Institut für Informationssicherheit

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Projekt-INF

Implementierung & Betrachtung der Offline Phase des Ordinos E-Voting Systems

André Sperrle & Jonas Geiselhart

Studiengang:	Informatik
Veranstaltung:	Bachelor Forschungsprojekt Informatik
Semester:	WiSe 2021/22

Prüfer/in:	Prof. Dr. Ralf Küsters
Betreuer/in:	Julian Liedtke, M.Sc.

Erklärung

Wir versicheren, diese Arbeit selbstständig verfasst zu haben. Wir haben 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. Wir haben diese Arbeit bisher weder teilweise noch vollständig veröffentlicht.

Ort, Datum, Unterschrift

Ort, Datum, Unterschrift

Inhaltsverzeichnis

1 Einführung

Das Ordinos Protokoll stellt ein sicheres E-Voting Protokoll dar. Es basiert auf dem Helios E-Voting Protokoll [1] und ergänzt dieses um ein tally-hiding Feature.
Tally-hiding bezeichnet hierbei die selektierte Ausgabe einzelner Teile der Wahl, im ursprünglichen Ordinos gestaffelt nach 3 Modi. Im Gegensatz zum full-tally voting können hier Anzahl der Stimmen pro Kandidat, bestimmte Ranking Positionen oder nur über-/unterschreiten von speziellen Schwellwerten verschlüsselt determiniert und dann entschlüsselt ausgegeben werden. Das kann benutzt werden um strategisches Wählerverhalten in einem iterativen Wahlvorgang zu verhindern, die Privacy von einzelnen Personen bei einem sehr kleinen Wählerkreis zu wahren, oder um Vorbehalte einzelner Kandidaten abzubauen.
Beispielsweise in Parteien könnte man eine solche tally-hiding Wahl durchführen, wenn man befürchtet, dass Gewinnende eine geringe Zustimmung erfahren werden und man verhindern will, dass diese direkt mit einer schwachen politischen Position, sowohl nach intern als auch nach außen, im Amt starten.
Ordinos ist strukturiert in eine Offline und eine Online Phase. Im folgenden wird sich unsere Nomenklatur an dem ursprünglichen Paper von *Küsters et al.* [9] orientieren.
Die Offline Phase beinhaltet die Konfiguration und das Setup der Wahl. Hier werden Trusted Channels zwischen den einzelnen Parteien geneneriert. Zusätzlich berechnet man hierbei idealerweise alle möglichen Parameter vor um die Wahlauswertung im Nachgang effizient zu gestalten.
Die Online Phase gliedert sich wiederum in mehrere Sektionen. Zuerst findet die Stimmabgabe der einzelnen Voter V_i bei einem Authentification Server AS statt. Dazu verwendet jeder Voter einen sogenannten Voter Supporting Device VSD_i und kann seine Stimmabgabe mittels eines Voter Verification Device VVD_i überprüfen. Als Ergebnis dieser Sektion entsteht eine Lister aller gültigen Stimmen b. Diese Liste wird vom AS auf einem *Bulletinboard* für alle einsehbar gepostet.
Im Anschluss kann jeder Voter überprüfen, dass seine Stimme gezählt wird. Dieser Sektion folgt die Tallying Phase, in der aus den gültigen auf dem Bulletin Board geposteten Stimmen das Ergebnis berechnet wird. Dies geschieht in zwei Schritten, zuerst werden die einzelnen Stimmen homomorph gegeben der Kandidatenoptionen aggregiert, dann wird eine verschlüsselte Berechnung des Tallys durchgeführt. Diese ist besonders zeitaufwändig und unterschiedlich je nach Modus. Es werden in [9] 3 verschiedene Modi zur Tallyberechnung beschrieben (immer $0 \leq n \leq n_{options}$):

1. Die ersten n Postionen der Wahl. In diesem Fall wird keine weitere Unterteilung der Gewinnenden oder Stimmen veröffentlicht.
2. Die ersten n Postionen der Wahl, jeweils untereinander geordnet nach Position im Ranking, also bspw. 1. Kandidat1 2. Kandidat2 ...
3. Jeweils Vorhergenannte Methode mit Anzahl der Stimmen pro Kandidat, für bestimmte Kandidaten

Anmerkung: Zusätzlich gibt es noch die Optionen, anstatt den ersten n Positionen, die letzten n Positionen auszugeben, einen Threshold-Test gegeben einer Stimmenanzahl n durchzuführen, oder eine Kombination der Modi durchzuführen.
Unser Ziel ist es den Offline Teil des Ordinos Protokolls in ein existierendes Projekt zu implementieren. Daher konzentriert sich unsere Ausarbeitung besonders auf die Algorithmen der Offline Phase. Die Online Stufe wird hier vor allem unter dem dem Gesichtspunkt der Effizienzsteigerung, primär durch Auslagerung einzelner Teilberechnungen in die Offline Phase, sowie Ersetzung einzelner Algorithmen durch andere mit geringerer Online Laufzeitkomplexität betrachtet.

2 Grundlegende Paradigmen

Dieser Abschnitt soll einen Einblick in die grundsätzlichen Techniken des Protokolls und der in der gegebenen Implementierung angewendete Techniken geben. Dazu betrachten wir zuerst in einer allgemeinen Weise den normalen Ablauf und gehen dann auf einzelne genutzte grundlegende Paradigmen zur Durchführung der Wahl und der Sicherung der Privacy des Wählenden an.

[1] https://vote.heliosvoting.org/

2.1 Grundlegende Struktur

Um eine bessere Übersicht über das Protokoll zu haben, wird die Durchführung des Protokoll in folgenden Phasen unterteilt. In der *Setup Phase* werden alle Wahlparameter vorbereitet und von einer Voting Authority auf einem Bulletin Board veröffentlicht. Diese Wahlparameter umfassen eine Liste der zugelassenen Wählenden, Beginn und Ende der Wahl, einen Wahlidentifier, ein korrektes Stimmformat, und eine gültige Schlüsselverbindung. Ein korrektes Stimmformat ist definiert durch Anzahl der Kandidaten und Stimmen pro Kandidat, sowie einer Enthaltungsoption. Diese Setup Phase umfasst dann weiter die Schlüsselgenerierung und das Erstellen von (ggfs. verschlüsselten) Kommunikationskanälen. Diese müssen jedoch nicht in der Vorbereitungsphase erfolgen, oftmals ist es allerdings sinnvoll diesen Prozess zumindest teilweise in der Setup Phase durchzuführen. Gleichwohl sollte es alle Teilnehmenden Entitäten (Voter, Voting support devices, Voter verification devices, Trustees, Authentification Server) zum Ende der Setup Phase möglich sein, sich gegenüber dem Bulletin Board unilateral zu identifizieren. Weitergehend sollte auch der Voter später die Möglichkeit haben sich sowohl gegenüber dem entsprechenden Voting support device/Voter verification device bilateral zu authentifizieren. Auch dem Voting support device sollte es möglich sein sich gegeüber dem Authenticated Server bilateral zu authentifizieren. Da diese Phase am wenigsten zeitkritisch ist werden wir im Folgenden versuchen so viele Berechnungen aus den anderen beiden Phasen in das Setup zu verlagern.
Danach folgt die *Voting Phase*, in der die Wähler sich für ihre Kandidaten entscheiden und ihre Stimmen verschlüsselt einreichen. In dieser Phase wird der Voter seine Stimme bei seinem zugewiesenen Voting Support Device (VSD) abgeben und dann mittels der Benaloh Challange oder einem vergleichbaren Protokoll die Korrektheit der von dem VSD errechneten Chiffre mittels des entsprechenden Voting verification Device überprüfen. Eine Stimme gilt als abgegeben wenn der Authenticated Server diesen erhalten und bestätigt hat. Bei dieser Benaloh Challange, kann jeder Wählende zuerst eine Stimme selektieren, das geschieht abhängig von der weitergehenden Verifikation beliebig oder nach eigentlichem Wahlinteresse. Der VSD verschlüsselt hierbei die Stimme und legt die verschlüsselte Stimme offen, nun kann der Wählende auswählen ob er den VSD prüfen möchte(Audit), in dem Fall werden alle Random-Tokens zur Berechnung des Ballots offengelegt, sodass der Wählende arithmetisch überprüfen kann, ob die richtige Eingabe verschlüsselt wurde, der Stimmvorgang wird nach der Überprüfung zurückgesetzt. Anderenfalls wird die Stimme (ohne Random-Tokens) an den Authentification Server versendet. [2] Das Resultat der Voting Phase ist eine Liste aller validen Stimmzettel, von denen Duplikate und invalide Stimmen entfernt wurden. Diese wird auf dem Bulletin Board veröffentlicht. Während des gesamten Prozesses können die Wählenden eine Beschwerde am Bulletin Board veröffentlichen, falls sie in ihrer Stimmabgabe Abweichungen von dem erwarteten Prozess registrieren.
Dieses Resultat wird in der *Tallying Phase* von den Trustees genommen, gelesen, verifiziert und in dem Fall, dass keine Unregelmäßigkeiten auftreten, die einzelnen Stimmen respektive der Kandidaten homomorph aggregiert und mittels Multi-party-computation ein Ergebnis berechnet. Diese Phase ist die rechenintensivste und erfordert Vorberechnungen. Dabei werden Trustees das Ergebnis in einem Ranking erstellen, bei der je nach spezifiziertem Modus unterschiedliche Berechnungen erforderlich werden. Jeder Trustee liefert entsprechend seinen Berchnungen nicht interaktive Zero Knowledge Proofs (NIZKP), die auf dem Bulletin Board veröffentlicht und von jedem verifiziert werden können. Das garantiert, das kein einzelner Trustee oder eine Kombination aus verschiedenen Trustees das Ergebnis abändern können.

2.2 Zero-Knowledge Proofs (ZKP)

Ein Zero Knowledge proof ist ein mathematisches Beweiskonzept, bei dem nur der Wahrheitsgehalt einer logischen Behauptung offengelegt wird. Ein Beweis wird typischerweise auf (mindestens teilweise) verschlüsselten Daten ausgeführt. ZKPs erfolgen in der Regel stochastisch, also beweisen eine Aussage nur zu einer bestimmten Wahrscheinlichkeit p. Dieser Beweis kann typischerweise wiederholt werden um die Wahrscheinlichkeit einen Fehler zu finden auf ein akzeptables Maß erhöhen. Generell ist ein ZKP über zwei Eigenschaften definiert, (1) *Completeness* Ein richtiger Beweis sollte mit einer hohen Wahrscheinlichkeit akzeptiert werden und (2) *Soundness* ein falscher Beweis sollte mit höchstens geringer Wahrscheinlichkeit akzeptiert werden. Ein ZKP basiert entweder auf *perfect zero-knowledge* oder *computational zero-knowledge*, je nach nötiger Sicherheitsstufe. Bei der Implementierung dieses Wahlverfahren ist letzteres ausreichend.

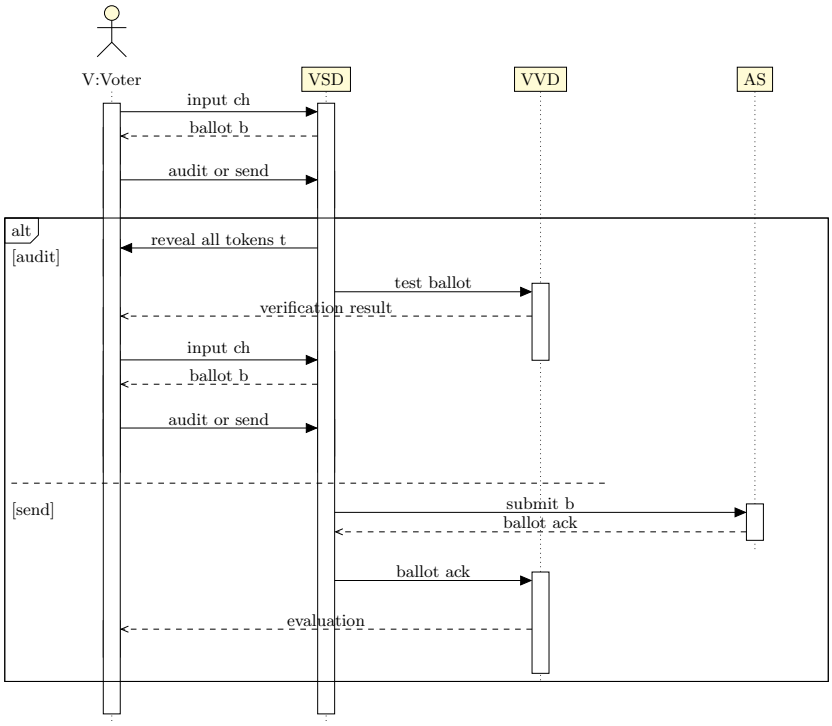


Abbildung 1: Erwarteter Ablauf einer Stimmabgabe

Der Beweis gliedert sich in der einfachsten Form in drei Phase *Commitment*, *Challange*, und *Response*. Ein Commitment zwingt den Sender/Prover sich auf einen geheimen Wert, der später den eigentlichen Wert maskieren soll festzulegen. Eine Challenge erfolgt von Seiten des Receiver/Verifier und ist der interaktive Teil des ZKP. Hierbei soll ein für den Prover zufälliger Wert sicherstellen, dass der Prover über ein besonderes Wissen verfügt (z.B. Random Tokens, Plaintext-knowledge, etc) und mit diesem arithmetische Aussagen über eine Chiffre treffen kann. Die Response geht wiederum von Prover aus und es kann nun vom Verifier mithilfe des Commitments und der Challange stochastisch verifiziert werden, dass eine bestimmte arithmetische Beziehung zwischen Chiffre und Response vorliegt, die eine logische Aussage über die Chiffre zulässt. Diese Aussage fällt je nach Beweis unterschiedlich aus, legt aber nicht mehr offen als den Wahrheitsgehalt der ursprünglichen Nachricht.
Non-interactive Zero Knowledge Proofs sind ein Spezialfall der normalen Zero Knowledge Beweise, bei denen kein aktiver Verifer benötigt wird um den Beweis durchzuführen. Hierbei erhalten Trusted Parties, hier Trustees, zu Beginn einen *common reference string*. Dieser wird dann zur Berechnung der Response und zur Verifikation genutzt. Dieser CRS ist ein Hashwert, definiert von einer festgelegten Eingabeform und pro Trustee unterschiedlich. In Ordinos werden verschiedene NIZKPs benutzt, u.A. zur korrekten Entschlüsselung, Kenntnis des diskreten Logarithmus, Wohlgeformtheit des DSA-Moduls, Klartextkenntis, Wertebereich einer Zahl, und korrekter Multiplikation. Ein nicht-interaktiver ZKP unterscheidet sich in der Berechnung der Challange, hier ist es nicht erforderlich einen Verifer anzufragen, somit kann von jedem sichergestellt werden, dass der Beweis korrekt durchgeführt wurde.

2.3 Multi-Party Computation (MPC)

Ein MPC Protokoll dient zur gemeinsamen verteilten Entschlüsselung von Werten. Dabei halten n Parteien jeweils ein Secrete Share und es müssen mindestens t Shares kombiniert werden um einen Wert offenzulegen. In der ursprünglichen Vorstellung des Ordinos System ist kein genauer MPC-Algorithmus gegeben. Die einzige Einschränkung ist, das die Chiffre IND-CPA-secure (unter Choosen-ciphertext-Attacken, nicht von einem zufälligen gleichlangen String unterscheidbar) sein muss. Im Folgenden erklären wir das in der Implementierung genutzte Protokoll.

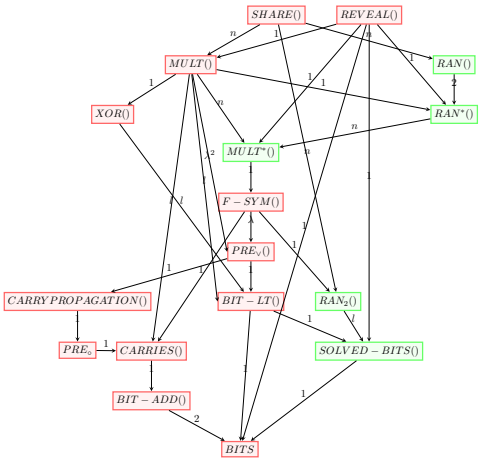
Threshold Pallier kann genutzt werden, da es nicht nur MPC unterstützt, sondern auch homomorph respektive der Addition ist. Dieses Kryptosystem nutzt die Shamir Secret Sharing Technik. In diesem Fall wird jeder verschlüsselte Wert nicht explizit, sondern implizit, beschrieben von einem Eingabewert und einem diskreten Polynom von Grad t, gespeichert. Hierbei stellt t das Threshold der nötigen Key-shares dar die zusammenkommen müssen um jeweils genügend Punkte des Polynoms zu erhalten, sodass das Polynom korrekt interpoliert wird und über den Eingabewert der verschlüsselte Wert ausgegeben werden kann. Es ist hierbei gesichert das nur falls mindestens t Key Shares vorhanden sind, eine Entschlüsselung stattfinden kann. In allen anderen Fällen, soll der Text wieder nicht von einem zufälligen String unterscheidbar sein.

2.4 Homomorphe Verschlüsselung

Ein homomorphes Verschlüsselungsprotokoll hat zusätzlich die Eigenschaft, das Ver- und Entschlüsselung $Enc(x)$ eines Wertes x bezüglich zweier Operationen $\circ$ und $*$ homomorph ist. Konkret bedeutet das $\forall x, y\ Enc(x \circ y) = Enc(x) * Enc(y)$. Das in der gegebenen Implementierung ist homomorph gegenüber der Addition, das ist notwendig, da dadurch die Aggregation der einzelnen Stimmvektoren respektive der Kandidaten effizient durchgeführt werden kann. Es gilt also $Dec_{sk}(Enk_{pk}(m_1) \cdot Enk_{pk}(m_2)) = m_1 + m_2$. Ein voll-homomorphes Kryptosystem würde es erlauben verschiedene andere Berechnungen potenziell deutlich effizienter zu implementieren wie wir später noch genauer ausführen, ist allerdings für unsere Fälle noch nicht zufriedenstellend implementiert.

3 Algorithmen zur Wertegenerierung in der Offlinephase

Die zeitkritische Berechnung des Tallys in der Onlinephase lässt sich in zwei Hauptteile gliedern. Zuerst die homomorphe Aggregation der Stimmen, was arithmetisch eine einfache Addition darstellt und somit auch von jeder Partei, egal ob an der verteilten Berechnung beteiligt oder nicht, einzeln und effizient durchgeführt werden kann. In [9] haben *Küsters et al.* festgestellt, dass diese Sektion der Berechnung einen für uns zu vernachlässigenden Teil darstellt. Deshalb liegt der Hauptfokus in unserer Implementierung und dieser Ausarbeitung in dem zweiten Hauptteil, der Berechnung des geheimen Rankings, je nach vorgegebener Konfiguration.
Die wohl wichtigste Vorberechnung ist die Berechnung der Offline Teile der Vergleichsprotokolle P^{Eq}_{MPC} und P^{gt}_{MPC}, dabei benötigen unterschiedliche Vergleiche von verschlüsselten Daten auch unterschiedliche Werte. Abbildung 2 stellt eine Übersicht aller Protokolle die genutzt werden können um Werte für diese Vergleichsprotokolle und andere Teile der Online Phase durch Vorberechnung zu beschleunigen (vgl. 4.3.4) dar. Dabei basiert die Betrachtung auf der Betrachtung in [1].
Zum einen enthält sie eine Übersicht, über die Beziehungen zwischen den Protokollen, welche Aufschluss geben soll, welche Teile besonders oft genutzt werden und wo potenzielle Optimierungen einen besonders



Alg.	Share	Reveal	Mult	Ran	Ran*
Time	1	n^2	n^2	n	n^2
Network	1	n	n	n	n
	Mult*	Xor	F-Sym	$\text{Pre}_\vee$	BIT-LT
Time	n^2	n^2	n^2	n^3	n^3
Network	n^2	n	n^2	n^3	n^3
	Ran_2	Solved Bits		Bit Add	Carries
Time	n^2	n^3		n^3	n^3
Network	n^2	n^3		n^3	n^3
	Pre_o	Carry Propagation		Bits	
Time	n^3	n^3		n^3	
Network	n^3	n^3		n^3	

Abbildung 2: Übersicht der Algorithmen

großen Effekt haben können. So ist zum Beispiel offensichtlich, das ein besser konditioniertes MULT Protokoll den größten Effekt haben könnte. Hierbei sind Algorithmen die von der Onlinephase aufgerufen werden grün markiert und Algorithmen die hier nur zur Unterstützung dienen sind hier rot markiert. Die nebenstehende Tabelle listet zudem die Zeitkomplexität in $\mathcal{O}$ zum Berechnen der einzelnen Algorithmen, unter üblichen Annahmen, also Addition/Subtraktion/Multiplikation in $\mathcal{O}(1)$, sowie die Netzwerk Interaktion , auch bezüglich $\mathcal{O}$, auf. Hierbei ist zu bemerken, dass es sich keineswegs um die Gesamtzahl der Nachrichten handelt, sondern jeweils nur der Nachrichtenein-/Ausgang (in unserem Fall i.A. pro $Trustee$) gewertet wird. Die einzelnen Protokolle enthalten sich hierbei wie in der Graphik gezeigt entweder konstant oft (1) oder sind von der spezifizerten Länge der Eingabe bzw. Ausgabe des darüberliegenden Protokolls abhängig. Hier beschreibt n einen linearen und $\lambda = n^2$ einen radialen Zusammenhang. Es können durch die Betrachteten Algorithmen folgende Werte betrachtet werden:

Share Teilen eines verschlüsselten Wertes zwischen verschiedenen Entitäten

Reveal Aufdecken/Entschlüsseln über MPC eines verteilt verschlüsselten Wertes.

MULT Verteilte Multiplikation auf zwei verschlüsselten Klartexten

RAN Ein zufälliger Wert
Das RAN-Protokoll liefert die Entschlüsselung einer gleich-verteilt, zufälligen Zahl aus dem Ring Z_N. Diese zufällige Zahl besteht aus die Summe aller r_j die von den Trustees $T_i, 1 \leq i \leq n$ zufällig gewählt, verschlüsselt, versendet, und aggregiert wurden.

RAN* Ein zufällig invertierbarer Wert und das Inverse aus dem Körper Z_N^*
Es werden zwei zufällige Werte gemeinsam berechnet und das Produkt auf Invertierbarkeit in Z_N getestet. Dies passiert so oft, bis eine zufällig invertierbare Zahl gefunden wird. Es ist also $a \cdot b = c$ mit $ggT(c, N) = 1$ und resultiert in den zwei Werten $(a, a^{-1}) = (a, b \cdot c^{-1})$

MULT* Präfixprodukte der Multiplikation von n Werten
Zur Berechnung der Präfixkomponente werden je Wert zwei verschlüsselte Werte, die invers zueinander sind, generiert. Diese maskieren dann die ursprünglichen einzelnen Produkte und die Berechnung kann im Klartextraum stattfinden. Durch Verschlüsselung und Kombination mit dem letztmöglichen Inversen kürzen sich die Zufallswerte wieder heraus und das verschlüsselte Präfixprodukt liegt vor. Das sichert einen theoretischen Effizienzgewinn gegenüber der naiven Implementierung mittels $MULT$ zu.

XOR Wendet XOR-Operator auf zwei verschlüsselten Bits an
Hier ist XOR als einfache arithmetische Operation implementiert. Diese Funktion ist nur für verschlüsselte Eingaben, die entweder zu eins oder null evaluieren definiert. In unserem weiteren Programmcode ist diese Bedingung sichergestellt.

F-SYM Verteilte Berechnung von symmetrischen boolschen Funktionen
Symetrische Funktionen werden definiert durch eine Liste von Stützstellen und Funktionswerten $(x_i, f(x_i))_{1 \leq i \leq n+1}$ für eine maximale Summe der Eingabewerte n. Demnach wird die Funktion ausgewertet indem man die Summe s über die Inputwerte s+1 bildet und verschlüsselt über die Gewichte der Lagrange Polynome auswertet. Hier ist zu beachten, dass die Gewichte in Z_N berechnet werden müssen, da die verschlüssselte Lagrange Interpolation nur in dieser Gruppe korrekt funktioniert.

PRE$_\vee$ Berechnet die Anwendung des logischen Oder-Operators auf aller Präfixe von n verschlüsselten Bits.
Die einzelne Oder-Operation wird durch eine symmetrischen Funktion realisiert. Durch geschickte Kombination und blockweiser Auswertung lässt sich das Präfix der einzelnen Werte in Linearzeit berechnen. Hierbei ist die Implementierung auf eine potenzielle Parallelisierung optimiert. Dennoch sollte man zu besserem Effizienzgewinn eine zusätzliche naive Implementierung in späteren Überlegungen nicht ausschließen.

PRE$_\wedge$ Berechnet die Anwendung des logischen Und-Operators auf alle Präfixe von n verschlüsselten Bits.
Wir berechnen das Resultat, indem wir die de-morgansche Gesetze auf Input und Output anwenden. Das ermöglicht uns $PRE_\vee$ zu nutzen. Der Mehraufwand ist vorerst vernachlässigbar, trozdem kann der Algorithmus an dieser Stelle noch optimiert werden.

$\mathbf{RAN}_2$ Generierung eins Zufallsbits
Jeder Akteur wählt ein Zufallsbit und beweist mittels $ZK - BOUND$, dass die Chiffre entweder 0 oder 1 verschlüsselt, es werden alle Chiffren addiert und mittels einer symmetrischen Funktion auf ein Bit abgebildet. Dabei ist das Resultat genau dann eins, falls die Aggregation zu einer geraden Zahl evaluiert und Null sonst.

BIT-LT Less-Then Vergleichsoperator auf zwei verschlüsselten Werten in Bitdarstellung.
Ergibt einen verschlüsselten Bit als Output, das zu eins evaluiert falls $x < y$ und zu null sonst. Dabei sind x, y bitweise verschlüsselte Eingabenzahlen.

SOLVED BITS Generiert einen zufälligen Wert aus $\mathcal{Z}_N$ welcher Bitweise verschlüsselt ist.
Der Wert wird hierbei zuerst bitweise zufällig generiert und dann in Basis 10 transformiert. Falls der verschlüsselte Wert größer N ist, wird das Protokoll erneut aufgerufen, deshalb sollte immer sichergestellt sein, dass die Anzahl der Bits des Schlüssels echt größer als die Anzahl der geforderten Bits ist.

BIT-ADD Summiert zwei Chiffretexte welche bitweise verschlüsselt sind auf
Dafür werden eine Reihe folgender Subprotokolle benutzt um jeweils ein 3-Tupel aus zwei Bits der korrespondierenden Stellwerte der Eingabe und ein Übertragsbit zu transformieren und ein stellwertiges Bitresultat sowie einen Übertrag zu erhaten

CARRIES Berechnet die verschlüsselten Carry Bits welche beim Summieren bitweise verschlüsselter Zahlen entstehen.
Dabei ist ein Carry bit, der bitweise Überlauf der Addition zweier Zahlen. Dieser Überlauf wird generell immer an die nächsthöhere Stelle durchpropagiert.

CARRY Propagation Berechnet aus mehreren SPK-Tupel, das an die nächste Stelle propagierte Tupel.
Nimmt ein Tupel aus (Bit)Listen des Formats (Set, Propagate, Kill) entgegen, bei denen jeweils genau einer der drei Werte gesetzt ist. Es resultiert in Set, falls eines der beiden Inputs setzt, es resultiert in Kill, falls einer der beiden Inputs "killt" und ansonsten resultiert die Eingabe in Propagate und der Übertrag wird auf die nächste Stelle durchpropagiert.

$\mathbf{PRE}_o$ Berechnet CARRY Propagation auf allen Präfixen eines Inputs.
Hierebei werden einzelne Blöcke, jeweils halb so groß wie das zu erwartende Ergebnis berechnet und geschickt miteinander kombiniert um die einzelnen Stellen zu ermitteln.

BITS Berechnet aus einem Ciphertext seinen bitweise verschlüsselten Pedant.
Dafür wird zuerst eine zufällige Bitzahl mittels Solved Bits generiert, die als Maske, sowohl für den bitweisen, als auch den dezimalen Wert fungiert. Nach Anwendung dieser Maske kann die Dezimaleingabe entschlüsselt, in Bits zerlegt, und wieder verschlüsselt werden. Nun kann die korrespondierende bitweise zufällige Zahl wieder abgezogen werden und wir erhalten das verschlüsselte Ergebnis.

Es ist noch zu notieren, dass $PRE_\vee$, $PRE_\wedge$ und Pre_o nicht naiv implementiert wurden sondern so umgeschrieben wurden, dass eine gute Parallelisierbarkeit möglich ist (vgl. [3] [4]). Die naive Implementation ist ohne Optimierungen deutlich schneller, doch hat weniger Raum für mögliche Verbesserungen. . Diese Auflistung dient hier nur zur Vollständigkeit eine genauere Betrachtung der einzelnen Protokolle und Werte findet sich in [1].
Da wir nun grundlegende Werte berechnen können, ist es uns möglich zwei Algorithmen zu definieren um die schon implementierten Vergleiche durchzuführen. Hierbei haben wir uns jeweils an den hammingbasierten Eq-/Gt-Tests spezifiziert von *Lipmaa und Toft* [10] orientiert.

3.1 Equality-Test Vorberechnungen

Nun können wir die konkreten Offlineberechnungen eines verschlüsselten Gleichheitsbeweises von zwei Zahlen wie in 1. Es werden hierbei zur Berechnung der verschlüsselten Hammingdistanz ein dezimal- sowie der equivalent bitverschlüsselte Wert generiert. Hierbei wird später die Differenz der Eingabewerte maskiert entschlüsselt, in Bits zerlegt und auf Gleichheit in der Hammingdistanz zu den zufälligen Bitwerten getestet. Zur effizienten verschlüsselten Auswertung der Hammingdistanz werden dann die Inversen sowie die Präfixe der Inversen benutzt.

Algorithm 1

Require: $bits$: *comparative values* $\leq 2^{bits}$

Ensure: $\exists n : 2^n = bits$

$r_{10}, r_2 \leftarrow SOLVED - BITS(bits)$

$r, r^{-1} \leftarrow RAN^*()$

$RandomPrefixes \leftarrow MULT^*((r)_{0\ldots(bits)})$

return $r_{10}, r_2, r, r^{-1}, RandomPrefixes$

3.2 Greater-Then-Test Vorberechnungen

Der Greater-Then Test erfordert auch Vorberechnung von Algorithmus 1, hier gekennzeichnet mit $EQ - DATA$ und einen rekursiven Aufruf mit $GT - DATA$, hierbei halbiert sich jeweils die Input Größe. Die Vorberechnung wird eine Liste ausgeben mit Werten aufsteigend von $2^2..2^{bits}$ mit den entsprechenden Daten für einen Greater-Vergleich und einen Equality-Vergleich. Der Algorithmus ist in 2 dargestellt.

4 Werteanforderungen der Onlinephase

Im folgenden Kapitel betrachten wir an welchen Stellen in dem Wahlprozess innerhalb der Onlinephase, besonders in der der Tallying-Phase, Werte benötigt werden, die man potenziell vorberechnen könnte, und die dadurch einen nicht vernachlässigbaren Zeitgewinn erbringen.
Hierbei ist besonders Kommunikation von verschiedenen Trustees maßgeblich. Wir werden zuerst betrachten welche Subprotokolle diese Werte benötigen und darauf aufbauend analysieren wir die verschieden Modi/Konfigurationen auf ihren Bedarf an einzelnen vorberechneten Werten.

4.1 Vorberechnungen der Subprotokolle

Während der Stimmabgabe ist das Optimierungspotenzial eher gering. Den größten Anteil der Nachrichten im Protokoll ist dezentral zwischen Voter V, VSD, und VVD. Alle andere Kommunikation entsteht

Algorithm 2

Require: $bits \leq key\ bits$

$\kappa \leftarrow \lfloor log_2(n) \rfloor - bits$ ▷ Statistical Security Parameter

$half \leftarrow \lfloor \frac{bits}{2} \rfloor$

if $half \neq 1$ **then**

  $e \leftarrow EQ - DATA(half)$

end if

$r \leftarrow (RAN_2)_{0..bits}$

$r_{bot} \leftarrow \sum_{i=0}^{half} r[i] * 2^i$

$r_{top} \leftarrow \sum_{i=0half}^{bits} r[i] * 2^{i-half}$

$s_2 \leftarrow SOLVED - BITS(\kappa)$ ▷ Here we had to adapt to MPC

$R \leftarrow s_2 * 2^{bits} + r_{top} * 2^{half} + r_{bot}$

if $half = 1$ **then**

  return $None, [R], [r_{bot}], [r_{top}]$

end if

$gt_{half} \leftarrow GT - Data(half)$

return $gt_{half}, [e, R, r_{bot}, r_{top}]$

zwischen AS und VSD, diese Nachrichten erfordern keine komplexen Berechnungen seitens des AS, die ausgelagert werden können.
Später in der Tallying-Phase wird die Liste der Wahlstimmen B vom AS überprüft, ggfs. bearbeitet, und veröffentlicht. Jeder einzelne Trustee T wird diese Liste nun lesen, verifizieren und die Stimmen homomorph auf die Kandidaten aggregieren, bis hier hin ist jede Berechnung nur von den Stimmen abhängig und hat wenig Optimierungspotenzial. Die darauffolgende Auswertung der Stimmen ist von der Konfiguration der Wahl abhängig und wird weiter unten getrennt betrachtet.

4.2 Konfigurationen der Wahl

Eine Konfiguration wird definiert durch den gegeben Evaluationsalgorithmus, dabei können verschiedene Versionen aufeinander aufbauen oder auf eine bestimmte Weise kombiniert werden. Es können so Wahlen von Kandidierenden, in einzelnen Bezirken, nach Parteien, oder andere unten beschriebene Kombinationen abgehalten werden.

4.2.1 Threshold Version

Die Standartvariante des Wahlsystems, es können aus der Stimmliste die n besten Positionen berechnet werden. Diese werden dann als Gewinner deklariert. In dieser Version wird eine Matrix berechnet in dem die einzelnen Kandidaten jeweils gegen alle anderen Kandidaten getestet werden und immer der Kandidat mit der größeren Anzahl von Stimmen den Eintrag 1, ansonsten den Eintrag 0 erhält. Diese verschlüsselte Matrix wird zu einem Vektor transformiert, indem man die Werte der einzelnen Einträge addiert. Nun erhält man eine Rangordnung der Kandidaten. Diese kann man gegen einen Threshold-Wert, den der Gewinner hat testen und dieses Ergebnis entschlüsseln. Hierbei haben die gewinnenden Parteien den Eintrag 1 und die Verlierenden 0.
Bei n Kandidaten, benötigt diese Variante $\frac{n\cdot(n-1)}{2}$ Greater-Then Tests und $\frac{n\cdot(n-1)}{2}$ zusätzliche Equality Tests. Diese ungerade Zahl lässt sich erklären, da für jeden Eintrag $m[i][j]$ gilt: $m[j][i] = eq(votes(i), votes(j)) - gt(votes(i), votes(j)) + 1$, wobei $eq(a, b)$ bzw. $gt(a, b)$ jeweils den Wahrheitswert der Aussage $a = b$ respektive $a < b$ verschlüsselt zurückgibt. Zudem gilt $m[i][i] = 0$ wodurch sich pro Kandidat eine Rechnung sparen lässt. Für den Vektor, der die Gewinnenden Parteien enthält gilt, je Kandidat wird ein Greater-Then Test gegenüber dem Threshold, also insgesamt n viele, benötigt. Es müssen also explizit insgesamt $\frac{n\cdot(n-1)}{2}$ Eq-Tests und $\frac{n\cdot(n-1)}{2} + n$ Gt-Tests durchgeführt werden.

4.2.2 Two Votes E-System

Ein Two Votes E-System modelliert eine Verhältniswahl. Dabei kann man mit einer Stimme eine Person oder mehrere Personen aus einer ggfs. kleineren Untermenge wählen und mit einer anderen das Verhältnis der gesamten Repräsentation der Gewählten mitbestimmen. Das Direktmandat wir in der Implementation als Subprotokoll realisiert. Als Resultat wir jeweils der Gewinner der Unterwahl offengelegt. Nun soll zusätzlich das Verhältnis der gewählten Parteien widergespiegelt werden, sodass kein Winner-Takes-All Prinzip realisiert ist, sondern auch Stimmen, die sich nicht in der Mehrheit befinden ein Gewicht haben. Dies wird bei wiederum als Unterprotokoll, der Seat Distribution Version, implementiert. Hier werden die bei dem Direktmandat Protokoll ermittelten Sitze erweitert und so aufgefüllt, sodass das Verhältnis der zweiten Stimme repräsentiert wird. Ausgegeben wird am Ende nur die endgültige Anzahl der Sitze die jeder Partei zufallen. Wir können aus den folgenden Abschätzungen schließen, dass besonders die Anzahl der Wahlbezirke (linear) und noch stärker die Anzahl der zur Wahl stehenden Kandidierenden und Parteien (quadratisch), die notwendigen Vorberechnungen negativ beeinflusst. Es ist festzuhalten, dass die Anzahl der vorbestimmten Sitze theoretisch einen kleineren Einfluss aufweist, was wichtig ist, da diese Kennzahl in Wahlen eine besonders hohe Spannweite hat.

Direct Mandate Version

Diese Version, baut auf der Standartversion auf. Hier ist das Ziel die Gewinnenden Parteien je nach Wahlkreis zu ermitteln. Demnach wird jeder Wahlbezirken mit den reduzierten Stimmen aus dem entsprechenden Kreis einzeln ausgewertet. Dies hat zur Folge, dass bei m Wahlkreisen die Threshold

Version m mal aufgerufen werden muss. Es müssen $m \cdot \frac{n \cdot (n-1)}{2}$ Eq-Tests explizit und $m \cdot (\frac{n \cdot (n-1)}{2} + n)$ Gt-Tests durchgeführt werden.

Seat Distribution Version/ Hare Niemeyer Distribution

Diese Version soll eine Art Parlamentswahl simulieren. Ziel ist es eine Sitzverteilung von q Sitzen und von n Parteien zu berechnen. Zuerst muss für jede Partei berechnet werden, ob die Anzahl der Stimmen über der Sperrklausel liegt, dafür sind n Gt-Vergleiche notwendig. Dann sind für alle Parteien zu berechnen wie viele Sitze notwendig sind um das Stimmverhältnis zu repräsentieren, dafür sind $n \cdot (q+1)$ GT-Vergleiche notwendig. Dannach müssen in einem Ausgleichsverfahren die Ausgleichssitze berechnet werden, dafür kann das Winner-Election Protokoll genutzt werden. Dieses nimmt hier $\frac{n \cdot (n-1)}{2}$ Eq-Tests und $\frac{n \cdot (n-1)}{2} + n$ Gt-Tests in Anspruch.
Damit liegt die gesamte Anzahl an zu Vorberechnenden Daten bei $\frac{n \cdot (n-1)}{2}$ Gleichheitstests und $n + n \cdot (q+1) + \frac{n \cdot (n-1)}{2} + n$ Greater-Vergleichen.

4.2.3 Instant Runoff Voting

Instant Runoff Voting wird hier betrachtet wie definiert in [8]. Dabei gibt jeder Wählende eine Stimme ab, die die persönlichen Präferenzliste darstellt. Es wird jeweils der höchste Eintrag, aus den Kandidaten die noch nicht eliminiert sind, als Zwischenstimme gezählt und in jedem Durchgang der Kandidat mit dem niedrigsten Ergebnis eliminiert. Es ist einfach zu sehen, dass Anzahl der Greater-Vergleichen in $\theta(n^2)$ liegt und noch zusätzlich $\theta(n^2)$ Equality-Vergleiche hinzukommen um zu ermitteln ob der letzten Kandidierende eindeutig ist, oder ob mehrere Parteien dasselbe schlechteste Ergebnis haben.

4.2.4 Condorcet Konfiguration

Ein Condorcet Wahlsystem ist hierbei allgemeiner definiert wie die vorherigen, es gibt mehrere Möglichkeiten ein solches System umzusetzen. Wir betrachten hier wieder die Möglichkeiten nach [8]. Das gemeinsame Ziel einer Condorcet Wahlmethoden ist es den Gewinner zu finden der die meisten Kandidierenden im paarweisen Vergleich schlägt. Dabei gibt ein Wähler ein volles/eindeutiges Ranking der Kandidierenden an, welche aggregiert und danach ausgewertet werden. Im Allgemeinen liegt hier die Anzahl der Vergleiche wieder in $\theta(n^2)$. Jedoch kann der Aufwand je nach implementierter Evaluationsmethode deutlich höher ausfallen.

4.2.5 Borda Wahlen

Eine Borda Wahl zeichnet sich dadurch aus, dass die Stimmen zu Evaluation eine Anzahl an Punkten pro Kandidat enthalten, diese kann entweder direkt abgegeben werden, oder aus Positionierungen von Kandidaten zueinander berechnet werden. Je nach Evaluation können die Ordinos Standartfunktionen wie sie schon in [9] vorgestellt wurden, diese benötigen in der Regel auch maximal $\theta(n^2)$ Vergleiche pro Test.
Die Borda Variante ist trotzdem interessant, da die Anzahl der gegebenen Punkte unter Umständen sehr hoch ausfallen kann, insbesondere wenn man auch gebrochene Zahlen darstellen will und dementsprechend die Punktzahlen um einen Faktor skalieren will. Daher betrachten wir die Generierung von vorberechneten Werten für höherstellige Bitlängen der zu vergleichenden Zahlen (wie in Abschnitt 5.1 gezeigt).

4.3 Mathematisch-kryptographische Primitive zur Ermittlung des Wahlergebnisses

Zur Auswertung der Wahl werden verschiedene mathematische Primitive genutzt, für uns beschränkt sich die Betrachtung auf drei relevante Operationen. Einerseits sind Equality und Greater-Then Tests wie von *Lipmaa und Toft* [10] relevant, andererseits verwenden wir einen $NIZKP$ im Subprotokoll RAN_2 zum Beweis, dass jeder Trustee einen Bit verschlüsselt hat und nicht höherwertige Zahlen. Dieser Beweis ist nötig, da sonst die symmetrische Funktionsauswertung Werte ausgibt die entschlüsselt zu Werten ungleich Null oder Eins evaluieren. Damit wird die Bitmaske zur bitweisen Entschlüsselung invalide und im u.U. die Ergebnisse der verschlüsselten Vergleichsoperatoren falsch.

4.3.1 Hammingbasierte Equality Test

Im hammingbasierten Equality Test wird zuerst der zu testende Wert maskiert und offengelegt. Dannach wird die Hammingdistanz zwischen diesem offengelegten in Bits zerlegten und wieder verschlüsselten Wert und der Maske berechnet. Dafür benötigen wir in der Onlinephase einen zufälligen verschlüsselten Wert in Dezimal- und Binärdarstellung. Die verschlüsselte Berechnung der Hammingdistanz h_d kann dank der Homomorphieeigenschaften mittels Daten aus der Offlinephase lokal und effizient berechnet werden. Dieser errechnete Wert $h_d + 1$ wird nun wieder mit einer neuen Maske versehen und offengelegt, die Maske ist notwendig um zu sichern, dass falls die Gleichheit nicht gilt, der Wert nicht bekanntgegeben wird. Dieser Maskierte Wert wird jetzt potenziert und mit der korrespondierenden Potenz der Inversen der Maske multipliziert. Die sich ergebenden Werte können nun wieder als symmetrische Funktion durch Lagrange Interpolation ausgewertet werden, es ist eine Bitposition genau dann Eins, falls die Gleichheit der Bits gegeben ist. Es sind alle Bitpositionen addiert und interpoliert genau dann Eins, falls alle Bitpositionen aggregiert zu Eins evaluieren, ansonsten wird alles zu Null interpoliert. Dieser errechnete Wert wird verschlüsselt zurückgegeben. Für den Grenzfall der Eingabe von einer Ein Bit Zahl (entweder 1 oder 0) lässt sich der Wert lokal mittels Nutzen der homomorphen Eigenschaften realisieren, da der binäre Wert in diesem Fall gleich dem dezimalen Wert ist.

4.3.2 Greater-Then Tests

Die zweite wichtige Testart sind Greater-Then Tests. Sie sind rekursiv implementiert und bauen auf einem Gleichheitstest eines bestimmten Bitbereichs der Zahl auf.
Ziel der rekursiven Phase ist es das Most-Significant-Bit zu ermitteln. Dafür errechnen wir als Maske einen verschlüsselten Wert, für den nicht nur die Chiffre, sondern auch separat verschlüsselt die niederwertigsten $\lfloor \frac{l}{2} \rfloor$ sowie die nächsten $\lfloor \frac{l}{2} \rfloor$ Bits als Dezimalwerte vorliegen. Zur statistischen Sicherheit wird dieser Wert weitergehend noch mit zufälligen Bits aufgefüllt, da sonst bestimmte Stellen der Zahl unmaskiert entschlüsselt werden würden.
Nun wird eine Eingabe y von Eingabe x subtrahiert und maskiert. Es wird die höherwertigere Hälfte der Bits auf Gleichheit mit dem höherwertigeren Teil der Maske getestet, falls eine Gleichheit besteht, wissen wir, dass das MSB in der anderen Hälfte ist, ansonsten ist es in dieser Hälfte. Nun kann dieser Algorithmus rekursiv mit der Hälfte des MSB m und der entsprechenden Hälfte der Maske r_H aufgerufen werden. Genau dann wenn $x - y$ kleiner als null war, ist nun m größer als r_H, also wird das Ergebnis des Aufrufs propagiert. Es ist wichtig zu beachten, dass für einen begrenzten Bitbereich x größer als y sein muss, hierbei kann um Korrektheit sicherzustellen 2^{bits} zu $x - y$ addiert werden.

4.3.3 DIE basierte Tests

In Disclose-If-Equal Protokollen, wie auch von *Lipmma und Toft* [10] beschrieben, erhalten wir das Ergebnis den richtigen Wert, falls wir eine Zahl verschlüsseln, die äquivalent zu unserer zu testenden Zahl ist, anderenfalls erhalten wir keine Aussage über den Wert. So kann man zwei Werte vergleichen und erhält am entweder eine Gleichheitsaussage oder einen zufällig gewählten Verschlüsselungstext. Es ist also wieder ein Nulltest bezüglich der Differenz der Werte. Dieser Algorithmus ist für die verteilt verschlüsselte Version linear in Anzahl der Parteien. Gegeben einem verschlüsseltem Wert $[x]$ und zwei von allen Parteien zufällig gewählten und verschlüsselten Werten r, β können die Trustees gemeinsam $y = (x_0 \cdot -1 \cdot [x])^{[r]} \cdot [\beta]$ berechnen. Dieser Wert kann entschlüsselt werden und evaluiert falls nun $x_0 = x$ gilt zu $y = r \cdot (x_0 - x) + \beta = \beta$ hier könnten die Trustees β aufdecken und auf Gleichheit prüfen.
Dieser Test ist schwer zu implementieren, da die Berechnung der Potenz eines verschlüsselten Wertes zu einem anderen verschlüsselten Wert einige kompliziertere Vorberechnungen erfordert und es in der verteilten Version dieses Algorithmus notwendig ist, das der Exponent geheim ist. Zudem sehen wir, als Vorgriff, in unserer Evaluation, dass die Equality Test Protokolle wenig Optimierungspotenzial zur Gesamtlaufzeit beitragen. Daher betrachten wir ihn nicht weiter. Dennoch ist dieser Test interessant, falls in der Zukunft ein Test benötigt wird, der auch sehr große Bitanzahlen testen soll, da der Test keinen Mehraufwand in der berechnung für größere Bitwerte benötigt.

Algorithm 3 1 out of 2 Proof

Require: n - public key, $[c]$ - test cipher, id - deterministic, unique per party,
p_0, p_1 - Allowed Plaintexts, v_{pal} - randomness used in encryption of cipher

$[g_0] \leftarrow Encrypt(p_0)$ ▷ No randomness
$[g_1] \leftarrow Encrypt(p_1)$
$[g_0]^{-1} \leftarrow invert([g_0], n^2)$
$[g_1]^{-1} \leftarrow invert([g_1], n^2)$
$[u_0] = [g_0]^{-1} \cdot [c] \ mod \ n^2$
$[u_1] = [g_1]^{-1} \cdot [c] \ mod \ n^2$
$v_{rand} \leftarrow Random(0...n-1) \times 3$
if $u_1 \neq v_{pal}^n \ mod \ n^2$ **then**
  $Swap([u_0], [u_1])$
end if
$a_0 = v_{rand}[0]^n \ mod \ n^2$
$a_1 \leftarrow honestM(n, u_1, bits, v_{rand})$ ▷ Simulates honest coversation
$s \leftarrow hash(n \oplus u_1 \oplus u_2 \oplus a_1 \oplus a_2 \oplus id)$ ▷ for our case $p_i = i$
$e_1 = v_{rand}[2] \qquad e_0 = (s - e_1) \ mod \ 2^k$
$z_1 = v_{rand}[1] \qquad z_0 = [r_0 \cdot v_{pal}^{e_0}] \ mod \ n^2$
return $a_0, e_0, z_0, a_1, e_2, z_2$

4.3.4 ZK Bounds

Bei der Erzeugung von zufällig verschlüsselten Bitwerten RAN_2 muss jede beteiligte Partei einen Wert, entweder $Null$ oder $Eins$, verschlüsseln. Die nachfolgende Kombination der Werte, sodass daraus genau ein zufälliges Bit entsteht erfordert aus Sicherheitsgründen einen Beweis, dass die verschlüsselte Zahl keinem anderen Wert entspricht. Dieser Beweis wird von jeder Partei als NIZKP erbracht und von allen anderen Parteien überprüft. Uns standen mehrere Beweise zur Verfügung, wir haben uns für denselben Beweis, wie er auch an anderen Stellen benutzt wird, dabei sehen wir hier vor allem den Vorteil, dass man zur selben Funktionalität denselben Beweis nur einmal liefern muss und damit die Implementierung insgesamt verständlicher wird.

Algorithm 4 honestM - Simulate honest verifier conversation

Require: n, u, v_{rand} - wie im Beweis spezifiziert.

$e = v_{rand}[2]$
$z = v_{rand}[1]$
$a = (z^n \ mod \ n^2 \cdot u^{-e}) \ mod \ n^2$
return a

ZK Bounds - 1 out of 2 Proof

Wir geben hier nur eine Intuition warum dieser Beweis funktioniert und was die einzelnen Schritte bedeuten. Eine ausführliche Erklärung finden sie von *Damgård und Jurik* [6].
Ziel des Beweises ist es einen Verifier davon zu überzeugen, dass der Chiffretext c entweder $p_0 = 0$ oder $p_1 = 1$ entschlüsselt. *Damgård und Jurik* argumentieren, dass dies gleichzusetzen ist mit dem Fakt, dass $c \cdot (g)^p$ eine Potenz von n^2 ist, wobei g hier ein verschlüsselter erlaubter Klartext. Dieser einfache Beweis wird in Algorithmus 3 dargestellt. Hier wird dieser Teil des ZKP allerdings modifiziert um Interaktion mit dem Verifier zu vermeiden, wird z als Commitment und e als Challange simuliert.
Nun wird dieses Protokoll erweitert um sicherzustellen, dass genau einer der zwei Werte verschlüsselt ist. Diese Erweiterung bietet keine allgemeine Zero-Knowledge sondern nur honest-verifier Zero-knowledge, das ist jedoch für nicht interaktive ZKP genug. Im folgenden ist $u = \frac{c}{g}$ und es wird bewiesen, dass dieser Text eine Potenz von n^2 ist. Es ist hier wieder a die Commitments, e die Challanges, und z die Responses. Dabei werden beide eine Challange e_1 zufällig gewählt und eine andere e_0 durch einen Hash erstellt, durch $s = e_0 + e_1$ sind beide eindeutig bestimmt. Falls nun g_0 oder g_1 dem Ciphertext entspricht, entspricht entweder u_0 oder u_1 der verwendeten Zufallsvariable, da genau dann $1 = g^{-1} \cdot c$ gilt. Diesen Ciphertext verwenden wir weiter und führen den vohrig beschriebenen Algorithmus durch, für den anderen Ciphertext simulieren wir Commitment, Challange, und Response Ergebnisse durch Algorithmus . Demnach kann nun der Verifer beide Protokollabläufe testen und wird für beide Korrektheit feststellen. Er erhält aber keine Information ob u_0 oder u_1 den Chiphertext verschlüsselt, da er dafür die zufällige Verschlüsselungsvariable kennen müsste. Auch kann er sicher sein, dass sowohl e_0 also auch e_1 richtig gewählt sind, da sonst $s \neq e_0 + e_1$ gelten müsste.

Algorithm 5 Verify Proof

Require: n - public key, $[c]$ - test cipher, id - same as previous, proof() - the input from the proofer

$a_0, e_0, z_0, a_1, e_1, z_1 \leftarrow proof()$ ▷ Assert bounds as given, if not compute modulo
$[g_0] \leftarrow Encrypt(p_0)$ ▷ As previously no randomness
$[g_1] \leftarrow Encrypt(p_1)$
$[g_0]^{-1} \leftarrow invert([g_0], n^2)$
$[g_1]^{-1} \leftarrow invert([g_1], n^2)$
$[u_0] = [g_0]^{-1} \cdot [c] \ mod \ n^2$
$[u_1] = [g_1]^{-1} \cdot [c] \ mod \ n^2$
$z_0^n = (z_0)^n \ mod \ N^2$
$z_1^n = (z_1)^n \ mod \ N^2$
$z_1^{test} = (a_0 \cdot u_0^{e_0}) mod N^2$
$z_2^{test} = (a_1 \cdot u_1^{e_1}) mod N^2$
$s \leftarrow hash(n \oplus u_0 \oplus u_1 \oplus a_0 \oplus a_1 \oplus id)$ ▷ for our case $p_i = i$
if $s \neq e_0 + e_1$ **or** $z_0^n \neq z_0^{test}$ **or** $z_1^n \neq z_1^{test}$ **then**
  Reject
end if
Accept

5 Implementierung

Wir haben unsere Implementierung in mehrere sematische Abschnitte untergliedert und diese nacheinander implementiert.

Als erstes haben wir die Protokolle, wie in [1] beschrieben implementiert, wir fassen alle Protokolle als Suite mit Basisklasse Preprotokoll. Diese Protokollsuite hilft uns zwei Funktionen zur Brechnung eines vorberechneten GT-Datensatzes nach Algorithmus 2 und eines vorberechneten EQ-Datensatzes nach Algorithmus 1 auszurechnen. Hierbei berechnen wir bei rekurisven Aufrufen alle unterliegenden Daten mit. Diese Designentscheidung ermöglicht es uns semantisch die Algorithmen von *Lipmaa et Toft*[10] näher abzubilden, inhaltlich wäre es genauso valide einfach jede rekursive Stufe einzeln auszurechnen und abzuspeichern.
Zu der Protokollsuite ist anzumerken, dass die implementierten Protokolle die Funktionalität der Precompute Protokolls übersteigen, wir haben uns hierbei strikt nach [1] gerichtet und konnten später bei der Implementierung der verteilten Version der Algorithmen aus [10] implementierte Subprotokolle ersetzen, diese Ersetzungen machen nicht nur das Programm verständlicher sondern bieten eine Basis für Optimierungen, dabei nutzen wir den Fakt, dass keine allgemeinen Zahlen, sondern nur Zahlen aus $\mathcal{Z}_{2^m}$ benötigt werden. Des weiteren ermöglicht diese Implementierung später mehr Optimierungen. Noch sind die Protokolle rein sequenziell implementiert, allerdings vorgeschlagene Änderungen in Vorbereitung auf spätere Parallelisierungen bereits implementiert.

Den zweiten semantischen Abschnitt stellt eine Klasse mit verwaltenden Funktionen dar. Hier wird insbesondere die Speicherung, Aufgabenverteilung, und Funktionen zum späteren Aufzeichnen der Funktionen umgesetzt. Es werden also die vorherig berechneten Daten genommen und gespeichert, hierbei können mehrere Trustees auf einem Computer in unterschiedlichen Prozessen gestartet werden, solange sie sich in der Nummer der Schlüssel-Shares unterscheiden. Von hier an verwenden wir den Teil des Schlüssels, der zur Vorberechnung benutzt wird als Diskriminante zwischen einzelnen Prozessen.
Des weiteren bietet die Klasse auch statische Funktionen zum Durchführen einer Dummy-Wahl die die Anzahl der Parameter zurücksetzt und misst, sowie Attribute für die Vorberechnung setzt. Wir haben auch Funktionen für die lokale parallele Berechnung der einzelnen Datensätze hinterlegt, diese funktionieren nicht für die verteilte Berechnung.

Um nun eine Wahl vorzubereiten, implementieren wir zwei Methoden um die Berechnung einzustellen. Dafür nimmt einmal der Authorithy Server die Wahlkonfiguration entgegen und benutzt diese um lokal eine Wahl zu simulieren, wobei er Dummy Werte benutzt und Speicherzugriffe aufgezeichnet werden. Es ist wichtig zu erwähnen, dass hierbei schon eine Wahl erstellt sein muss, da sonst die eigentlichen Trustee- und Bulletinboards nicht initialisiert werden. Hier kann die Vorbereitung gestartet werden.

Weitere Schritte die wir lokal implementiert oder vorbereitet haben sind zu Teil im nächsten Abschnitt unter Optimierungen aufgelistet.

5.1 Evaluation

Die Messungen wurden auf den selben Computern wie [9] durchgeführt und sind dementsprechend kompatibel. Unser Setup besteht aus zwei PCs wobei einer die Authentification Instanz und den Bulletinboard-Webserver und ein zweiter den Trustee-Webserver, auf dem die Trustees operieren, darstellt. Alle sind über ein lokales Netzwerk verbunden.
Wir nutzen zwei Trustees zur Berechnung mit einem Threshold von zwei Trustees für unsere Berechnungen. Es ist klar zu sehen, dass die komplette Berechungszeit von den GT-Tests dominiert wird (Wir können sagen, dass die EQ-Zeit unter 1024 Bits bei meist deutlich unter 4 Minuten liegt und ansonsten weniger als 2% der GT-Zeit benötigt). Weiterhin ist die Schlüssellänge auch zeitbestimmend, es ist deutlich zu sehen, dass in der direkten Berechnung, das $SOLVED - BITS$ zur Berechnung eines zufälligen Wertes aus $\mathcal{Z}_m$ mit $m = keylength - bits - 1$ die meiste Zeit einnimmt. Hierzu ist zu erwähnen, dass eine Parallelisierung der $RAN2$-Aufrufe wie in Abschnitt 6 beschrieben lokal ein deutlich besseres Ergebnis erzielt. Auch eine parallele Berechnung der Top-Protokolle zeigt in der lokalen Version keine signifikante Verlängerung in der einzelnen Berechnungen. Da der Overhead in der verteilten Version hauptsächlich durch die längere Dauer der verteilten Operationen die ein Austausch über das Netzwerk erfordern entsteht, ist davon auszugehen,

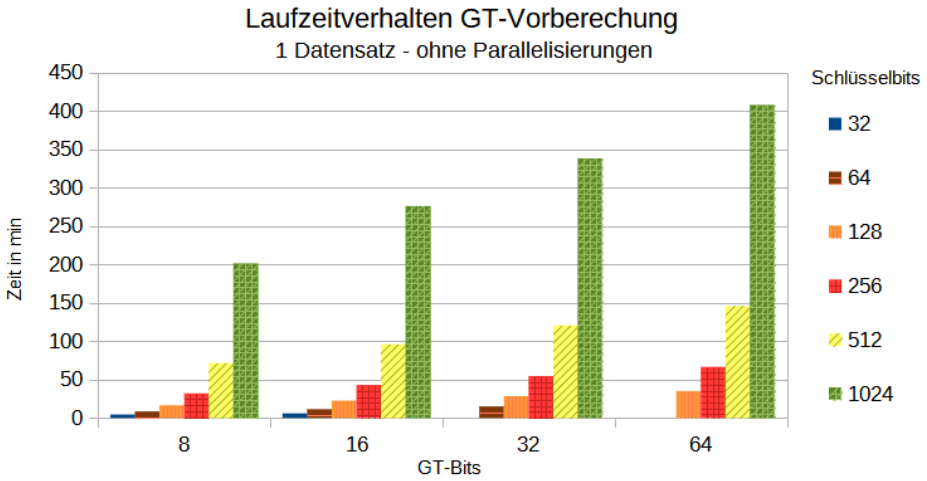


Abbildung 3: Auswertung

dass eine entsprechende Implementierung hier auch mindestens genauso signifikaten tendentiell eher sogar besseren Speedup erzeugt. Wir sehen, dass die Berechnungszeit wenig von der Anzahl der Bits abhängt. Hier ist nicht davon auszugehen, dass mehr als 2^{64} aggregierte Punkte nötig werden, dennoch würden selbst höhere Anzahlen gut berechenbar sein, da die zusätzliche Berechenzeit logarithmisch wächst, mit der Anzahl der zusätzlichen Bits, da jede Rekursionsebene durch die Berechnung der GT-Werte unabhängig von der Bitanzahl dominiert wird und somit nur die Gesamtanzahl der Rekursionswerte ausschlaggebend ist.
Komplizierter ist die Erhöhung der Schlüsselwerte. Eine Erhöhung hat ein effektiv lineares Wachstum in der Berechnungszeit zur Folge. Wie schon vorher erwähnt besteht hier großes Parallelisierungspotenzial, was wir im folgenden Abschnitt genauer betrachten werden. Allgemein sind die Werte gut genug um einzelne Berechnungen in der Offline-Phase, also nicht zeitkritisch betrachtet, durchzuführen, jedoch werden weitere Parallelisierungen nötig um eine allgemein akzeptable Zeit umzusetzen.

6 Optimierungen

Wir schlagen eine Reihe von Optimierungen vor. Einen Teil davon haben wir bereits implementiert, andere betrachten wir nur theoretisch. Wir verknüpfen die Optimierungen nicht mit End-to-end Sicherheitsbeweisen, sondern nur Intuitionen warum diese Beweise so sicher sind. Einige Optimierungen werden schon in [1] und [10] erwähnt.
Folgende Optimierungen wurden bereits implementiert:

- Wir implementieren einen Early-Break in Solved-Bits. Die Spezifikation der Bitlänge ermöglicht uns Daten zurückzugeben ohne auf einen Overflow in $\mathcal{Z}_n$ zu Testen, da, falls die vorgegebene Anzahl der Bits kleiner ist als die Anzahl der Bits des Schlüssels (beginnend mit der ersten Eins), der durch die Kombination der einzelnen Bits erreichbare Höchstwert kleiner als n ist.

- Wir berechnen die einzelnen Greater-/Equality Sets parallel. Dafür rechnen wir zuerst aus wie viele einzelne Berechnungen nötig sind. Wir haben uns entschieden die Anzahl der insgesamten parallel laufenden Berechnungen auf ein einstellbares Maximus zu limitieren, da diese Parallelität unterschiedliche Verbindungen nutzt. Damit verhindern wir, dass die Verbindungsschnittstelle ausgelastet ist und einen Bottleneck darstellt.
  Aus Zeitgründen haben wir diese Implementierung nur für eine lokale Berechnung implementiert, eine Implementierung für Berechnung mittels Kommunikation über ein verteiltes Setup ist allerdings nicht viel schwerer.

- Wir haben zwei ausgewählte Protokollaufrufe parallelisiert.
  Einmal der Aufruf von $RAN2$ in Solved-Bits, dieser läuft komplett parallel ab und erzeugt eine spezifizierte Anzahl an verschlüsselten Zufallsbits. Wir erhoffen uns davon insbesondere die Berechnung des Statistischen Sicherheitsparamters κ nach [10] zu beschleunigen, einzelne vorhergegangene Tests zeigen, dass die Schlüssellänge schnell zum Limitierenden Faktor wird, da man bei jeder Aufdeckung der verschlüsselten Zahl alle Bits maskieren muss und das mit einer großen Menge an Maskenbits einhergeht.
  Als zweiten Aufruf haben wir wieder $RAN2$, diesmal direkt im Protokoll zur Generierung der Greater-Then Daten, parallelisiert. Das soll die benötigte Zeit unabhängiger von der erwartenden Größe der zu testenden Zahlen machen. Diese Zeit ist nicht maßgebend für die gesamte Berechnungszeit, macht das ganze Protokoll allerdings im allgemeinen skalierbarer zur Verwendung höherer Zahlen.
  Weiter Parallelisierungen haben wir nicht durchgeführt, da wir befürchten, dass der zusätzliche Arbeitsaufwand zum verteilen der Nachrichten nicht dem Zeitgewinn durch das Parallele Ausführen gerecht wird. Eine genauere Untersuchung dieser Hypothese ist noch ausstehend.
  Auch diese Optimierung haben wir aus Zeitgründen nur lokal implementiert.

Wir halten die Betrachtung folgender Optimierungen sinnvoll:

- Wir halten die Implementierung von DISCLOSE-IF-EQUAL(DIE) Equality Tests für sinnvoll. Eine Untersuchung könnte insbesondere die Hypothese, dass DIE Tests, besonders für höhere Anzahl an Bitzahlen, einen praktischen Effizienzgewinn einbringen in den Fokus nehmen. Das Protokoll benötigt andere, weniger komplizierte, allerdings einschränkendere vorberechnete Daten, insbesondere die Potenzierung einer Verschlüsselten Zahl zu einer anderen Chiffre, ist ein lösbares, aber nicht-triviales Problem.

- Eine Verbesserung wäre die Einführung eines effizienten Kryptosystems, dass homomorph nicht nur respektive der Addition, sondern auch der Multiplikation ist. Diese Optimierung ist offensichtlich und nur zur Vollständigkeit erwähnt.

- Es ist zu untersuchen wie viel weiteres Optimierungspotenzial in der Berechnung einer verschlüsselt zufällig gleichverteilten Chiffre mit Wert in $\mathcal{Z}_{2^k}$ mit beliebig positiven k liegt. Aktuell wird jedes Bit einzeln Berechnet und am Ende alle einzelnen Bits zusammengerechnet, dabei ist die Erzeugung dieser randomisierten Bits aufwändig, da jede Partei genau einen Wert beisteuert, den Beweis liefern aus Algorithmus 3 liefert, und dann ein gemeinsames Bit mittels Auswertung einer symmetrischen Funktion bestimmt wird. Dieser Aufwand dominiert einen Großteil der Berechnungszeit beider Protokolle.

7 Zusammenfassung

In dieser Ausarbeitung haben wir grundlegende Paradigmen für die Offlinephase von Ordinos betrachtet, unsere Implementierung vorgestellt, und alle dafür wichtigen Algorithmen, Protokolle, Techniken vorgestellt. Wir haben eine kurze Evaluierung unserer Arbeit beigefügt und ausführlich durchgeführte sowie zukünftige Optimierungen vorgestellt. Unsere Implementierung umfasst hier nicht nur für uns relevante, sondern auch zusätzliche, für potenziell zukünftige Tests relevante Algorithmen. Es ist uns gelungen unseren Projektcode so in das gegebene Projekt einzufügen, dass es möglich ist mit nur eine Konfigurationsdatei die Vorberechnung zu starten, damit war das Projekt erfolgreich.
Mögliche Defizite zeichnen sich besonders in der Parallelisierung ab, hier ist es uns nicht mehr gelungen unsere implementierten Optimierungen sicher auf die über das Netzwerk verteilte Version zu integrieren. Das kann in einer zukünftigen Arbeit in schon vorbereitet allgemeinere Mechanismen zur Parallelisierung des Codes zusammengefasst werden.

Literatur

[1] Christina Bauer. Untersuchung der durchführung der offline phase von mpc protokollen. Master's thesis, Universität Stuttgart, oct 2020. URL http://dx.doi.org/10.18419/opus-10699.

[2] Josh Benaloh. Ballot casting assurance via voter-initiated poll station auditing. 08 2007.

[3] Ashok K. Chandra, Steven Fortune, and Richard Lipton. Lower bounds for constant depth circuits for prefix problems. In Josep Diaz, editor, *Automata, Languages and Programming*, pages 109–117, Berlin, Heidelberg, 1983. Springer Berlin Heidelberg. ISBN 978-3-540-40038-7.

[4] Ashok K. Chandra, Steven Fortune, and Richard Lipton. Unbounded fan-in circuits and associative functions. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing*, STOC '83, page 52–60, New York, NY, USA, 1983. Association for Computing Machinery. ISBN 0897910990. doi: 10.1145/800061.808732. URL https://doi.org/10.1145/800061.808732.

[5] Ronald Cramer, Rosario Gennaro, and Berry Schoenmakers. A secure and optimally efficient multi-authority election scheme. *European transactions on Telecommunications*, 8(5):481–490, 1997.

[6] Ivan Damgård and Mads Jurik. A generalisation, a simplification and some applications of paillier's probabilistic public-key system. In *LNCS*, pages 119–136. Springer-Verlag, 2001.

[7] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient rsa key generation and threshold paillier in the two-party setting. *Journal of Cryptology*, 32:265–323, 2019. doi: 10.1007/s00145-017-9275-7.

[8] Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Kuesters, Julian Liedtke, and Daniel Rausch. Extending the tally-hiding ordinos system: Implementations for borda, hare-niemeyer, condorcet, and instant-runoff voting. Cryptology ePrint Archive, Report 2021/1420, 2021. https://ia.cr/2021/1420.

[9] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding Remote E-Voting System. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 216–235. IEEE, 2020. URL https://publ.sec.uni-stuttgart.de/kuestersliedtkemuellerrauschvogt-eurosp-2020.pdf.

[10] Helger Lipmaa and Tomas Toft. Secure equality and greater-than tests with sublinear online complexity. In Fedor V. Fomin, Rūsiņš Freivalds, Marta Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming*, pages 645–656, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-39212-2.