüssel. $E : P \times K \rightarrow C$ *ist eine Verschlüsselungsfunktion.* $E : C \times K \rightarrow P$ *ist eine Entschlüsselungsfunktion. So dass für jedes* $e \in K$ *ein* $d \in K$ *existiert, sodass D(E(x,e),d)= x für alle* $p \in P$.

Ein Kryptosystem kann folgende Eigenschaften haben:

- ***Homomorphic:*** $E(m_1; r_1) \cdot E(m_2; r_2) = E(m_1 + m_2; r_1 + r_2)$
- ***Root Extraction:*** *Für einen Ciphertext C,* $e \neq 0$ *und* $C^e = E(M; R)$*, muss es möglich sein* μ, ρ *zu finden so, dass* $M = e\mu$*,* $R = e\rho$ *und* $C = E(\mu; \rho)$ *gilt.*

Bei einem *Public-Key* System ist der Schlüssel zum Verschlüsseln öffentlich. Der Schlüssel zum Entschlüsseln ist jedoch nur dem/der EmpfängerIn bekannt, also privat. Ein $(t, n) - treshold$ Kryptosystem besteht aus n Instanzen. Der private Schlüssel ist dabei auf diese Instanzen verteilt, und es werden mindesten t gebraucht um eine Entschlüsselung durchzuführen.

2.3 Commitment

Bei einem Commitment Verfahren soll sich auf einen bestimmten Wert festgelegt werden. Um ein Commitment zu überprüfen gibt es einen *Open* Algorithmus. Oft wird jedoch einfach der festgelegte Wert und die Zufallswerte veröffentlicht und das Commitment kann einfach nachgerechnet werden. Meistens ist so ein Verfahren *binding*. Das heißt, der Wert kann nicht im Nachhinein geändert werden. Außerdem ist das Commitment *hiding*, wenn der Wert nicht vor der Veröffentlichung ausgelesen werden kann. Die Definition sind an [BBB+18] angelehnt.

Definition 2.3.1 (Commitment Verfahren)
Ein non-interactive Commitment Verfahren besteht aus zwei polynomial Zeit Algorithmen Setup und com.

- $pp \leftarrow Setup(1^\lambda)$*: generiert öffentliche Parameter, wobei* λ *ein Sicherheitsparameter ist.*
- $c \leftarrow com(x;r) : M_{pp} \times R_{pp} \rightarrow C_{pp}$ *wobei* M_{pp} *der Message Space,* R_{pp} *Randomness Space und* C_{pp} *Commitment Space ist.* $r \xleftarrow{\$} R_{pp}$ *wird zufällig gewählt.*

Ein Commitment Verfahren kann folgende Eigenschaften haben:

- ***Homomorphic:*** $com(x_1;r_1) \cdot com(x_2;r_2) = com(x_1 + x_2; r_1 + r_2)$
- ***Hiding:*** *Ein Commitment verrät den Wert* $x \in M_{pp}$ *nicht. Ein Angreifer kann also anhand des Commitments und ohne den Zufallswert r nicht erkennen, auf welches x verwendet wurde.*
- ***Binding:*** *Der Ersteller des Commitments sollte das Commitment nicht mit einem anderen Wert öffnen können. Zwei unterschiedliche Werte haben also mit hoher Wahrscheinlichkeit ein unterschiedliches Commitment.*

$$x_0 \neq x_1 : com(x_0;r_0) \neq com(x_1;r_1)$$

- ***Root Extraction:*** *Für ein Commitment* c, $m_1, \ldots m_k, r$ *und* $e \neq 0$, *sodass* $c^e = com(m_1, \ldots m_k; r)$, *dann folgt* $e|m_1, \ldots, e|m_k$ *und* $c = com(\mu_1, \ldots, \mu_k; \rho)$ *mit* $\mu_i = \frac{m_i}{e}$

Die Eigenschaften *binding* und *hiding* können wieder in den drei Abstufung *computational, statistical and perfect* erreicht werden.

Definition 2.3.2 (Discrete Logarithm Assumption)
Sei $\mathbb{G}$ *eine Gruppe von Ordnung p, p eine Primzahl, g ein Generator für* $\mathbb{G}$. *Dann ist es für ein* $y \in \mathbb{G}$ *nicht möglich, eine Zahl x zu finden, sodass* $y = g^x$ *gilt.*

Eine Umsetzung ist das Pedersen-Commitment Verfahren [Ped91].

Definition 2.3.3 (Pedersen-Commitment)
Das Pedersen-Commitment ist perfectly hiding und computationally binding unter der discrete Logarithm Assumption 2.3.2. $M_{pp}, R_{pp} = \mathbb{Z}_{pp}$, $C_{pp} = \mathbb{G}$ *mit Ordnung p.*

- *Setup:* $g, h \xleftarrow{\$} \mathbb{G}$
- $com(x;r) = g^x h^r$

Definition 2.3.4 (Pedersen-Vektor-Commitment)
Das Pedersen-Vektor-Commitment ist perfectly hiding und computationally binding unter der discrete Logarithm Assumption 2.3.2. $M_{pp} = \mathbb{Z}_{pp}^n, R_{pp} = \mathbb{Z}_{pp}, C_{pp} = \mathbb{G}$ *mit Ordnung p.*

- *Setup:* $(g_1, \ldots g_n), h \xleftarrow{\$} \mathbb{G}$
- $com(x_1, \ldots, x_n; r) = h^r \prod_{i=1}^n g_i^{x_i}$

3 E-Voting

In diesem Kapitel geht es um E-Voting allgemein, ohne Fokus auf ZKP. Außerdem wird Ordinos beschrieben und dessen Anforderungen an die kryptografischen Primitiven festgelegt.

Es gibt viele Anforderungen an E-Voting Systeme, die wie in [LK02] beschrieben, in zwei Kategorien eingeteilt werden können. Das sind zum einen die Basis Anforderungen, die jedes System erfüllen sollte. So wird durch *Privacy* gefordert, dass alle Stimmzettel geheim bleiben. Das alle gültigen Stimmen auch korrekt gezählt wurden, wird durch *Completeness* verlangt. Im Gegensatz sollen ungültige Stimmen nicht mitgezählt werden. Dies wird durch *Soundness* garantiert. *Unreusability* fordert, dass kein/e WählerIn mehrfach die Stimme abgeben kann. Außerdem sollen nur Berechtigte WählerInnen abstimmen. Diese Eigenschaft heißt *Eligibility*. Die letzte Basis Anforderung ist *Fairness*, durch die Garantiert wird, dass nichts die Abstimmung beeinflussen kann.

Einige E-Voting Systeme haben jedoch noch weitere Eigenschaften als Ziel. Diese werden dann erweitere Anforderungen genannt. Beispiele hierfür sind *Robustness*, bei der eine Wahl erfolgreich abgeschlossen werden kann, auch wenn ein Teil des Systems ausfällt. Um Stimmenkauf zu verhindern wird die Eigenschaft *Receipt-freeness* gefordert. Durch diese wird verhindert, dass ein/e WählerIn beweisen kann, wie der Stimmzettel ausgefüllt wurde. Ein System, das dies erfüllt wird in [LK02] beschrieben. Eine noch stärkere Eigenschaft ist *Incoercibility*, durch die ein/e Wählerin nicht zu einer bestimmen Wahl gezwungen werden kann. Eine sehr wichtige Anforderung ist *Universal Verifiability*, durch die verlangt wird, dass jeder, auch externe Beobachter, verifizieren kann, dass die Wahl fair war und das Ergebnis korrekt ist. Es gibt noch viele weitere solcher Anforderungen.

Die Wahlsysteme lassen sich in drei Verfahrensarten einteilen. Die erste benutzt Mix-Nets, um die Stimme abzugeben. Ein Mix-Net besteht aus mehreren Mix-Knoten, die nur ihre direkte Umgebung kennen. Jeder dieser Knoten im Netz verschlüsselt die eingehenden Stimmzettel, sodass eine verschachtelte Verschlüsselung entsteht. Dabei soll die Beziehung zwischen WählerIn und Stimmzettel verschleiert werden. Es gibt einige Systeme mit Mix-Nets [Adi08], [JJR02], [Abe98]. Eine weitere Möglichkeit E-Voting Systeme umzusetzen, ist mithilfe von *blind Signatures*. Dabei signiert eine Autorität den Stimmzettel dessen Inhalt verschleiert wurde. So wird garantiert, dass nur berechtigte WählerInnen teilnehmen und nur einmal wählen. Im Nachhinein soll keine Verbindung zwischen WählerIn und Stimmzettel festgestellt werden können. Beispiel Systeme sind [OMA+99],[FOO92]. Der letzte Ansatz benutzt homomorphe Verschlüsselung, durch die mehrere Ciphertexte addiert werden können, ohne diese zu entschlüsseln. Die Korrektheit der einzelnen Stimmen wird durch Zero-Knowledge Beweise gezeigt.

3.0.1 Ordinos

Ordinos [KLM+20] ist ein E-Voting System, dass auf homomorpher Verschlüsselung basiert. Der Fokus liegt dabei darauf eine Wahl mit der Eigenschaft *tally-hiding* zu ermöglichen. *Tally-hiding* bedeutet, dass nicht das ganze Ergebnis der Wahl veröffentlicht, da zum Beispiel nur die ersten k

Plätze von Interesse sind. Einige Wahlarten bestehen aus mehreren Runden und eine komplette Veröffentlichung der Zwischenergebnisse könnte einen Wähler beeinflussen. Außerdem ist diese Eigenschaft schwierig in klassischen Wahlen umzusetzen, da dadurch das Vertrauen in das Ergebnis stark geschwächt wird. Ordinos besitzt des weiteren *end-to-end Verifiability*. Das heißt WählerInnen, sowie externe BeobachterInnen, können überprüfen, ob alle Stimmen mitgezählt wurden und, dass das Ergebnis korrekt berechnet wurde. Es wird sogar die stärkere Eigenschaft *Accountability* erfüllt, wobei ein falsches Ergebnis oder ein Fehler nicht nur erkannt wird, sondern auch der Schuldige ausgemacht werden kann. Ordinos benötigt ein homomorphes, IND-CPA sicheres (t, n)-treshold public-key Kryptosystem (Definition 2.2.1), wie zum Beispiel ElGamal [ElG85] oder Paillier [Pai99]. Zur Überprüfung des Stimmzettels wird ein *non-interactive* Zero-Knowledge Beweis benötigt, dass dieser gültig, entsprechend der Wahlart, ist.

Die Wahlarten können ungefähr in drei Kategorien eingeteilt werden. Die einfachste Variante ist *Multiple Choice*, bei der ein/e WählerIn eine/n KandidatIn entweder auswählen kann oder nicht. Dabei kann eine Obergrenze an Stimmen festgelegt werden. Eine komplizierte Variante ist das *Ranking*, wobei der/die Wählende eine Rangliste der KandidatInnen erstellt. Ihrer Platzierung entsprechend bekommen die KandidatInnen unterschiedlich viele Stimmen. Die Abstufungen der Punkte kann dabei frei gewählt werden. Die letzte Kategorie ist *Rating*, bei der/die WählerIn eine bestimmte Anzahl N an Stimmen bekommt, die er/sie verteilen kann. Entweder können die Stimmen an alle KandidatInnen zusammen vergeben werden, oder für jeden einzeln ist eine Bewertung von 0 bis N möglich. Da die erlaubten Stimmzettel je nach Wahlart sehr unterschiedlich aussehen können, variieren die Anforderungen an die Beweise und es werden verschiedene benötigt.

4 Zero-Knowledge Proofs

In diesem Kapitel werden zunächst die Eigenschaften der Beweise diskutiert. Dabei geht es darum, welche Anforderungen gefordert sind und welche Eigenschaften am sinnvollsten sind. Danach wird ein Argument vorgestellt, dass eine allgemeine Relation zeigt. Diese wird für jeden Beweis gebraucht, da sie das Format des Ciphertextes beschreibt. Die vollständigen Beweise für die Wahlarten werden danach vorgestellt und die Funktionsweise erklärt. Für die Kategorie *Rating* wird zusätzlich ein *Range Proof* (Abschnitt 2.1) benötigt. Diese werden ausführlicher diskutiert.

Viele Zero-Knowledge Beweissysteme (Abschnitt 2.1) sind allgemein für beliebige NP-Sprachen. Oder beschäftigen sich mit Problemen, wie das Erfüllbarkeitsproblem für arithmetische Schaltkreise, bei denen das Statement ein Schaltkreis und die Witness eine Lösung dafür ist. Man kann diese Systeme anpassen, indem die gesuchte Relation auf ein Schaltkreis abbildet. Jedoch sind solche Ansätze in der Praxis oft zu ineffizient. Man benötigt also speziell an die Relation oder das Problem angepasste Beweise, um ein effizientes System zu erhalten. Dazu werden auch häufig einfache Relationen als Bausteine benutzt, um über komplexere Relationen zu argumentieren. Zum Beispiel, dass der Verifier die Lösung eines diskreten Logarithmus kennt oder, dass zwei Zahlen in einem Multiplikativen Verhältnis stehen [GK93]. Zuletzt können auch bestimmte Eigenschaften des Problems oder der Relation ausgenutzt werden, um angepasste ZKP zu generieren. Deswegen sind viele Beweissysteme im E-Voting Bereich, speziell auf das Wahlsystem und/oder Kryptosystem angepasst und können nicht in jedem Setting benutzt werden [DGS03],[BM09],[PR18], [Smi05],[BFP+01],[CGS97],[LK02],[SK94],[DJ01].

Wir wollen mithilfe von ZKP beweisen, dass ein Stimmzettel richtig ausgefüllt wurde, ohne den Inhalt zu verraten. Das Statement ist also die Wohlgeformtheit des verschlüsselten Stimmzettels. Die Witness ist dann der Klartext also die Wahl des/der WählerIn. In Ordinos wird gefordert, dass auch externe Beobachter das Ergebnis der Wahl nachvollziehen können. *Designated* Verifier Beweise sind deswegen uninteressant. *Non-interactive* Beweise (Definition 2.1.5) sind in E-Voting erwünscht, da während des Ausfüllens keine Interaktion zwischen potenziell sehr vielen WählerInnen und einer Wahlautorität stattfinden muss. Durch die Effizienz und einfache Umwandlung in *non-interactive* Protkolle, bieten sich Σ-Protokolle (Definition 2.1.1) an. Es handelt sich bei den vorgestellten Beweisen um *honest Verifier* Zero-Knowledge Σ-Protokolle (Definition 2.1.4). Damit Verifier und Prover nicht kommunizieren müssen, wird die Fiat-Shamir Transformation auf diese Argumente angewendet. Dadurch sind die Argumente *ZKP of Knowledge*. Da der die WählerIn den Inhalt kennen sollte, ist dies auch erwünscht. Und durch das Verwenden von Hash-Funktionen befinden wir uns im Random Oracle Model. Für die Beweise werden, die von Groth [Gro05] entwickelten Argumente angepasst und erweitert. Dabei wird von einem homomorphen *treshold* Kryptosystem ausgegangen. Zusätzlich wird die Eigenschaft *Root Extraction*, wie in Definition 2.2.1 erklärt, gefordert. Wie bei Ordinos erfüllt ElGamal [ElG85] und Paillier [Pai99] diese Anforderungen. Außerdem wird ein Commitment-Verfahren (Definition 2.3.1) benötigt. Dieses muss auch homomorph sein und die

Root Extraction Eigenschaft erfüllen. Die Beweise basieren je nachdem welches Kyrptosystem und Commitment-Verfahren verwendet wird, deren *cryptographic Assumptions*. Wie zum Beispiel der *discrete Logarithm Assumption* (Definition 2.3.2).

Zunächst müssen wir einige Parameter festlegen. L ist die Anzahl der KandidatInnen, die gewählt werden können. Durch M wird die maximale Stimmenanzahl, die ein/e KandidatIn erreichen kann, präsentiert. Außerdem ist N die Anzahl der Stimmen je WählerIn. Die maximale Bit-Länge einer Stimme ist $l_v = 2\lceil L \cdot log(M)/2\rceil$. Für das Kryptosystem (Definition 2.2.1) und Commitment Scheme (Definition 2.3.1) werden l_R und l_r-randomizer verwendet. Die Hash-Funktion hat einen l_e-Bit Output. Zuletzt wird ein Sicherheitsparameter l_s benötigt. Dieser sollte so gewählt werden, dass für jeden Wert a, $a + r_a$ und r_a nicht zu unterscheiden sind. Dabei ist r_a eine $|a| + l_s$-Bit Zahl. Der Nachrichten Raum ist $\mathbb{Z}_n$. Wobei n keine Primfaktoren besitzt, die kleiner als 2^{l_e} sind. Außerdem sollte $M^L \leq n$ sein.

Die Zahlen $0, ..., L-1$ repräsentieren die KandidatInnen. Dadurch kann ein Ergebnis durch $\sum_{i=0}^{L-1} a_i M^i$ dargestellt werden, wobei a_i für die Anzahl an Stimmen für KandidatIn i steht. M^i wird hierbei verwendet, damit die Stimmzahl der KandidatInnen weit genug auseinander liegen, da M die maximale Anzahl der Stimmen pro KandidatIn ist. Es gilt also $a_i < M$ und M ist wie eine Basis in einem Zahlensystem. Das Ergebnis kann dadurch einfach ausgelesen werden. Durch die homomorphe Eigenschaft des Kryptosystems kann man die Stimmzettel verschlüsselt addieren und das Ergebnis $E(\sum_{i=0}^{L-1} a_i M^i)$ kann einfach entschlüsselt und ausgelesen werden.

In Ordinos sind die Stimmen jedoch einzeln verschlüsselt und deswegen muss der Ciphertext zunächst umgewandelt werden.

$$C = E(\sum_{i=0}^{L-1} a_i M^i; R) \tag{4.1}$$

$$= E(a_0 M^0 + a_1 M^1 + \cdots + a_{L-1} M^{L-1}; R) \tag{4.2}$$

$$= E(a_0; r_0)^{M^0} \cdot E(a_1; r_1)^{M^1} \ldots E(a_{L-1}; r_{L-1})^{M^{L-1}} \tag{4.3}$$

Diese Transformation muss der Verifier vor der Verifikation berechnen. Da der Prover die Stimmen a_i kennt, muss er diese Umwandlung nicht vorher durchführen. Er muss nur die Zufallswerte unverschlüsselt addieren, da $R = \sum_0^{L-1} r_i$.

4.1 Format

Alle Beweise der Stimmzettelarten folgen demselben Prinzip. Da das Format der Stimmen als Summe jeweils dasselbe ist, wird dieser Teil in jedem Beweis gleich bewiesen. Dazu muss der Prover zum einen zeigen, dass er die einzelnen Stimmen a_i pro KandidatIn kennt. Zum anderen muss er beweisen, dass diese auch zur Berechnung des Ciphertextes verwendet wurden. Es wird also folgende Relation, in Algorithmus 4.1 gezeigt.

$$R = \left\{ (C, (V, R, a_0, \ldots, a_{L-1})) : C = E(V; R) \text{ und } V = \sum_{i=0}^{L-1} a_i M^i \right\} \tag{4.4}$$

Da die Wahl geheim bleiben muss, dürfen die Stimmen nicht im Klartext an den Verifier geschickt

Algorithmus 4.1 Allgemeines Argument für gültiges Format

Input: C
Prover Input: $a_0, \ldots, a_{L-1} \in \{0,1\}$ und $R \in \{0,1^{l_R}\}$, sodass $C = E(\sum_{i=0}^{L-1} a_i M^i; R)$

function ARGUMENT($a_0, \ldots, a_{L-1}$ und R)
  $r \xleftarrow{\$} \{0,1\}^{l_r}; r_{a_0}, \ldots, r_{a_{L-1}} \xleftarrow{\$} \{0,1\}^{1+l_e+l_s}$ // Generiere Zufallswerte
  $r_r \xleftarrow{\$} \{0,1\}^{l_r+l_e+l_s}; R_R \xleftarrow{\$} \{0,1\}^{l_R+l_e+l_s}$
  $c = com(a_0, \ldots, a_{L-1}; r)$ // Berechne Commitments
  $c_r = com(r_{a_0}, \ldots, r_{a_{L-1}}; r_r)$
  $R_V = \sum_{i=0}^{L-1} r_{a_i} M^i$
  $C_R = E(R_V; R_R)$
  $e = hash(C, C_R, c, c_r, aux)$ // Challenge berechnen
  $\boxed{R} = eR + R_R; \boxed{a_i} = ea_i + r_{a_i}; \boxed{r} = er + r_r$ // Verschleiern
  return $C_R, c, c_r, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$
end function

function VERIFIKATION($C, C_R, c, c_r, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$)
  $e = hash(C, C_R, c, c_r, aux)$
  $\boxed{V} = \sum_{i=0}^{L-1} \boxed{a_i} M^i$ // Stimmzettel erstellen
  if $C^e C_R = E(\boxed{V}; \boxed{R})$ and $c^e c_r = com(\boxed{a_1}, \ldots, \boxed{a_{L-1}}; \boxed{r})$ **then**
    return True
  end if
  return False
end function

werden. Die Challenge e wird dabei genutzt um die Werte des Ciphertextes und Commitments an den Beweis anzupassen. Zusätzlich wird eine Zufallszahl addiert, um den richtigen Wert zu verschleiern. Dafür wird $\boxed{a_i} = ea_i + r_{a_i}$ berechnet und stattdessen verschickt. Hierbei steht die Box $\boxed{a_i}$ dafür, dass das ursprüngliche a_i nicht auslesbar ist. Dieses Prinzip der Blackbox wird immer wieder verwendet, damit keine Information preisgegeben wird, man jedoch diese verschleierten Werte trotzdem zum Beweisen benutzen kann. Durch die homomorphen Eigenschaften des Kryptosystems und Commitment Verfahren kann der Verifier die Challenge e, sowie die Zufallswerte auf den Klartext multiplizieren/addieren ohne zu entschlüsseln oder das Commitment zu öffnen. Im Argument stehen die $\boxed{a_i}$ und der Prover berechnet ein Commitment auf diese. Deswegen wird überprüft, ob

diese Werte auch zur Erstellung der Commitments, sowie des Ciphertextes genutzt wurden. Dazu überprüft er zwei Gleichungen. Zunächst für den Ciphertext C, ob Gleichung (4.10) wahr ist.

$$C^e C_r = E(eV + R_v; eR + R_R) \tag{4.5}$$

$$= E(e \sum_{i=0}^{L-1} a_i M^i + \sum_{i=0}^{L-1} r_{a_i} M^i; \boxed{R}) \tag{4.6}$$

$$= E(\sum_{i=0}^{L-1} e a_i M^i + r_{a_i} M^i; \boxed{R}) \tag{4.7}$$

$$= E(\sum_{i=0}^{L-1} (e a_i + r_{a_i}) M^i; \boxed{R}) \tag{4.8}$$

$$= E(\sum_{i=0}^{L-1} \boxed{a_i} M^i; \boxed{R}) \tag{4.9}$$

$$= E(\boxed{V}, \boxed{R}) \tag{4.10}$$

Auf der linken Seite von Gleichung (4.5) wird das Statement, also der Ciphertext, verwendet. Wenn der Prover ein Commitment auf die richtigen $\boxed{a_i}$ berechnet hat, gilt Gleichung (4.10) und der Verifier weiß, dass die verschleierten Stimmen auch zum Ciphertext gehören. Für die Überprüfung des Commitments kann man dasselbe Prinzip verwenden, indem die Gleichung (4.14) getestet wird.

$$c^e c_r = c(a_0, \ldots, a_{L-1}; r) \tag{4.11}$$

$$= c(a_0, \ldots, a_{L-1}; r) \tag{4.12}$$

$$= c(e a_0 + r_{a_0}, \ldots, e a_{L-1} + r_{a_{L-1}}; er + r_r) \tag{4.13}$$

$$= c(\boxed{a_0}, \ldots, \boxed{a_{L-1}}; \boxed{r}) \tag{4.14}$$

4.2 Multiple Choice

Bei der Wahlart Multiple Choice kann der/die WählerIn N verschiedene KandidatInnen auswählen. Dazu wählt er/sie $a_i = 1$, wenn er/sie KandidatIn i wählen will und $a_i = 0$, wenn nicht. Dazu wird das obige Argument 4.1 für das richtige Format verwendet. Außerdem muss noch zusätzlich bewiesen werden, dass alle a_i entweder 0 oder 1 sind. Auch, dass alle Stimmen aufsummiert N ergeben muss gezeigt werden. Deswegen ist Protokoll 4.2 ein Argument für folgende Relation.

$$R = \left\{ ((C, N), (V, R, a_0, \ldots, a_{L-1})) : \begin{array}{l} C = E(V; R) \text{ und } V = \sum_{i=0}^{L-1} a_i M^i \text{ und} \\ \sum_{i=0}^{L-1} (a_i^2 - a_i) = 0 \text{ und } \sum_{i=0}^{L-1} a_i = N \end{array} \right\} \tag{4.15}$$

Algorithmus 4.2 Multiple Choice

Input: C, N
Prover Input: $a_0, \ldots, a_{L-1} \in \{0,1\}$ und $R \in \{0, 1^{l_R}\}$, sodass $C = E(\sum_{i=0}^{L-1} a_i M^i; R)$

function ARGUMENT($a_0, \ldots, a_{L-1}$ und R)
  $r \xleftarrow{\$} \{0,1\}^{l_r}; r_{a_0}, \ldots, r_{a_{L-1}} \xleftarrow{\$} \{0,1\}^{1+l_e+l_s}$ // Generiere Zufallswerte
  $r_r \xleftarrow{\$} \{0,1\}^{l_r+l_e+l_s}; R_R \xleftarrow{\$} \{0,1\}^{l_R+l_e+l_s}$
  $\Delta = \sum_{i=0}^{L-1}(2a_i - 1)r_{a_i}$
  $c = com(a_0, \ldots, a_{L-1}, \Delta; r)$ // Berechne Commitments
  $c_r = com(r_{a_0}, \ldots, r_{a_{L-1}}, \sum_{i=0}^{L-1} r_{a_i}^2; r_r)$
  $R_V = \sum_{i=0}^{L-1} r_{a_i} M^i$
  $r_\Sigma = \sum_{i=0}^{L-1} r_{a_i}$
  $C_R = E(R_V; R_R)$
  $e = hash(C, C_R, c, c_r, r_\Sigma, aux)$ // Challenge berechnen
  $\boxed{R} = eR + R_R; \boxed{a_i} = ea_i + r_{a_i}; \boxed{r} = er + r_r$ // Verschleiern
  return $C_R, c, c_r, r_\Sigma, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$
end function

function VERIFIKATION(C,$C_R, c, c_r, r_\Sigma, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$)
  $e = hash(C, C_R, c, c_r, r_\Sigma, aux)$
  $\boxed{V} = \sum_{i=0}^{L-1} \boxed{a_i} M^i$ // Stimmzettel erstellen
  $\boxed{\Delta} = \sum_{i=0}^{L-1} (\boxed{a_i}^2 - e\boxed{a_i})$
  if $C^e C_R = E(\boxed{V}; \boxed{R}) \wedge c^e c_r = com(\boxed{a_1}, \ldots, \boxed{a_{L-1}}, \boxed{\Delta}; \boxed{r}) \wedge \sum_{i=0}^{L-1} \boxed{a_i} = eN + r_\Sigma$ **then**
    return True
  end if
  return False
end function

Für jede ganze Zahl x außer 0 und 1 gilt $x^2 > x$. Für 0 und 1 gilt jedoch $x^2 = x$. Diese Eigenschaft kann genutzt werden, da aus $\sum_{i=0}^{L-1}(a_i^2 - a_i) = 0$ nun folgt, dass für jedes $a_i \in \{0,1\}$ gilt. Der Verifier überprüft $c^e c_r = com(\boxed{a_1}, \ldots, \boxed{a_{L-1}}, \boxed{\Delta}; \boxed{r})$. Die ersten $L-1$ Einträge gehören zum allgemeinen Argument 4.1 und bestätigen die Korrektheit der $\boxed{a_i}$. Mit dem Eintrag $\boxed{\Delta}$ wird die Gleichung (4.21) geprüft.

(4.16)
$$\sum_{i=0}^{L-1}(\boxed{a_i}^2 - e\boxed{a_i}) = \sum_{i=0}^{L-1}(ea_i + r_{a_i})^2 - e(ea_i + r_{a_i})$$

(4.17)
$$= \sum_{i=0}^{L-1} e^2 {a_i}^2 + 2ea_i r_{a_i} + r_{a_i}^2 - e^2 a_i - e r_{a_i}$$

(4.18)
$$= e^2 \sum_{i=0}^{L-1} a_i^2 - a_i + \sum_{i=0}^{L-1} 2a_i r a_i - r_{a_i} + \sum_{i=0}^{L-1} r_{a_i}^2$$

(4.19)
$$= e^2 \sum_{i=0}^{L-1} a_i^2 - a_i + e \sum_{i=0}^{L-1} (2a_i - 1) r_{a_i} + \sum_{i=0}^{L-1} r_{a_i}^2$$

(4.20)
$$= e^2 \sum_{i=0}^{L-1} a_i^2 - a_i + e\Delta + \sum_{i=0}^{L-1} r_{a_i}^2$$

(4.21)
$$= e\Delta + \sum_{i=0}^{L-1} r_{a_i}^2$$

Die letzte Gleichung (4.21) gilt nur, wenn $\sum_{i=0}^{L-1} a_i^2 - a_i = 0$, da $e^2 > 0$. Daraus folgt, wie erklärt, dass alle $a_i \in \{0, 1\}$.

Um $\sum_{i=0}^{L-1} a_i = N$ zu zeigen, überprüft der Verifier Gleichung (4.24). Wenn N=L, also alle KandidatInnen gewählt werden können, kann dieser Teil auch aus dem Protokoll entfernt werden.

(4.22)
$$\sum_{i=0}^{L-1} \boxed{a_i} = \sum_{i=0}^{L-1} ea_i + r_{a_i}$$

(4.23)
$$= \sum_{i=0}^{L-1} ea_i + \sum_{i=0}^{L-1} r_{a_i}$$

(4.24)
$$= \sum_{i=0}^{L-1} eN + r_\Sigma$$

Die letzte Gleichung (4.24) gilt nur dann, wenn $\sum_{i=0}^{L-1} a_i = N$ gilt. Sollte bei dieser Wahlart noch zusätzlich ein Minimum an Stimmen gewünscht sein, die der/die WählerIn verteilen muss. Dann können hier die Bulletproofs aus Abschnitt 4.5 verwendet werden.

4.3 Ranking

Bei der Wahlart Ranking erstellt der/die WählerIn eine geordnete Rangliste. Außerdem ist spezifiziert, wie die Punktverteilung P aussieht. Diese kann mit verschiedene Abstufungen und Rängen mit keiner Stimme definiert werden. Zum Beispiel bekommt der erste Platz L Stimmen, der zweite $L-1$, bis der letzte noch eine Stimme bekommt. Deswegen ordnet eine Funktion π den KandidatInnen die Stimmzahl, also nach der festgelegten Punkteverteilung P, zu. Es muss also überprüft werden, ob es sich auch um eine Permutation handelt. Das heißt, jeder Wert wird nur einmal zugeordnet und es wird keine Stimmzahl außerhalb der definierten Punkteverteilung vergeben. Dafür wird ein zusätzlicher Beweis 4.4 verwendet der für zwei Listen A, B zeigt, dass A eine Permutation von B ist. Diese Art von Beweise wird auch *Proof of Shuffle* (Abschnitt 2.1) genannt. Das allgemeine Argument 4.1 wird natürlich wieder benutzt. Der Algorithmus 4.3 zeigt also folgende Relation.

(4.25)
$$R = \left\{ ((C,P),(V,R,\pi,a_0,\ldots,a_{L-1})) : C = E(V;R) \text{ und } V = \sum_{i=0}^{L-1} a_i M^i \text{ und } \bigwedge_{i=0}^{L-1} a_i = \pi(i) \right\}$$

Bei dem *Argument of Shuffle* handelt es sich um ein *special Honest Verifier* Zero-Knowledge Argument, dass zeigt, dass ein Commitment eine Permutation einer bekannten Menge enthält. Dabei werden wieder geheime Werte durch die Challenge und ein Zufallswert verschleiert. In [Gro10] wird das Protokoll interaktiv angegeben. Mithilfe der Fiat-Shamir Transformation, kann diese Kommunikation aber entfernt werden.

Im Beweis wird genutzt, dass ein Polynomial sich unter einer Permutation der Nullstellen nicht verändert. Es gilt also $p(x) = \Pi_{i=1}^{n}(m_i - x) = \Pi_{i=1}^{n}(m_{\pi(i)} - x)$, da die Multiplikation kommutativ ist. Es wird deswegen bewiesen, dass man $m_{\pi(1)}, \ldots, m_{\pi(L)}, r$ kennt, so dass $c = com(m_{\pi(1)}, \ldots m_{\pi(L)}; r)$ und, dass $\Pi_{i=1}^{L}(m_i - x) = \Pi_{i=1}^{L}(m_{\pi(i)} - x)$ gilt. Für ein zufälliges z zeigt man $\Pi_{i=1}^{L}(m_i - z) = \Pi_{i=1}^{L}(m_{\pi(i)} - z)$ und mit überwältigender Wahrscheinlichkeit stimmt das nicht, außer $\Pi_{i=1}^{L}(m_i - x) = \Pi_{i=1}^{L}(\pi(i) - x)$ gilt für jedes x.

In Algorithmus 4.4 gilt $\boxed{f_i} = em_{\pi(i)} + d_i$, daraus folgt $\boxed{f_i} - ex = e(m_{\pi(i)} - x) + d_i$. Daraus folgt wiederum, dass $\Pi_{i=1}^{L}(\boxed{f_i} - ex) = e^L \Pi_{i=1}^{L}(m_{\pi(i)} - x) + p_{n-1}(e)$, wobei p_{n-1} ein Polynom $(n-1)$-Grades ist. Wenn nun $\Pi_{i=1}^{L}(\boxed{f_i} - ex) = e^L \Pi_{i=1}^{L}(m_i - x) + p_{n-1}(e)$ gilt, folgt daraus, dass $\Pi_{i=1}^{L}(m_i - x) = \Pi_{i=1}^{L}(\pi(i) - x)$, da in den $\boxed{f_i}$ die $m_{\pi(i)}$ verschleiert werden. Um diese Gleichung zu zeigen, definieren wir $F_j = e\Pi_{i=1}^{j}(m_{\pi(i)} - x) + \Delta_j$, wobei $\Delta_1 = d_1$, $\Delta_L = 0$ und $\Delta_2, \ldots, \Delta_{L-1}$ zufällige Werte sind. Daraus folgt $F_1 = \boxed{f_1} - ex$ und $F_L = e\Pi_{i=1}^{L}(m_{\pi(i)} - x)$. Der Verifier berechnet $eF_{i+1} = F_i(f_{i+1} - ex) + f_{\Delta_i}$ und überprüft, ob $F_L = e\Pi_{i=1}^{L}(m_i - x)$. Dadurch gilt aus seiner Sicht:

(4.26)
$$e^L \Pi_{i=1}^{L}(m_i - x) = e^{L-1} F_L = e^{L-2} F_{L-1}(\boxed{f_L} - ex) + \boxed{f_{\Delta_L}}$$

(4.27)
$$= \cdots = \Pi_{i=1}^{L}(\boxed{f_i} - ex) - p_{L-1}(e)$$

Es gilt also die obige Gleichung und damit mit überwältigender Wahrscheinlichkeit $\Pi_{i=1}^{L}(m_i - x) = \Pi_{i=1}^{L}(m_{\pi(i)} - x)$. Damit sind die Nullstellen identisch und m_π ist eine Permutation.

Algorithmus 4.3 Ranking

Input: C, P
Prover Input: Permutation π und $R \in \{0,1^{l_R}\}$, sodass $C = E(\sum_{i=0}^{L-1} \pi(i)M^i; R)$

function ARGUMENT(π und R)
  $r \xleftarrow{\$} \{0,1\}^{l_r}; r_{a_0}, \ldots, r_{a_{L-1}} \xleftarrow{\$} \{0,1\}^{1+l_e+l_s}$ // Generiere Zufallswerte
  $r_r \xleftarrow{\$} \{0,1\}^{l_r+l_e+l_s}; R_R \xleftarrow{\$} \{0,1\}^{l_R+l_e+l_s}$
  $a_i = \pi(i)$
  $c = com(a_0, \ldots, a_{L-1}; r)$ // Berechne Commitments
  $c_r = com(r_{a_0}, \ldots, r_{a_{L-1}}; r_r)$
  $p =$ PERMUTATION-ARGUMENT(c, P) // Permutation
  $R_V = \sum_{i=0}^{L-1} r_{a_i} M^i$
  $C_R = E(R_V; R_R)$
  $e = hash(C, C_R, c, c_r, p, aux)$ // Challenge berechnen
  $\boxed{R} = eR + R_R; \boxed{a_i} = ea_i + r_{a_i}; \boxed{r} = er + r_r$ // Verschleiern
  return $C_R, c, c_r, p, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$
end function

function VERIFIKATION(C,$C_R, c, c_r, p, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$)
  $e = hash(C, C_R, c, c_r, p, aux)$
  $shuffle =$ PERMUTATION-VERIFIKATION(p) // Permutation verifizieren
  $\boxed{V} = \sum_{i=0}^{L-1} \boxed{a_i} M^i$ // Stimmzettel erstellen
  if $C^e C_R = E(\boxed{V}; \boxed{R}) \wedge c^e c_r = com(\boxed{a_1}, \ldots, \boxed{a_{L-1}}; \boxed{r}) \wedge shuffle$ **then**
    return True
  end if
  return False
end function

4.4 Rating

Bei der Wahlart Rating gibt es zwei verschiedene Varianten. Entweder die N Stimmen, die der/die WählerIn verteilen kann, dürfen an alle KandidatInnen vergeben werden. Oder jede/r KandidatIn wird einzeln mit 0 bis N Stimmen bewertet. Da diese Varianten sich stark unterscheiden sind zwei verschiedene Argumente notwendig.

4.4.1 N Stimmen für Alle

Die N Stimmen werden an alle KandidatInnen zusammen verteilt. Das Argument für das korrekte Format 4.1 des Ciphertextes wird wieder verwendet. Damit Niemand negative Punkte an einzelne KandidatInnen verteilen kann, muss überprüft werden, ob alle a_i größer oder gleich 0 sind. Eine Zahl der Form $4a_i + 1$ kann immer als Summe von drei Quadraten geschrieben werden. Aus dieser

Algorithmus 4.4 Permutation

Input: $c, (1, \ldots, L)$
Prover Input: Permutation π und $r \in \{0, 1^{l_r}\}$, sodass $c = com(m_{\pi(1)}, \ldots, m_{\pi(L-1)}; r)$

function ARGUMENT(π, r, c)
  $x = hash(c, (1, \ldots, L), aux)$ // Erste Verifiernachricht
  $d_1, \ldots, d_L, r_d, r_\Delta, r_a \xleftarrow{\$} \mathbb{Z}_q$ // Generiere Zufallswerte
  $\Delta_1 = d_1; \Delta_2, \ldots, \Delta_{L-1} \xleftarrow{\$} \mathbb{Z}_q; \Delta_L = 0$
  $a_i = \Pi_{j=1}^{i}(m_{\pi(j)} - x)$
  $c_d = com(a_1, \ldots, a_L; r_d)$ // Berechne Commitments
  $c_\Delta = com(-\Delta_1 d_2, \ldots, -\Delta_{L-1} d_L; r_\Delta)$
  $c_a = com(\Delta_2 - (m_{\pi(2)} - x)\Delta_1 - a_1 d_2, \ldots, \Delta_L - (m_{\pi(L)} - x)\Delta_{L-1} - a_{L-1} d_L; r_a)$
  $e = hash(c_d, c_\Delta, c_a)$ // Challenge berechnen
  $\boxed{f_i} = em_{\pi(i)} + d_i; \boxed{z} = er + r_d$ // Verschleiern
  $\boxed{f_{\Delta_i}} = e(\Delta_{i+1} - (m_{\pi(i+1)} - x)\Delta_i - a_i d_{i+1}) - \Delta_i d_{i+1}; \boxed{z_\Delta} = er_a + r_\Delta$
  return $\boxed{f_1}, \ldots, \boxed{f_L}, z, \boxed{f_{\Delta_1}}, \ldots \boxed{f_{\Delta_{L-1}}}, \boxed{z_\Delta}$
end function

function VERIFIKATION($c, \boxed{f_1}, \ldots, \boxed{f_L}, z, \boxed{f_{\Delta_1}}, \ldots \boxed{f_{\Delta_{L-1}}}, \boxed{z_\Delta}$)
  $x = hash(c, (1, \ldots, L), aux)$
  $e = hash(c_d, c_\Delta, c_a)$
  $check1 \leftarrow c^e c_d = com(\boxed{f_1}, \ldots, \boxed{f_L}; z)$
  $check2 \leftarrow c_a^e c_\Delta = com(\boxed{f_{\Delta_1}}, \ldots, \boxed{f_{\Delta_L}}; z_\Delta)$
  $F_1 = \boxed{f_1} - ex$
  $eF_2 = F_1(\boxed{f_2} - ex) + \boxed{f_{\Delta_1}}; \ldots; eF_L = F_{L-1}(\boxed{f_n} - ex) + \boxed{f_{\Delta_{L-1}}}$
  $check3 \leftarrow F_L = e\Pi_{i=1}^{L}(m_i - x)$
  if $check1 \wedge ceck2 \wedge check3$ **then**
    return True
  end if
  return False
end function

Eigenschaft folgt, dass a_i nicht negativ ist. Außerdem dürfen nur N Stimmen vergeben werden. Deswegen wird in Protokoll 4.5 folgende Relation bewiesen.

(4.28)

$$R = \left\{ \begin{array}{l} ((C, N), (V, R, a_0, b_0, c_0, d_0 \ldots, a_{L-1}, b_{L-1}, c_{L-1}, d_{L-1})) : \\ C = E(V; R) \text{ und } V = \sum_{i=0}^{L-1} a_i M^i \text{ und } \bigwedge_{i=0}^{L-1} 4a_i + 1 = b_i^2 + c_i^2 + d_i^2 \text{ und } \sum_{i=0}^{L-1} a_i = N \end{array} \right\}$$

Für die nicht Negativität der Stimmen gibt es auch Alternativen, wie Boudot [Bou00]. Hier wird eine Variation von [LAN02] benutzt.

Um die 3 Quadrate zu finden, nutzt der Prover folgende Variante von Robin Shallit [RS86].

1. Rate gerades b_i^2 zufällig, sodass $4a_i + 1 - b_i^2$ ein Produkt von Primzahlen mit $1 \; mod \; 4$ ist.

Algorithmus 4.5 Rating

Input: C, N
Prover Input: $0 \leq a_0, \ldots, a_{L-1}$ und $R \in \{0,1^{l_R}\}$, sodass $C = E(\sum_{i=0}^{L-1} a_i M^i; R)$

function ARGUMENT($a_0, \ldots, a_{L-1}$ und R)
  Finde b_i, c_i, d_i, sodass $4a_i = b_i^2 + c_i^2 + d_i^2$
  $r \xleftarrow{\$} \{0,1\}^{l_r}$; $r_{a_i}, r_{b_i}, r_{c_i}, r_{d_i} \xleftarrow{\$} \{0,1\}^{log(N)+l_e+l_s}$ // Generiere
  $r_r \xleftarrow{\$} \{0,1\}^{l_r+l_e+l_s}$; $R_R \xleftarrow{\$} \{0,1\}^{l_R+l_e+l_s}$ // Zufallswerte
  $\Delta_i = 4r_{a_i} - 2b_i r_{b_i} - 2c_i r_{c_i} - 2d_i r_{d_i}$
  $\Delta = \sum_{i=0}^{L-1}(2a_i - 1)r_{a_i}$
  $c = com(a_0, b_0, c_0, d_0, \Delta_0, \ldots, a_{L-1}, b_{L-1}, c_{L-1}, d_{L-1}\Delta_{L-1}; r)$
  $c_r = com(r_{a_0}, r_{b_0}, r_{c_0}, r_{d_0}, -r_{a_0}^2 - r_{b_0}^2 - r_{c_0}^2 - r_{d_0}^2,$
  $\ldots, r_{a_{L-1}}, r_{b_{L-1}}, r_{c_{L-1}}, r_{d_{L-1}}, -r_{a_{L-1}}^2 - r_{b_{L-1}}^2 - r_{c_{L-1}}^2 - r_{d_{L-1}}^2; r_r)$
  $R_V = \sum_{i=0}^{L-1} r_{a_i} M^i$
  $r_\Sigma = \sum_{i=0}^{L-1} r_{a_i}$
  $C_R = E(R_V; R_R)$
  $e = hash(C, C_R, c, c_r, r_\Sigma, aux)$ // Challenge berechnen
  $\boxed{R} = eR + R_R$; $\boxed{a_i} = ea_i + r_{a_i}$; $\boxed{r} = er + r_r$ // Verschleiern
  $\boxed{b_i} = eb_i + r_{b_i}$; $\boxed{c_i} = ec_i + r_{c_i}$; $\boxed{d_i} = ed_i + r_{d_i}$
  return $C_R, c, c_r, r_\Sigma, \boxed{R}, \boxed{a_0}, \boxed{b_0}, \boxed{c_0}, \boxed{d_0} \ldots, \boxed{a_{L-1}}, \boxed{b_{L-1}}, \boxed{c_{L-1}}, \boxed{d_{L-1}}\boxed{r}$
end function

function VERIFIKATION(C,$C_R, c, c_r, r_\Sigma, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$)
  $e = hash(C, C_R, c, c_r, r_\Sigma, aux)$
  $\boxed{V} = \sum_{i=0}^{L-1} \boxed{a_i} M^i$ // Stimmzettel erstellen
  $\boxed{\Delta_i} = e(4\boxed{a_i} + e) - \boxed{b_i}^2 - \boxed{c_i}^2 - \boxed{d_i}^2$
  if $C^e C_R = E(\boxed{V}; \boxed{R}) \wedge c^e c_r = com(\boxed{a_1}, \ldots, \boxed{\Delta_{L-1}}; \boxed{r}) \wedge \sum_{i=0}^{L-1} \boxed{a_i} = eN + r_\Sigma$ **then**
    return True
  end if
  return False
end function

2. Diese Primzahlen als Summe von 2 Quadraten mit Cornacchias Algorithmus [Coh13] schreiben. Dieser Algorithmus bekommt eine Primzahl als Input und berechnet zwei Quadrate die summiert, die Primzahl ergeben.

3. Mit $X = a^2 + b^2 \wedge Y = c^2 + d^2 \Longrightarrow XY = (ac + bd)^2 + (ad - bc)^2$, c_i und d_i berechnen, so dass $4a_i + 1 - b_i^2 = c_i^2 + d_i^2$.

In der Verifikation wird zu jedem i für die verschleierten Zahlen $\boxed{a_i}$, $\boxed{b_i}$, $\boxed{c_i}$, $\boxed{d_i}$ ein Commitment berechnet. Mit $c^e c_r = com(\boxed{a_0}, \ldots, \boxed{\Delta_{L-1}})$ wird dabei überprüft, ob diese Zahlen auch im Commitment verwendet wurden. Der wichtige Teil passiert mit den $\boxed{\Delta_i}$.

(4.29)
$$\boxed{\Delta_i} = e(4\boxed{a_i} + e) - \boxed{b_i}^2 - \boxed{c_i}^2 - \boxed{d_i}^2$$

(4.30)
$$= e(4(ea_i + r_{a_i}) + e) - (eb_i + r_{b_i})^2 - (ec_i + r_{c_i})^2 - (ed_i + r_{d_i})^2$$

(4.31)
$$= e(4ea_i + 4r_{a_i} + e) - (e^2 b_i^2 + 2eb_i r_{b_i} + r_{b_i}^2) - (e^2 c_i^2 + 2ec_i r_{c_i} + r_{c_i}^2) - (e^2 d_i^2 + 2ed_i r_{d_i} + r_{d_i}^2)$$

(4.32)
$$= e(4ea_i + 4r_{a_i} + e) - (e^2 b_i^2 + 2eb_i r_{b_i} + r_{b_i}^2) - (e^2 c_i^2 + 2ec_i r_{c_i} + r_{c_i}^2) - (e^2 d_i^2 + 2ed_i r_{d_i} + r_{d_i}^2)$$

(4.33)
$$= 4e^2 a_i + e4r_{a_i} + e^2 - e^2 b_i^2 - 2eb_i r_{b_i} - r_{b_i}^2 - e^2 c_i^2 - 2ec_i r_{c_i} - r_{c_i}^2 - e^2 d_i^2 - 2ed_i r_{d_i} - r_{d_i}^2$$

(4.34)
$$= e^2(4a_i + 1 - b_i^2 - c_i^2 - d_i^2) + e(4r_{a_i} - 2b_i r_{a_i} - 2c_i r_{c_i} - 2d_i r_{d_i}) - r_{b_i}^2 - r_{c_i}^2 - r_{d_i}^2$$

(4.35)
$$= e(4r_{a_i} - 2b_i r_{b_i} - 2c_i r_{c_i} 2d_i r_{d_i}) - r_{b_i}^2 - r_{c_i}^2 - r_{d_i}^2$$

(4.36)
$$= e\Delta_i - r_{b_i}^2 - r_{c_i}^2 - r_{d_i}^2$$

Da $e^2 \geq 0$, muss $4a_i + 1 - b_i^2 - c_i^2 - d_i^2 = 0$ sein, damit Gleichung (4.35) gilt. Dies Impliziert $4a_i + 1 = b_i^2 + c_i^2 + d_i^2$ und damit auch, dass a_i nicht negativ ist. Dass die Summe der Stimmen N ist, wird wie im Multiple Choice Argument 4.2 gezeigt.

4.4.2 N Stimmen für jede/n KandidatIn

Bei der zweiten Variante kann der/die WählerIn jede/n KandidatIn einzeln mit 0 bis N Stimmen bewerten. Das heißt zusätzlich zum allgemeinen Argument 4.1, muss gezeigt werden, dass $a_i \in \{0, \ldots, N\}$. Es muss also gezeigt werden, dass die Stimmen in einem bestimmten Intervall liegen. Solche Beweise nennt man *Range Proof* (Abschnitt 2.1). Dies wird, wie in Abschnitt 4.5 beschrieben mithilfe von Bulletproofs erreicht. Das Argument ähnelt dem des Rankings, nur dass, statt des

Algorithmus 4.6 Rating Einzeln

Input: C, N
Prover Input: $0 \leq a_0, \ldots, a_{L-1}$ und $R \in \{0,1^{l_R}\}$, sodass $C = E(\sum_{i=0}^{L-1} a_i M^i; R)$

function ARGUMENT($a_0, \ldots, a_{L-1}$ und R)
  $r \xleftarrow{\$} \{0,1\}^{l_r}; r_{a_0}, \ldots, r_{a_{L-1}} \xleftarrow{\$} \{0,1\}^{1+l_e+l_s}$ // Generiere Zufallswerte
  $r_r \xleftarrow{\$} \{0,1\}^{l_r+l_e+l_s}; R_R \xleftarrow{\$} \{0,1\}^{l_R+l_e+l_s}$
  $c = com(a_0, \ldots, a_{L-1}; r)$ // Berechne Commitments
  $c_r = com(r_{a_0}, \ldots, r_{a_{L-1}}; r_r)$
  $p = $ BULLETPROOF-ARGUMENT(c, N) // Intervall
  $R_V = \sum_{i=0}^{L-1} r_{a_i} M^i$
  $C_R = E(R_V; R_R)$
  $e = hash(C, C_R, c, c_r, p, aux)$ // Challenge berechnen
  $\boxed{R} = eR + R_R; \boxed{a_i} = ea_i + r_{a_i}; \boxed{r} = er + r_r$ // Verschleiern
  return $C_R, c, c_r, p, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$
end function

function VERIFIKATION(C,$C_R, c, c_r, p, \boxed{R}, \boxed{a_0}, \ldots, \boxed{a_{L-1}}, \boxed{r}$)
  $e = hash(C, C_R, c, c_r, p, aux)$
  $intervall = $ PERMUTATION-VERIFIKATION(p) // Bulletproof verifizieren
  $\boxed{V} = \sum_{i=0}^{L-1} \boxed{a_i} M^i$ // Stimmzettel erstellen
  if $C^e C_R = E(\boxed{V}; \boxed{R}) \wedge c^e c_r = com(\boxed{a_1}, \ldots, \boxed{a_{L-1}}; \boxed{r}) \wedge intervall$ **then**
    return True
  end if
  return False
end function

Beweis einer Permutation, ein *Range Proof* eingebunden wird. In Protokoll 4.6 wir folgende Relation gezeigt.

(4.37)
$$R = \left\{ ((C, N), (V, R, a_0 \ldots, a_{L-1})) : C = E(V; R) \text{ und } V = \sum_{i=0}^{L-1} a_i M^i \text{ und } \bigwedge_{i=0}^{L-1} a_i \in [0, N] \right\}$$

4.5 Range Proofs

Range Proofs (Abschnitt 2.1) sind Beweise, dass ein Wert in einem bestimmten Intervall liegt. Dabei wird jedoch keine weitere Information preisgegeben. Deswegen spricht man auch von Zero-Knowledge Range Proofs ZKRP. Das eine Zahl in einen festgelegtem Bereich liegt, kann von vielseitig Nutzen sein. In [MKVK19] werden die Anwendungsgebiete diskutiert. Zum Beispiel könnte eine Person beweisen, dass sie älter als 18, und somit volljährig, ist ohne das wirkliche Alter zu verraten. Aber interessanter für diese Arbeit ist die Anwendung in E-Voting.

So muss in der Wahlart Rating 4.4 gezeigt werden, dass die Stimmanzahl in einem Intervall liegen und ein/e WählerIn nicht mehr Stimmen als erlaubt vergeben kann.

Ein Ansatz für solche Beweise ist, zu zeigen, dass $x - a$ und $b - x$ nicht negativ sind. Daraus folgt, dass x im Intervall $[a, b]$ liegt. Dazu reicht es diese zwei Terme jeweils, als Summe von Quadraten zu schreiben, und diese somit positiv sind. Dies wurde zuerst von Boudot [Bou00] verwirklicht. Lipmaa [Lip03] entwickelte eine effizientere Methode mit der nur vier Quadrate gebraucht werden. In [Gro05] wird dieser Ansatz nochmals verbessert, da nur 3 Quadraten benötigt werden, wenn eine Zahl die Form $4a + 1$ hat. Daraus folgt, dass a positiv ist. Diese drei Ansätze sind in der Größe des Beweises konstant, aber vor allem im Vergleich mit anderen Ansätzen bei kleinen Intervallen, sehr groß. Eine andere Methode basiert auf Signaturen. Dabei werden im Voraus alle Zahlen in gewünschten Bereich signiert. Der Prover beweist nun, dass er eine Signatur kennt und die Zahl damit im Intervall liegt. Ein Beispiel für diese Methode ist der Beweis von Camenisch, Chaabouni et al. [CC+08]. Der letzte Ansatz ist, die Zahl in ihrer Bit-Repräsentation darzustellen. Dabei wird bewiesen, dass es sich auch wirklich um Bits handelt und die Repräsentation zur Zahl passt. Durch eine Darstellung zu einer anderen Basis können diese noch verbessert werden. Zu dieser Kategorie gehören auch die nachfolgenden Bulletproofs.

4.5.1 Bulletproofs

Eine effiziente Methode sind die Bulletproofs [BBB+18]. Diese brauchen im Gegensatz zu vielen Alternativen kein vertrauenswürdiges Setup und sind deswegen auch in Einsatzgebieten, wie Kryptowährungen interessant. Dabei wird ein Argument über das innere Produkt zweier Vektoren benutzt. Als gemeinsamen Input bekommen der Prover und Verifier ein Commitment auf eine Zahl, die in einem bestimmten Intervall liegen soll. Es wird ein Pedersen-Commitment (Definition 2.3.3) [Ped91] benutzt, da der Beweis bestimmte Eigenschaften dieses Verfahrens ausnutzt. Für den Einsatz in dem Beweissystem von [Gro05] muss das Commitment Verfahren jedoch zwei Eigenschaften erfüllen. Zum einen muss es homomorph sein. Und zum anderen muss die sogenannte *Root Extraction* Eigenschaft besitzen. Siehe Definition 2.3.1.

Zunächst der Homomorphismus.

(4.38)
$$com(\mathbf{x}; r_1) \cdot com(\mathbf{y}; r_2) = h^{r_1} \Pi_{i=1}^{n} \mathbf{g}^{\mathbf{x}} \cdot h^{r_2} \Pi_{i=1}^{n} \mathbf{g}^{\mathbf{y}}$$

(4.39)
$$= h^{r_1} \Pi_{i=1}^{n} g_i^{x_i} \cdot h^{r_2} \Pi_{i=1}^{n} g_i^{y_i}$$

(4.40)
$$= h^{r_1 + r_2} \Pi_{i=1}^{n} g_i^{x_i + y_i}$$

(4.41)
$$= com(x_1 + y_1, \ldots, x_n + y_n; r_1 + r_2)$$

Das Perdersen-Commitment besitzt auch die *Root-Extraction* Eigenschaft

(4.42)
$$c = com(x_1, \ldots, x_n; r)$$
(4.43)
$$\implies c^e = (h^r \Pi_{i=1}^n \mathbf{g}^{\mathbf{x}})^e$$
(4.44)
$$= h^{er} (\Pi_{i=1}^n g_i^{x_i})^e$$
(4.45)
$$= h^{er} \Pi_{i=1}^n g_i^{ex_i}$$
(4.46)
$$= com(ex_1, \ldots, ex_n; er)$$
(4.47)
$$\implies e | ex_i \; \forall i \in \{1, n\}$$
(4.48)
$$\implies com(\frac{ex_1}{e}, \ldots, \frac{ex_n}{e}; \frac{er}{e}) = com(x_1, \ldots, x_n; r) = c$$

Die *Bulletproofs* basieren auf den Ideen von Bootle et al. [BCC+16] und verbessert das Argument für das innere Produkt. Die Grundlage ist das Problem des Lösens von diskreten Logarithmen. Also die *discrete Logarithm Assumption* (Definition 2.3.2). Des Weiteren kann die Kommunikation zwischen Prover und Verifier durch die Fiat-Shamir Transformation entfernt werden. Zusätzlich zum Beweis, das eine Zahl in einem Intervall liegt, können damit Lösungen für arithmetische Schaltkreise und allgemeine NP-Sprachen bewiesen werden. Das Argument kann auch in ein MPC-Protokoll umgewandelt werden, dies ist jedoch für unseren Einsatz nicht relevant. Interessanter ist die Eigenschaft, dass mehrere Beweise für dasselbe Intervall kostengünstig in nur einen Beweis zusammengefasst werden können.

4.5.2 Notation

$\mathbb{G}$ sei eine zyklische Gruppe mit Ordnung p, wobei p eine Primzahl ist. Außerdem sei $\mathbb{G}^n$ und $\mathbb{Z}_p^n$ Vektorräume mit Dimension n. Weiter sei $Z_p^\star = \mathbb{Z}_p \setminus \{0\}$. Dass g und h Generatoren der Gruppe $\mathbb{G}$ sind, wird durch $g, h \in \mathbb{G}$ beschrieben. Fettgedruckte Buchstaben sind Vektoren $\mathbf{a}$ oder Matrizen $\mathbf{V}$. Das innere Produkt wird folgendermaßen definiert $\langle \mathbf{a}, \mathbf{b} \rangle = \sum_{i=0}^n a_i b_i$. Des Weiteren ist $\mathbf{a} \circ \mathbf{b} = (a_1 b_1, \ldots, a_n b_n)$ die elementweise Multiplikation und $\mathbf{g}^{\mathbf{a}} = \Pi_{i=1}^n g_i^{a_i}$. Für ein $k \in \mathbb{Z}_p^\star$ ist $\mathbf{k}^n = (1, k, k^2, \ldots, k^{n-1})$.

4.5.3 Argument für inneres Produkt

Für den Intervall-Beweis wird ein Argument benötigt, dass der Prover ein inneres Produkt zweier Vektoren eines Vektor-Commitments kennt. Zunächst wird jedoch ein Hilfsprotokoll 4.7 benötigt, dass folgende Relation zeigt.

(4.49)
$$R = \left\{((\mathbf{a}, \mathbf{b}), (\mathbf{g}, \mathbf{h}, u, P)) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}u^{\langle \mathbf{a},\mathbf{b}\rangle}\right\}$$

Der Algorithmus 4.7 ruft sich selbst rekursiv auf. Dabei halbiert sich die Dimension der Vektoren $\mathbf{a}, \mathbf{b}$ jede Iteration. Das folgende Beispiel demonstriert das Prinzip dahinter. Wenn das Wissen von a_1 und a_2 von einem *binding* Commitment $P = g_1^{a_1} g_2^{a_2}$ gezeigt werden soll, kann einfach a_1, a_2 geschickt werden. Doch falls der Verifier $R = g_1^{a_2}$ und $L = g_2^{a_1}$ kennt, kann er $g' = g_1^{(x^{-1})} g_2^{x}$ und $P' = L^{(x^2)} P R^{(x^{-2})}$ berechnen. Daraus folgt.

(4.50)
$$P' = (g_2^{a_1})^{(x^2)} g_1^{a_1} g_2^{a_2} (g_1^{a_2})^{(x^{-2})}$$
(4.51)
$$= g_1^{a_1 + a_2 x^{-2}} g_2^{a_2 + a_1 x^2}$$
(4.52)
$$= g_1^{x^{-1}(a_1 x + a_2 x^{-1})} g_2^{x(a_2 x^{-1} + a_1 x)}$$
(4.53)
$$= (g')^{a_1 x + a_2 x^{-1}}$$

Statt a_1, a_2 zu senden kann nun $a' = a_1 x + a_2 x^{-1}$ gesendet werden. Wenn $P' = (g')^{a'}$ gilt, akzeptiert der Verifier. Da L, R und a' gesendet werden, ist das Beispiel nicht besser als a_1, a_2 zu senden. Jedoch skaliert dieses Prinzip in höhere Dimensionen. Außerdem können zwei Vektoren parallel bearbeitet werden, da nur ein Korrekturfaktor verändert wird.

Für das innere Produkt wird dann folgende Relation in Protokoll 4.8 gezeigt.

(4.54)
$$R = \left\{((\mathbf{a}, \mathbf{b}), (\mathbf{g}, \mathbf{h}, P, c)) : P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}} \text{ und } c = \langle \mathbf{a}, \mathbf{b}\rangle\right\}$$

Das Protokoll 4.8 ruft dabei Algorithmus 4.7 mit $P = Pu^{xc}$ und $u = u^x$ auf. Dadurch wird getestet, ob $P = \mathbf{g}^{\mathbf{a}}\mathbf{h}^{\mathbf{b}}(u^x)^{\langle \mathbf{a},\mathbf{b}\rangle}$. Dies gilt nur, wenn $c = \langle \mathbf{a}, \mathbf{b}\rangle$ und damit wird die gewollte Relation 4.54 gezeigt.

Algorithmus 4.7 Hilfsprotokoll

Input: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n; u, P \in \mathbb{G}; a, b \in \mathbb{Z}_p^n; \mathbf{l}, \mathbf{r} \in \mathbb{G}^n$,
Prover Input: $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n; c \in \mathbb{Z}_p$

function ARGUMENT($\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{h}, P, c, \mathbf{l}, \mathbf{r}$)
    $x = hash(\mathbf{g}, \mathbf{h}, P, c)$ // Erste Verifiernachricht
    if n = 1 **then**
        return $\mathbf{g}, \mathbf{h}, P, u, a, b, \mathbf{l}, \mathbf{r}$
    else
        $n' = \frac{n}{2}$
        $c_L = \langle \mathbf{a}_{[:n']}, \mathbf{b}_{[n':]} \rangle; c_R = \langle \mathbf{a}_{[n':]}, \mathbf{b}_{[:n']} \rangle$
        $L = \mathbf{g}_{[n':]}^{a_{[:n']}} \mathbf{h}_{[:n']}^{b_{[n':]}} u^{c_L}; R = \mathbf{g}_{[:n']}^{a_{[n':]}} \mathbf{h}_{[n':]}^{b_{[:n']}} u^{c_R}$
        $\mathbf{l}.append(L); \mathbf{r}.append(R)$
        $x = hash(L, R)$
        $g' = \mathbf{g}_{[:n']}^{x^{-1}} \mathbf{g}_{[n':]}^{x}; h' = \mathbf{g}_{[:n']}^{x} \mathbf{g}_{[n':]}^{x^{-1}}$
        $P' = L^{x^2} P R^{x^{-2}}$
        $\mathbf{a}' = \mathbf{a}_{[:n']} x + \mathbf{a}_{[n':]} x^{-1}; \mathbf{b}' = \mathbf{b}_{[:n']} x^{-1} + \mathbf{b}_{[n':]} x$
        HILFSPROTOKOLL-ARGUMENT($\mathbf{g}', \mathbf{h}', P', u, \mathbf{a}', \mathbf{b}', \mathbf{l}, \mathbf{r}$)
    end if
end function

function VERIFIKATION($\mathbf{g}, \mathbf{h}, P, u, a, b, \mathbf{l}, \mathbf{r}$)
    $i = 0$
    while $i \leq log(n)$ **do**
        $n' = \frac{n}{2}$
        $x = hash(\mathbf{l}[i], \mathbf{r}[i])$
        $g' = \mathbf{g}_{[:n']}^{x^{-1}} \mathbf{g}_{[n':]}^{x}; h' = \mathbf{g}_{[:n']}^{x} \mathbf{g}_{[n':]}^{x^{-1}}$
        $P' = L^{x^2} P R^{x^{-2}}$
        i = i+1
    end while
    $c = ab$
    if $P = g^a h^b u^c$ **then**
        return True
    end if
    return False
end function

Algorithmus 4.8 Inneres Produkt

Input: $\mathbf{g}, \mathbf{h} \in \mathbb{G}^n; P \in \mathbb{G}; c \in \mathbb{Z}_p$,
Prover Input: $\mathbf{a}, \mathbf{b} \in \mathbb{Z}_p^n$

function ARGUMENT($\mathbf{a}, \mathbf{b}, \mathbf{g}, \mathbf{h}, P$)
    $x = hash(\mathbf{g}, \mathbf{h}, P, c)$ // Erste Verifiernachricht
    $P' = Pu^{xc}$
    $\mathbf{l}, \mathbf{r} \in \mathbb{G}^n$ // Initialisieren
    return HILFSPROTOKOLL-ARGUMENT($\mathbf{g}, \mathbf{h}, u^x, P', c, \mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$)
end function

function VERIFIKATION($\mathbf{g}, \mathbf{h}, u, P, \mathbf{a}, \mathbf{b}, \mathbf{l}, \mathbf{r}$)
    return HILFSPROTOKOLL-VERIFIKATION($\mathbf{g}, \mathbf{h}, u, P, a, b, \mathbf{l}, \mathbf{r}$)
end function

4.5.4 Intervall Argument

Bei dem Bulletproof wird gezeigt, dass eine Zahl aus einem Pedersen-Commitment in einem festgelegten Intervall liegt. Um Beweise in einem Intervall mit allgemeinerer Obergrenze zu verwirklichen kann [CLS10] verwendet werden. Es wird also folgende Relation in Algorithmus 4.9 gezeigt.

(4.55)
$$R = \{((v, \gamma), (V, g, h, \mathbf{g}, \mathbf{h})) : V = h^\gamma g^v \text{ und } v \in [0, 2^n - 1]\}$$

Die Idee des Bulletproof basiert darauf, dass ein v in einem bestimmten Intervall liegt, genau dann, wenn dass innere Produkt zweier Polynome $l(X), r(X)$ ein bestimmtes Format hat. Dabei ist $\mathbf{a}_L$ die Bit-Repräsentation von v. Es gilt also $v = \langle \mathbf{a}_L, \mathbf{2}^n \rangle$. Außerdem ist $\mathbf{a}_R = \neg \mathbf{a}_L$ und $\mathbf{s}_L, \mathbf{s}_R$ sind Werte zur Verschleierung von $\mathbf{a}_L, \mathbf{a}_R$. Die beiden Polynome werden nun folgendermaßen gewählt:

(4.56)
$$l(X) = \mathbf{a}_L - z\mathbf{1}^n + \mathbf{s}_L X$$

(4.57)
$$r(X) = \mathbf{y}^n \circ \mathbf{a}_r + z\mathbf{1}^n + \mathbf{s}_R X) + z^2 \mathbf{2}^n$$

Das innere Produkt ist deswegen

(4.58)
$$t(X) = \langle l(X), r(X) \rangle = \sum_{i=0}^{2} t_i X^i \text{ , mit}$$

(4.59)
$$t_0 = \langle \mathbf{a}_L, \mathbf{a}_R \circ \mathbf{y}^n \rangle + z\langle \mathbf{a}_L - \mathbf{a}_R, \mathbf{y}^n \rangle + z^2 \langle \mathbf{2}^n, \mathbf{a}_L \rangle + k(y, z)$$

(4.60)
$$k(y, z) = -z^2 \langle \mathbf{1}^n, \mathbf{y}^n \rangle - z^3 \langle \mathbf{1}^n, \mathbf{2}^n \rangle$$

Algorithmus 4.9 Bulletproof

Input: $V, g, h \in \mathbb{G}; \mathbf{g}, \mathbf{h} \in \mathbb{G}^n$,
Prover Input: $v, \gamma \in \mathbb{Z}_p$

function ARGUMENT($V, g, h, \mathbf{g}, \mathbf{h}, v, \gamma$)
  $\mathbf{a}_L \in \{0,1\}^n$, sodass $\langle \mathbf{a}_L, \mathbf{2}^n \rangle = v$
  $\mathbf{a}_R = \mathbf{a}_L - \mathbf{1}^n$
  $\mathbf{s}_L, \mathbf{s}_R \xleftarrow{\$} \mathbb{Z}_p^n$
  $\alpha, \rho \xleftarrow{\$} \mathbb{Z}_p$
  $A = h^\alpha \mathbf{g}^{\mathbf{a}_L} \mathbf{h}^{\mathbf{a}_R}; S = h^\rho \mathbf{g}^{\mathbf{s}_L} \mathbf{h}^{\mathbf{s}_R}$
  $y = hash(A, S)$
  $z = hash(A, S, y)$
  $\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_p$
  $T_1 = g^{t_1} h^{\tau_1}; T_2 = g^{t_2} h^{\tau_2}$
  $x = hash(T_1, T_2)$
  $\mathbf{l} = l(x) = \mathbf{a}_L - z\mathbf{1}^n + \mathbf{s}_L x$
  $\mathbf{r} = r(x) = \mathbf{y}^n \circ (\mathbf{a}_R + z\mathbf{1}^n + \mathbf{s}_R x) + z^2 \mathbf{2}^n$
  $t = \langle \mathbf{l}, \mathbf{r} \rangle$
  $\tau_x = \tau_1 x + \tau_2 x^2 + z^2 \gamma$
  $\mu = \alpha + \rho x$
  $h_i' = h_i^{y^{-i+1}}$
  $proof\_Inner$ = INNERES PRODUKT-ARGUMENT($\mathbf{g}, \mathbf{h}', g, P \cdot h^{-\mu}, t$)
  return $\tau_x, \mu, t, proof\_Inner$
end function

function VERIFIKATION($V, g, h, \mathbf{g}, \mathbf{h}, \tau_x, \mu, t, proof\_Inner$)
  $y = hash(A, S); z = hash(A, S, y); x = hash(T_1, T_2)$
  $h_i' = h_i^{y^{-i+1}}$
  $P = AS^x \mathbf{g}^{-z} \mathbf{h}'^{zy^n + z^2 \mathbf{2}^n}$
  $verify\_Inner$ = INNERES PRODUKT-ARGUMENT($\mathbf{g}, \mathbf{h}', g, P \cdot h^{-\mu}, t$)
  if $g^t h^{\tau_x} = g^{k(y,z) + z\langle \mathbf{1}^n, \mathbf{y}^n \rangle} V^{z^2} T_1^x T_2^{x^2} \wedge verify\_Inner$ **then**
    return True
  end if
  return False
end function

Wenn nun t_0 diese Form annimmt:

(4.61)
$$t_0 = z\langle \mathbf{1}^n, \mathbf{y}^n \rangle + z^2 v + k(y,z)$$

Dann folgt daraus:

(4.62)
$$\mathbf{a}_R \overset{1.}{=} \mathbf{a}_L - \mathbf{1}^n \text{ und } \mathbf{a}_L \circ \mathbf{a}_R \overset{2.}{=} \mathbf{0}^n \text{ und } \langle \mathbf{a}_L, \mathbf{2}^n \rangle \overset{3.}{=} v$$

Dabei zeigen 1. und 2., dass $\mathbf{a}_L \in \{0,1\}^n$ ist. Aus 3. folgt, dass $\mathbf{a}_L$ die Bit-Repräsentation von v ist. Damit muss $v \in [0, 2^n - 1]$, liegt also im gewünschten Intervall.

Der Verifier testet dazu erst, ob $t(X)$ auch das innere Produkt von $\mathbf{l}, \mathbf{r}$ ist. Dies kann er mit dem Protokoll 4.8 effizient machen. Ob t_0 das gewünschte Format hat, wird mit Gleichung (4.67) überprüft.

(4.63)
$$g^t h^{\tau_x} = g^{\langle \mathbf{l}, \mathbf{r} \rangle} h^{\tau_1 x + \tau_2 x^2 + z^2 \gamma}$$

(4.64)
$$= g^{t_0 + t_1 x + t_2 x^2} h^{\tau_1 x + \tau_2 x^2 + z^2 \gamma} \text{ , mit } t_0 \text{ im richtigem Format}$$

(4.65)
$$= g^{z^2 v + z\langle \mathbf{1}^n, \mathbf{y}^n \rangle + k(y,z) + t_1 X + t_2 X^2} h^{\tau_1 x + \tau_2 x^2 + z^2 \gamma}$$

(4.66)
$$= g^{k(y,z) + z\langle \mathbf{1}^n, \mathbf{y}^n \rangle} (g^v h^\gamma)^{z^2} (g^{t_1} h^{\tau_1})^x (g^{t_2} h^{\tau_2})^{x^2}$$

(4.67)
$$= g^{k(y,z) + z\langle \mathbf{1}^n, \mathbf{y}^n \rangle} V^{z^2} T_1^x T_2^{x^2}$$

Wenn mehrere Beweise für das gleiche Intervall gebraucht werden, können diese in einen Beweis zusammengefasst werden, anstatt jeden einzeln zu zeigen. Dazu kann Protokoll 4.9 angepasst werden, sodass folgende Relation gezeigt wird.

(4.68)
$$R = \left\{ ((v,\gamma), (\mathbf{V}, g, h, \mathbf{g}, \mathbf{h})) : V_j = h_j^\gamma g^{v_j} \text{ und } v_j \in [0, 2^n - 1] \forall j \in [1, m] \right\}$$

Die Bit-Repräsentation $\mathbf{a}_L$ von v, ist dann die Konkatenation aller Bits der v_j. Deswegen muss $r(X)$ angepasst werden.

$$r(X) = \mathbf{y}^{nm} \circ (\mathbf{a}_R + z\mathbf{1}^{nm} + \mathbf{s}_R X) + \sum_{j=1}^{m} z^{1+j} \mathbf{0}^{(j-1)n} || \mathbf{2}^n || \mathbf{0}^{()m-j)n}$$

τ_x muss jetzt die ganzen Zufallswerte der Commitments V_j enthalten und somit muss auch $k(y,z)$ modifiziert werden.

$$\tau_x = \tau_1 x + \tau_2 x^2 + \sum_{j=1}^{m} z^{1+j} \gamma_j$$

$$k(y,z) = -z^2 \langle \mathbf{1}^{nm}, \mathbf{y}^{nm} \rangle - \sum_{j=1}^{m} z^{j+2} \langle \mathbf{1}^n, \mathbf{2}^n \rangle$$

Der Test, ob t_0 das richtige Format hat, muss auch alle V_j enthalten.

$$g^t h^{\tau_x} = g^{k(y,z)+z\langle \mathbf{1}^{nm}, \mathbf{y}^{nm}\rangle} \mathbf{V}^{z^2\mathbf{z}^m} T_1^x T_2^{x^2}$$

Zuletzt wird noch das letzte Commitment an das neue $\mathbf{r}$ angepasst.

$$P = AS^x \mathbf{g}^{-z} \mathbf{h}'^{z\mathbf{y}^{nm}} \prod_{j=1}^{m} \mathbf{h}'^{z^{j+1}\mathbf{2}^n}_{[(j-1)m:jm]}$$

Durch diese Modifikationen wächst die Beweisgröße nicht um den Faktor m, sondern lediglich um einen additiven Term $2 \cdot log(m)$.

Die Beweise für *Completeness*, *Soundness* und *Zero-Knowledge* können für alle Argumente in [Gro05],[Gro10] und [BBB+18] nachgelesen werden.

5 Performance

In diesem Kapitel wird die theoretische Performance der Argumente besprochen. Zunächst werden die verschiedenen Vergleichswerte erläutert.

Es gibt verschieden Größen, um die Effizienz von Zero-Knowledge Beweissystemen zu bewerten. Die *Round Complexity* beschreibt die Anzahl der Nachrichten in einem interaktiven System. Bei *non-interactive* ZKP gibt es nur eine Nachricht und deswegen ist die Größe dieser Nachricht, also dem Beweis, ein guter Vergleichswert. Dies wird entweder in der Bit-Größe der Nachrichten oder der Anzahl der verschickten Elemente gemessen. Der Aufwand für Prover und Verifier wird entweder in Sekunden oder als Anzahl der dominierenden Operationen angegeben. In unserem Fall ist der Aufwand des Prover interessanter, da dieser auf dem Endgerät des Wählenden ausgeführt wird. Dieses besitzt unter Umständen nur begrenzte Rechenleistung. Es sollte das Ziel sein auch damit die Abstimmung in einer sinnvollen Zeit zu bewältigen. Bei der Verifizierung ist deutlich mehr Spielraum, da das MPC-Protokoll zur Auswertung wahrscheinlich diesen Prozess dominiert. Oft ist ein Ergebnis nicht direkt nötig, da nicht alle WählerInnen gleichzeitig wählen. Die ZKP können jedoch schon vor der Auszählung bestätigt werden. Auch die Rechenleistung der Wahlautorität wird meistens deutlich stärker sein.

5.1 Transformation

Der Verifier muss vor der Verifikation eine Transformation durchführen. (Siehe Kapitel 4 auf Seite 20). Diese benötigt zusätzliche L Potenzen und $L-1$ Multiplikationen. Alternativ könnten die Beweise auch so angepasst werden, dass die einzelnen Verschlüsselten Stimmen, statt der Summen Schreibweise verwendet werden. Dann müsste die Transformation nicht gemacht werden. Jedoch müssten Prover und Verifier L Verschlüsselungen, anstatt nur einer, durchführen. Vor allem der Aufwand für den Prover sollte vermieden werden.

5.2 Multiple Choice

Das Argument für Multiple Choice besteht aus L+6 Elementen. Die Verschlüsselung und Berechnung der Commitments c und c_r dominieren in diesem Fall. Der Prover muss eine Verschlüsselung und $2L+4$ Potenzen für die Berechnung der Commitments durchführen. Dabei wird von einem Commitment Verfahren ausgegangen, dass so viele für n Eingabewerte $n+1$ Potenzen und n Multiplikationen benötigt, wie zum Beispiel das Pedersen-Commitment-Scheme (Definition 2.3.4). Der Aufwand für den Verifier ist eine Verschlüsselung, eine Potenz des Ciphertextes und $L+3$ Potenzen.

5.3 Ranking

Das Argument für Ranking besteht aus $3L+4$ Elementen. Für den Beweis der Permutation berechnet der Prover $2L + 1$ Potenzen. Im restlichen Beweis kommen nochmal $2L + 1$ Potenzen dazu. So, dass insgesamt eine Verschlüsselung und $4L + 2$ Potenzen benötigt werden. Die Verifizierung der Permutation braucht $2L + 2$, der Rest $L + 1$, also zusammen $3L + 3$ Potenzen. Dazu kommt wieder eine Verschlüsselung und eine Potenz des Ciphertextes.

5.4 Rating

Das Argument für das Rating, bei der die N Stimmen auf alle KandidatInnen verteilt werden können, besteht aus $4L+6$ Elementen. Ansonsten muss der Prover $10L+4$ Potenzen und eine Verschlüsselung durchführen. Der Verifier berechnet hingegen $5L + 2$ Potenzen, eine Verschlüsselung eine Potenz des Ciphertextes.

5.5 Bulletproofs

Da der Bulletprof den Beweis für das Wissen eines inneren Produktes benutzt, zunächst die Performance des entsprechenden Algorithmus.

Das Argument für das innere Produkt besteht aus $2 \cdot \lceil log(n) \rceil$ Elemente aus $\mathbb{G}$ und zusätzlich 2 Elemente aus $\mathbb{Z}_p$. Hierbei ist n die Bit Länge der Zahl v. Der Prover berechnet $4n$ Potenzen in der Gruppe $\mathbb{G}$. In der Verifizierung müssen nur $2n$ Potenzen berechnen werden.

Das Argument für den ganzen Bulletproof besteht aus $2 \cdot \lceil log(n) \rceil + 4$ Gruppenelementen und 5 Element aus $\mathbb{Z}_p$. Da für jede/n KandidatIn ein Beweis gebraucht wird, dass die Stimmzahl im erlaubten Intervall liegt, braucht man L Argumente. Dabei ist immer dasselbe Intervall erlaubt und Bulletproofs können dadurch günstig zusammen berechnet werden. Der Beweis ist nun $2 \cdot \lceil log(n \cdot L) \rceil + 4 = 2 \cdot \lceil log(n) + log(L) \rceil + 4$ groß und somit nur um den additiven Term $2\lceil log(L) \rceil$ größer als ein einzelnes Argument. Die Erstellung des Beweises und die Verifikation sind beide linear in n.

Man muss bei den Berechnungen der Potenzen bedenken, dass die Exponenten unterschiedlich groß sein können. Somit kann die Berechnung unterschiedlichen Aufwand bedeuten.

Da die Berechnungen für Verifier und Prover immer linear zur Anzahl der KandidatInnen sind, sind diese sehr effizient.

6 Zusammenfassung und Ausblick

Zunächst wurden die Grundlagen erläutert. Der Fokus lag hierbei bei den Definitionen von Zero-Knowledge Beweisen. Im Anschluss wurde in Kapitel 3 die Ziele von E-Voting Systemen diskutiert. Zusätzlich wurde Ordinos und dessen Anforderungen zusammengefasst. Die Anforderungen an die Beweise und welche Arten am sinnvollsten sind, wurde in Kapitel 4 diskutiert. Danach wurden die resultierenden ZKP vorgestellt. Dabei wurden auch die Beweisideen erläutert und erklärt. Ein wichtiger Bestandteil sind die *Range Proofs*. Deren Anwendungsgebiete wurden in Abschnitt 4.5 diskutiert und in Kapitel 5 wird die theoretische Performance der Beweise besprochen. Der Aufwand des Prover ist dabei immer linear zur Anzahl der KandidatInnen und somit sehr effizient.

Ausblick

Die Implementierung der theoretischen Algorithmen wäre ein wichtiger Schritt um die Praxistauglichkeit der Argumente festzustellen. Dann könnte die Performance in Testwahlen zusammen mit dem Ordinossystem getestet werden. Außerdem gibt es sehr viele verschiedene Wahlarten, die nicht unbedingt in die, in dieser Arbeit benutzten, Kategorien passen. Oft könnte es reichen eines der Argumente mit einer weiteren Beschränkung zu erweitern und/oder einen anderen Teil wegzulassen. Für komplexere Wahlen, die zum Beispiel mehrere verschiedene KandidatInnenlisten in einer Wahl zusammenfassen, müssten die Algorithmen deutlich stärker erweitert oder kombiniert werden. Bei der deutschen Bundestagswahl hat der/die WählerIn eine Erst- und eine Zweitstimme. Die einzelne Stimme passt zwar zur Kategorie *Multiple Choice*, jedoch braucht man nun zwei Beweise für die Wohlgeformtheit des Stimmzettels. Durch die fortschreitende Entwicklung von Quanten-Computern, stellt sich die Frage nach Argumenten und Beweisen, die auch Post-Quantum noch die gewünschten Sicherheitseigenschaften erfüllen. Da die Argumente auf der *discrete Logarithm Assumption* (Definition 2.3.2) basieren, würden diese nicht mehr halten und neue Beweise müssen entwickelt werden. Auch in diesem Bereich gibt es bereits Systeme, die auch Post-Quantum noch funktionieren können. Libert et al. entwickelten ein Beweissystem für *Range Proofs* [LLNW18].

Literaturverzeichnis

[Abe98] M. Abe. „Universally verifiable mix-net with verification work independent of the number of mix-servers“. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1998, S. 437–447 (zitiert auf S. 17).

[Adi08] B. Adida. „Helios: Web-based Open-Audit Voting.“ In: *USENIX security symposium*. Bd. 17. 2008, S. 335–348 (zitiert auf S. 17).

[BBB+18] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, G. Maxwell. „Bulletproofs: Short proofs for confidential transactions and more“. In: *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2018, S. 315–334 (zitiert auf S. 15, 31, 38).

[BCC+16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth, C. Petit. „Efficient zero-knowledge arguments for arithmetic circuits in the discrete log setting“. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2016, S. 327–357 (zitiert auf S. 32).

[BCCG16] J. Bootle, A. Cerulli, P. Chaidos, J. Groth. „Efficient zero-knowledge proof systems“. In: *Foundations of security analysis and design VIII*. Springer, 2016, S. 1–31 (zitiert auf S. 12).

[BFM19] M. Blum, P. Feldman, S. Micali. „Non-interactive zero-knowledge and its applications“. In: *Providing Sound Foundations for Cryptography: On the Work of Shafi Goldwasser and Silvio Micali*. 2019, S. 329–349 (zitiert auf S. 13).

[BFP+01] O. Baudron, P.-A. Fouque, D. Pointcheval, J. Stern, G. Poupard. „Practical multi-candidate election system“. In: *Proceedings of the twentieth annual ACM symposium on Principles of distributed computing*. 2001, S. 274–283 (zitiert auf S. 19).

[BM09] M. A. Based, S. F. Mjølsnes. „A non-interactive zero knowledge proof protocol in an internet voting scheme“. In: *Proceedings of the the 2nd Norwegian Security Conference (NISK 2009), Tapir Akademisk Forlag*. Citeseer. 2009, S. 148–160 (zitiert auf S. 19).

[Bou00] F. Boudot. „Efficient proofs that a committed number lies in an interval“. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, S. 431–444 (zitiert auf S. 27, 31).

[CC+08] J. Camenisch, R. Chaabouni et al. „Efficient protocols for set membership and range proofs“. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2008, S. 234–252 (zitiert auf S. 31).

[CDN01] R. Cramer, I. Damgård, J. B. Nielsen. „Multiparty computation from threshold homomorphic encryption“. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 2001, S. 280–300 (zitiert auf S. 12).

[CGH04] R. Canetti, O. Goldreich, S. Halevi. „The random oracle methodology, revisited“. In: *Journal of the ACM (JACM)* 51.4 (2004), S. 557–594 (zitiert auf S. 15).

[CGS97] R. Cramer, R. Gennaro, B. Schoenmakers. „A secure and optimally efficient multi-authority election scheme“. In: *European transactions on Telecommunications* 8.5 (1997), S. 481–490 (zitiert auf S. 19).

[CLS10] R. Chaabouni, H. Lipmaa, A. Shelat. „Additive combinatorics and discrete logarithm based range protocols“. In: *Australasian Conference on Information Security and Privacy*. Springer. 2010, S. 336–351 (zitiert auf S. 35).

[Coh13] H. Cohen. *A course in computational algebraic number theory*. Bd. 138. Springer Science & Business Media, 2013 (zitiert auf S. 28).

[DGS03] I. Damgård, J. Groth, G. Salomonsen. „The theory and implementation of an electronic voting system“. In: *Secure Electronic Voting*. Springer, 2003, S. 77–99 (zitiert auf S. 19).

[DJ01] I. Damgård, M. Jurik. „A generalisation, a simpli. cation and some applications of paillier's probabilistic public-key system“. In: *International workshop on public key cryptography*. Springer. 2001, S. 119–136 (zitiert auf S. 19).

[ElG85] T. ElGamal. „A public key cryptosystem and a signature scheme based on discrete logarithms“. In: *IEEE transactions on information theory* 31.4 (1985), S. 469–472 (zitiert auf S. 18, 19).

[FOO92] A. Fujioka, T. Okamoto, K. Ohta. „A practical secret voting scheme for large scale elections“. In: *International Workshop on the Theory and Application of Cryptographic Techniques*. Springer. 1992, S. 244–251 (zitiert auf S. 17).

[FS86] A. Fiat, A. Shamir. „How to prove yourself: Practical solutions to identification and signature problems“. In: *Conference on the theory and application of cryptographic techniques*. Springer. 1986, S. 186–194 (zitiert auf S. 15).

[GK93] O. Goldreich, E. Kushilevitz. „A perfect zero-knowledge proof system for a problem equivalent to the discrete logarithm“. In: *Journal of Cryptology* 6.2 (1993), S. 97–116 (zitiert auf S. 19).

[GMR89] S. Goldwasser, S. Micali, C. Rackoff. „The knowledge complexity of interactive proof systems“. In: *SIAM Journal on computing* 18.1 (1989), S. 186–208 (zitiert auf S. 11).

[GMW91] O. Goldreich, S. Micali, A. Wigderson. „Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems“. In: *Journal of the ACM (JACM)* 38.3 (1991), S. 690–728 (zitiert auf S. 12).

[GOS06] J. Groth, R. Ostrovsky, A. Sahai. „Perfect non-interactive zero knowledge for NP“. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2006, S. 339–358 (zitiert auf S. 12).

[Gro05] J. Groth. „Non-interactive zero-knowledge arguments for voting“. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2005, S. 467–482 (zitiert auf S. 19, 31, 38).

[Gro09] J. Groth. „Linear algebra with sub-linear zero-knowledge arguments“. In: *Annual International Cryptology Conference*. Springer. 2009, S. 192–208 (zitiert auf S. 12).

[Gro10] J. Groth. „A verifiable secret shuffle of homomorphic encryptions“. In: *Journal of Cryptology* 23.4 (2010), S. 546–579 (zitiert auf S. 25, 38).

[JJR02] M. Jakobsson, A. Juels, R. L. Rivest. „Making mix nets robust for electronic voting by randomized partial checking.“ In: *USENIX security symposium*. San Francisco, USA. 2002, S. 339–353 (zitiert auf S. 17).

[JSI96] M. Jakobsson, K. Sako, R. Impagliazzo. „Designated verifier proofs and their applications“. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1996, S. 143–154 (zitiert auf S. 12).

[KLM+20] R. Küsters, J. Liedtke, J. Müller, D. Rausch, A. Vogt. „Ordinos: a verifiable tally-hiding e-voting system“. In: *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE. 2020, S. 216–235 (zitiert auf S. 9, 17).

[LAN02] H. Lipmaa, N. Asokan, V. Niemi. „Secure Vickrey auctions without threshold trust“. In: *International Conference on Financial Cryptography*. Springer. 2002, S. 87–101 (zitiert auf S. 27).

[Lip03] H. Lipmaa. „On diophantine complexity and statistical zero-knowledge arguments“. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2003, S. 398–415 (zitiert auf S. 31).

[LK02] B. Lee, K. Kim. „Receipt-free electronic voting scheme with a tamper-resistant randomizer“. In: *International Conference on Information Security and Cryptology*. Springer. 2002, S. 389–406 (zitiert auf S. 17, 19).

[LLNW18] B. Libert, S. Ling, K. Nguyen, H. Wang. „Lattice-based zero-knowledge arguments for integer relations“. In: *Annual International Cryptology Conference*. Springer. 2018, S. 700–732 (zitiert auf S. 41).

[MKVK19] E. Morais, T. Koens, C. Van Wijk, A. Koren. „A survey on zero knowledge range proofs and applications“. In: *SN Applied Sciences* 1.8 (2019), S. 1–17 (zitiert auf S. 30).

[OMA+99] M. Ohkubo, F. Miura, M. Abe, A. Fujioka, T. Okamoto. „An improvement on a practical secret voting scheme“. In: *International Workshop on Information Security*. Springer. 1999, S. 225–234 (zitiert auf S. 17).

[Pai99] P. Paillier. „Public-key cryptosystems based on composite degree residuosity classes“. In: *International conference on the theory and applications of cryptographic techniques*. Springer. 1999, S. 223–238 (zitiert auf S. 18, 19).

[Ped91] T. P. Pedersen. „Non-interactive and information-theoretic secure verifiable secret sharing“. In: *Annual international cryptology conference*. Springer. 1991, S. 129–140 (zitiert auf S. 16, 31).

[PR18] S. Panja, B. K. Roy. „A secure end-to-end verifiable e-voting system using zero knowledge based blockchain.“ In: *IACR Cryptol. ePrint Arch.* 2018 (2018), S. 466 (zitiert auf S. 19).

[RS86] M. O. Rabin, J. O. Shallit. „Randomized algorithms in number theory“. In: *Communications on Pure and Applied Mathematics* 39.S1 (1986), S239–S256 (zitiert auf S. 27).

[SK94] K. Sako, J. Kilian. „Secure voting using partially compatible homomorphisms“. In: *Annual International Cryptology Conference*. Springer. 1994, S. 411–424 (zitiert auf S. 19).

[Smi05] W. D. Smith. „Cryptography meets voting“. In: *September* 10 (2005), S. 80 (zitiert auf S. 19).

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