Karlsruhe Institute of Technology

Kryptographische Protokolle mit einfachen Hardwarebausteinen als Vertrauensanker

mit unterschiedlichen Vertrauensmodellen am Beispiel von kryptographischen Wahlverfahren, sicheren Bezahlsystemen und Softwareschutzverfahren

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
des Karlsruher Instituts für Technologie (KIT)

genehmigte

Dissertation

von

Bernhard Löwe

aus Heilbronn

Tag der mündlichen Prüfung:	12.01.2018
Erster Gutachter:	Prof. Dr. Jörn Müller-Quade
Zweiter Gutachter:	Prof. Dr. Bernhard Beckert

Ich versichere wahrheitsgemäß, die Dissertation bis auf die dort angegebenen Hilfen selbständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer und eigenen Veröffentlichungen unverändert oder mit Änderungen entnommen wurde.

Karlsruhe, den 12. November 2018 Bernhard Löwe

In dieser Arbeit werden kryptographische Konzepte beschrieben, die den Rahmen der bisherigen Möglichkeiten erweitern. Eine wesentliche Rolle in diesen Konzepten haben die Nutzerinnen und Nutzer. Nur Konzepte, die bequem und einfach einsetzbar sind, werden sich durchsetzen können und Anwenderinnen und Anwendern den Fortschritt bieten, den Forscherinnen und Forscher erreichen wollen. Dennoch stehen Nutzerinnen und Nutzer nicht im Fokus der Beschreibungen. Daher und der besseren Lesbarkeit wegen verwenden wir in diesem Dokument das generische Maskulinum. Damit sind alle *Nutzer* gleichermaßen gemeint, ganz gleich welcher Herkunft, welcher Nationalität, welchen Geschlechts (m, w, x) oder welches andere für die Anwendung unbedeutende Merkmal sie haben. Dies gilt selbstverständlich auch für jede Rolle, die ein Nutzer einnehmen kann: Wähler, Administrator, Hersteller usw.

Zusammenfassung

Kryptographische Protokolle sind ein essentieller Bestandteil unseres Alltags geworden. Sie finden Verwendung bei nahezu jedem Webseitenaufruf, beim Telefonieren mit dem Mobiltelefon oder beim bargeldlosen Bezahlen. Von der korrekten Ausführung der Protokolle hängt unsere Privatsphäre, der Schutz privater und geschäftlicher Daten, der Schutz unseres Vermögens ab.

Die Protokolle weisen in der Ausführung im Allgemeinen eine Komplexität auf, die vom Menschen nicht in ausreichender Geschwindigkeit beherrscht wird. Daher bedient sich der Nutzer in der Regel einer Hardware, die die Ausführung für ihn vornimmt. Dazu muss er sich auf die korrekte Funktionsweise der Hardware verlassen. Wir halten daher zwei Dinge in diesem Kontext für erstrebenswert: Einerseits sollte der Nutzer auf eine Art und Weise in die Ausführung des Protokolls einbezogen werden, die für ihn einfach zu leisten ist und anhand der er sich von der korrekten Ausführung überzeugen kann. Andererseits sollte der kritische Teil der Ausführung von einem möglichst geringen Teil im Protokoll abhängen, der dann idealerweise durch eine Hardware des Nutzers ausgeführt wird, der der Nutzer vertrauen kann.

Sind beide Bedingungen erfüllt, ist es einem Angreifer, der andere Teile als die vertrauenswürdige Hardware korrumpiert hat, nicht mehr möglich, den Ablauf des Protokolls – mit Ausnahme eines erzwungenen Abbruchs – zu stören. Wir zeigen dies in dieser Arbeit anhand von drei Beispielen.

Im ersten Szenario betrachten wir das Geldabheben am Automaten mit einer EC-Karte (*Girocard*). Das bisher eingesetzte EMV-Verfahren nimmt hierzu den ganzen Automaten und die Kommunikation zur Bank als sicher an. Uns ist gelungen, die Annahmen deutlich zu reduzieren: In unserem Protokoll muss die Steuerung des Automaten, die bisher auf herkömmlichen PCs – meist mit einem Betriebssystem der Firma Microsoft – aufbauen, nicht mehr als vertrauenswürdig angenommen werden; die Sicherheit des Protokollablaufs beruht nunmehr auf der Vertrauenswürdigkeit der Girocard und der physischen Sicherheit der Geldkassette.

Bereits in diesem Szenario stellt sich die Frage, wann eine Hardware als vertrauenswürdig angenommen werden kann. Wir beleuchten in jedem Szenario, das wir betrachten, auf welchen Annahmen das Vertrauen der Nutzer beruhen kann/muss. Im zweiten Szenario – Kopierschutz für Softwareprodukte – setzen wir eine technisch sehr aufwändige Lösung als vertrauenswürdige Hardware ein, bei der wir dennoch davon ausgehen, dass Kunden ihr vertrauen können. Die Hardware ermöglicht effiziente Protokolle, die die Interessen des Softwareherstellers außerhalb seiner direkten Kontrolle durchsetzen. Der

Softwarehersteller vertraut dazu dem Hersteller des Kopierschutzes, dass die Leistungsfähigkeit seines Produktes den Erwartungen entspricht, ohne die genaue Funktionsweise der vertrauenswürdigen Hardware zu kennen.

Anders stellt sich das bei kryptographischen Wahlverfahren dar. Der Wähler hat im Allgemeinen keine freie Wahl, welche Technik bei seiner Stimmabgabe zum Einsatz kommt. Dennoch ist es seine freie Wahlentscheidung, die geschützt werden soll. Daher muss für den Wähler nachvollziehbar sein, was die Hardware leisten muss und ob die eingesetzte dies auch tut. Wir stellen in dieser Arbeit ein Verfahren vor, in dem ein einfacher Hardwarebaustein, der aus einfachen elektronischen Bausteinen aufgebaut ist, als vertrauenswürdige Hardware fungiert. Durch den Einsatz dieser Hardware und der Erweiterung der Protokolle kann der durch Software gesteuerte Teil der Wahlmaschine passiv korrumpiert sein, ohne dass der Wähler erpressbar wird. Einfache vertrauenswürdige Hardwarebausteine ermöglichen dem Wähler mittels einer einfachen Aktion, seinen Wunsch zum Ausdruck zu bringen, ohne dass dieser einem Angreifer bekannt wird oder er diesen ändern, löschen oder überschreiben kann.

Die Auswertung der gesammelten Daten kann im Falle von (kryptographischen) Wahlverfahren öffentlich verifizierbar geschehen. Dabei ist entscheidend, dass die Berechnung keine Informationen über das Wählerverhalten preisgibt, die über das Auszählungsergebnis hinausgehen. Das ist uns sowohl bei der Präsenzwahl gelungen als auch bei einem Verfahren, das sich der Aussortierung überschriebener Stimmen widmet: Hat ein Wähler die Möglichkeit, seine zuvor abgegebene Stimme durch eine erneute Stimmabgabe zu überschreiben, darf – sofern das als Ausweichstrategie gegen Erpressungen verwendet wird – die Berechnung, welche Stimmen nicht überschrieben wurden, das Wählerverhalten nicht preisgeben.

Inhaltsverzeichnis

1. Vorwort

1.1. Motivation

Die Kryptographie befasste sich ursprünglich ausschließlich mit der Kunst, Nachrichten zu verschlüsseln. Anwendung fanden Verschlüsselungsverfahren anfangs hauptsächlich in der militärischen Kommunikation, um die Vertraulichkeit der Nachrichten zu sichern. Inzwischen wird jedoch nicht nur die Kryptoanalyse – die Analyse der Verschlüsselungsverfahren, um sie zu brechen – zum Bereich der Kryptographie hinzugerechnet; die Kryptographie beschäftigt sich auch mit anderen Bausteinen als der Verschlüsselung. Mit Signaturen und Commitments sollen nur zwei Vertreter genannt sein. Signaturen helfen beispielsweise, die Integrität und die Authentizität von Nachrichten zu schützen.

Diese Bausteine werden jedoch nur selten alleinstehend verwendet. Schon für die *sichere* Kommunikation ist die Vertraulichkeit der Nachrichten nicht ausreichend; erst durch die durch Signaturverfahren geschaffene Integrität und Authentizität wird eine Kommunikation ermöglicht, die wir als sicher bezeichnen. Verfügbarkeit, Vollständigkeit oder Laufzeiten müssen je nach Anwendung selbstverständlich auch berücksichtigt werden. Durch eine geschickt gewählte Sequenz von auszutauschenden Nachrichten – einem *Protokoll* – und dem Einsatz eines oder mehrerer kryptographischer Bausteine können Ziele erreicht werden, die auf den ersten Blick unmöglich erscheinen.

Die Sicherheit der kryptographischen Bausteine stützt sich im Allgemeinen auf mathematisch gut untersuchte Probleme: die Schwierigkeit, die Primfaktoren großer Zahlen zu bestimmen, oder die Decisional-Diffie-Hellman-Annahme sind zwei Beispiele. Die Sicherheit der daraus konstruierten Protokolle lässt sich dann ebenfalls auf diese Annahmen reduzieren. Zur Lösung einiger Probleme lassen sich effizientere Protokolle entwerfen, wenn zusätzliche Annahmen hinzugenommen werden.

In dieser Arbeit widmen wir uns daher Protokollen, die zusätzlich zu den mathematischen Annahmen Hardwarebausteine als weiteren Vertrauensanker verwenden. Diese Hardwarebausteine verfügen im Allgemeinen über einen sehr begrenzten Funktionsumfang; dadurch lassen sie sich einfacher *sicher* realisieren. Je nach Anwendungsfall ist es von großer Bedeutung, dass die Umsetzung nachvollziehbar ist. Anhand von drei Beispielen, in denen dieses Konzept bereits in unterschiedlichen Konfigurationen eingesetzt wird, betrachten wir, welche Eigenschaften die Hardware erfüllen müsste, um das notwendige Vertrauen erzeugen zu können, und welche dieser Annahmen nicht erfüllt werden können. Aufgrund der Erkenntnisse können wir alle drei Protokolle neu- oder weiterentwickeln und so das Vertrauen der Nutzer in die Systeme steigern.

1.2. Überblick über diese Arbeit

Bevor wir uns in Kapitel 2 den Grundlagen widmen, die für die später vorgestellten Konzepte notwendig sind, geben wir noch einen Überblick über die in dieser Arbeit behandelten Themen.

- In Kapitel 3 diskutieren wir am Beispiel *Geldautomaten* unseren ersten Anwendungsfall, in dem Hardware bereits als Vertrauensanker eingesetzt wird. Das bestehende Protokoll legt einige Annahmen zugrunde, die, wie sich in der Vergangenheit gezeigt hat, nicht gehalten werden können. Wir skizzieren ein Protokoll, das ohne diese Annahmen auskommt und dennoch eine höhere Sicherheit erzielen kann.
- In Kapitel 4 betrachten wir das zweite Anwendungsszenario: Kopierschutz für Software. Hier ist der Kontext, in dem wir sichere Hardware einsetzen, grundlegend anders. Das spiegelt sich in der Art der Hardware wider. Uns ist es gelungen, das erste Kopierschutzkonzept zu entwickeln, dessen Methoden offen diskutiert werden können, ohne dass es einem Angreifer durch diese Informationen möglich ist, den Schutz effizient zu brechen.
- Kapitel 5 befasst sich mit kryptographischen Wahlverfahren und stellt den Kern dieser Arbeit dar. Wir widmen uns darin einerseits der Fragestellung, ob es möglich ist, einen Wahlcomputer zu konzipieren, der keine Information über den Wählerwunsch erhält, dem Wähler aber dennoch einen Beweis für die korrekte Aufzeichnung seiner Stimme und die korrekte Auszählung geben kann. Andererseits betrachten wir das Problem der Aussortierung überschriebener Stimmzettel in einem Onlinewahlverfahren, ohne dass eine an dem Prozess beteiligte Partei Informationen über das Wählerverhalten extrahieren kann.
  - In Kapitel 5.3 beschreiben wir eine Weiterentwicklung des 2007 veröffentlichten Bingo-Voting-Wahlverfahrens. Der Fokus der Weiterentwicklung lag einerseits auf der Geheimhaltung der Füllstimmen und andererseits auf der Geheimhaltung des Wählerwunsches vor Komponenten der Maschine, die über Rechenleistung oder Speicher verfügen.
  - In Kapitel 5.4 beschreiben wir, wie die Mitglieder der Wahlleitung einer Onlinewahl gemeinsam überschriebene Stimmen beweisbar korrekt aussortieren können, ohne dass sie dabei Informationen erhalten, mit deren Hilfe ein Wähler erpresst werden kann.
- In Kapitel 6 fassen wir die Ergebnisse dieser Arbeit zusammen und geben einen Ausblick auf zukünftige Arbeiten.

2. Grundlagen

Die in dieser Arbeit vorgestellten Lösungen, Protokolle, Vorschläge und Skizzen bauen auf vielerlei Bausteinen auf. Wir geben in diesem Abschnitt einen Überblick über die wichtigsten davon. Der Überblick erhebt keinen Anspruch auf Vollständigkeit. Er soll lediglich kurze Beschreibungen beinhalten, um einen Eindruck der Eigenschaften der Bausteine zu vermitteln.

Die Details der Implementierung der Bausteine spielen in den vorgestellten Ideen nur in seltenen Fällen eine Rolle. Muss die Implementierung eines Bausteins für die Realisierung einer Idee eine weitergehende Eigenschaft erfüllen, gehen wir im Kontext des speziellen Lösungsvorschlags näher darauf ein.

2.1. Verschlüsselung

Verschlüsselungsverfahren bestehen im Allgemeinen aus den drei Algorithmen `KeyGen`, `ENC` und `DEC`. `KeyGen` erzeugt – je nach Art der Verschlüsselung – einen Schlüssel oder ein Schlüsselpaar. $\texttt{ENC}_{key}(m, r)$ und $\texttt{ENC}_{key}(m)$ verschlüsseln eine Nachricht m mit einem Schlüssel key und einem optionalen oder optional angegebenen Zufallswert r. $\texttt{DEC}_{key}(m)$ entschlüsselt eine Nachricht m unter Zuhilfenahme des Schlüssels key. Vereinfacht schreiben wir stattdessen auch $\texttt{ENC}(m)$ oder $\texttt{DEC}(m)$.

Bei Verschlüsselungsverfahren wird zwischen symmetrischen und asymmetrischen Verfahren unterschieden. Bei symmetrischen Verfahren kann der gleiche Schlüssel zum Ver- und Entschlüsseln verwendet werden. Bei asymmetrischen Verfahren besteht der Schlüssel aus zwei Teilen: einem öffentlichen Teil, der zur Verschlüsselung verwendet wird, und einem geheimen Schlüssel, mit dem das Chiffrat wieder entschlüsselt werden kann.

2.1.1. Symmetrische Verschlüsselung

Symmetrische Verschlüsselungsverfahren können weiter in Strom- und Blockchiffren unterteilt werden. In dieser Arbeit spielen Stromchiffren jedoch keine Rolle. Blockchiffren werden wir als Black-Box-Baustein verwenden – beispielsweise in Kapitel 4 über das Kopierschutzverfahren *Blurry-Box*. Ein Beispiel für eine Blockchiffre ist der Advanced Encryption Standard (AES) [43]. Wir nehmen stets die Verwendung eines geeigneten Betriebsmodus [49] an.

2.1.2. Asymmetrische Verschlüsselung

Asymmetrische Verschlüsselungsverfahren unterscheiden sich von symmetrischen insbesondere dadurch, dass jeder verschlüsseln können soll, aber nur der Inhaber des geheimen Schlüssels entschlüsseln kann. Der Algorithmus `KeyGen` erzeugt ein Schlüsselpaar – bestehend aus dem öffentlichen Schlüssel pk und dem geheimen Schlüssel sk. Mit dem öffentlichen Schlüssel kann ver-, aber nicht entschlüsselt werden. Der geheime Schlüssel lässt sich aus dem öffentlichen Schlüssel nicht (effizient) ableiten.

Im Folgenden stellen wir einige Eigenschaften vor, die ein asymmetrisches Verschlüsselungsverfahren haben kann, sowie einige Verfahren, die für diese Arbeit relevant sind.

Homomorphie

Asymmetrische Verfahren besitzen oft eine homomorphe Eigenschaft: Eine Verknüpfung zweier Chiffrate führt zu einer (gewünschten) Verknüpfung der beiden Klartexte. So ist Elgamal [50] beispielsweise multiplikativ homomorph und Paillier [94] additiv homomorph: Vereinfacht lässt sich sagen, dass eine Multiplikation zweier Elgamal-Chiffrate ein Chiffrat ergibt, das das Produkt der beiden Klartexte enthält ($\texttt{ENC}(a) \cdot \texttt{ENC}(b) = \texttt{ENC}(a \cdot b)$). Ähnliches gilt für Paillier-Chiffrate ($\texttt{ENC}(a) \cdot \texttt{ENC}(b) = \texttt{ENC}(a + b)$).

Wir nutzen diese Eigenschaft – die auch manche Commitment-Verfahren besitzen – in Kapitel 5.3 (Oblivious Bingo Voting).

Homomorphe Verschlüsselungsverfahren können den Sicherheitsbegriff IND-CCA-2 offensichtlich nicht mehr erfüllen. Diese *Schwäche* der Verfahren wird jedoch dort, wo sie eingesetzt werden, meistens gezielt genutzt.

Threshold-Verfahren

Bei einem k-aus-n-Treshold-Verfahren (oder auch Schwellenwertverfahren) wird der geheime Schlüssel auf n Parteien verteilt, so dass eine einzelne Partei Chiffrate nicht entschlüsseln kann. Nur wenn sich mindestens k Parteien zusammentun, sind sie in der Lage, Chiffrate zu entschlüsseln. Mit dem Wissen von $k - 1$ Parteien können keine Rückschlüsse auf den geheimen Schlüssel oder den Klartext von Chiffraten gezogen werden.

Im Allgemeinen führen die k Parteien eine Teilentschlüsselung des Chiffrates durch, aus denen dann der Klartext berechnet werden kann. Aus den Teilentschlüsselungen sind weder die Teilschlüssel der Parteien noch der geheime Schlüssel ableitbar. Auf diese Art wird vermieden, dass zur Entschlüsselung die Geheimnisse zusammengeführt oder der geheime Schlüssel berechnet werden müssen. Das Geheimnis des Entschlüsselungsschlüssels bleibt weiterhin auf die Parteien verteilt.

Paillier-Kryptosystem

Pascal Paillier hat 1999 ein asymmetrisches Verschlüsselungsverfahren [93] vorgestellt, das additiv homomorph ist. Es beruht auf der *Decisional-Composite-Residuosity*-Annahme. Wir nutzen es in einer von Ronald Cramer, Ivan Damgård, Jesper Buus Nielsen und Mads Jurik angepassten Variante [42, 44] im Kontext von Oblivious Bingo Voting in Kapitel 5.3.

Elgamal-Verschlüsselung

Taher Elgamal entwarf in den achtziger Jahren ein asymmetrisches, multiplikativ homomorphes Verschlüsselungsverfahren [50, 51]. Es ist unter der Annahme, dass das *Decisional-Diffie-Hellman-Problem* (DDH) in der zugrunde liegenden Gruppe schwierig zu lösen ist, IND-CPA-sicher.

Der öffentliche Parameter g ist Erzeuger einer zyklischen Gruppe G mit Ordnung p. Für die Berechnung des öffentlichen Schlüssels wählt der Empfänger eine zufällige Zahl $x < p$ als geheimen Schlüssel, für den $\texttt{ggT}(x,\, p) = 1$ gilt. Der öffentliche Schlüssel berechnet sich als $\mathrm{pk} := g^x$.

Ein Chiffrat auf den Klartext m mit dem Verschlüsselungszufall r besteht aus dem Tupel $(a,\, b) := (g^r,\, m \cdot \mathrm{pk}^r)$. Zur Entschlüsselung berechnet der Schlüsselinhaber $a^{-x} \cdot b = g^{-rx} \cdot m \cdot g^{xr} = m$.

Modifizierte Elgamal-Verschlüsselung (m-Elgamal)

M-Elgamal ist eine modifizierte Variante des oben dargestellten Elgamal-Verschlüsselungssystems. Juels, Catalano und Jakobsson nutzen diese Variante für den Beweis ihres Schemas [73].

Gegeben sei eine multiplikative zyklische Gruppe G mit Prim-Ordnung p, in der das DDH-Problem schwierig zu lösen ist. Weiterhin seien g_1 und g_2 zwei Erzeuger der Gruppe G. Der geheime Schlüssel $(x_1,\, x_2)$ wird zufällig, gleichverteilt aus $\mathbb{Z}_p$ gezogen. Der öffentliche Schlüssel berechnet sich dann als $y := g_1^{x_1} g_2^{x_2}$.

Für die Verschlüsselung einer Nachricht m wähle ein zufälliges $r \in \mathbb{Z}_p$ und berechne das Chiffrat als $\hat{c} = \texttt{ENC}\,(m, y) := (g_1^r,\, g_2^r,\, y^r m)$. Für die Entschlüsselung eines Chiffrats $\hat{c} = (A, B, C)$ mit dem geheimen Schlüssel (x_1, x_2) berechne $m = A^{-x_1} B^{-x_2} C$.

2.1.3. Sicherheitsbegriffe

Die Sicherheitseigenschaften von Bausteinen, die wir verwenden, sind essentiell für die in dieser Arbeit vorgestellten Konstruktionen. Wir gehen hauptsächlich in den Sicherheitsbeweisen auf die in unseren Protokollen notwendigen Eigenschaften ein. Im Allgemeinen fordern wir von Verschlüsselungsverfahren IND-CPA-, IND-CCA-1- oder IND-CCA-2-Sicherheit. Jonathan Katz and Yehuda Lindell beschreiben diese Begriffe in ihrem Buch *Introduction to modern cryptography* [77] ausführlich.

IND-CPA-Sicherheit

Der Sicherheitsbegriff IND-CPA (*Security under Chosen-Plaintext Attacks*) für symmetrische Verschlüsselungsverfahren ermöglicht es einem Angreifer, ein Orakel mehrfach nach Verschlüsselungen adaptiv gewählter Klartexte zu fragen. Der Angreifer darf im Experiment $\mathrm{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$ zwei von ihm gewählten Nachrichten nur mit vernachlässigbarer Wahrscheinlichkeit mit höherer Wahrscheinlichkeit als $1/2$ ihren Chiffraten zuordnen können. Das Experiment gibt 1 aus, wenn dem Angreifer die Zuordnung gelungen ist, sonst den Wert 0.

Ein symmetrisches Verschlüsselungsverfahren $\Pi = (\mathtt{Gen}, \mathtt{Enc}, \mathtt{Dec})$ ist IND-CPA-sicher, wenn für alle probabilistischen Polynomialzeit-Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion negl existiert, für die gilt:

$$\Pr[\mathrm{PrivK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

IND-CCA-1-Sicherheit

In der Sicherheitsdefinition IND-CCA-1 (*Security Against Chosen-Ciphertext Attacks*) für (a)symmetrische Verschlüsselungsverfahren steht dem Angreifer im Sicherheitsexperiment $\mathrm{PrivK}^{\mathsf{cca\text{-}1}}_{\mathcal{A},\Pi}$ ein (Ver- und) Entschlüsselungsorakel zur Verfügung. Er kann sich, bevor er zwei Klartexte für die Challenge wählt, beliebige Chiffrate entschlüsseln lassen. Nachdem er das Chiffrat einer der Klartexte erhalten hat, steht ihm das Orakel nicht mehr zur Verfügung. Das Experiment gibt 1 aus, wenn der Angreifer erfolgreich bestimmen konnte, welchen der beiden Klartexte das Chiffrat im Experiment für die Challenge gewählt wurde, sonst 0.

Ein Verschlüsselungsverfahren Π ist IND-CCA-1-sicher, wenn für alle probabilistischen Polynomialzeit-Angreifer $\mathcal{A}$ eine vernachlässigbare Funktion negl existiert, so dass gilt:

$$\Pr[\mathrm{PrivK}^{\mathsf{cca\text{-}1}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \mathrm{negl}(n)$$

IND-CCA-2-Sicherheit

Der Sicherheitsbegriff IND-CCA-2 unterscheidet sich von IND-CCA-1 dadurch, dass dem Angreifer das Entschlüsselungsorakel auch nach Erhalt des Challenge-Chiffrats noch zur Verfügung steht. Es entschlüsselt alle Chiffrate außer dem Challenge-Chiffrat.

2.1.4. Plaintext-Equality-Test und encrypted Plaintext-Equality-Test

Plaintext-Equality-Tests (PETs) prüfen die Gleichheit der Klartexte zweier Chiffrate, ohne diese preiszugeben. Sie wurden von Juels und Jakobsson [70] im Jahr 2000 eingeführt. Die Funktion $\mathtt{PET}(\hat{c}_1, \hat{c}_2, t) \mapsto (b, \mathcal{P})$ erhält als Eingabe zwei Chiffrate $\hat{c}_1$ und $\hat{c}_2$ und eine Trapdoor-Funktion. Sie gibt ein Bit b, das aussagt, ob die Chiffrate den gleichen Klartext enthalten, und einen Korrektheitsbeweis $\mathcal{P}$ aus.

Für die Berechnung nutzen wir die homomorphen Eigenschaften des zugrunde liegenden Verschlüsselungsverfahrens:

$$\mathtt{ENC}(a) \div \mathtt{ENC}(b) = \mathtt{ENC}\left(\frac{a}{b}\right) \qquad\qquad \mathtt{ENC}(a) \star r = \mathtt{ENC}(a^r)$$

Die zweite Operation kann pro PET von unterschiedlichen Parteien (beziehungsweise Servern) mehrfach ausgeführt werden. Für diese Operation zieht der Server $r \neq 0$ zufällig und gleichverteilt. Wenn die Klartexte ungleich sind, wird dadurch ihr Quotient verschleiert.

Gegeben zwei Chiffrate $c_1 = \mathtt{ENC}_{\mathrm{pk}}(a)$ und $c_2 = \mathtt{ENC}(b)$ – verschlüsselt mit dem öffentlichen Schlüssel pk zu dem geheimen Schlüssel sk –, lässt sich die Funktion eines PET wie folgt beschreiben:

$$\begin{aligned}
\mathtt{PET}(c_1, c_2, \mathtt{DEC}_{sk}(\cdot)) &= \mathtt{DEC}_{\mathrm{sk}}\left(\left(\left((c_1 \div c_2) \star r_1\right) \star r_2\right) \cdots \star r_n\right) \\
&= \mathtt{DEC}_{\mathrm{sk}}\left(\left(\left(\mathtt{ENC}_{\mathrm{pk}}\left(\tfrac{a}{b}\right) \star r_1\right) \star r_2\right) \cdots \star r_n\right) \\
&= \mathtt{DEC}_{\mathrm{sk}}\left(\mathtt{ENC}_{\mathrm{pk}}\left(\left(\tfrac{a}{b}\right)^{r_1 \cdot r_2 \cdot \ldots \cdot r_n}\right)\right) \\
&= \left(\tfrac{a}{b}\right)^{r_1 \cdot r_2 \cdot \ldots \cdot r_n}
\end{aligned}$$

Der PET gibt mit überwältigender Wahrscheinlichkeit 1 aus, wenn $a = b$, und eine zufällige Zahl, wenn $a \neq b$. Der Korrektheitsbeweis $\mathcal{P}$ enthält einerseits Beweise, dass die Server die Operation $\star$ korrekt ausgeführt haben, und andererseits, dass die Entschlüsselung korrekt ist.

In manchen Kontexten soll das Ergebnis des PETs lediglich verschlüsselt zur Verfügung gestellt werden. Es ist erforderlich, dass das Chiffrat bereitgestellt wird, ohne dass das Ergebnis einer Partei bekannt wurde. Dafür erstellen wir einen *encrypted Plaintext-Equality-Test (ePET)*. Der ePET entspricht einem PET ohne die Anwendung der Trapdoor-Funktion.

2.2. Digitale Signaturen

Eine digitale Signatur berechnet eine Partei mit Hilfe ihres geheimen Schlüssels, um die Echtheit der signierten Daten für Dritte prüfbar zu machen. Die digitale Signatur kann von Dritten mit dem öffentlichen Schlüssel und den signierten Daten geprüft werden.

Ein Signaturverfahren besteht aus drei Algorithmen: `KeyGen`, `Sign` und `Verif`. `KeyGen` erstellt ein Schlüsselpaar, das aus einem geheimen Schlüssel, der für die Erstellung der Signatur verwendet wird, und einem öffentlichen Schlüssel für die Verifikation besteht. `Sign` erhält als Eingabe die zu signierende Nachricht und den geheimen Schlüssel und gibt eine Signatur aus. `Verif` erhält als Eingabe eine Signatur, eine Nachricht und einen öffentlichen Schlüssel; er gibt `True` aus, wenn die Signatur zum Schlüssel und der Nachricht passt; `False` sonst.

2.2.1. EUF-CMA sichere digitale Signaturen

Ein Signaturverfahren (`KeyGen`, `Sign`, `Verif`) ist EUF-CMA-sicher [63] (*existentially unforgeable against adaptive chosen message attacks*), wenn – mit der Verwendung eines Signaturorakels $O_{Sign(.)}$ – die Wahrscheinlichkeit

$$Pr[(vk, sk) \leftarrow \mathsf{KeyGen}(1_k), (\sigma, m) \leftarrow \mathcal{A}^{O_{Sign(.)}}(vk) : \mathsf{Verify}(vk, \sigma, m) = 1]$$

vernachlässigbar im Sicherheitsparameter k ist. Das Signaturorakel darf selbstverständlich nie eine Signatur auf m ausgegeben haben.

2.3. Commitment

Ein Commitment c ist ein Wert, mit dem sich eine Partei gegenüber einer zweiten Partei auf eine Information m (ein Bit, eine Zahl, ...) festlegen kann, ohne dass die zweite Partei m kennenlernt. Der Übergabe des Commitments – in der Commitment-Phase – folgt die Unveil-Phase, in der die erste Partei der zweiten Partei anhand des Commitments beweist, dass es sich bei c um ein Commitment auf m handelt.

Commitments müssen zwei Eigenschaften besitzen: sie müssen *verbergend* (*hiding*) und *bindend* (*binding*) sein.

hiding Die Eigenschaft verbergend zu sein bedeutet, dass die zweite Partei anhand des Commitments c nicht auf die dazugehörige Information m schließen kann.

binding Durch die bindende Eigenschaft ist es der ersten Partei nicht möglich, eine Information $m' \neq m$ zu finden, für die sie die zweite Partei davon überzeugen kann, dass c ein Commitment auf m' ist.

Ein Beispiel für ein Commitment-Verfahren ist das Pedersen-Commitment [96]: Mit Hilfe zweier (vom Empfänger gewählten) Erzeuger g und h einer Gruppe G mit Prim-Ordnung und einer für jedes Commitment frisch gezogenen Zufallszahl r berechnet sich das Commitment auf m als $c = \texttt{COM}(m, r) := g^m h^r$. Die Unveil-Informationen bestehen aus der Nachricht m und dem Zufall r. Pedersen-Commitments sind informationstheoretisch verbergend. Mit genügend Rechenleistung (oder zusätzlichem Wissen) lässt sich jedoch ein $m' \neq m$ und ein r' finden, für das $c = g^m h^r = g^{m'} h^{r'}$ gilt.

2.4. Blum-Coin-Toss

Manuel Blum hat ein Protokoll vorgeschlagen [19] (siehe Abbildung 2.1), wie zwei Parteien (Alice und Bob), die sich nicht im gleichen Raum befinden, einen fairen Münzwurf ausführen können. Dabei commitet sich eine Partei gegenüber der zweiten Partei auf ein (zufällig gewähltes) Bit b_A. Die zweite Partei wählt ihrerseits ein (zufälliges) Bit b_B und sendet dies an die erste Partei. Daraufhin sendet die erste Partei die Unveil-Informationen an die zweite Partei. Beide können nun das Ergebnis $b = b_A \oplus b_B$ berechnen.

Blum-Coin-Toss

Alice		Bob
$b_A \in \{0,1\}$		
$c_A := \texttt{COM}(b_A, r_A)$	$\xrightarrow{c_A}$	c_A
b_B	$\xleftarrow{b_B}$	$b_B \in \{0,1\}$
	$\xrightarrow{b_A,\, r_A}$	$\texttt{COM}(b_A, r_A) \stackrel{?}{=} c_A$
$b := b_A \oplus b_B$		$b := b_A \oplus b_B$

Abbildung 2.1.: Coin-Toss-Protokoll, wie von Manuel Blum vorgeschlagen.

Da die erste Partei das Ergebnis berechnen kann, bevor sie der zweiten Partei die dazu notwendigen Informationen zur Verfügung stellen muss, garantiert das Protokoll keine Fairness. Für unsere Zwecke ist das jedoch ausreichend. Wichtig ist für uns die Betrachtung einer Modifikation des Protokolls und der Erweiterung des Protokolls auf mehr als zwei Parteien.

2.4.1. Unsichere Erweiterung des Blum-Coin-Toss

Wir ändern nun das Protokoll so ab, dass Bob auch ein Commitment erstellt und an Alice übergibt (siehe Abbildung 2.2); die Unveil-Informationen sendet Bob, nachdem er die Unveil-Informationen von Alice erhalten hat.

Das ermöglicht Bob, statt mit eines eigenen Commitments – wie oben dargestellt – mit Alice's Commitment (c_A) oder einer Maskierung davon zu antworten. Da er die Unveil-Informationen von c_A in dem Moment, in dem er Alice das Commitment schickt, noch

Blum-Coin-Toss

Alice		Bob
$b_A \in \{0,1\}$		$b_B \in \{0,1\}$
$c_A := \texttt{COM}(b_A, r_A)$	$\xrightarrow{c_A}$	c_A
c_B	$\xleftarrow{c_B}$	$c_B := \texttt{COM}(b_B, r_B)$
	$\xrightarrow{b_A, r_A}$	$\texttt{COM}(b_A, r_A) \stackrel{?}{=} c_A$
$\texttt{COM}(b_B, r_B) \stackrel{?}{=} c_B$	$\xleftarrow{b_B, r_B}$	
$b := b_A \oplus b_B$		$b := b_A \oplus b_B$

Abbildung 2.2.: Unsichere Erweiterung des Coin-Toss-Protokolls

nicht kennt, ist es ihm zu diesem Zeitpunkt noch nicht möglich, das von ihm gesendete Commitment zu öffnen. Nachdem er die Unveil-Information von Alice erhalten hat, kann er die zu seinem Commitment gehörenden berechnen. Dadurch ist es Bob möglich, das Ergebnis $b = 0$ zu erzwingen.

Das Problem kann durch eine Invertierung der Unveil-Reihenfolge behoben werden: Muss Bob vor Alice die Unveil-Informationen preisgeben, ist es Bob nicht mehr möglich, mit einer Maskierung von Alices Commitment ein bestimmtes Ergebnis zu erzwingen. Durch die korrekte Unveil-Reihenfolge erhalten wir eine sichere Erweiterung des Blum-Coin-Toss.

2.4.2. Erweiterung des Protokolls auf mehrere Parteien

Durch die sichere Erweiterung des Blum-Coin-Toss durch die Invertierung der Unveil-Reihenfolge kann das Protokoll um weitere Teilnehmer ergänzt werden (siehe Abbildung 2.3). Im Folgenden wird nur noch ein Protokollteilnehmer P_i dargestellt. Dieser ist über den angedeuteten Broadcast-Kanal mit allen anderen Teilnehmern verbunden.

Die dargestellte Variante lässt sich als sicher beweisen. Kein Protokollteilnehmer kennt sowohl das Commitment eines anderen Teilnehmers, bevor er sein eigenes Commitment erstellen, als auch dessen Unveil-Informationen, bevor er seine eigenen Unveil-Informationen versenden muss. Wir nutzen ein Protokoll dieser Art in Kapitel 5.3 für die verteilte Berechnung der Füllstimmen in Bingo Voting.

2.5. Non-Interactive Zero-Knowledge-Beweis (of Knowledge)

Wir verwenden in unseren Protokollen Zero-Knowledge-Beweise (ZK), nicht-interaktive Zero-Knowledge-Beweise (NIZK) und nicht-interaktive Kenntnisbeweise mit Zero-Knowledge-Eingenschaft – sogenannte *non-interactive Zero-Knowledge Proofs of Knowledge* (NIZKPOK). Zero-Knowledge-Beweise sind Beweise, mit deren Hilfe ein Beweiser (*prover*)

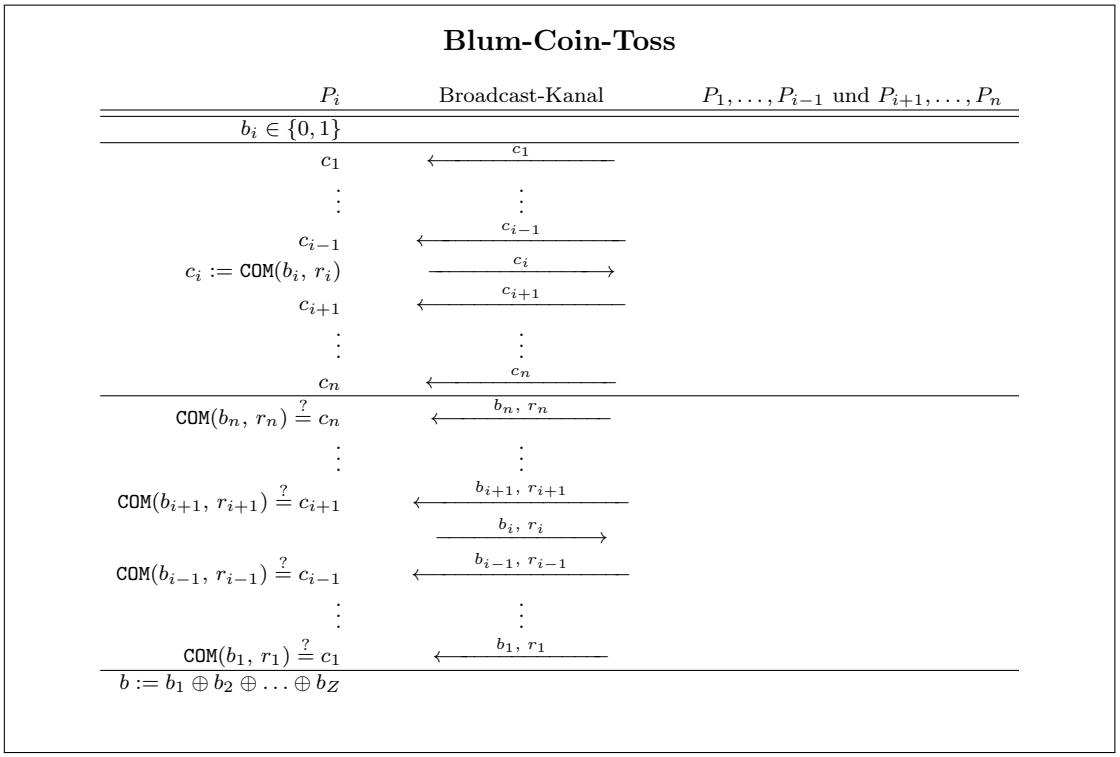

Abbildung 2.3.: Sichere Mehrparteien-Erweiterung des Coin-Toss-Protokolls

einen Verifizierer (*verifier*) von der Gültigkeit einer Aussage überzeugen kann, ohne dass diesem dabei weitere Informationen – wie beispielsweise den Zeugen, der die Aussage belegt – bekannt werden. Unter der Annahme der Existenz von Einwegfunktionen gibt es solche Beweise für beliebige NP-Probleme ([59], Theorem 4.7.7). Eine Einführung in diese Beweistechnik ist in Goldreichs *Foundations Of Cryptography – Basic Tools* [59] zu finden. Für korrektes Entschlüsseln kann beispielsweise das Chaum-Pedersen-Protokoll [34] verwendet werden. Das Schorr-Protokoll [107] beweist die Kenntnis eines Logarithmus eines Wertes zu gegebener Basis. Der Beweis über die Kenntnis einer Signatur wird von Camenisch et al. [30] vorgestellt.

Im Folgenden geben wir ein Beispiel für ein Zero-Knowledge-Protokoll: Verifizierbare und geheime Mix-Nets zur beweisbar korrekten Rerandomisierung und Permutation einer Liste von Chiffraten oder Commitments.

2.5.1. Verifizierbares und geheimes Shuffle

Ein verifizierbares `Shuffle` erhält als Eingabe eine Liste $C := [c_1, \ldots, c_n]$ rerandomisierbarer Elemente. Für die Ausführung des `Shuffle` werden eine zufällige Permutation π und n Zufallswerte r_i für die Rerandomisierung gezogen. Der `Shuffle` gibt eine Liste $C' := [c'_{\pi^{-1}(1)}, \ldots, c'_{\pi^{-1}(n)}]$ und einen Korrektheitsbeweis $\mathcal{P}$ aus. Für $1 \leq i \leq n$ ist $c'_{\pi^{-1}(i)}$

das mit $r_{\pi^{-1}(i)}$ rerandomisierte Element, das entsprechend der Permutation π an die i-te Stelle der Ausgabeliste C' verschoben wird. Mit Hilfe von $\mathcal{P}$ kann verifiziert werden, dass es sich bei C' um eine Permutation der rerandomisierten Elemente aus C handelt.

Wir fordern, dass das `Shuffle` *geheim* und *verifizierbar* ist. Geheim bedeutet in diesem Kontext, dass dem Verifizierer für jedes Element aus C nicht mit besserer Wahrscheinlichkeit als raten möglich ist, das korrespondierende Element aus C' zu benennen. Verifizierbar ist ein `Shuffle`, wenn ein Beweis $\mathcal{P}$ existiert, der nachweist, dass C' eine permutierte Liste der rerandomisierten Elemente aus C ist.

Für den Beweis gibt es verschiedene Herangehensweisen. Die Lösung, die sich in der Literatur aufgrund ihrer Sicherheitseigenschaften durchgesetzt hat, nennt sich *Shadow-Mixes*. Dabei werden k zufällige Permutationen $\pi_{j,L}$ erstellt, die jeweils zusammen mit einer berechneten Permutation $\pi_{j,R}$ die Gesamtpermutation ergeben: $\pi_{j,R} \circ \pi_{j,L} = \pi$. Mit den Zufallswerten für die während der Permutation angewendeten Rerandomisierungen – die im Weiteren immer implizit bei jeder Permutation ausgeführt werden – wird ebenso verfahren. Während des Beweises werden – ausgehend von den k Zwischenpermutationen, die sich durch die Anwendung von $\pi_{j,L}$ auf die Eingabeliste C ergeben – die Verbindungen entweder zu C oder zu C' aufgedeckt, jedoch niemals zu beiden Seiten.

Wir gehen weiter davon aus, dass ein `Shuffle` mehrfach hintereinander von unterschiedlichen Servern ausgeführt wird, damit die Gesamtpermutation zwischen der Eingabeliste des ersten Servers und der Ausgabeliste des letzten Servers niemandem bekannt ist, solange mindestens ein Server vertrauenswürdig ist. Die Mehrfachausführung wird als Mixnetz (*Mix-Net*) bezeichnet. Wenn wir von einem `Shuffle` oder dessen Anwendung sprechen, meinen wir im Allgemeinen ein Mix-Net respektive dessen Anwendung.

Mix-Nets finden bei vielen Wahlverfahren Anwendung und können als einer der Standardbausteine betrachtet werden. Beispiele für geheime und verifizierbare Umsetzungen finden sich in zahlreichen Veröffentlichungen [90, 106, 68, 78, 5, 66].

2.6. Hash-Funktion

Eine Hash-Funktion bildet eine Eingabe beliebiger Länge deterministisch auf eine Ausgabe fester Länge ab. Eine kryptographische Hash-Funktion $\mathtt{H}(\cdot)$ erfüllt die folgenden Eigenschaften:

Die Hash-Funktion ist eine Einwegfunktion: Es ist nicht effizient möglich, ein Urbild zu einem gegebenen Hashwert zu berechnen.

Die Hash-Funktion ist kollisionsresistent: Es ist nicht effizient möglich, zwei Klartexte $m \neq m'$ zu finden, die den gleichen Hashwert besitzen.

2.7. Bulletin Board

Bei vielen Wahlverfahren gibt es die Notwendigkeit, Informationen öffentlich bekannt zu machen. Eine gängige Möglichkeit bietet ein sogenanntes *Bulletin Board* – ein „Schwarzes Brett“. Auf die technische Umsetzung werden wir nicht näher eingehen. Stattdessen nehmen wir eine Umsetzung mit den gewünschten Eigenschaften als gegeben an. In den meisten Fällen sind die folgenden Eigenschaften ausreichend:

- Jede autorisierte Partei darf Nachrichten an die bereits bestehenden anhängen. Autorisiert sind genau die Parteien, für die eine Notwendigkeit besteht, Nachrichten zu veröffentlichen. In den meisten Fällen sind das einzelne Server und die Wahlleitung, in manchen Fällen die Wähler selbst.
- Zusätzlich kann angenommen werden, dass die Form dessen, was auf das Board geschrieben wird, genau spezifiziert ist. So kann ein Wähler im Allgemeinen lediglich einen (verschlüsselten) Stimmzettel veröffentlichen. Server hingegen müssen oftmals Beweise für die korrekte Ausführung einer Berechnung veröffentlichen.
- Keine Nachricht, die auf das Bulletin Board geschrieben wurde, kann verändert oder gelöscht werden. Technisch stellt dies die größte Herausforderung dar.
- Das Lesen des gesamten Inhalts des Bulletin Boards ist jederzeit ohne Einschränkung möglich. Selbstverständlich ist es sinnvoll, einen Zeitpunkt zu definieren, zu dem das Bulletin Board offline genommen werden kann. Wie viel Zeit nach der Veröffentlichung der Wahlergebnisse vergehen muss, bis das Bulletin Board archiviert werden kann, bedarf einer gesetzlichen Regelung und liegt außerhalb des Fokus dieser Arbeit.

In unserer Erweiterung von Bingo Voting werden wir ein Bulletin Board ($\mathcal{BB}$) mit einer weiteren Eigenschaft benötigen. Ein Teil der Inhalte ist ausschließlich für die Mitglieder der Wahlleitung einsehbar. Alle anderen Daten sind öffentlich und können von jedem eingesehen werden.

Benötigt wird das $\mathcal{BB}$ unter anderem auch für die Veröffentlichung von Beweisen, wie sie für das korrekte Entschlüsseln oder das ordnungsgemäße Mischen (siehe 2.5.1) erstellt werden. Techniken, die bei der Umsetzung eines Bulletin Boards hilfreich sind, schließen Hashes, Signaturen und „Secure Logging“ ein.

3. Bezahlprotokolle

Ein prominentes Beispiel für eine sichere Hardware, die in unserem Alltag Verwendung findet, ist die *Girocard*, die umgangssprachlich meist noch als *EC-Karte* bezeichnet wird. Eine Belastung des zugeordneten Kontos soll nur mit Zugriff auf die Girocard und den dazugehörigen Authentifikationsinformationen – zum Beispiel einer PIN – möglich sein. Die Entwickler streben damit an, dass sich der Kunde mit zwei Faktoren (*Chip-und-PIN-Verfahren*) ausweisen muss: Er muss im *Besitz* der Girocard sein und *Kenntnis* der PIN haben.

Ermöglicht wird das durch den Chip, den die Girocard enthält. Er verfügt über eine digitale Schnittstelle, etwas Speicher und ein wenig Rechenleistung. Für das bei Online-Transaktionen eingesetzte Chip-TAN-Verfahren wurde die Girocard mittels einer weiteren vertrauenswürdigen Hardware um eine optische Schnittstelle und ein einfaches Display erweitert.

Wir werden uns in diesem Kapitel mit der Frage beschäftigen, wie die Girocard als sichere Hardware für das Geldabheben an Automaten verwendet werden kann. Dabei stehen Ideen und Konzepte im Mittelpunkt. Dieser Anwendungsfall dient als einführendes Beispiel; wir wollen betrachten, wie die Verwendung vertrauenswürdiger Hardware zur Erstellung einfacher, möglichst modularer Protokolle genutzt werden kann. Ziel dieser Arbeit ist es nicht, ein Protokoll zur Ablösung des zur Zeit (2017) eingesetzten EMV-Verfahrens zu erarbeiten. Vielmehr interessieren uns mögliche Designziele und ob diese mit Hilfe vertrauenswürdiger Hardware mit minimalen Mitteln erreicht werden können.

Die drei für diesen Anwendungsfall wichtigsten Designparadigmen, die wir untersuchen und umsetzen wollen, sind die folgenden:

- Komplexe Maschinen sind (im Allgemeinen) nicht vertrauenswürdig.
- Abweichungen im Protokollablauf müssen erkennbar und beweisbar sein.
- Komponenten ohne zufällige Session-Nonce sind anfällig für Replay-Angriffe.

Aktuelle Protokolle nehmen sowohl den Geldautomaten als sicher als auch die Verbindung als vertraulich und unmanipulierbar an. Die Vergangenheit hat gezeigt, dass diese Annahmen nicht erfüllt werden; die Sicherheitsziele, die mit der Einführung des Chip-und-PIN-Verfahrens angestrebt waren, konnten daher nicht vollständig erreicht werden. Wir schlagen eine Struktur für Protokolle vor, sodass die Sicherheit der Protokolle auf schwächeren Annahmen beruht und sie dennoch die gewünschten Sicherheitsziele erreichen.

Zuerst werfen wir in Abschnitt 3.1 einen Blick auf das zur Zeit eingesetzte EMV-Verfahren, seine Stärken und Schwächen. Aus den Schwächen können wir einige Fehler im Entwurf des Verfahrens ableiten. Die Basis, auf der wir die Sicherheit unseres Verfahrens aufbauen, stellen wir in Abschnitt 3.2 vor. In Abschnitt 3.3 und 3.4 beschreiben wir über die Manipulationsresistenz hinausgehende Ziele und Methoden und wie wir sie erreichen können. Aus diesen Grundlagen konstruieren wir ein einfach strukturiertes Protokoll, das wir in Abschnitt 3.5 vorstellen. Wir schließen das Kapitel mit einem Ausblick, in welche Richtung sich die weitere Forschung entwickeln kann, und einem Fazit.

3.1. Das EMV-Protokoll

Bevor wir auf unseren Ansatz eingehen, werfen wir einen Blick auf das aktuelle EMV-Protokoll. EMV (**E**uropay International, **M**asterCard und **V**ISA) ist eine Spezifikation für Zahlungskarten [52]. Sie beschreibt die Schnittstelle zwischen Karte und Terminal, die Anwendungsauswahl, Sicherheit und Schlüssel-Handhabung, die Anwendungsspezifikation und die Schnittstellen-Anforderungen. Für die Authentifizierung des Karteninhabers gibt es verschiedene Verfahren (*cardholder verification methods*).

Wir wollen keine vollumfängliche Analyse des Kommunikationsprotokolls oder anderer Spezifikationen vornehmen. Uns interessieren lediglich die zugrunde liegenden Konzepte sowie ihre Stärken und Schwächen.

3.1.1. Angriffe

Auf das EMV-Protokoll gibt es eine Vielzahl unterschiedlicher Angriffe. Der zu den bekanntesten zählende nutzt einen Fehler in der Implementierung der Authentifizierung aus. Er wurde im Jahr 2010 von Murdoch, Drimer, Anderson und Bond vorgestellt [88] und macht sich zunutze, dass zwei unterschiedliche Authentifizierungsmethoden implementiert wurden. Besonders interessant sind für uns zwei weitere Angriffstypen: Es gibt verschiedene Berichte von Angriffen auf die Steuerungssysteme der Bankautomaten, die es ermöglichen, dem Automaten den Befehl zur Geldausgabe zu geben (*Jackpotting*). Ein anderer Angriffstyp ermöglicht es einem Angreifer, Geld von einem Konto abzuheben, ohne zu diesem Zeitpunkt Zugriff auf die zugehörende Karte zu haben (*Preplay-Angriffe*).

Chip und PIN

Bei der Einführung des Chips, der die Authentifizierung des Kunden über die PIN übernimmt[1], wollten die Entwickler die Kompatibilität zu Systemen, die das Chip-und-PIN-Verfahren noch nicht unterstützen, aufrecht erhalten. Daher ist es auch bei der Karte,

[1] Die Karte authentifiziert sich außerdem gegenüber dem Terminal und überprüft die Authentifizierung des Terminals.

die einen Chip besitzt, möglich, dass sich der Kunde mit seiner Unterschrift authentifiziert. Durch einen Fehler in der Implementierung des Systems ist es einem *Man-in-the-Middle*-Angreifer möglich, das Terminal glauben zu lassen, dass sich der Kunde mit einer PIN authentifiziert, während der Chip eine Authentifizierung über die Unterschrift des Kunden erwartet: Der Chip akzeptiert in diesem Fall jede PIN und meldet die erfolgreiche PIN-Eingabe zurück; aus Sicht des Terminals findet eine erfolgreiche PIN-Authentifizierung statt [88].

Jackpotting

Bankautomaten dürfen nur dann Geld an einen Kunden ausgeben, wenn die Auszahlung von der Bank autorisiert ist. Ross Anderson berichtet bereits 1993 [9] von einem Angriff, bei dem der Automat eine gefälschte Anweisung erhält, Geld auszugeben. In dem von ihm berichteten Fall war es mangels korrekter Authentifikation möglich, dem Geldautomaten die Nachricht der Bank, das Geld auszuzahlen, erneut einzuspielen.

Bei zuletzt ausgeführten Angriffen dieser Art werden Schwachstellen des Steuerungssystems des Bankautomaten ausgenutzt. Da es sich bei den Steuerungssystemen in den meisten Fällen um handelsübliche PCs handelt, deren Betriebssystem oft nicht auf dem aktuellen Stand ist, können Angreifer bekannte Lücken nutzen, um das System unter ihre Kontrolle zu bringen. In Esslingen und Berlin wurden am 9. August 2015 Geldautomaten mit dieser Technik ausgeraubt [15].

Preplay-Angriff

Die Idee von Chip und PIN des EMV-Verfahrens ist, dass der Kunde, um zu bezahlen oder Geld abzuheben, über zwei Dinge verfügen muss: Die Karte selbst und die dazugehörige PIN.

Eine Designschwäche im EMV-Verfahren ermöglicht es einem Angreifer, der Bank vorzutäuschen, gegenwärtig Zugriff auf eine Karte zu haben, obwohl er nicht mehr darüber verfügt. Ein Angreifer kann, wenn er die PIN der Karte kennt, Daten aus der Karte auslesen, die es ihm später ermöglichen – ohne Zugriff auf die Karte – Geld an einem Geldautomaten abzuheben.

Die Schwäche, die dieser Angriff ausnutzt, liegt in der Wahl der Zufallszahlen, die als Nonce dienen. Diese werden vom Geldautomaten gewählt. Das wäre kein Problem, wenn der Zufall, den der Automat wählt, unvorhersagbar und der Automat sowie die Kommunikation zur Bank vom Angreifer nicht beeinflusst werden könnten. Wie Bond, Choudary, Murdoch, Skorobogatov und Anderson in ihren Veröffentlichungen gezeigt haben, gilt keine der drei Annahmen [23, 22].

Kann der Angreifer die Nonce entweder selbst wählen oder vorhersagen, welche Nonce der Automat wählen wird, kann er sich die für das Protokoll notwendigen (symmetrischen)

Signaturen (MAC) zuvor von der Karte erstellen lassen und während des Protokollablaufs einspielen. Die Karte muss zum Zeitpunkt der Kommunikation mit der Bank demzufolge nicht verfügbar sein. Das verletzt die Anforderung, dass der Kunde für einen Bezahlvorgang in dem Moment der Transaktion über die Karte, die belastet werden soll, verfügen muss.

3.2. Vertrauen der Inhaber

Das Vertrauen der Nutzer basiert (respektive *sollte basieren*) auf der Sicherheit des Systems. Die Sicherheit des Systems beruht auf der Gültigkeit der dafür gemachten Annahmen und der Umsetzung. Darauf werfen wir in diesem Abschnitt einen genaueren Blick. In den folgenden Kapiteln werden wir diese Diskussion – speziell die über das Vertrauen in die Hardware – unter den dann veränderten Umständen wiederholen.

Die Sicherheit der Geldautomaten, wie sie aktuell eingesetzt werden, beruht unter anderem auf zwei Annahmen: Zum einen gehen die Hersteller davon aus, dass die Automatensteuerung sicher ist. Zum anderen ist es notwendig, dass die Kommunikation mit der Bank geschützt ist.

Es scheint plausibel anzunehmen, dass die Automatensteuerung sicher umgesetzt werden muss; kann dieser doch die Auszahlung von Bargeld anweisen. Bei dem Steuerungssystem handelt es sich jedoch um handelsübliche Computer, auf denen häufig Windows als Betriebssystem eingesetzt wird. Laut einem Bericht von Bloomberg [115] liefen im Jahr 2014 95 Prozent aller Maschinen mit dem Betriebssystem Windows XP. Die Firma *Diebold Nixdorf* bereitet sich nach eigenen Angaben [91] aktuell auf den Umstieg von Windows 7 auf Windows 10 vor. Komplexe Software ist jedoch selten – vielleicht sogar nie – frei von Fehlern [54], die zu Sicherheitslücken führen.

Außerdem wird davon ausgegangen, dass die Kommunikation zur Bank gesichert ist. Unter Berücksichtigung der Tatsache, dass Banken in der Lage sind, ihr eigenes Netz für diese Zwecke aufzubauen und zu betreiben, scheint auch diese Annahme plausibel. Doch die Schnittstellen sind in vielen Fällen nur unzureichend geschützt. So zeigt Olga Kochetova in ihrem Bericht [79] beispielsweise auf, dass die Anschlüsse von freistehenden Geldautomaten oft leicht zugänglich sind.

Die Vergangenheit hat gezeigt, dass es eine Vielzahl von Angriffen aufgrund dieser zwei nicht haltbaren Annahmen gibt. So häufen sich in den letzten Jahren Berichte über Angreifer, die beispielsweise Fernzugriff [83] oder Direktzugriff [85] auf das Steuersystem erlangen.

Die Annahmen, die die Banken bisher gemacht haben, sind demzufolge praktisch nicht erfüllbar. Betrachten wir, worauf die Bank (und die Kunden) ihr Vertrauen stützen können.

Vertrauen in die Hardware

Das Stützen einer Annahme auf eine Hardware ist nur sinnvoll, wenn die Hardware diese Annahme erfüllen kann. Ob der Nutzer glaubt, dass die Hardware die Annahmen erfüllt, ist entscheidend für das Vertrauen des Nutzers. In diesem Abschnitt beschreiben wir den speziellen Fall der Girocard als vertrauenswürdige Hardware.

Das Konzept von Hardware als Vertrauensanker basiert – wie der Name schon sagt – darauf, dass die Person, deren Gut geschützt werden soll, der Hardware vertraut. Wir nennen diese Person im Weiteren auch den *Inhaber*. *Vertrauen* bedeutet an dieser Stelle, dass der Nutzer davon ausgeht, dass die Hardware wie erwartet arbeitet und ein Angreifer die Funktion nicht beeinflussen kann. Das Gut, das der Nutzer geschützt wissen will, ist in diesem Szenario das Guthaben auf seinem Konto: sein Geld. Wir werden in den weiteren Kapiteln dieser Arbeit noch auf andere Werte zu sprechen kommen, die wir durch den Einsatz vertrauenswürdiger Hardware schützen wollen. Wir werden uns bei den zentralen, als vertrauenswürdig angenommenen Hardwarebausteinen die Frage stellen, warum derjenige, dessen Gut geschützt werden soll, Vertrauen in die Hardware haben können soll. Die Frage nach dem Vertrauen der Kunden in die Girocard lässt sich leicht beantworten:

Zunächst muss erwähnt werden, dass sich der Kunde – so hoffen wir – selbst (weit genug) vertraut. Diese Eigenschaft wirkt überflüssig oder trivial, da der Inhaber des Gutes, das geschützt werden soll, und der Nutzer der Hardware ein und dieselbe Person ist. In den folgenden Kapiteln wird diese Beziehung interessanter.

Wir nehmen an, dass der Kunde seiner Bank Vertrauen schenkt; würde er das nicht tun, so überließe er ihr sicherlich nicht sein Geld: Die Bank kann das Geld veruntreuen auch ohne die Girocard zu nutzen. Wir nehmen des Weiteren an, dass die Bank dem Hersteller der Girocard vertraut. Dieses Vertrauensverhältnis beruht im Allgemeinen auf Zertifizierungen, die die korrekte Funktionsweise des Produktes nachweisen.

Der Inhaber des zu schützenden Guts ...
... ist Nutzer der vertrauenswürdigen Hardware.
... vertraut dem Hersteller der Hardware.

Zusammenfassend kann gesagt werden, dass der Inhaber dem Nutzer und dem Hersteller der Girocard vertraut; dafür setzen wir die Transitivität von Vertrauen in diesem speziellen Kontext voraus. Umgekehrt formuliert bedeutet das, dass der Angreifer die vertrauenswürdige Hardware weder hergestellt hat noch dauerhaft über sie verfügt. Das ist der einfachste Fall; in den folgenden Kapiteln werden wir andere Konstellationen betrachten.

Sowohl die Annahme über die korrekte Kontoführung durch die Bank als auch die Erfüllung der Anforderungen an die Girocard können und sollten in Frage gestellt werden; uns sind jedoch keine Vorfälle bekannt, in denen diese Annahmen gebrochen wurden. Die (für jede Girocard individuellen) Geheimnisse können mit Spezialhardware ausgelesen werden. Aber dieser Aufwand übersteigt im Allgemeinen den Gewinn eines Angriffs. Daher können die Kunden die Girocard zurecht als Vertrauensanker betrachten.

Mit der Girocard alleine wird es uns nicht gelingen, den Vorgang des Geldabhebens sicher zu gestalten. Auch wir müssen uns für unseren Protokollentwurf auf verschiedene unkorrumpierbare Komponenten im Automaten stützen. Im Automaten ist das allen voran die Geldkassette: das Bauteil, das die Geldscheine, die an die Kunden ausgegeben werden, bereithält und bei Bedarf ausgibt. Erhält der Angreifer die Kontrolle über dieses Bauteil, hat er Zugriff auf das darin lagernde Geld. Der Unterschied zu der Annahme über die sichere Steuerung des gesamten Geldautomaten liegt in der Einfachheit der Programmierung der Geldkassette im Vergleich zu der des gesamten Systems. Wie wir später zeigen werden, muss die Geldkassette nur wenige, einfache Operationen ausführen. Die Auszahlungsanweisung erhält die Geldkassette auch nicht von dem Steuerungssystem des Geldautomaten, sondern von der Bank selbst. Das Vertrauen in die Geldkassette ließe sich auf mehrere Komponenten aufteilen. Der Einfachheit halber werden wir das hier nicht tun.

Es stellt sich demzufolge die Frage, ob ein Protokoll realisierbar ist, das Schutz vor Angriffen bietet, obwohl alle Nachrichten (im Klartext) vom Sender über den Angreifer an den Empfänger weitergeleitet werden, und obwohl der Computer, der den Automaten steuert, durch den Angreifer kontrolliert wird. Unser Ziel ist es, dass nur kleinere Komponenten mit geringerem Funktionsumfang als vertrauenswürdig angenommen werden müssen. Über die Blockierung der Kommunikation (und das Senden beliebiger Nachrichten) hinaus sollen dem Angreifer keine weiteren Angriffe möglich sein.

3.2.1. Unser Modell

Wie bereits beschrieben, möchten wir in unserem Modell, das die parallele Ausführung einer Transaktion pro Kunde zulässt, insbesondere zwei Annahmen ändern: Zunächst soll die komplexe Maschine, die die Steuerung des Automaten übernimmt, für die Sicherheit des Vorgangs des Geldabhebens nicht relevant sein; wir nehmen sie als durch den Angreifer korrumpiert an.

Komplexe Maschinen eignen sich nicht als vertrauenswürdige Hardware
Nachvollziehbare, verifizierbar korrekte Umsetzung ist erreichbar durch einfache Aufgaben im Protokollablauf und minimale, klar definierte Schnittstellen

Die zweite Annahme – dass die Kommunikation sicher ist – wollen wir (fast) ganz verwerfen. Sämtliche Kommunikation zwischen den Komponenten des Geldautomaten und der Bank läuft in unserem Modell über den Angreifer. Das deckt sich mit der ersten Modifikation: Die unkorrumpierbaren Komponenten sind an den potentiell korrumpierten Steuerungsrechner angeschlossen. An einer Stelle im Protokoll werden wir jedoch noch einen unkorrumpierbaren Kanal von der Bank zum Kunden benötigen.

Mächtigkeit des Angreifers

Durch die Kontrolle über die Kommunikation hat der Angreifer nicht nur die Macht, Nachrichten aktiv zu verändern, sondern auch sie zu unterdrücken. Dadurch ist es dem Angreifer möglich, das Protokoll an Stellen, an denen Kommunikation stattfindet, abzubrechen. Da sich das nicht verhindern lässt, müssen wir damit umgehen, dass das Protokoll zu einem von uns nicht vorgesehenen Zeitpunkt unterbrochen wird. Die Anforderung, die wir an das Protokoll stellen, ist, dass es sich in keinem Zustand befinden kann, in dem der Kunde oder die Bank einen Schaden nicht erkennen und ausgleichen kann. Zusätzlich möchten wir erreichen, dass der Nutzer immer Klarheit über den Status des Protokolls hat. Angriffe, wie es sie auf herkömmliche Automaten gibt [101], in denen der Nutzer das Verhalten des Automaten nicht nachvollziehen und belegen kann, sollen ausgeschlossen werden.

3.3. Kontrollmöglichkeit durch den Kunden als Designziel

Weder die Bank noch der Geldautomat oder die Girocard können – sehen wir von heuristischen Methoden zur Fehlerfindung ab – entscheiden, ob eine Transaktion tatsächlich vom Inhaber der Girocard gewünscht ist. Das kann lediglich der Kunde.

Die Kontrolle der Korrektheit der Transaktion durch den Kunden kann in drei Schritte gegliedert werden:

- Der Kunde muss sicherstellen können, dass die vom System angenommene Transaktion der gewünschten entspricht.
- Der Kunde muss sich sicher sein, dass er durch seine Authentifizierung keine andere Transaktion autorisiert.
- Der Kunde muss kontrollieren können, dass genau diese Transaktion ausgeführt wurde.

Die drei Schritte sind vergleichbar mit einem Sicherheitsaspekt bei kryptographischen Wahlverfahren: *cast as intended*, *recorded as cast* und *counted as cast*. Bei Wahlverfahren soll zunächst der Wähler sicherstellen können, dass der Stimmzettel, den er ausgefüllt hat, eine Stimme für genau den Kandidaten repräsentiert, den er wählen will. Der Stimmzettel

soll unverändert angenommen und ausgezählt werden. Dieses Konzept wollen wir auch bei diesem Protokoll umsetzen.

Abweichungen im Protokollablauf müssen *erkennbar* und beweisbar sein.
Der Kunde muss die Kontrolle über die auszuführende Aktion haben.

3.3.1. Nachvollziehbarkeit der Korrektheit

Der Schritt vom EMV-Protokoll zu unserem Protokoll weist viele Parallelen zum Schritt von TAN-Nummern zu dem Chip-TAN-Verfahren für Online-Zahlungsanweisungen (Onlinebanking) auf. Noch bevor wir auf das Protokoll eingehen, möchten wir zwei Aspekte des Chip-TAN-Verfahrens hervorheben.

Vertrauenswürdige Anzeige Das Chip-TAN-Gerät erweitert die Girocard nicht nur um eine optische Schnittstelle, sondern auch um ein Display, das die Auftragsdaten, die der Kunde bestätigen soll, anzeigt. Das Chip-TAN-Gerät liest über die optische Schnittstelle Daten vom Bildschirm des Kunden; die Daten enthalten die Transaktionsinformationen. Das Gerät zeigt dem Kunden diese Informationen an und leitet sie bei Bestätigung an die Girocard weiter. Die Girocard berechnet eine auf den Transaktionsinformationen beruhende TAN, die auf dem Display des Chip-TAN-Geräts angezeigt wird und mit der der Kunde die Transaktion autorisiert.

Durch die einfache Hard- und Software des Chip-TAN-Gerätes sowie die minimalistische Schnittstelle ist anzunehmen, dass das Gerät nicht korrumpiert werden kann. Da das Gerät – wenn es unkorrumpiert ist – die Transaktionsinformationen auf seinem Display anzeigt, die es auch an die Girocard weitergibt, kann der Kunde die Eingabe, die die Girocard erhält, beobachten;

Das Display garantiert – unter der Annahme der Unkorrumpiertheit des Chip-TAN-Gerätes – die Eigenschaft, die *cast as intended* entspricht.

Verknüpfung zwischen Authentifizierung und Auftrag Die TAN, die die Girocard berechnet, ist unter anderem von den Auftragsdaten und einem Geheimnis, das sich die Girocard und die Bank teilen, abhängig. Die TAN, mit der der Kunde die Überweisung autorisiert, ist dementsprechend fest den Auftragsdaten zugeordnet, die ihm das Gerät anzeigt. Die Bank erwartet vom Kunden die von ihr – basierend auf den ihr bekannten Auftragsdaten – selbst berechnete TAN als Autorisierungscode. Die Annahme hierbei ist, dass ein Angreifer ohne Zugriff auf die Girocard keine gültige TAN zu selbst gewählten Auftragsdaten erzeugen kann.

Offensichtlich ist, dass die TAN im Chip-TAN-Verfahren zusätzlich von einer einmal verwendbaren Nummer – beispielsweise Datum und Uhrzeit oder einer einmal verwendeten Zufallszahl (Nonce) – abhängen muss, um Replay-Angriffe zu vermeiden. Die Wahl darf hierbei nicht dem Chip-TAN-Gerät überlassen werden. Für ein sicheres

Protokoll muss diese von der prüfenden Instanz – in diesem Fall der Bank – gewählt werden.

Technische Bemerkung: Wahl einer Nonce
Den Fehler, dass nicht die Bank die für den Protokollablauf zu verwendende Zufallszahl (*Nonce*) wählt, ermöglicht im EMV-Verfahren den Preplay-Angriff: Hier wird die für jede Transaktion verwendete Zufallszahl von dem Geldautomaten gewählt. Dadurch ist es einem Angreifer, der den Automaten unter Kontrolle hat, möglich, eine Transaktion zu tätigen, ohne in diesem Moment Zugriff auf die Girocard zu haben.

Wenn die Bank nur Aufträge annimmt, die durch eine so erzeugte TAN autorisiert wurden, war die Girocard des Kunden in den Prozess involviert. Daraus folgt zwar noch nicht, dass diese durch den rechtmäßigen Kunden bedient wurde, aber dass die Auftragsdaten auf dem Übertragungsweg nicht manipuliert wurden. Damit ist die Eigenschaft *recorded as cast* erfüllt. Die Überprüfung der Legitimität des Nutzers wird durch ein Login-Verfahren auf der Webseite der Bank übernommen.

Das Chip-TAN-Verfahren zeigt, wie zwei wichtige Eigenschaften im Prozess der Auftragsabwicklung erreicht werden können. Die dritte Eigenschaft – *counted as cast* – ist im Gegensatz zu Wahlverfahren, die wir später betrachten werden, einfach zu erreichen, da Transaktionen einzeln, gemeinsam mit ihren Metadaten, ausgewiesen werden können: Der Kunde erhält einen Kontoauszug, auf dem die Transaktion wie ausgeführt vermerkt ist. Der Beweis im Fehlerfall ist damit jedoch noch nicht möglich.

Um die gleichen Ziele an einem Geldautomaten zu erreichen, benötigen wir äquivalente Lösungen. Da wir dem Kunden nicht zumuten möchten, dass er – parallel zur Bank – eine Berechnung ausführt, verwenden wir statt der TAN einen authentifizierten Kanal zwischen Bank und Kunde. Die Autorisierung des Auftrags kann dann losgelöst von den Auftragsdaten gegenüber der Girocard ablaufen, da wir davon ausgehen, dass der Kunde den Auftrag nur autorisiert, wenn die ihm von der Bank gesendeten Informationen korrekt sind. Eine Verbindung der Autorisierung zur Transaktion ist dennoch erforderlich.

Wir werden den authentifizierten Kanal und die Authentifizierung gegenüber der Girocard zunächst als gegeben annehmen. Wie diese realisiert werden könnten, werden wir im Anschluss diskutieren.

3.3.2. Nachweis im Fehlerfall

Die Möglichkeit für den Kunden, den korrekten Ablauf zu überprüfen, ist nur eine Seite der Medaille. Läuft das Protokoll nicht korrekt ab, muss der Kunde dies einerseits feststellen und andererseits auch beweisen können.

Abweichungen im Protokollablauf müssen *erkennbar* und beweisbar sein.
Der Kunde muss handlungsfähig sein, wenn ein Fehler auftritt.

Wie wir bereits erwähnt haben, ist es dem Angreifer in unserem Modell immer möglich, das Protokoll an einem beliebigen Punkt, an dem Kommunikation zwischen den Komponenten stattfindet, abzubrechen. Der Angreifer kann beispielsweise nach der Autorisierung der Transaktion durch den Kunden das Protokoll unterbrechen. Für den Kunden ist in diesem Moment ohne Weiteres nicht klar, in welchem Zustand sich das Protokoll befindet. Einerseits hätte die Bank die Autorisierung noch erhalten können; andererseits hätte diese schon fehlschlagen können.

Das zeigt, dass wir nicht alle Probleme mit Hilfe von kryptographischen Mitteln lösen können. Wir müssen dem Kunden jedoch ausreichend Informationen zukommen lassen, dass er im Zweifelsfall eine Lösung außerhalb des Protokolls wählen kann: die Polizei rufen. Für diese muss genug Evidenz vorhanden sein, um den bisherigen Verlauf nachvollziehen zu können, ohne dabei anwesend gewesen zu sein.

3.4. Methoden

Neben einem modularen Aufbau des Protokolls bedienen wir uns einfacher Techniken, um Angriffe auszuschließen. Einerseits muss das Einspielen falscher oder alter Nachrichten erkannt werden können. Andererseits muss verhindert werden, dass der Angreifer die Modularisierung des Automaten für seine Zwecke nutzt.

3.4.1. Nonces gegen Preplay und Replay

Alle an der Transaktion beteiligten Instanzen müssen sich mit den Parametern der Transaktion einverstanden erklären. Für die Bank bedeutet das beispielsweise, dass der Kunde über ein ausreichend hohes Guthaben auf seinem Konto verfügt und die Girocard bestätigt, dass sich der Nutzer authentifiziert hat; für die Geldkassette bedeutet das, dass sie über den auszuzahlenden Betrag verfügt und dass die Auszahlungsanweisung von der Bank signiert ist. Ob jedoch die Transaktion an sich ausgeführt werden soll, kann kein Computer entscheiden: Die Transaktion muss vom Kunden autorisiert werden.

Es muss dementsprechend nicht nur sichergestellt werden, dass alle Komponenten aktiv an der Transaktion beteiligt sind (Schutz gegen den Preplay-Angriff), sondern auch, dass Transaktionen (oder Teile der Protokollabläufe) nicht ohne die Einwilligung des Kunden wiederholt werden können (Schutz gegen Replay-Angriffe). Da wir annehmen, dass die vertrauenswürdigen Komponenten (abgesehen von der Bank) nur über wenig Speicher verfügen, in dem sie einen Zustand speichern können, solange sie mit Strom versorgt werden, muss verhindert werden, dass sie erneut eingespielte Nachrichten erneut akzeptieren.

Komponenten ohne zufällige Session-Nonce sind anfällig für Replay-Angriffe.
Jede Komponente wählt zu Beginn des Protokolls eine Nonce. Diese erwartet sie in jeder an sie gesendeten Nachricht.

Um das zu erreichen, wählt sich jede Komponente für jede Transaktion eine Zufallszahl (*Nonce*), die sie für diesen einen Protokollablauf verwendet. Sie erwartet diese Nonce, die sie als erstes an die Bank sendet, nun in jeder Nachricht, die sie von der Bank – der zentralen Stelle der Kommunikation – erhält. Erhält sie, nachdem sie kurzzeitig vom Strom getrennt wurde, eine Nachricht aus einem alten Protokollablauf, wird diese nicht zu ihrer nun neu gewählten Nonce passen: alte Nachrichten werden erkannt und verworfen. Diese (und andere Überprüfungen) sind implizit Teil des Protokolls.

Welche Angriffe möglich werden, wenn sich eine Komponente keine Nonce wählt, werden wir später noch ausführlicher erläutern. Je nach Implementierung des authentifizierten Kanals und der Methode, wie sich der Kunde gegenüber der Girocard authentifiziert, führt das Weglassen der Nonce einer Komponente zu der Möglichkeit für Replay-Angriffe. Nur beim Kunden nicht: Der Kunde hat den eigenen Willen, der die Nonce ersetzt. Er wird, davon gehen wir aus, kein zweites Mal Geld abheben, obwohl er das nicht möchte. Die Nachrichten, die die Komponenten untereinander austauschen, müssen nun so mit den Auftragsdaten und der Autorisierung des Kunden verwoben werden, dass sichergestellt ist, dass nur vom Kunden gewünschte Transaktionen ausgeführt werden.

3.4.2. Schutz vor „Frankenstein-Automaten“

Das Konzept von Frankenstein-Automaten wird durch die Modularisierung der Automaten ermöglicht. Jede Komponente (Modul) des Automaten kommuniziert über einen Kanal, den der Angreifer unter Kontrolle hat. Es ist für den Angreifer daher möglich, einen Automaten aus den Komponenten vieler verschiedener Automaten *zusammenzustellen*. Das einfachste Beispiel ist, dass er alle Komponenten außer der Geldkassette eines Automaten A_1 mit der Geldkassette eines anderen Automaten A_2 verbindet. Dadurch würde das Geld, das der Kunde an Automat A_1 abheben will, an Geldautomat A_2 ausgegeben werden.

Wir stellen zwei Konzepte vor, um den Angriff zu erschweren oder sogar unmöglich zu machen.

Verknüpfung der Komponenten: Basierend auf der bisherigen Beschreibung verfügt ein Geldautomat nur über eine unkorrumpierte Komponente: die Geldkassette. Für die Realisierung des authentifizierten Kanals und der Authentifizierung des Kunden gegenüber der Girocard können noch zwei weitere unkorrumpierbare Komponenten hinzukommen: zum Beispiel ein Drucker und ein PIN-Pad. Die Bank muss wissen, welche Geräte sich in einem Automaten befinden. So kann garantiert werden, dass ein Beleg, den der Drucker bereits vor der Autorisierung zu Teilen gedruckt hat, am gleichen Automaten erscheint wie das auszugebende Geld. Davon ausgehend, dass

der Angreifer nicht nur die Software des Automaten manipuliert, ist der Ausgabeort des Beleges alleine selbstverständlich keine Garantie für den korrekten Standort.

Information des Kunden: Die Auftragsdaten, die der Kunde durch die Autorisierung bestätigt, müssen die Information über den Empfänger enthalten. Im Fall von Geldautomaten ist der Empfänger des *digitalen Geldes* eine Geldkassette. Diese gibt das *physische Geld* (die Geldscheine) aus.

Der Kunde muss, bevor er die Transaktion autorisiert, zweifelsfrei wissen, welche Geldkassette die Bank anweisen wird, Geld auszugeben. Wir sind der Meinung, dass die Geldkassette sowohl durch eine Nummer auf dem Geldautomaten als auch durch die Adresse, an der er steht, und gegebenenfalls weiteren Informationen gekennzeichnet sein muss. In einem mehrstöckigen Kaufhaus mit Automaten auf jeder Etage ist beispielsweise die Information, in welcher Etage der Automat steht, zusätzlich zu der Adresse notwendig.

3.5. Unser Ansatz

Die Schwächen des EMV-Protokolls kennend, haben wir uns Gedanken über Anforderungen an ein neues Protokoll gemacht und darüber, welche Techniken zum Einsatz kommen müssen, um Angriffe, wie sie auf das EMV-Protokoll möglich sind, zu vermeiden und die Nachvollziebarkeit für den Kunden zu gewährleisten. Unserem Ziel, dass ein korrumpiertes Steuersystem des Geldautomaten die Sicherheit nicht beeinflussen kann, Rechnung tragend, setzen wir in den folgenden Beschreibungen das Steuersystem mit dem Angreifer gleich.

3.5.1. Protokoll

Wir schlagen zunächst ein Basisprotokoll vor, das im Anschluss erweitert werden kann. Das Protokoll lässt sich in drei Phasen einteilen. In der ersten Phase (Abbildung 3.1) werden die Auftragsdaten gesammelt und über den Angreifer an die Bank weitergeleitet. In der zweiten Phase (Abbildung 3.2) informiert die Bank den Kunden über den Auftrag, den sie erhalten hat; der Kunde autorisiert den Auftrag, wenn er den Wünschen entspricht. In Phase drei (Abbildung 3.3) wird der Auftrag ausgeführt; die Bank erhält von der Geldkassette eine Bestätigung, dass der Auftrag ausgeführt wurde.

Phase 1: Session-Informationen

Das Protokoll beginnt damit, dass der Kunde eine Verbindung zwischen Girocard und Angreifer herstellt (Einführen der Girocard in das Lesegerät des Automaten) und die Transaktionsdaten an den Angreifer übergibt (Schritt 1). In den Schritten 2 und 3 erzeugen sich die Komponenten Session-Nonces und senden diese gemeinsam mit ihren IDs an den Angreifer. Die Bank wählt und übergibt ihre Session-Nonces in Schritt 5.

Phase I – Session-Informationen

1	Kunde (S)	$\xrightarrow{S, E, B}$	$\mathcal{A}$
2	Girocard (T)	$\xrightarrow{ID_T, N_T}$	$\mathcal{A}$
3	Geldkassette (E)	$\xrightarrow{ID_E, N_E}$	$\mathcal{A}$
4	$\mathcal{A}$	$\xrightarrow{S, E, B, ID_T, N_T, ID_E, N_E}$	Bank
5	Bank	$\xrightarrow{N'_T, N'_E}$	$\mathcal{A}$
6	$\mathcal{A}$	$\xrightarrow{N'_T}$	Girocard
7	$\mathcal{A}$	$\xrightarrow{N'_E}$	Geldkassette

Legende:

T	Girocard (Token)	B	Betrag
S	Kunde (Sender)	ID_x	ID der Komponente x
E	Geldkassette (Empfänger)	N_x	Nonce der Komponente x
$\mathcal{A}$	Angreifer (Geldautomat / Kanal)	N'_x	Nonce der Bank für Komponente x

Abbildung 3.1.: Angreifer erhält die Informationen der Transaktion und verteilt sie an die Beteiligten

Der Angreifer leitet die an ihn gesendeten Daten in den Schritten 4, 6 und 7 an die entsprechenden Empfänger weiter. Damit modellieren wir explizit, dass der Angreifer diese Informationen auch verändern oder deren Auslieferung unterdrücken kann. Die Reihenfolge der Nachrichten und deren genauen Inhalt stellen wir in Abbildung 3.1 dar. Dabei hat die Reihenfolge der Schritte 4 und 5 keine besondere Bewandtnis; die Bank muss die Nonces N'_T und N'_E nicht in Abhängigkeit der in Schritt 4 erhaltenen Informationen wählen. Lediglich die Zuordnung der Werte zu einer Transaktion durch die Bank ist von Bedeutung. Die gewählte Variante spart eine Nachricht an die Bank, dass ein neuer Protokollablauf beginnt. Die Geldkassette und die Girocard geben diese Informationen bei Aktivierung (Stromversorgung) aus.

Wie eingangs schon erwähnt, erlauben wir dem Angreifer, die Auftragsdaten zu manipulieren. Das Protokoll muss gewährleisten, dass die Bank nur Aufträge ausführt, die vom Kunden gewünscht sind. Der Kunde muss demzufolge später die Möglichkeit zur Kontrolle der der Bank bekannten Daten haben.

Die Nonces werden verwendet, um Replay-Angriffe zu verhindern. Sie werden in späteren Nachrichten Teil der signierten Inhalte sein. Wir werden in der Version, die wir skizzieren, nicht jede Nonce benötigen; wir erläutern aber später, wie die hier abstrahierten Protokolle umgesetzt werden können. Dort finden sie Verwendung.

Phase 2: Autorisierung

Im zweiten Teil stellt die Bank sicher, dass die ihr übermittelten Auftragsdaten unverändert sind. Dazu verfügt sie über einen authentifizierten, integritätsgesicherten Kanal zum Kunden. Wir nehmen diesen zunächst als gegeben an.

Die Bank schickt dem Kunden – direkt, nicht über den Angreifer – in Schritt 8 die ihr vorliegenden Auftragsdaten und eine Nonce, die sie mit dem Auftrag verbindet. Diese

Phase II – Autorisierung

8	Bank	$\xrightarrow{\text{S, E, B, } N'_S}$	Kunde
9	Kunde	$\xrightarrow{\text{Autorisierung, } N'_S}$	Girocard
10	Girocard	$\xrightarrow{\sigma_1 := \mathtt{SIG}_{sk}(N_T, N'_T, N'_S)}$	$\mathcal{A}$
11	$\mathcal{A}$	$\xrightarrow{\sigma_1}$	Bank

Kommentar:

Im Protokoll nicht explizit erwähnt sind die Kontrollen, die der Kunde und die Bank ausführen: Der Kunde überprüft, ob Sender, Empfänger und Betrag stimmen. Nur dann sollte er seine Authentifizierungsinformationen und die Nonce der Bank (N'_S) eingeben. Die Bank überprüft die Signatur und ob Girocard und Sender zusammengehören; der auszuzahlende Betrag sollte nicht größer sein als das Guthaben des Kunden.

Abbildung 3.2.: Die Transaktion wird anhand der Auftragsdaten vom Kunden autorisiert.

Nonce (N'_S) ist notwendig, da der Angreifer die Autorisierung des Kunden sonst für einen anderen Auftrag nutzen kann.

Aus dem Protokollabschnitt (Abbildung 3.2) geht im Schritt 9 auch hervor, dass sich der Kunde gegenüber der Girocard und nicht gegenüber der Bank authentifiziert; die Bank verlässt sich auf die sichere Hardware: sie prüft nur die Beteiligung der Girocard des Kunden (Sender S). Die Bank nimmt an der Stelle an, dass die Girocard nur dann eine Signatur berechnet, wenn sich der Nutzer der Girocard gegenüber erfolgreich authentifiziert hat.

Die Nachricht, zu der die Girocard in Schritt 10 eine Signatur erstellt, enthält nicht die Auftragsdaten. Das wäre nur sinnvoll, wenn der Kunde diese der Girocard direkt übergeben würde. Im Vergleich dazu gehen beim Chip-TAN-Verfahren die Auftragsdaten in die Berechnung der TAN mit ein; der Kunde *beobachtet* die Daten auf dem Weg zur Girocard. Wir verknüpfen die Auftragsdaten stattdessen mit einer vom Kunden zusammen mit der Autorisierung an die Girocard übergebenen Nonce, die Teil der Signatur ist, sowohl auf der Seite der Bank als auch auf der Seite des Kunden. Da es dem Angreifer durch den authentifizierten Kanal nicht möglich ist, unterschiedliche Sichten der Bank und des Kunden zu erzeugen, erreichen wir durch die Verwendung der Nonce (N'_S) das gleiche Ziel.

Im Protokoll wird implizit angenommen, dass Bank und Kunde die Korrektheit der Daten, die sie erhalten, prüfen. Für die Bank bedeutet das, dass sie prüft, dass es sich um die Girocard des Senders handelt, dass die Signatur gültig ist und dass die Nonces den erwarteten entsprechen. Dass die Nonces mit den erwarteten übereinstimmen, ist dadurch sichergestellt, dass die Bank die Nonces in diesem Schritt nicht mehr erneut geschickt bekommt, sondern auf die Daten aus Phase I zurückgreifen muss.

Phase III – Auszahlung

12	Bank	$\xrightarrow{\sigma_2 := \mathtt{SIG}_{\mathrm{sk}}(B, \mathrm{ID_E}, \mathrm{N_E}, \mathrm{N'_E}, \text{„auszahlen“}),\ \mathrm{B}}$	$\mathcal{A}$
13	$\mathcal{A}$	$\xrightarrow{\sigma_2,\ \mathrm{B}}$	Geldkassette
14	Geldkassette	$\xrightarrow{\text{Bargeld}}$	Kunde
15	Geldkassette	$\xrightarrow{\sigma_3 := \mathtt{SIG}_{\mathrm{sk}}(\mathrm{N_E}, \mathrm{N'_E}, \text{„Done“})}$	$\mathcal{A}$
16	$\mathcal{A}$	$\xrightarrow{\sigma_3}$	Bank
17	Bank	$\xrightarrow{\sigma_4 := \mathtt{SIG}_{\mathrm{sk}}(\mathrm{N_E}, \mathrm{N'_E}, \text{„finish“})}$	$\mathcal{A}$
18	$\mathcal{A}$	$\xrightarrow{\sigma_4}$	Geldkassette

Kommentar:

Die Geldkassette speichert die Signatur σ_3 und die dazugehörige Nachricht, bis sie die Signatur σ_4 von der Bank erhalten hat. Das führt dazu, dass die Bank – sollte der Angreifer die Auslieferung von σ_3 (oder σ_4) unterbinden – sich die Bestätigung der Auszahlung beim nächsten Auffüllen der Geldkassette mit Bargeld abholen kann. Der Handlungsbedarf liegt in einem Fehlerfall bei der Bank und nicht beim Kunden.

Abbildung 3.3.: Die Transaktion wird durchgeführt.

Phase 3: Auszahlung

Im dritten Abschnitt des Protokolls gibt die Bank die Zahlungsanweisung an die Geldkassette (siehe Abbildung 3.3, Schritt 12). Da die Bank damit rechnen muss, dass die Zahlungsanweisung (σ_2, B) nicht bei der Geldkassette ankommt, kann sie zu diesem Zeitpunkt das Geld noch nicht vom Konto des Kunden abbuchen. Erst wenn sie die Bestätigung der Geldkassette in Schritt 15 erhält (σ_3), kann sie sich dessen sicher sein. Da diese Nachricht ebenfalls vom Angreifer blockiert werden kann, speichert die Geldkassette σ_3. Um möglichst wenig Platz im Speicher der Geldkassette zu benötigen, sendet die Bank in Schritt 17 eine Signatur (σ_4), die bestätigt, dass sie σ_3 erhalten hat. Die Geldkassette kann nach Erhalt von σ_4 die gespeicherte Signatur σ_3 löschen.

Abweichungen im Protokollablauf müssen erkennbar und *beweisbar* sein.

Die Beweislast muss bei der Bank liegen.

Im Fehlerfall kann die Bank die in der Geldkassette gespeicherten Daten vor Ort auslesen und so die Nachrichten am Angreifer vorbei abrufen. Sie muss nun die schon behandelten Signaturen (σ_4 wurde blockiert) von den neuen Signaturen (σ_3 wurde blockiert) trennen.

Annahmen, Methoden, Garantien

Dieses Protokoll dient uns als Basis. Unterschiedliche Instanziierungen des sicheren Kanals von der Bank zum Kunden und der Authentifizierungsmethode sind möglich. Dabei kommt der Girocard als vertrauenswürdiger Hardware eine zentrale Rolle zu: Der Kunde und die Bank verlassen sich darauf, dass ohne Zugriff auf die Girocard, der nur mittels

der Authentifizierung gegenüber der Girocard möglich ist, kein Geld abgehoben werden kann.

Weiterhin muss die Geldkassette als vertrauenswürdig angenommen werden. Das Protokoll ist so entworfen, dass dem Kunden keine Nachweispflicht zukommt, wenn kein Geld ausgegeben wurde. Die Geldkassette erstellt nur nach erfolgreicher Auszahlung die Signatur, auf der basierend die Bank das Geld vom Konto des Kunden abbucht. Zwischen Auszahlung und Erstellung der Signatur kann der Angreifer das Protokoll nicht abbrechen, da keine Kommunikation nach außen vorhanden ist. Erhält die Bank diese Bestätigung nicht, kann sie – den Angreifer umgehend – diese direkt aus der Geldkassette entnehmen.

Technische Bemerkung: Atomare Schritte im Protokoll
Wir nehmen an, dass der Angreifer nur dann, wenn die Komponenten miteinander kommunizieren, das Protokoll unterbrechen kann, nicht aber, wenn die Komponente Berechnungen durchführt. De facto kann der Angreifer jedoch die Ausführung des Protokolls auch durch das Abstellen der Stromversorgung der vom Automaten versorgten Geräte unterbrechen. Insbesondere bei der Geldkassette ist dies entscheidend, da sie die Auszahlung korrekt protokollieren muss. Ein Protokolleintrag ohne Auszahlung ist ebenso zu vermeiden wie eine Auszahlung ohne Protokolleintrag. Damit ein Angreifer der Geldkassette nicht im entscheidenden Moment den Strom abstellen kann, wäre es denkbar, dass die Geldkassette über einen kleinen Stromspeicher verfügt, der die Geldkassette für die Dauer der Auszahlung, Erstellung der Signatur und Speicherung derselben mit Strom versorgen kann. Die Auszahlung des Geldes ist dann an eine zusätzliche Bedingung geknüpft: der Stromspeicher ist voll aufgeladen.

Nonces Jeder Teilnehmer – der Kunde ausgenommen – erstellt für eine Transaktion eine zufällige Nonce. Die Girocard tut dies, wenn sie Strom erhält, die Geldkassette, wenn eine neue Transaktion angestoßen wird. Beide Geräte führen nach Ablauf eines Protokolldurchlaufs keinen zweiten Durchlauf mit der gleichen Nonce aus. Dadurch wird es unmöglich, einen mit einem Gerät angefangenen Protokolldurchlauf mit einem anderen zu beenden oder Nachrichten einer Aufzeichnung eines Protokolldurchlaufs einem Gerät als aktuelle Nachricht unterzuschieben.

Zieht die Geldkassette keine, ist ein Replayangriff auf die Auszahlung möglich, da die Geldkassette – abgesehen von den Informationen für den aktuellen Protokolldurchlauf und den gespeicherten Signaturen – zustandslos ist. Ähnliches gilt für die Girocard: Ohne frischen Zufall ist ein Replayangriff auf die Authentifizierung möglich. Preplay-Angriffe, wie sie in der Literatur beschrieben sind, werden durch die Nonces der Bank unterbunden: Sie lässt sich eine frisch gezogene Nonce von der Girocard signieren; dadurch kann sich die Bank sicher sein, dass die Girocard während der Transaktion online ist.

3.5.2. Der sichere Kanal

Der sichere Kanal von der Bank zum Kunden ist von zentraler Bedeutung für das Protokoll, da sich die Bank durch dessen Verwendung sicher sein kann, dass der Kunde die Daten angezeigt bekommt, von denen die Bank ausgeht, dass der Kunde sie sieht. Wie bereits

erwähnt, ist das Modell etwas anders als das des Chip-TAN-Verfahrens, bei dem der Nutzer die Daten auf dem Weg zur Girocard zu sehen bekommt.

Wir nehmen an, dass ein Drucker, der die Informationen, die der Kunde prüfen soll, ausdruckt, eine geeignete Realisierung ist. Dieser Drucker sollte vertrauenswürdig sein. Da wir dies aber nicht garantieren können, muss die Bank Signaturen mitschicken, die der Drucker mit ausdruckt, so, dass sie der Kunde mit einem Smartphone (oder einem anderen Gerät) prüfen kann.

Dazu nehmen wir weiter an, dass ein Angreifer nicht sowohl den Drucker des Geldautomaten als auch das Smartphone des Kunden korrumpieren kann.

Der Beleg soll den Kunden in die Lage versetzen, jederzeit zu erkennen, in welchem Zustand sich der Protokollablauf befindet, und zu beweisen, wenn ein Fehler auftritt. Die Schwierigkeit besteht darin, dass anhand der von der Bank gesendeten, auf dem Beleg gedruckten Daten nicht erkennbar sein kann, ob ein Angreifer das Protokoll abgebrochen hat oder ob der Kunde (in täuschender Absicht) nur den Teil des Belegs präsentiert, der vor dem scheinbaren Abbruch ausgedruckt wurde. Insbesondere, wenn der Kunde eine dritte, unabhängige Instanz hinzuziehen kann – die Polizei –, ist es ihm möglich, Fehlverhalten des Automaten nachzuweisen, ohne dass der Dritte bis zu diesem Zeitpunkt anwesend war. Der Beleg, der noch nicht abgetrennt im Drucker steckt, zeigt, dass der Beleg vollständig ist.

Mit diesen zwei Techniken – Signaturen und dem Nachweis auf Vollständigkeit des Beleges – und der Annahme, dass das Protokoll nicht beliebig lange Pausen zwischen zwei Ausgaben zulässt, erreichen wir das Ziel, dass der Kunde informiert und handlungsfähig ist. Welche Daten auf den Beleg gedruckt werden, muss sorgfältig gewählt werden, um dem Kunden die Möglichkeit zu nehmen, eigene Angriffe wie Angriffe eines Angreifers aussehen zu lassen und dem Angreifer die Möglichkeit zu nehmen, nicht nachweisbare Manipulationen vorzunehmen.

3.5.3. Die Authentifizierung gegenüber der Girocard

In unserem Protokoll wird angenommen, dass sich der Kunde gegenüber der Girocard authentifiziert. Dafür benötigen wir ein geeignetes Verfahren und eine Umsetzung im Rahmen unseres Protokolls. Wir betrachten zunächst das Verfahren, wie es im EMV-Verfahren eingesetzt wird (PIN-Eingabe) und fügen im Anschluss einen Vorschlag zu dessen Umsetzung im Kontext unseres Protokolls hinzu.

Das EMV-Protokoll nutzt dafür eine (im Allgemeinen vierstellige) PIN, für die der Kunde drei Versuche hat, sie korrekt einzugeben; die Girocard sperrt sich bei mehreren Fehlversuchen in Folge.

Dass die Authentifizierung mittels einer PIN keine gute Lösung ist, zeigen viele Angriffe auf dieses Verfahren. Die PIN kann beispielsweise mit Hilfe einer Kamera oder mit

einer über das PIN-Pad gelegten Maske mitgelesen werden. Besser geeignet wäre ein Zero-Knowledge-Verfahren, dessen Realisierung aber nicht unser Ziel ist.

Wir beschreiben an dieser Stelle, wie ein PIN-Verfahren bei Verwendung eines vertrauenswürdigen PIN-Pads in unseren Protokollvorschlag integriert werden könnte. Wir wählen hierzu beispielhaft eine Realisierung, die nicht auf einer Public-Key-Infrastruktur basiert. Die Girocard und das PIN-Pad teilen sich mit der Bank einen geheimen Schlüssel. Das PIN-Pad kann sich jedoch nicht gegenüber der Girocard authentifizieren.

Technische Bemerkung: Symmetrische oder asymmetrische Verfahren
Statt eines symmetrischen Verfahrens könnte auch ein asymmetrisches Verfahren zum Einsatz kommen, ohne eine Public-Key-Infrastruktur (PKI) verwenden zu müssen. Dazu muss die Bank ebenso wie bei der Verwendung symmetrischer Schlüssel die Zuordnung zwischen Gerät und dessen öffentlichem Schlüssel speichern. Der Speicherbedarf kann durch erhöhten organisatorischen Aufwand reduziert werden, indem eine PKI verwendet wird.

Authentifizierung mit PIN

Die Bank teilt sich mit dem PIN-Pad und der Girocard je einen symmetrischen Schlüssel für Verschlüsselung und einen für die Signatur. Der Einfachheit halber bezeichnen wir sie jeweils mit $\texttt{key}_{\text{pad}}$ bzw. $\texttt{key}_{\text{card}}$. In der ersten Phase hat sich das PIN-Pad ebenso wie die Girocard eine Nonce gewählt, die sie – zusammen mit ihrer ID – dem Angreifer übermittelt hat: N_P. Dieser hat diese (oder eine andere) an die Bank weitergeleitet.

1	Bank	$m := \texttt{ENC}_{\texttt{key}_{\text{pad}}}\left(\texttt{key}_{\text{pin}}\right),\ \sigma_{\text{pad}} := \texttt{SIG}_{\texttt{key}_{\text{pad}}}(m, \text{N}_\text{P})$	$\mathcal{A}$
2	$\mathcal{A}$	$m,\ \sigma_{\text{pad}}$	PIN-Pad
3	Bank	$m := \texttt{ENC}_{\texttt{key}_{\text{card}}}\left(\texttt{key}_{\text{pin}}\right),\ \sigma_{\text{card}} := \texttt{SIG}_{\texttt{key}_{\text{card}}}(m, \text{N}_\text{T})$	$\mathcal{A}$
4	$\mathcal{A}$	$m,\ \sigma_{\text{card}}$	Girocard
5	Kunde	$a_{\text{plain}} := \{PIN, ...\}$	PIN-Pad
6	PIN-Pad	$a_{\text{enc}} := \texttt{ENC}_{\texttt{key}_{\text{pin}}}(a)$	$\mathcal{A}$
7	$\mathcal{A}$	a_{enc}	Girocard

Kommentar:

Sowohl das PIN-Pad als auch die Girocard prüfen die Signaturen. Für eine erfolgreiche Prüfung muss die korrekte Nonce verwendet worden sein.
Das PIN-Pad kann sich nicht gegenüber der Girocard ausweisen. Die Authentifizierung findet implizit über die Kenntnis des korrekten Schlüssels $\texttt{key}_{\text{pin}}$ statt: Nur ein PIN-Pad, das von der Bank als gültig angesehen wird, ist in der Lage, den Schlüssel zu empfangen.

Abbildung 3.4.: Der Kunde authentifiziert sich gegenüber der Girocard mittels Eingabe seiner PIN. Dazu verwenden wir ein vertrauenswürdiges PIN-Pad, das sich – ebenso wie die Girocard – einen Schlüssel mit der Bank teilt.

Die Abbildung 3.4 zeigt den Ablauf des Protokolls. Beide Parteien erhalten einen von der Bank frisch gezogenen zufälligen symmetrischen Schlüssel (Schritte 1 bis 4). Dieser ist zusammen mit der Nonce, die sich PIN-Pad und Girocard zu Beginn des gesamten Protokolls ziehen und an die Bank übermitteln, von der Bank signiert. Ein Replayangriff ist nicht möglich, da das PIN-Pad und die Girocard einen (frischen) Schlüssel erwarten, der mit ihrer frisch gezogenen Nonce verknüpft ist. Der Kunde gibt in Schritt 5 über das PIN-Pad seine PIN und alle anderen für die Authentifizierung und Autorisierung notwendigen Informationen ein. Das PIN-Pad verschlüsselt in Schritt 6 die Daten mit dem Schlüssel, den es in Schritt 2 erhalten hat.

3.5.4. Naheliegende Modifikationen

Das hier vorgestellte Protokoll ist möglichst knapp gehalten, um das Konzept leichter verständlich zu machen. Für einen realen Einsatz muss das Protokoll um einige Standardschutzmaßnahmen erweitert werden. Die erste und offensichtlichste ist die Verschlüsselung der gesamten Kommunikation. Wir haben uns bei dem Entwurf des Protokolls bewusst dafür entschieden, die Informationen auf dem Weg zur Bank nicht zu verschlüsseln und nur die entscheidenden Nachrichten zu signieren. Das zeigt, dass ein Angreifer selbst mit dem Wissen über Transaktion und den gewählten Parametern die Transaktion nicht verändern kann. Jedoch lernt er die Transaktionsdaten. Das kann verhindert werden, indem die Daten verschlüsselt übertragen werden.

Des weiteren können Ansätze betrachtet werden, die durch eine (vertrauenswürdige) Ein- und/oder Ausgabe an der Girocard möglich werden. Über ein einfaches Sieben-Segment-Display mit wenigen Stellen könnte die Girocard dem Kunden ähnlich dem Chip-TAN-Verfahren mitteilen, welche Transaktion er autorisieren würde. Auch für die Authentifikationsmethode eröffnen sich dadurch neue Möglichkeiten.

Eine dritte Stelle, an der eine weitere Aufteilung der Komponenten sinnvoll erscheint, ist die Geldkassette: Ein unabhängiges System sollte prüfen können, ob die geforderte Auszahlung tatsächlich durchgeführt wurde. Das Ergebnis, zu dem dieser Baustein kommt, muss in den Protokollablauf eingeflochten werden. Das Konto des Kunden kann nur um den tatsächlich ausgezahlten Betrag belastet werden. Zusätzlich könnte überprüft werden, ob der Automat frei von einigen *bekannten* Manipulationen ist, wie beispielsweise dem Blockieren der Klappe, die das Geld freigibt.

3.6. Weitere Forschungsansätze

Für dieses Protokoll nehmen wir die Bank als vertrauenswürdige Instanz an. Ziel einer weiteren Arbeit könnte es sein, ein Protokoll zu entwerfen, bei dem lediglich eine Bank aus einem Netzwerk vertrauenswürdig sein muss, obwohl nur die Bank des Kunden die genauen Transaktionsinformationen erhält. Weiter könnte das Kundenverhalten – wann wird wo wie viel Geld abgehoben – vor der Bank verborgen werden. All diese Modifikationen stellen das Vertrauensverhältnis Kunde zu Bank in Frage; ob in einem solchen Szenario die Girocard noch als vertrauenswürdig angenommen werden kann, ist anzuzweifeln.

3.7. Fazit

Mit Hilfe einer vertrauenswürdigen Komponente für den Kunden, deren Funktionsumfang dem einer Girocard entspricht, kann ein sicheres Protokoll zum Abheben von Geld an Automaten umgesetzt werden. Die anfangs in diesem Kapitel geforderten Eigenschaften konnten wir durch die folgenden einfachen Konzepte erfüllen.

1. Als vertrauenswürdig angenommene Hardware führt nur einfache Operationen innerhalb des Protokolls aus.
2. Der Protokollfortschritt wird für den Kunden und Dritte nachvollziehbar protokolliert.
3. Alle Komponenten wählen ihre eigene Session-Nonce.

Mit Ausnahme der Bank verfügen alle Komponenten, die wir in unserem Protokoll als vertrauenswürdig angenommen haben, über einen begrenzten Funktionsumfang, der, verglichen mit dem eines komplexen Betriebssystems, einfach zu realisieren ist. Das ermöglicht ein vergleichsweise einfaches Protokoll zur Realisierung des Geldabhebens, das den Kunden im Fehlerfall durch gezielte Information über den aktuellen Status des Protokolls handlungsfähig macht. Ohne vertrauenswürdige Hardware würden die Protokolle, um die gleiche Sicherheit zu gewährleisten, ungleich komplexer. Dennoch können viele, schwierig zu kontrollierende Bausteine wie das Netzwerk oder der Steuerungsrechner des Geldautomaten als korrumpiert angenommen werden. Durch Session-Nonces können wir verhindern, dass der begrenzte Überblick der vertrauenswürdigen Hardware für Pre- oder Replay-Angriffe ausgenutzt wird.

4. Softwareschutz Blurry-Box

Bei dem Beispiel des Geldabhebens war die Person, deren Rechte wir schützen wollen, auch die Person, die die sichere Hardware bedient (beziehungsweise verwendet). Wir betrachten nun mit dem Kopierschutz von Software ein Szenario, bei dem das nicht mehr zutrifft. Die (Soft- und) Hardware wird zwar vom Rechteinhaber hergestellt, aber vom Angreifer betrieben, und ist damit außerhalb der Kontrolle des Rechteinhabers. Dennoch soll sie seine Interessen schützen.

Das Interesse des Rechteinhabers – dem Softwarehersteller – ist der Schutz vor nicht lizenzierter Verwendung seines Know-Hows. Anders ausgedrückt: Einem Angreifer soll es nicht möglich sein, ein gleichwertiges Produkt anbieten zu können, ohne den Aufwand betreiben zu müssen, sich das Know-How anzueignen.

Mit allgemeinen Mehrparteienberechnungen und Obfuscation gibt es zwar mindestens zwei Werkzeuge, mit denen dieses Ziel – gegebenenfalls im Zusammenspiel mit einer unkorrumpierbaren Hardware – erreichbar ist, die jedoch für den praktischen Einsatz einen zu hohen Aufwand bei der Ausführung haben. Wie bei dem bisher betrachteten Szenario, in dem die Verwendung des Systems für den Anwender (Bankkunde) so effizient wie möglich gemacht werden muss, damit es angenommen wird, muss die effiziente Ausführung des geschützten Programmes für einen Kopierschutz gewährleistet sein.

Unser Ziel ist es, die Ausführung der Software an eine Hardware (ein *Dongle*) zu knüpfen, die der Angreifer nicht kopieren kann. Sie enthält beispielsweise zur Ausführung des Programms notwendiges Schlüsselmaterial, das der Angreifer nicht extrahieren kann. Die Kosten und die Zeit, die notwendig sind, um die Software losgelöst vom Dongle betreiben zu können, sollen dabei so groß sein, dass sie den Gewinn, den der Angreifer mit dem Verkauf des kopierten Programms erzielen kann, übersteigt oder er die Software selbst hätte schreiben können. Ab diesem Punkt ist es für einen Angreifer wirtschaftlich nicht sinnvoll, den Schutz einer einzelnen Software anzugreifen. Sinkt der Aufwand für Angriffe auf weitere, mit demselben Verfahren geschützte Programme, müssen die Gesamtkosten und der Gesamtgewinn aller Angriffe in Betracht gezogen werden.

4.1. Vertrauen in die Hardware

Wie bereits im ersten Beispiel stützen wir uns in diesem Szenario darauf, dass die Person, deren Gut geschützt werden soll, der Hardware vertraut. Wir nennen diese Person auch in diesem Szenario den *Inhaber*. Das Gut, das der Inhaber in diesem Szenario schützen

will, ist sowohl das Ausführungsrecht seiner Software als auch das Know-How, das zur Erstellung der Software notwendig ist; kann ein Angreifer das Know-How in Erfahrung bringen, kann er eine eigene Software schreiben, die funktionsgleich mit der geschützten Software ist und damit den Kopierschutz indirekt aushebeln. *Vertrauen* muss der Inhaber, dass das Dongle wie erwartet arbeitet und ein Angreifer nicht an die Geheimnisse des Dongles gelangen kann.

Anders als bei der Girocard ist der Inhaber nicht der Nutzer der Hardware; der Nutzer ist ein potentieller Angreifer, der sich auf legalem Weg eine Lizenz kauft und versucht, durch das Betreiben der Software an eine ungeschützte Version zu gelangen.

Der Inhaber vertraut jedoch dem Hersteller des Dongles; würde er das nicht, investierte der Inhaber auch nicht in das Schutzsystem, von dem das Dongle ein Teil ist.

Der Inhaber des zu schützenden Guts ...
... ist *nicht* Nutzer der vertrauenswürdigen Hardware.
... vertraut dem Hersteller der Hardware.

Zusammenfassend kann gesagt werden, dass der Inhaber nur dem Hersteller des Dongles vertraut. In diesem Fall kann das Dongle beliebig komplex gebaut werden. Es muss Angriffen standhalten können.

Wir werden nicht auf die Konstruktion eines Dongles eingehen. Wir nehmen stattdessen ein *sicheres* Dongle an und bauen darauf basierend einen Softwareschutz auf, der einen nachvollziehbaren Schutz gewährleistet. *Sicher* heißt in diesem Kontext, dass es einem Angreifer nicht möglich ist, die geheimen Daten, wie beispielsweise Schlüsselmaterial, aus dem Dongle auszulesen.

4.2. Verwandte Arbeiten und Ansätze

Die Funktionsprinzipien von Softwareschutzverfahren, wie sie bisher eingesetzt werden, unterliegen der Geheimhaltung. Firmen, die solche Systeme entwickeln, wissen um deren Verletzlichkeit bei Bekanntwerden der Mechanismen.

Einerseits macht die Geheimhaltung der Verfahren es Interessenten des Schutzes unmöglich, sich von der Qualität des Produktes zu überzeugen. Andererseits kann der Schutz – ist das Funktionsprinzip bekannt – von Produkten schneller entfernt werden.

Im Gegensatz zu den kommerziellen Schutzverfahren stehen akademische Ansätze: zum Beispiel funktionale Verschlüsselung für alle polynomialgroßen Schaltungen [56], abstreitbare Verschlüsselung [105], nicht-interaktiver Mehrparteien-Schlüsselaustausch ohne vertrauenswürdiges Setup [25]. Sie können oftmals Garantien bieten, sind aber im Allgemeinen

nicht effizient genug für den praktischen Einsatz [64]. Die White-Box-Kryptographie stellt diesbezüglich in gewisser Hinsicht eine Ausnahme dar. Für bestimmte Anwendungen gibt es effiziente Ansätze. Für unseren Anwendungsfall ist uns jedoch keine Lösung bekannt. Der Black-Box-Ansatz, bei dem der Angreifer keinen Zugriff auf den die Ausgabe berechnenden Code erhält, steht hingegen für das anzustrebende Ideal.

4.2.1. White-Box-Kryptographie

Das Ziel von White-Box-Kryptographie ist es, den damit geschützten Algorithmus oder Daten, die er enthält, gegen Reverse-Engeneering-Techniken zu schützen. Ein prominentes Beispiel, für das White-Box-Kryptographie zum Einsatz kommen kann, ist die Nutzung eines symmetrischen Verschlüsselungsverfahrens als asymmetrisches Verschlüsselungsverfahren. Dabei wird der symmetrische Schlüssel sowohl in den Verschlüsselungs- als auch in den Entschlüsselungsalgorithmus fest einprogrammiert [38, 16]. Aus den mit White-Box-Kryptographie geschützten Algorithmen kann – so die Annahme – ein Angreifer den Schlüssel nicht extrahieren. Ist einem Nutzer nur der (geschützte) Verschlüsselungsalgorithmus bekannt, kann er – wie bei asymmetrischen Verschlüsselungsverfahren mit dem öffentlichen Schlüssel – Chiffrate erstellen, aber nicht entschlüsseln.

Der Angreifer bekommt Zugriff auf den geschützten Algorithmus und kann ihn selbst ausführen. Dennoch soll es ihm nicht möglich sein, nachvollziehen zu können, wie der Algorithmus funktioniert. Die Technik alleine schützt das Programm noch nicht davor, ohne Genehmigung des Lizenzausstellers ausgeführt zu werden. Diese Technik könnte verwendet werden, um zu verhindern, dass die Kommunikation mit einem Dongle vom Angreifer aus dem Code entfernt werden kann. Es gibt jedoch nur für spezielle Anwendungsfälle Vorschläge, wie das Ziel der White-Box-Kryptographie erreicht werden kann [72].

4.2.2. Black-Box versus Copy&Paste

Bei Black-Box-Verfahren erhält der Angreifer im Vergleich zur White-Box-Kryptographie keinen Zugriff auf die Routinen, die zur Berechnung der Ausgabe zu einer Eingabe ausgeführt werden. Der Angreifer erhält zu von ihm gewählten Eingaben lediglich die dazu passenden Ausgaben. Ein Tabellieren der Ein- und Ausgabepaare kann nicht verhindert werden. Daher ist für Programme mit kleinem Eingabebereich ein vollständiges Tabellieren möglich. Mit dieser Tabelle, die die Ausgaben zu allen möglichen Eingaben enthält, kann eine vom Kopierschutz befreite Version der Software hergestellt werden. Der Black-Box-Begriff stellt das Optimum des zu erreichenden Schutzes dar und dient uns daher als Referenz.

Im Weiteren werden wir einen ähnlichen, aber etwas stärkeren Angreifer betrachten: den Copy&Paste-Angreifer. Er erhält den Code, der für die Berechnung des Ergebnisses verwendet wurde. Verfügt der Angreifer anders als in den unter anderem von Goldreich beschriebenen Modellen *oblivious RAM* [61, 62, 97, 45, 67, 24, 92, 114, 109] oder *physically*

shielded CPU [61, 92, 62] über die Kontrolle über die Maschine, auf der der Code ausgeführt wird, kann das ebenfalls nicht verhindert werden. Eine neue Technik, die vor kurzem in handelsüblichen CPUs (der Firma Intel) Einzug gehalten hat, ist SGX. Sie birgt das Potential, einzelne Aufgaben, die bisher von einem externen, beispielsweise per USB angeschlossenen Dongle ausgeführt wurden, zu übernehmen oder das Dongle möglicherweise ganz zu ersetzen. Noch ist diese Technik in der breiten Masse der bei den Kunden stehenden PCs nicht verfügbar.

Eine vollständige Abdeckung aller Codestücke ist für kleine Programme denkbar. Für große und komplexe Programme stellt das selbst Entwickler, die die Anwendung kennen, bei der Erstellung der Testfälle für die Qualitätskontrolle vor eine große Herausforderung.

4.2.3. Hardware-Token

Neben den bereits beschriebenen Techniken mit Hardwareunterstützung gibt es auch Obfuskationstechniken, die auf unmanipulierbare Hardware-Token aufbauen. Hardware-Token, wie sie unter anderem von Katz [76] beschrieben werden, erlauben Black-Box-Obfuskation. Goldwasser gibt eine allgemeine Konstruktion für zustandsbehaftete Token an [65], die auf One-Time-Programmen aufbaut. One-Time-Programme sind Programme, die nur eine einzige Ausführung zulassen [71, 80, 26].

Leider betrachten diese Konstruktionen die Programme (beziehungsweise deren Funktionalität) als Schaltkreise, die Gatter für Gatter evaluiert werden. Dabei finden im Allgemeinen multilineare Abbildungen [56], vollhomomorphe Verschlüsselung [17, 46] oder PCP-Beweise [47] Anwendung. Diese Ansätze sind für den von uns anvisierten Einsatzzweck nicht effizient genug: Für den Einsatz in der Industrie ist lediglich eine geringfügige Verlangsamung vertretbar. Die Ausführung muss schnell genug sein, um mit den Maschinen, die sie steuert, oder mit den Menschen, von denen sie bedient wird, mitzuhalten.

Wir werden in unserer Lösung ebenfalls ein Hardware-Token verwenden. Jedoch verfügt es nur über begrenzte Ressourcen; die Hauptlast trägt während der Ausführung ein korrumpierbarer PC.

4.3. Blurry-Box

Mit Blurry-Box schlagen wir ein Konzept vor, das einen höheren Schutz als bekannte, in der Industrie eingesetzte Verfahren bietet, obwohl die Methodik bei unserem Verfahren offengelegt werden kann, ohne den Schutz dadurch zu schwächen. Wir orientieren uns an einem der Grundsätze von Kerckhoffs: Das Verfahren – in unserem Fall die Konzepte – müssen für eine Qualitätskontrolle öffentlich bekannt sein.

Das Verfahren basiert auf dem von der Firma Wibu-Systems eingesetzten Softwareschutz. Welche Techniken dabei bereits verwendet werden, erläutern wir im folgenden. Die zusätzlichen Techniken basieren auf einer Idee von Herrn Müller-Quade. In Zusammenarbeit mit Matthias Huber und Brandon Broadnax habe ich das Verfahren weiter ausgebaut. Der Beweis der Sicherheit des Verfahrens wurde von Brandon Broadnax erstellt. Konzepte zur speicher- und recheneffizienten Umsetzung stammen von mir. Veröffentlicht wurden die Ergebnisse in dem Papier *Towards Efficient Software Protection Obeying Kerckhoffs's Principle using Tamper-proof Hardware* [4].

4.3.1. Ziele

Der Schutz einer Software in unserem Sinne soll einen Angreifer daran hindern, Kopien der Software zu betreiben oder zu verkaufen, an denen der Hersteller keine Lizenzgebühren verdient. Im Unterschied zu DRM-Verfahren geht es nicht darum, die Nutzung der legal erworbenen Software(lizenz) einzuschränken. Dazu soll die Ausführung an den Zugriff auf ein Dongle gekoppelt sein, das beispielsweise über die USB-Schnittstelle mit dem PC, auf dem die Software ausgeführt wird, verbunden ist.

Der Schutz zielt nicht darauf ab, das Kopieren der Software unmöglich zu machen; vielmehr soll der Aufwand nur soweit erhöht werden, dass es sich für einen Angreifer wirtschaftlich nicht mehr lohnt. Das kann einerseits dadurch erreicht werden, dass die Zeit oder die Kosten, die für das Entfernen des Schutzes notwendig sind, in die Höhe getrieben werden, oder dadurch, dass der Angreifer zu viel eigenes Know-How aufbringen muss, um eine funktionsfähige Kopie zu erstellen. Beschränkt sind die Ziele durch die Sicherheit der Hardware; auch diese ist nicht uneingeschränkt sicher. Mit ausreichend finanziellen Mitteln können die Geheimnisse des Dongles ausgelesen werden. Die Geheimnisse – kryptographisches Schlüsselmaterial – ermöglichen das Aufheben des Schutzes.

4.3.2. Grundlegende Techniken

Blurry-Box baut auf dem Softwareschutzverfahren der Firma Wibu-Systems auf. Im Folgenden werden die grundlegenden Methoden des Schutzverfahrens zusammengefasst.

Die geschützte Software wird verschlüsselt ausgeliefert. Dazu wird die Software in einzelne Abschnitte unterteilt. Diese werden einzeln verschlüsselt und während der Ausführung der Software nur bei Bedarf entschlüsselt. Um die Beschreibung einfach zu halten, nehmen wir an, dass das Dongle die Blöcke entschlüsselt. Der geheime Schlüssel ist im Dongle gespeichert und verlässt das Dongle nicht. Zur Entschlüsselung sendet die Software den Block, der als nächstes ausgeführt werden soll, an das Dongle und erhält den entschlüsselten Code zurück. Der Code enthält die IDs der Blöcke, die – in Abhängigkeit des Zustandes – als nächstes angesprungen werden können. Der Zustand umfasst beispielsweise Variablenbelegungen oder Eingabewerte.

Da das Dongle alle Anfragen beantwortet, müssen – um zu verhindern, dass der Angreifer das Dongle als Entschlüsselungsorakel verwendet – Fallen eingefügt werden, die bei Entschlüsselung das Dongle sperren. Die Fallen werden bei (korrekter) Ausführung des Programmes nie angesprungen. Für den Angreifer sind die Chiffrate der Fallen nicht von den Chiffraten der korrekten Code-Blöcke zu unterscheiden.

Diese Technik funktioniert nur dann, wenn in dem entschlüsselten Code nicht nur auf die korrekten Blöcke verwiesen wird, sondern auch auf die Fallen; der Angreifer darf aus dem entschlüsselten Code nicht zwischen Verweisen auf Fallen und gültige Codeblöcke unterscheiden können. Ist er dazu in der Lage, kann er die Fallen umgehen und die Software entschlüsseln, ohne Gefahr zu laufen, beim Dongle die Entschlüsselung einer Falle anzufragen. Dieses Problem löst Blurry-Box, indem die Berechnung des nächsten auszuführenden Blockes im Dongle stattfindet.

4.3.3. Angreifermodell

Das Ziel des Softwareschutzes ist es, dass keine Methode verbleibt, eine ungeschützte Version der Software zu erzeugen, die ohne das Hardware-Token funktionsfähig ist. Dieses Ziel ist im Allgemeinen nicht zu erreichen. Dem Angreifer bleibt – selbst bei Black-Box-Sicherheit – immer die Möglichkeit, eine Lizenz der Software zu erwerben und alle Ausgabewerte für alle Eingabewerte zu berechnen. Mit der daraus entstandenen Tabelle ist er in der Lage, eine – wenngleich auch viel größere – Software zu erstellen, die funktional identisch ist.

Im praktischen Einsatz muss ein Softwareschutz nur so gut sein, dass es sich für einen Angreifer finanziell nicht lohnt, ihn zu brechen. In dem Moment, in dem die Kosten für das Brechen des Schutzes die Kosten einer Neuentwicklung der Software übersteigen, wird es für einen Angreifer uninteressant, den Schutz anzugreifen. Kann er jedoch die Erfahrung aus einem erfolgreichen Hack nutzen, um eine zweite Software effizienter anzugreifen, verschiebt sich das Kosten-Nutzen-Verhältnis wieder. Als zweites Ziel kann daher formuliert werden, dass es für den Angreifer nicht leichter sein soll, die Software B vom Kopierschutz zu befreien, wenn er zuvor den Schutz der Software A entfernt hat.

Im Folgenden beschreiben wir Angriffe, die wir systembedingt nicht ausschließen können, jedoch zu ineffizient sind. Für alle anderen Angriffe kann gezeigt werden, dass sie im Allgemeinen nicht effizienter sind.

Statische Angreifer

Der erste Angreifer, den wir betrachten, versucht das Dongle als Entschlüsselungsorakel zu nutzen und sendet Block für Block zur Entschlüsselung an das Dongle. Er erhält bis zur ersten Falle (siehe Abschnitt 4.3.2) den entschlüsselten Code des angefragten Blocks. Die Sicherheit, wie viele Code-Blöcke der Angreifer entschlüsseln kann, hängt somit vom Verhältnis der Fallen zu realen Code-Blöcken ab. Das Vorgehen des statischen Angreifers

ist unabhängig von dem entschlüsselten Code. Der Angreifer analysiert ihn nicht und führt ihn nicht aus. Das bedeutet auch, dass der Angreifer weder einen Debugger verwenden noch Einfluss auf die Variablenbelegungen während der Ausführung nehmen kann.

Der entsprechende reale Angreifer erhält mit jeder Entschlüsselung einen Code-Block. Findet er alle Code-Blöcke, ist er in der Lage, das Programm zusammenzusetzen, und erhält damit eine funktionsfähige Kopie. Je größer die Zahl der Code-Blöcke ist, desto besser ist der Schutz.

Eine der Techniken des Blurry-Box-Verfahrens hat das zum Ziel: Sie erhöht die Zahl der Code-Blöcke durch eine Reduzierung des in einem Block enthaltenen Codes. Dies geschieht nicht durch eine Verkürzung eines Code-Blocks; die Codeblöcke werden auf einen eingeschränkten Eingabebereich spezialisiert. Einen spezialisierten Block nennen wir Variante. Für jeden Eingabebereich wird eine Variante erstellt. Die Zahl der Entschlüsselungsaufrufe bleibt für einen Programmdurchlauf dadurch konstant. Detaillierter wird die Technik in Abschnitt 4.3.5 beschrieben.

Dynamischer Angreifer

Im Vergleich zu dem statischen Angreifer betrachtet der dynamische Angreifer den entschlüsselten Code und passt seine Strategie an die Antwort des Entschlüsselungsorakels an. Die in Abschnitt 4.3.2 beschriebene Vorgehensweise zugrunde legend, wäre ein dynamischer Angreifer dann mit einer einfachen Strategie erfolgreich, wenn im Code nie auf die IDs der Fallen verwiesen wird. Das Konzept beruht darauf, dass der Angreifer – gegeben eine Sprungbedingung – nicht entscheiden kann, ob ein Pfad in einer realen Ausführung nie gewählt werden würde. Nutzt der Angreifer das Programm wie ein normaler Nutzer, werden Fallen nie angesprungen. Es ist jedoch auch mit Kenntnis des Programmcodes sehr schwierig, Eingabedaten für eine vollständige Codeüberdeckung zu finden.

Blurry-Box nutzt die Variantenbildung und *Code-Moving*, um dem Angreifer diesen Vorteil zu nehmen. In Abschnitt 4.3.5 werden die Techniken beschrieben.

4.3.4. Grundlagen für den Schutz

Unser Schutzkonzept basiert auf drei Annahmen: der Komplexität der Software, einem *sicheren Hardware-Token* und der Sicherheit des eingesetzten symmetrischen Verschlüsselungsverfahrens. Die Komplexität der Software steht stellvertretend für die Fachlichkeitsannahme: dem Angreifer fehlt das Domänenwissen aus dem Gebiet, aus der die Software stammt. Das hindert ihn daran, den Softwareschutz zu umgehen, indem er die Anwendung oder relevante Teile davon neu schreibt. Das Token (oder Dongle) verfügt über eine geringe Menge Speicher, auf den der Angreifer nicht zugreifen kann, sowie eine (leistungsschwache) CPU, deren Operationen vom Angreifer ebenfalls nicht überwacht oder beeinflusst werden können.

Auch das vollständige Tabellieren der Ergebnisse aller Eingabewerte schließen wir dadurch implizit aus; einerseits ist der Eingabebereich zu umfangreich, und andererseits kann der Angreifer nicht beurteilen, welche Ergebnisse korrekt sind. Es steht zwar außer Frage, dass die Ergebnisse, die die Originalsoftware ausgibt, korrekt sind. Der Angreifer kann jedoch nicht aus (selbst gewählten) Eingabe-Ausgabe-Paaren weitere Paare berechnen, ohne die Software zu verwenden.

4.3.5. Techniken

In diesem Abschnitt betrachten wir Schritt für Schritt, welche Modifikationen des Programms den Blurry-Box-Schutz ausmachen.

Wir beginnen mit der Verschlüsselung des Programms (siehe Abschnitt *Verschlüsselung*). Um zu verhindern, dass der Angreifer mit einer Entschlüsselung das ganze Programm ungeschützt erhält, wird das Programm in einzelne, logische Abschnitte unterteilt (beschrieben in *Aufteilung des Programms*). Die Abschnitte werden noch weiter untergliedert (*Variantenbildung*) und im Anschluss alle dadurch entstandenen Teile einzeln verschlüsselt (*Verschlüsselung der Varianten*) Die Varianten werden nur dann entschlüsselt, wenn diese für die Programmausführung notwendig sind.

Damit ein Angreifer nicht wahllos Chiffrate entschlüsseln kann, bis er das gesamte Programm entschlüsselt vorliegen hat, werden *Fallen* eingefügt; dieser Vorgang wird im Abschnitt *Einfügen der Fallen* beschrieben. Um zu erreichen, dass nur die Varianten entschlüsselt werden, die für den Programmablauf notwendig sind, verwenden wir *Code-Moving* und den *Zustandsspeicher* des Dongles; siehe dazu die Abschnitte *Code-Moving – Variantenauswahl im Dongle* und *Zustandsspeicher*.

Verschlüsselung Die Software wird verschlüsselt ausgeliefert und ist auf der unsicheren Hardware (dem PC) gespeichert. Dazu wird ein (IND-CPA-sicheres) symmetrisches Verschlüsselungsverfahren verwendet. Der Schlüssel ist – geschützt mit dem geheimen Schlüssel des Dongles – ebenfalls auf dem PC des potentiellen Angreifers abgelegt. Zur Ausführung sendet der Angreifer das Chiffrat, das den Schlüssel enthält, an das Dongle. Dieses entschlüsselt den Schlüssel und sendet ihn zurück an den PC. Dort kann nun auf der (im Vergleich zum Dongle) schnellen Hardware der Code entschlüsselt werden.

Da auf diese Art und Weise ein Angreifer mit einer gültigen Lizenz an das komplett entschlüsselte Programm gelangt, sind weitere Maßnahmen notwendig. Die Art und Weise, wie das Programm verschlüsselt wird, wird dementsprechend angepasst: statt das Programm mit einem einzelnen Schlüssel zu verschlüsseln, wird das Programm in Blöcke unterteilt und jeder Block mit einem eigenen Schlüssel verschlüsselt.

Aufteilung des Programms Das zu schützende Programm wird in einzelne Codeabschnitte (Blöcke) unterteilt. Diese Technik verwendet die Firma Wibu-Systems bereits im Zusammenspiel mit der Verschlüsselung und dem Einfügen von Fallen in ihrem aktuellen Produkt (siehe Abbildung 4.1). Codezeilen, die stets zusammen ausgeführt werden, bleiben in einem Code-Block. Welcher Block als nächstes ausgeführt wird, ist abhängig von der Variablenbelegung – dem Zustand des Programms. Die Funktion, die berechnet, welcher Block als nächstes ausgeführt wird, enthält alle Sprünge aus dem Block heraus; wir nennen sie die *Sprungauswahlfunktion*. Abgesehen von der Information, in welchem Block sich ein bestimmtes Stück Code befindet (beispielsweise eine Methode), sind alle für die Sprungauswahlfunktion notwendigen Informationen bereits im Originalcode enthalten. Es handelt sich dabei um Funktionsaufrufe, Sprunganweisungen oder ähnliche Konstrukte zur Ablaufsteuerung eines Programms.

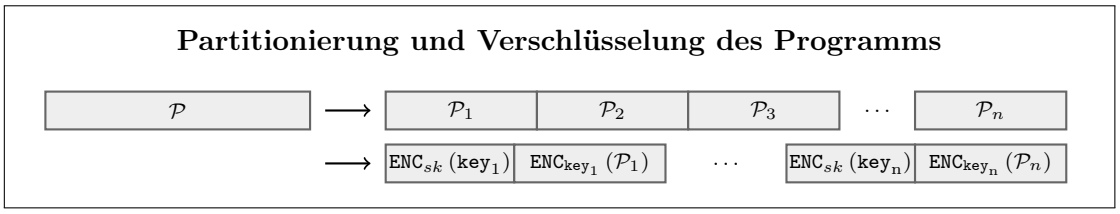

Abbildung 4.1.: Das Programm $\mathcal{P}$ wird in n Blöcke partitioniert. Jeder Block wird separat verschlüsselt.

Variantenbildung Gegeben ein Code-Abschnitt, der einen oder mehrere Eingabeparameter erhält und sich anhand derer in unterschiedliche Subroutinen verzweigt, erstellen wir Varianten. Auf die Verzweigung, die auch künstlich herbeigeführt werden kann, gehen wir später ein. Zunächst gehen wir davon aus, dass der Eingabeparameter einen Wert enthält, anhand dessen im späteren Verlauf eine von mehreren Codestellen ausgeführt wird. Die Auswahl kann durch ein If-Statement, eine Switch-Case-Anweisung oder ähnliche Sprungbedingungen getroffen werden.

Nehmen wir für ein einfaches Beispiel an, dass der Abschnitt mit einem Parameter p vom Typ *Boolean* aufgerufen wird und je nach Belegung im Verlauf des Abschnitts mit Hilfe eines If-Statements ein Codeblock A oder B ausgeführt wird. (Wir nehmen hier implizit an, dass p nach dem Aufruf nicht mehr verändert wird.) Das Beispiel ist als Pseudocode in Algorithmus 1 dargestellt.

Algorithmus 1 Beispiel-Code für die Zerlegung in zwei Varianten

```
1: some code;
2: if p then
3:     Code A;
4: else
5:     Code B;
6: end if
7: some other code;
```

Für die Variantenerstellung wird der Codeabschnitt dupliziert; die If-Else-Anweisung wird an dieser Stelle nicht mehr benötigt. In der ersten Kopie verbleibt der If-Teil A im Code; der Else-Teil wird entfernt. In der zweiten Kopie verbleibt der Else-Teil B im Code und der If-Teil wird entfernt. Das Ergebnis stellt Algorithmus 2 dar. Statt von Kopien des Codes sprechen wir nun von Varianten.

Algorithmus 2 Zwei Varianten des Algorithmus 1

Variante 1:	Variante 2:
1: some code;	1: some code;
2: code A;	2: code B;
3: some other code;	3: some other code;

Welche Variante gewählt wird, hängt von der Belegung des Parameters ab und wird vor Eintritt in den Abschnitt entschieden. Dafür wird ein Wrapper erstellt (siehe Algorithmus 3 und Abbildung 4.2), der anhand der Parameter die richtige Variante auswählt.

Algorithmus 3 Wrapper für Variantenauswahl

```
1: if p then
2:     Variante 1;
3: else
4:     Variante 2;
5: end if
```

Die Wrapper enthalten zusammen mit den Sprungauswahlfunktionen alle Informationen über die Gültigkeitsbereiche der einzelnen Varianten.

Varianten können auch anhand anderer Kriterien erstellt werden als der Programmflusssteuerung. Mathematische Funktionen, die abschnittsweise approximiert werden, sind ein Beispiel dafür.

Verschlüsselung der Varianten Die Varianten werden einzeln verschlüsselt auf der unsicheren Hardware (dem PC) abgelegt. Jede Variante ist mit einem eigenen, zufällig gewählten Schlüssel verschlüsselt. Der Schlüssel ist der Variante mit dem geheimen Schlüssel des Dongles verschlüsselt angehängt.

Welches Verschlüsselungsverfahren zum Einsatz kommt, ist für die Anwendung nicht wichtig, solange sie die notwendigen Sicherheitsanforderungen erfüllt. Die Sicherheit des Verfahrens ist jedoch eine der drei Säulen, auf die sich das Blurry-Box-Prinzip stützt. Kann der Angreifer ohne Kenntnis des Schlüssels an den Klartext der Chiffrate gelangen, kann er den Schutz ohne Weiteres aufheben und eine ohne das Dongle lauffähige Kopie erstellen. Zu dem Zeitpunkt, zu dem dieses Dokument verfasst wurde, war der AES [110] das zu wählende Verschlüsselungsverfahren.

Einfügen der Fallen Im ursprünglichen Verfahren der Firma Wibu-Systems werden verschlüsselte Blöcke unter die verschlüsselten Varianten gemischt, die keinen (sinnvollen) Code enthalten; sogenannte *Fallen*. Ein Angreifer kann eine verschlüsselte Variante von einer verschlüsselten Falle nicht unterscheiden (siehe Abbildung 4.2). Dies wird genutzt, um Angreifer, die das Dongle als Entschlüsselungsorakel für alle Chiffrate verwenden wollen, zu erkennen.

Um eine Falle zu verschlüsseln, wird ein Schlüssel einer besonderen Form verwendet, die das Dongle als solche erkennt und die nicht für die Verschlüsselung von Varianten genutzt wird. Das Kriterium könnte beispielsweise sein, dass die Ausgabe einer Keyed-Hash-Funktion[1], angewendet auf den Schlüssel, mit einer Eins beginnt. Dieses Vorgehen partitioniert den Schlüsselraum, aus dem Schlüssel für die Verschlüsselung von Varianten und Fallen gezogen werden, (bei idealer Hashfunktion) in zwei gleich große Teile.

Erkennt das Dongle einen Schlüssel als Schlüssel einer Falle, wird es keine weiteren Entschlüsselungsanfragen beantworten; es löscht die Lizenz der Software. Ein Angreifer, der wahllos Chiffrate entschlüsseln lässt, wird – in Abhängigkeit des Verhältnisses zwischen Anzahl verschlüsselter Varianten und verschlüsselter Fallen – früher oder später das Dongle sperren.

Durch Kombination von *Code-Moving* und *Zustandsspeicher* wird es möglich, alle Chiffrate, die nicht den als nächstes auszuführenden Codeblock enthalten, als Falle zu interpretieren. Das Einfügen von Chiffraten, die keinen Code enthalten, bleibt dennoch weiterhin eine Technik des Blurry-Box-Prinzips; durch sie kann der Angreifer nie Sicherheit darüber haben, ob er das Programm bereits in Gänze gesehen hat.

Code-Moving – Variantenauswahl im Dongle *Code-Moving* bedeutet, dass Code im Dongle statt auf dem PC ausgeführt wird. Die vollständige Ausführung des Programms im Dongle stellt die einfachste Methode dar, das Programm vor illegalem Kopieren zu schützen. Durch den Performanceunterschied zwischen PC und Dongle ist das keine sinnvolle Alternative. Für einzelne Codezeilen ist Code-Moving jedoch ohne große Leistungseinbußen möglich.

Mit dem Wissen, welche Codezeilen das Fachwissen darstellen, können diese gezielt verschoben werden. Ein generischer Ansatz ist die Verschiebung der *Variantenauswahl*. Die Variantenauswahl besteht auf der einen Seite – der Seite der gerade ausgeführten Variante – aus der Sprungauswahlfunktion, die den nächsten Block identifiziert, und auf der anderen Seite – der Seite der als nächstes auszuführenden Variante – aus der Wrapperfunktion, die die richtige Variante des Blockes auswählt. Je nach Struktur des Programmes können diese beiden Schritte zur Bestimmung der als nächstes auszuführenden Variante zusammengefasst werden.

Sie werden gemeinsam mit dem geheimen Schlüssel des Dongles verschlüsselt und zusammen mit der Variante auf der unsicheren Hardware gespeichert. Das Chiffrat wird

[1] Der Key der Hashfunktion ist ein Geheimnis des Dongles und wird dort sicher verwahrt.

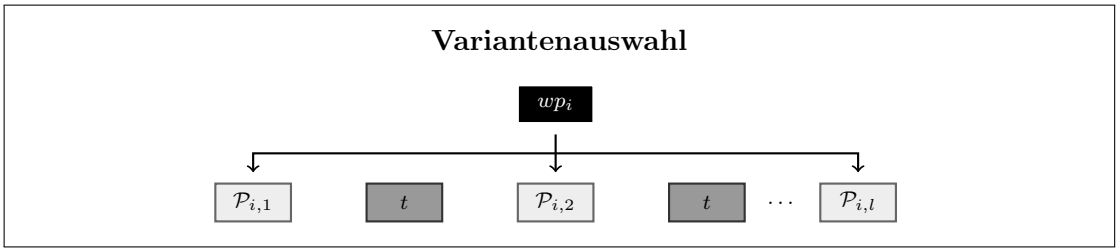

Abbildung 4.2.: Die nächste auszuführende Variante wird anhand des Zustands des Programms vom Wrapper bestimmt. Der (in schwarz dargestellte) Wrapper wird im Dongle ausgeführt. Dass es zu den (in dunkelgrau dargestellten) Fallen keine Verbindung gibt, ist für den Angreifer nicht erkennbar.

gemeinsam mit den notwendigen Parametern an das Dongle gesendet, das sie entschlüsselt und ausführt. Das Ergebnis kann das Dongle an den PC zurückgeben. Dieser weiß nun, welchen Block er sich als nächstes entschlüsseln lassen kann und muss, um das Programm weiter auszuführen.

Effiziente Variantenauswahl im Dongle

Wir betrachten die Auswahl der nächsten Variante, die sowohl aus der Sprungauswahlfunktion der zuletzt ausgeführten Variante als auch der Wrapperfunktion des nächsten Codeblockes besteht. Für die folgende Betrachtung müssen wir nicht zwischen den beiden Teilen der Berechnung unterscheiden. Wir nehmen eine kurze Folge von Codezeilen an, die im Dongle ausgeführt wird und ein Tupel ausgibt: (Blocknummer, Variantennummer).

Das Tupel identifiziert die auszuführende Variante. Wären die Varianten jedoch so bezeichnet, würde es einem Angreifer leicht fallen zu überprüfen, welche Variante er noch nicht entschlüsselt hat. Stellen wir uns eine Tabelle mit einer Abbildung von dem Tupel auf eine Zufallszahl vor. So könnten wir die Zufallszahl als Varianten-ID verwenden. Um die ID der nächsten Variante zu bestimmen, kann anhand der Blocknummer und Variantennummer in der Tabelle nachgeschlagen werden. Das Vorgehen verhindert Rückschlüsse von der Varianten-ID auf ihren Inhalt.

Bei vielen Varianten wächst diese Tabelle auf eine Größe, die im Dongle nicht mehr effizient zu verarbeiten ist. Daher verwenden wir eine Hash-Funktion, die die Abbildung dynamisch berechnet:

$$\text{Varianten-ID} = \texttt{H}\,(\text{Donglegeheimnis},\ \text{Blocknummer},\ \text{Variantennummer})$$

Damit das Ergebnis der Funktion als Varianten-ID genutzt werden kann, werden die Varianten-IDs bei der Erstellung des geschützten Programms nicht zufällig gewählt. Die Varianten-IDs werden mit Hilfe der Hashfunktion, die während der Programmausführung auch das Dongle verwendet, gewählt.
Mit dieser Methode können Varianten-IDs effizient berechnet werden, ohne dazu Speicher im Dongle oder auf dem PC zu verbrauchen.

Abbildung 4.3.: Die Variantenauswahl im Dongle durchzuführen stellt den Angreifer vor das Problem, dass er einen kleinen, aber wichtigen Teil des Programmcodes nicht zu sehen bekommt. Die Effizienz dieser Verschiebung ist dabei von großer Relevanz. Wir geben hier eine mögliche Umsetzung an.

Zustandsspeicher Um den Spielraum eines Angreifers weiter einzuschränken, erhält das Dongle Informationen über den aktuellen Zustand, in dem sich die Software befindet. Darüber soll es dem Dongle möglich sein, Angriffe, die die Ablaufreihenfolge beeinflussen, besser feststellen zu können.

Im Zusammenspiel mit der Verschiebung der Variantenauswahl in das Dongle ist es möglich, aus allen Varianten, außer der vom Dongle für den nächsten Schritt gewählten, Fallen zu machen. Das zwingt den Angreifer zur Manipulation der an das Dongle übergebenen Parameter, um vom normalen Ausführungspfad abzuweichen. Verfügt das Dongle über ein Modell, welche Werte die Parameter annehmen können, ergibt sich dadurch eine weitere Möglichkeit, den Angriff eines Angreifers ohne Domänenwissen festzustellen; dieser verfügt – so die Annahme – nicht über das Fachwissen, in welchem Bereich sich die Werte, die die Parameter annehmen können, bewegen. Das Modell über die Parameter kann – mit dem geheimen Schlüssel des Dongels verschlüsselt – Teil der Sprungauswahlfunktion und auf der unsicheren Hardware abgespeichert sein.

Diese Methodik verhindert auch, dass ein Anwender mit einer einzelnen gültigen Lizenz die Software mehrfach ausführt.

4.3.6. Sicherheitsanalyse

In diesem Kapitel betrachten wir die Sicherheit von Blurry-Box und skizzieren die Sicherheitsanalyse. Um die Diskussion übersichtlich zu halten, vereinfachen wir das Verfahren in zwei Punkten.

1. Das Dongle entschlüsselt die Varianten selbst.
2. Die Sprungadressen sind im Dongle gespeichert.

Wir betrachten für die Analyse ein Programm, das mit den Methoden des Blurry-Box-Schemas in verschlüsselte Blöcke aufgeteilt wurde. Wir betrachten weiterhin das Dongle als Orakel. Es fungiert einerseits als Entschlüsselungsorakel, andererseits als Orakel, welcher Block zur korrekten Programmausführung als nächstes entschlüsselt und ausgeführt werden muss. Das Orakel ist nur so lange verfügbar, bis die erste Falle entschlüsselt wurde. Wir werden im Folgenden wieder *statische* und *dynamische* Angreifer betrachten.

Statischer Angreifer

Der statische Angreifer handelt unabhängig vom Programmcode. Der Angreifer verwendet keinen Debugger, nimmt keinen Einfluss auf Variablenbelegungen, analysiert weder das Programm noch führt er es aus. Angreifer dieses Typs versuchen den Softwareschutz zu brechen, indem sie das Dongle als Entschlüsselungsorakel verwenden. Da wir die Sprungadressen im Dongle verbergen, ist das ein realistischer erster Ansatz, den Schutz zu brechen. Wir können die Sicherheit des Schutzes vor diesem Angreifertyp zeigen, indem wir nachweisen, dass seine Erfolgswahrscheinlichkeit sehr gering ist.

Dynamischer Angreifer

Der dynamische Angreifer führt das Programm aus, nutzt Debugger und versucht, die Struktur des Programmes ohne spezifisches Domänenwissen der Anwendung zu analysieren. Für einen aussagekräftigen Sicherheitsbegriff müssen wir die Annahme machen, dass der dynamische Angreifer nicht in der Lage ist, das für die Anwendung

notwendige Domänenwissen zu extrahieren. Das bedeutet insbesondere, dass er fehlende Teile nicht rekonstruieren kann. Damit kann gezeigt werden, dass die Rekonstruktion des gesamten Programmes nur mit einem sehr hohem Einsatz möglich ist.

Um die Aussagen über den statischen und den dynamischen Angreifer zu zeigen, definieren wir für beide je ein Sicherheitsmodell. Alle Angreifer können auf eine kleine Menge an Angreifern reduziert werden. Für die reduzierte Menge kann die Sicherheit des Schemas bewiesen werden.

Für die Klasse der statischen Angreifer kann auf die Angreifer reduziert werden, die nur Entschlüsselungsanfragen an das Dongle stellen und weder den Programmcode analysieren noch das Verschlüsselungsverfahren brechen. Durch die Fallen, die im geschützten Programmcode enthalten sind, ist es ihnen nur mit vernachlässigbarer Wahrscheinlichkeit möglich, das Programm erfolgreich zu rekonstruieren. Der dynamische Angreifer erhält auf seine Entschlüsselungsanfrage hin den Programmcode (ohne die Sprungbedingungen und -adressen) und kann darauf basierend seinen Angriff optimieren. Einem solchen Angreifer ist es in jedem Falle möglich, das Programm wie ein normaler Nutzer auszuführen und die entschlüsselten Stücke zusammenzusetzen. Wir nennen einen solchen Angreifer *Copy&Paste-Angreifer*. Dieser kann nicht ausgeschlossen werden. Für große und komplexe Programme ist das vollständige Traversieren des Programmes jedoch zu aufwändig. Wir zeigen, dass ein Angreifer, der keinen Einblick in die Funktionsweise des Programmes hat, nicht effizienter ist als der Copy&Paste-Angreifer. Anhand von Beispielen zeigen wir, wie die Methoden der Variantenbildung umgesetzt werden könnten.

Notation

- Ein *Programm* $\mathcal{P}$ ist eine Menge maschinenlesbarer Code-Blöcke $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\} \notin \{11\ldots1,\, 00\ldots0\}$ und einer Funktion $\texttt{jump}_{\mathcal{P}}$, die basierend auf einem Index i und einem Vektor $\bar{x}$ als Eingabe einen Index j oder „Falle“ als Ausgabe berechnet; außerdem eine Liste $\texttt{Legal}_{\mathcal{P}}$ mit Index-Paaren (i, j). Wir nennen $\mathcal{P}_i$ Varianten. Für den Beweis nehmen wir an, dass alle Varianten die gleiche Länge $\text{len}(\mathcal{P})$ haben. Des Weiteren sei $\text{size}(\mathcal{P})$ die Anzahl der Varianten, die ein Programm enthält, und $|\mathcal{P}|$ die Bitlänge des Programms.
- Des Weiteren sei $\text{size}(\mathcal{P})$ die Anzahl der Varianten; jede sei $\text{len}(\mathcal{P})$ groß.
- Wir codieren Fallen als Bitstrings, die nur aus Einsen bestehen. Dadurch werden Fallen für das Dongle leicht erkennbar, was für den Beweis benötigt wird.
- Wir definieren `InsertTraps` als Vorgang, der – gegeben eine ganzzahligen Variable $T > 0$ und ein Programm $\mathcal{P} = (\mathcal{P}_1, \ldots, \mathcal{P}_{\text{size}(\mathcal{P})}, \texttt{jump}_{\mathcal{P}})$ – insgesamt T Fallen $\mathcal{P}_{i+\text{size}(\mathcal{P})} := 1^{\text{len}(\mathcal{P})}$ $(i = 1, \ldots, T)$ erstellt, eine Permutation π wählt und $(\widetilde{\mathcal{P}}_1, \ldots, \widetilde{\mathcal{P}}_{\text{size}(\mathcal{P})+T}, \overline{jump}_{\mathcal{P}})$ ausgibt. Dabei gilt $\widetilde{\mathcal{P}}_i := \mathcal{P}_{\pi(i)}$, und $\overline{jump}_{\mathcal{P}}$ gibt den

Wert $\pi(j)$ aus, wenn $\texttt{jump}_{\mathcal{P}}$ den Wert j ausgibt und „trap“ in allen anderen Fällen. Wir nennen $\widetilde{\mathcal{P}}_i$ einen *Code-Block*.

- Mit $\Delta(A, B)$ bezeichnen wir den statistischen Abstand zwischen zwei zufälligen Variablen A, B: $\Delta(A, B) = \frac{1}{2} \sum_{v \in V} |\Pr[A = v] - \Pr[B = v]|$.
- Wir verwenden die Standarddefinition für komplexitätstheoretische Ununterscheidbarkeit (siehe [60]) und schreiben $A \overset{c}{\approx} B$, wenn zwei Variablen A, B komplexitätstheoretisch ununterscheidbar sind.
- Außerdem verwenden wir die Standarddefinition von IND-CPA-Sicherheit für asymmetrische Verschlüsselungsverfahren (siehe [60]).

Die Sicherheit des Blurry-Box-Prinzips gegen statische Angreifer lässt sich ähnlich wie die gegen dynamische Angreifer zeigen. Der statische Angreifer wird im Wesentlichen im letzten Schritt des Modells eingeschränkt sein. Daher gehen wir hier ausschließlich auf den dynamischen Angreifer ein.

Dynamischer Angreifer

In diesem Abschnitt betrachten wir den dynamischen Angreifer. Der dynamische Angreifer kann – im Gegensatz zum statischen Angreifer – das Programm ausführen, den Code analysieren und einen Debugger während der Ausführung verwenden. Prinzipiell ist ein dynamischer Angreifer daher in der Lage, das ganze Programm auszuführen und zu kopieren. Für unsere Analyse gehen wir davon aus, dass der Angreifer den entschlüsselten Programmcode aus dem Speicher kopieren und zu einer funktionsfähigen Software zusammensetzen kann. Wir nennen diesen Angreifer Copy&Paste-Angreifer. Um eine vollständige Kopie zu erhalten, ist eine häufige Ausführung des Programmes mit geschickt gewählten Eingabedaten notwendig, so dass jeder Teil des Programms mindestens einmal benötigt wird. Für komplexe Programme ist dieser Angriff zu ineffizient. Für einen effizienteren Angriff müsste der Angreifer die Struktur des Programmes und seine Funktionsweise kennenlernen. Wir zeigen, dass er dazu nicht in der Lage ist.

Wir nutzen hierzu das Real-Ideal-Paradigma. Bevor wir das Experiment im idealen Modell in Abbildung 4.5 definieren, diskutieren wir das reale Modell in Abbildung 4.4.

Das Experiment in Abbildung 4.4 beschreibt einen realen dynamischen Angreifer auf Blurry-Box. Es unterscheidet sich aber vom statischen Fall dadurch, dass es dem Angreifer ermöglicht, das Programm auszuführen: Der Angreifer kann das Experiment fragen, welcher Programm-Block als nächstes ausgeführt werden soll, relativ zu der aktuellen Position im Programm (i) und dem Zustand des Programms (x) (Schritt 3a). Diese Sprunganfragen zeigen dem Angreifer, welchen Block er als nächstes gefahrlos entschlüsseln lassen kann (Schritt 3b). Eine abwechselnde Sprung- und Entschlüsselungsanfrage entspricht dem normalen Programmablauf. Den Zustand des Programms (Variablenbelegungen) erhält der Angreifer durch die Ausführung des Programms. Der Angreifer kann in Schritt 4

Experiment $\mathrm{DynReal}^{\mathrm{SKE},T}_{\mathcal{P},\mathcal{A}}(n)$ **im realen Modell**

Definition 1 (Experiment für das reale Modell) *Sei* SKE *ein symmetrisches Verschlüsselungsverfahren und* $T > 0$*. Gegeben ein Programm* $\mathcal{P}$ *und ein Angreifer* $\mathcal{A}$*, ist das Experiment* $\mathrm{DynReal}^{\mathrm{SKE},T}_{\mathcal{P},\mathcal{A}}(n)$ *wie folgt definiert:*

1. *Der Angreifer* $\mathcal{A}$ *erhält* $1^{n+|\mathcal{P}|}$ *als* Eingabe.
2. *Das Experiment berechnet*
   $(\widetilde{\mathcal{P}}_1, \ldots, \widetilde{\mathcal{P}}_{\mathrm{size}(\mathcal{P})+T}, \overline{jump}_{\mathcal{P}}, \overline{\texttt{Legal}}_{\mathcal{P}}) \leftarrow \texttt{InsertTraps}(\mathcal{P}, T)$
   und sendet die verschlüsselten Blöcke
   $\{\mathcal{C}_i = \mathrm{Enc}_{\mathrm{SKE}}(\widetilde{\mathcal{P}}_i) |\ i \in \{1, \ldots, \mathrm{size}(\mathcal{P}) + T\}\}$ *an* $\mathcal{A}$*.*
3. *Das Experiment initialisiert einen Zähler* $C = 0$ *und eine Statusvariable* $\texttt{state} = 0$*.*
   Der Angreifer $\mathcal{A}$ *kann (adaptiv) zwei Typen von Anfragen an das Experiment senden:*
   a) Index-Anfragen*: Der Angreifer sendet einen Vektor* $\bar{x}$ *an das Experiment. Wenn es die erste Index-Anfrage ist, berechnet das Experiment* $\overline{jump}_{\mathcal{P}}(0, \bar{x})$*, sonst* $\overline{jump}_{\mathcal{P}}(i, \bar{x})$ *wenn* $\texttt{state} = i$ *Wenn das Ergebnis „Falle“ ist, antwortet das Experiment mit „Falle“ und ignoriert alle weiteren Anfragen; sonst antwortet das Experiment mit dem berechneten Index.*
   b) Entschlüsselungsanfragen*: Für jede Entschlüsselungsanfrage* $j \in \{1, \ldots, \mathrm{size}(\mathcal{P}) + T\}$ *verfährt das Experiment wie folgt:*
      - *Wenn* $\texttt{state} = i$ *und* (i, j) *nicht in* $\overline{\texttt{Legal}}_{\mathcal{P}}$ *oder wenn* $\widetilde{\mathcal{P}}_j$ *eine Falle ist, sendet das Experiment „Falle“ an* $\mathcal{A}$ *und ignoriert alle weiteren Anfragen.*
      - *Wenn* j *bereits angefragt wurde, ignoriert das Experiment die Anfrage.*
      - *Sonst setzt das Experiment* $C = C + 1$ *und* $\texttt{state} = j$ *und antwortet mit* $\widetilde{\mathcal{P}}_j$*.*
4. *Sei* $\mathcal{Q}$ *die Anzahl der Entschlüsselungsanfragen aus Schritt 3. Der Angreifer kann bis zu* $\mathrm{size}(\mathcal{P}) - \mathcal{Q}$ *Codeblöcke* $Q_i \in \{0,1\}^{\mathrm{len}(\mathcal{P})}$ *an das Experiment senden. Für jeden Index* i *setzt das Experiment* $C = C + 1$*, wenn eine Variante* $\widetilde{\mathcal{P}}_j$ *existiert, so dass*
   (1) $Q_i = \widetilde{\mathcal{P}}_j$*,*
   (2) j *noch nicht in Schritt 3b angefragt wurde und*
   (3) $\widetilde{\mathcal{P}}_j$ *nicht identisch mit einem Block zuvor (*Q_l*,* $l < i$*) war.*

Abbildung 4.4.: Reales Modell für den dynamischen Angreifer

Code an das Experiment senden; das entspricht der Möglichkeit, dass der Angreifer eigene (gültige) Varianten erzeugt.

Ein Angreifer ist im Allgemeinen primär nicht daran interessiert, eine exakte Kopie des Programmes zu erhalten. Angreifer, wie sie in der realen Welt zu erwarten sind, haben zum Ziel, eine *funktional gleichwertige* Software zur Verfügung zu haben. Ein Angreifer, der von der ursprünglichen Implementierung abweichen und eigene Varianten nutzen möchte, muss aber – was den elementaren Teil der Software betrifft – verstehen, wie die Software funktioniert. Ohne die Kenntnis des Know-Hows entspricht daher die Kopie der Software nicht nur funktional dem Original.

Abbildung 4.5 zeigt nun die Definition des Experiments in der idealen Welt. Wie im statischen Fall erlaubt das ideale Modell nur Aktionen des Angreifers, die nicht verhindert werden können. Das bedeutet, dass der Angreifer insbesondere einen Copy&Paste-Angriff durchführen kann.

Die Experimente $\text{DynReal}^{\text{SKE},T}_{\mathcal{P},\mathcal{A}}(n)$ und $\text{DynIdeal}^{T}_{\mathcal{P},\mathcal{A}}(n)$ unterscheiden sich jedoch in einigen Punkten. Der Angreifer erhält nur noch die Informationen über die Anzahl der Codeblöcke und deren Länge. Außerdem kann er in Schritt vier keine weiteren, eigenen Codeblöcke an das Experiment senden.

Das Modell lässt lediglich Copy&Paste-Angreifer zu. Enthält das reale Modell keine Angriffe, die nicht auch im idealen möglich sind, gibt es keinen Angreifer, der einen höheren Erfolg haben kann als ein Copy&Paste-Angreifer.

Experiment $\text{DynIdeal}^{T}_{\mathcal{P},\mathcal{A}}(n)$ **im idealen Modell**

Definition 2 (Experiment für ideales Modell) *Sei $T > 0$. Gegeben ein Angreifer $\mathcal{A}$ und ein Programm $\mathcal{P}$, sei das Experiment* $\text{DynIdeal}^{T}_{\mathcal{P},\mathcal{A}}(n)$ *wie folgt definiert:*

1. *Dem Experiment wird $1^{n+|\mathcal{P}|}$ als* Eingabe *gegeben.*
2. *Das Experiment sendet* $\text{size}(\mathcal{P}) + T$ *und* $\text{len}(\mathcal{P})$ *an den Angreifer $\mathcal{A}$.*
3. *Das Experiment berechnet*
   $(\widetilde{\mathcal{P}}_1, \ldots, \widetilde{\mathcal{P}}_{\text{size}(\mathcal{P})+T}, \overline{jump}_{\mathcal{P}}, \overline{\texttt{Legal}}_{\mathcal{P}}) \leftarrow \texttt{InsertTraps}(\mathcal{P}, T)$.
   Das Experiment initialisiert einen Zähler $C = 0$ und eine Statusvariable $\texttt{state} = 0$.
   Der Angreifer $\mathcal{A}$ kann (adaptiv) Anfragen zweier Typen stellen:
   a) Index-Anfragen*: Der Angreifer sendet einen Vektor $\bar{x}$ an das Experiment. Wenn es die erste Index-Anfrage ist, berechnet das Experiment $\overline{jump}_{\mathcal{P}}(0, \bar{x})$, sonst $\overline{jump}_{\mathcal{P}}(i, \bar{x})$ if $\texttt{state} = i$. Wenn das Ergebnis „Falle“ ist, antwortet das Experiment mit „trap“ und ignoriert alle weiteren Anfragen; sonst antwortet das Experiment mit dem berechneten Index.*
   b) Entschlüsselungsanfragen*: Für jede Entschlüsselungsanfrage $j \in \{1, \ldots, \text{size}(\mathcal{P}) + T\}$ verfährt das Experiment wie folgt:*
      - *Wenn $\texttt{state} = i$ und (i, j) nicht in $\overline{\texttt{Legal}}_{\mathcal{P}}$ enthalten ist oder wenn $\widetilde{\mathcal{P}}_j$ eine Falle ist, antwortet das Experiment mit „Falle“ und ignoriert alle weiteren Anfragen.*
      - *Wenn j bereits angefragt wurde, ignoriert das Experiment die Anfrage.*
      - *Sonst setzt das Experiment $C = C + 1$ und $\texttt{state} = j$ und antwortet mit $\widetilde{\mathcal{P}}_j$.*
4. *Die Ausgabe des Experiments ist C.*

Abbildung 4.5.: Ideales Modell für den dynamischen Angreifer

Um zu zeigen, dass das reale Modell keinen zusätzlichen Angriff zulässt, müssen wir eine Annahme machen: Der Angreifer kann weder die Anwendung als Ganzes noch Teile davon selbst erzeugen. (Siehe Abbildung 4.6.)

Fachlichkeitsannahme – Variant Security Assumption (VSA)

Fachlichkeitsannahme (*Variant Security Assumption (VSA)*) *Gegeben eine Teilmenge der Varianten $\mathcal{P}_{i_1}, \ldots \mathcal{P}_{i_k}$, darf ein Angreifer $\mathcal{A}$ nicht in der Lage sein, selbst weitere, gültige Varianten zu erzeugen.*

Abbildung 4.6.: Die Sicherheit des Blurry-Box-Schemas basiert auf der Annahme, dass ein Angreifer die Software nicht versteht und Teile davon nicht neu schreiben kann.

Es kann gezeigt werden, dass die folgende Aussage gilt: „Wenn ein Angreifer die Fachlichkeitsannahme für ein Programm $\mathcal{P}$ nicht brechen kann, dann existiert ein Copy&Paste-Angreifer, der die gleiche Menge Varianten erhalten kann."

> **Technische Bemerkung: Fachlichkeitsannahme nicht für das gesamte Programm**
> Die Fachlichkeitsannahme (VSA) besagt, dass ein Angreifer keine Varianten, die bisher nicht entschlüsselt sind, selbst schreiben können darf. Der Gedanke hinter dieser Forderung ist, dass es dem Angreifer unmöglich sein muss, die Varianten, die er bei seiner Ausführung des Programms nicht erreichen konnte, durch eigenen Code zu ersetzen.
>
> Es ist offensichtlich, dass diese Annahme nicht für das gesamte Programm gelten muss: Ein Programm, das diese Annahme erfüllt, kann mit einem sicheren Schutz versehen werden. Wird dieses Programm um banalen, leicht zu ersetzenden Code erweitert, wird der Schutz der Bestandteile, die im ursprünglichen Programm enthalten waren, nicht schlechter. Lediglich die neu hinzugefügten Teile, deren Funktion der Angreifer leicht herausfinden und selbst programmieren kann, lassen sich nicht schützen.
>
> Die VSA muss für den zentralen, zu schützenden Teil des Programmes gelten; nicht für die peripheren Teile.

Erfolgsanalyse Wenn ein Angreifer $\mathcal{A}$ blind Varianten entschlüsseln lässt, wie es der statische Angreifer tut, wird er mit hoher Wahrscheinlichkeit auf eine Falle treffen, was es ihm unmöglich macht, weitere Varianten zu entschlüsseln. Selbst das Entschlüsseln beliebiger Code-Blöcke, die keine Fallen sind, löst ebenfalls eine Sperre des Dongles aus, da es die mögliche Ablaufreihenfolge der Blöcke kennt. Das Dongle zwingt den Angreifer damit zu einer Aufrufreihenfolge, die einem plausiblen Programmlauf entspricht.

Um das Auslösen der Sperre zu vermeiden, muss der Angreifer das Programm ausführen, um zusätzliche Informationen zu erhalten. Für die Pfade durch das Programm, die er durch das Ausführen mit von ihm gewählten Eingabeparametern kennt, kennt er auch die Zusammenhänge der dabei benötigten Varianten. Alle anderen Zusammenhänge bleiben ihm unbekannt, da die Variantenauswahl durch die Sprungberechnung im Dongle vor ihm verborgen bleibt. Das bedeutet insbesondere auch, dass er aus den Informationen, die er gewinnen kann, nicht ableiten kann, bei welchen bisher unentschlüsselten Blöcken es sich um Fallen handelt. Das bedeutet, dass der Angreifer nicht mächtiger ist als der Copy&Paste-Angreifer.

Der Angreifer könnte – statt das Programm mit verschiedenen Eingaben auszuführen – während der Ausführung die Belegung der Variablen ändern und so versuchen, vom Dongle einen Sprung auf eine bisher unentschlüsselte Variante zu erhalten. Dazu muss er – um zu vermeiden, dass $\overline{jump}_{\mathcal{P}}(i, x)$ eine „Falle" ausgibt – den Gültigkeitsbereich der Variablen kennen, der gegebenenfalls noch von weiteren Variablen abhängig ist. Dazu ist domänenspezifisches Wissen notwendig.

Zusammenfassung Um einen Copy&Paste-Angreifer zu übertreffen – das folgt aus der Sicherheitsanalyse –, muss ein Angreifer die Funktionsweise des Programms lernen. Das bedeutet, dass Programme, die eine hohe Komplexität aufweisen, Kandidaten sind, die

sich potentiell mit diesem Schema schützen lassen. Einige Ansätze, wie sich der Schutz umsetzen lässt, werden im folgenden Abschnitt diskutiert.

4.3.7. Realisierungen

Nach Fertigstellung des Konzeptes wurde zunächst ein Demonstrator für Messen und Ausstellungen angefertigt, der die Funktionsweise erläutert. Das hierfür speziell entwickelte Programm/Spiel konnte durch seine Struktur automatisiert in Varianten aufgeteilt und die Wrapper automatisiert erstellt werden. Dennoch konnten aus zeitlichen Gründen nicht alle Blurry-Box-Methoden angewendet werden. Für einen Hackers-Contest wurde ein zweites Programm von Wibu-Systems erstellt und mit allen zur Verfügung stehenden Blurry-Box-Techniken geschützt. Weitere Schutzmaßnahmen – wie Schutz vor Debuggern – kamen für den Hackers-Contest bewusst nicht zum Einsatz.

CeBIT-Exponat

Im Rahmen der Vorstellung des Konzeptes auf der CeBIT 2016 ist ein Exponat entstanden, das die Praktikabilität des Ansatzes demonstriert. Das Exponat – bestehend aus einem LED-Cube mit $16 \times 16 \times 16$ LEDs, einem PC und einem Raspberry Pi – ist die Realisierung eines Spiels mit dreidimensionaler Ausgabe. Das Spiel, das an einen simplen Flugsimulator angelehnt ist, ist mit dreien der oben beschriebenen Methoden geschützt. Sowohl die Variantenbildung als auch die Verschiebung des Sprungcodes in das Dongle – dargestellt durch den Raspberry Pi – wurden umgesetzt. Das Dongle kennt den Zustand des Programmes und lässt keine Sprünge außerhalb des durch die Programmausführung gewählten Pfades zu. Weitere Techniken konnten aufgrund der kurzen Zeit, in der das Exponat aufgebaut werden musste, nicht mehr umgesetzt werden.

Der Spieler kann sich in die Lage eines Angreifers versetzen, das Spiel spielen und testen, welchen Erfolg er hätte, weitere Varianten ohne weiteres Spielen zu entschlüsseln. Dabei stellt er fest, dass einfaches Durchprobieren im Allgemeinen nicht zum Erfolg führt. Jeder Misserfolg entspricht jedoch einer gesperrten Lizenz.

Der Demonstrator zeigt einerseits, dass die Ausführung trotz der Aufteilung des Spiels flüssig läuft. Andererseits gibt sie dem Betrachter einen Eindruck, welcher Aufwand – trotz der Einfachheit des Spiels – für einen Copy&Paste-Angreifer notwendig wäre, um durch die Verwendung des Programmes jede Variante zu durchlaufen.

Für diesen Demonstrator wurde ein Automatismus entworfen, der – gegeben das ungeschützte Spiel – eine geschützte Variante des Spiels erstellt. Im Allgemeinen lässt sich dieser Automatismus nicht anwenden, da er spezielle Eigenschaften der Umsetzung des Spiels nutzt, die nicht für jede Anwendung gelten.

Hackers-Contest

Im Mai 2017 hat die Fima Wibu-Systems einen mit 50.000 € Preisgeld dotierten Hackers-Contest veranstaltet. Ziel des Contests war es, ein mit den Blurry-Box-Methoden geschütztes Spiel so zu modifizieren, dass es sich ohne Dongle ebenso verhält wie die Originalversion des Spiels mit Dongle. An dem Contest nahmen über dreihundert Teilnehmer aus der ganzen Welt teil. Keinem der Hacker ist es gelungen, eine ohne das Dongle lauffähige Version des Spiel bei der unabhängigen Jury einzureichen.

Das Spiel Bei dem Spiel handelte es sich um einfaches *Deutschlandreise-Spiel*, bei dem der Spieler mit verschiedenen Verkehrsmitteln von Stadt zu Stadt reist und an jedem Reiseziel Fragen gestellt bekommt, mit denen er sein Guthaben auffüllen kann. Reisemöglichkeiten, Reisekosten, Fragen und die Gewinnfunktion stellen dabei die Varianten dar. Die Übergangsfunktion zwischen den einzelnen Varianten wird im Dongle berechnet und ist so vor den Blicken der Hacker geschützt. Die Auswahlfunktion ließ sich in wenigen Codezeilen darstellen, enthielt aber einige Ausnahmen, die für den Spieler nicht ohne Weiteres vorhersehbar waren. Die Implementierung entspricht der in Abbildung 4.3 beschriebenen Technik. Skizzen zur Umsetzung des Schutzes sind im Anhang B zu finden.

Zur leichteren Bedienung war eine den Hackern bekannte Schnittstelle implementiert, die das gescriptete Ausführen des Programms ermöglicht. Zu erwähnen ist an dieser Stelle noch, dass die Korrektheit der Antworten im Dongle geprüft wird und daher eine automatisierte Ausführung des Spiels, so dass alle Antworten immer korrekt gegeben werden, nicht möglich ist.

Ein Angriff, wie wir ihn erwartet haben, lässt sich in zwei Phasen einteilen:

Entschlüsseln der Varianten Der erste Schritt für einen erfolgreichen Hack bestand darin, alle gültigen Varianten des Spiels zu entschlüsseln. Das Spiel war so ausgelegt, dass ausreichend Zeit zur Verfügung stand, dass dies durch normale Nutzung gelingen kann. Die Komplexität des Spiels wird durch die Zahl der Reisemöglichkeiten, der Fragen und der Reisekosten erreicht. Das Spiel enthält 491 Städte, 2817 Fragen unterschiedlicher Schwierigkeitsstufen, 10 Varianten für die Reisekosten. Der Gewinn wurde anhand 5 unterschiedlicher Funktionen berechnet. Insgesamt enthielt das Spiel 34 Fallen.

Bereits dieser Schritt ist keinem der Angreifer gelungen.

Rekonstruieren der Auswahlfunktion Sind alle Varianten bekannt, muss die Auswahlfunktion rekonstruiert werden. Hierfür war der Aufwand so hoch, dass es den Hackern auch in größeren Teams nicht gereicht hätte, alle Spielpfade zu durchlaufen. Hier hätten die Teilnehmer zu einer eigenen Strategie greifen müssen.

Die Auswahlfunktion zu rekonstruieren war unter anderem durch seltene Ereignisse (*Easter Eggs*) schwierig. Die Easter Eggs wurden beispielsweise durch eine ausreichend lange

Reihe richtig beantworteter Fragen oder eine bestimmte Reihenfolge der besuchten Städte ausgelöst. Auch die Auswahl der Fragen wurde durch den Verlauf des Spiels bestimmt.

Das Entwicklerteam selbst ist davon überzeugt, dass es hier keinen effizienteren Weg gibt als alle Pfade zu durchlaufen, um Ausnahmen im Verhalten zu finden.

Schlussfolgerung Ob der Schutz die erwartete Stärke hat, lässt sich nur dann (negativ) beantworten, wenn ein Hacker das Ziel erreicht. Solange das nicht der Fall ist, sprechen wir davon, dass der Schutz ungebrochen ist; wir können aber nicht ausschließen, dass es einem Hacker gelungen ist, den Schutz zu umgehen, er die Lösung aber nicht eingereicht hat. Während der formale Beweis nur die Sicherheit des (theoretischen) Konzepts beweist, aber nicht die Implementierung an sich, zeigt sich erst durch den praktischen Einsatz, ob die Umsetzung frei von Fehlern ist, die die Sicherheit beeinträchtigen.

5. Kryptographische Wahlverfahren

Kryptographische Wahlverfahren haben zum Ziel, den papierbasierten Prozess der Stimmabgabe bei politischen Wahlen zu ersetzen. Kryptographische Wahlverfahren haben das Potential, Auszählungsergebnisse schneller zur Verfügung stellen zu können, die zudem weniger fehleranfällig sind. Außerdem ist es möglich, Schutz gegen Angriffe zu bieten, die bei herkömmlichen Wahlverfahren nicht ausgeschlossen werden können. Durch die Ergänzung der Präsenzwahlen durch Onlinewahlen versprechen sich viele eine höhere Wahlbeteiligung – insbesondere bei jungen Wählern. Die etablierten papierbasierten Wahlverfahren sollten aber erst dann durch ihre digitale Alternative ersetzt werden, wenn die Sicherheitseigenschaften des Gesamtsystems das Niveau des bislang eingesetzten Verfahrens erreichen. Das kann durch unterschiedliche Bedrohungslagen von Land zu Land und von Abstimmung zu Abstimmung variieren:

Die Gründe, aus denen Beteiligte oder Dritte die Art und Weise, wie die Wahl durchgeführt wurde, oder deren Ergebnis anzweifeln, sind vielfältig. Als drei unterschiedliche Beispiele können die kurzfristigen Änderungen der Wahlgesetzreform in Kenia (wie Wähler und Wahlstimmen registriert werden) [113], die zusätzlichen Stimmen, die bei der Wahl für eine verfassungsgebende Versammlung in Venezuela eingefügt worden sein sollen [100, 119, 48], oder die Unstimmigkeiten bei der Wahl des Vorstandsvorsitzende der Jüdischen Gemeinde in Berlin [104, 103, 116] genannt werden.

In dieser Arbeit werden zwei von uns entworfene (bzw. weiterentwickelte) Verfahren diskutiert. Das erste Verfahren basiert auf einer Wahlmaschine für ein Präsenzwahlverfahren. Das zweite Verfahren ist Teil eines Onlinewahlverfahren.

Bei Präsenzwahlverfahren gibt der Wähler seine Stimme – wie bei der bislang verwendeten Papierwahl – in einer Wahlkabine in dem ihm zugeordneten Wahllokal ab. Statt seine Stimme durch ein Kreuz auf Papier zu markieren, gibt er die Stimme in ein Computersystem ein. Durch die digitale Erfassung entsteht die Schwierigkeit, die Korrektheit und die Erpressungsfreiheit zu gewährleisten. Die Authentifizierung des Wählers findet wie bisher losgelöst vom Prozess der Stimmabgabe und der Auszählung statt.

Bei Onlinewahlverfahren kann der Wähler zuhause an seinem eigenen PC (oder einem ähnlichen Gerät) wählen. Auch hier müssen Korrektheit, Erpressungsfreiheit und eine fehlerfreie Autorisierung gewährleistet sein. Die fehlende Wahlkabine und die Verwendung durch den Wähler (respektive durch den Angreifer) kontrollierter Hardware stellen hier eine besondere Herausforderung dar. Hat der Wähler seine Stimme erfolgreich abgegeben, so soll er nur noch zu der optionalen Verifikation der Korrektheit, jedoch nicht zur Auszählung aktiv werden müssen.

5.1. Vertrauen in die Hardware

Auch in dem dritten von uns betrachteten Szenario setzen wir Hardware als Vertrauensanker in einer Wahlmaschine ein. Der *Inhaber* ist in diesem Szenario der Wähler. Das Gut, das der Inhaber in diesem Szenario schützen will, ist seine Wahlentscheidung. *Vertrauen* bedeutet an dieser Stelle, dass der Inhaber oder Personen, denen er vertraut, überprüfen können, dass die Hardware des Vertrauensankers genau der Spezifikation entspricht.

Wie beim ersten Beispiel, der Girocard, ist der Inhaber auch der Nutzer der Hardware; sie soll den Wähler davor schützen, dass seine Wahlentscheidung der Maschine oder Dritten bekannt wird.

Der Inhaber vertraut dem Hersteller der Hardware jedoch nicht; wäre dem so, wäre ein einfacher PC, der die Stimmen speichert, hinreichend.

Der Inhaber des zu schützenden Guts ...
... ist Nutzer der vertrauenswürdigen Hardware.
... vertraut dem Hersteller der Hardware *nicht*.

Der entscheidende Punkt in diesem Szenario ist, dass die Hardware so einfach gebaut sein muss, dass sie von einem Wähler überblickt werden kann. Das bedeutet zumindest, dass sie – ähnlich einer Wahlurne – vor der Wahl und/oder im Nachhinein genauestens überprüft werden kann. Anders als bei einem komplexen Computer gehen wir davon aus, dass eine Schaltung, die aus wenigen Bausteinen aufgebaut ist, dieses Kriterium erfüllt.

5.2. Das Fundament und die Ziele kryptographischer Wahlverfahren

In einer idealen Lösung muss sich jeder Wähler und jedes Mitglied des Veranstalters der Wahl nur selbst vertrauen. Dieses Ziel wird jedoch von papierbasierten Wahlverfahren nicht erreicht und scheint leider auch nicht erreichbar zu sein. Daher muss über die Annahmen gesprochen werden, auf die jeder Beteiligte sein Vertrauen stützen kann. Diese können andere, an dem Prozess beteiligte Personen, die verwendeten Protokolle, Hard- oder Software und mathematische Beweise sein.

Für Berechnungen eines Ergebnisses, das auf mehreren geheimen Eingaben verschiedener Personen beruht, sind Mehrparteienberechnungen ein klassischer Lösungsansatz. Im Kontext von Wahlverfahren sind die Personen die Wähler und die geheimen Eingaben die Wahlentscheidung, die die Wähler treffen. Das Ergebnis ist die Auszählung, wie viele Stimmen ein Kandidat bekommen hat. Es gibt Mehrparteienberechnungen, die bei einer

ehrlichen Mehrheit ein ehrliches Ergebnis garantieren. Spezielle Konstruktionen erlauben sogar einen noch größeren Anteil korrumpierter Teilnehmer.

Die Vorschläge, die in dieser Arbeit vorgestellt werden, verwenden jedoch keine Mehrparteienberechnungen, bei denen die Protokollteilnehmer die Wähler sind. Dazu müssten die Wähler (mehr oder weniger) gleichzeitig an der Berechnung teilnehmen. Wir wollen den Wählern jedoch nicht zumuten, dass sie sich entweder mehrfach oder über einen längeren Zeitraum hinweg an der Berechnung beteiligen müssen. Ein Wähler soll lediglich eine (geheime) Eingabe tätigen müssen. Die Mehrparteienberechnung zur Berechnung des Auszählungsergebnisses wird dann von einer anderen Gruppe – der Wahlleitung – ausgeführt.

Erstellung der Eingabe in die Mehrparteienberechnung Bleiben wir bei unserer Betrachtung aber zunächst bei der Abgabe der Stimme, bevor wir uns der Wahlleitung widmen: der *Erstellung* der geheimen Eingabe. Eine grundlegende Annahme, die üblicherweise gemacht wird, ist, dass die Maschinen (PCs), an denen die Stimme abgegeben und die Eingabe in die Mehrparteienberechnung erstellt wird, vertrauenswürdig sind. Wir halten diese Annahme im Allgemeinen für nicht haltbar.

Außerdem gilt, dass die Wahl des Wählers in ihrem Wert unmanipuliert, aber in ihrer Darstellung nicht unverändert in die Mehrparteienberechnung eingehen darf. Das ist offensichtlich, da sonst die Geheimhaltung gegenüber der Wahlleitung, die diese ausführt, nicht gewährleistet ist. Zusammen mit der Anforderung, dass der Wähler seine Stimme direkt[1] eingeben kann, folgt daraus jedoch auch, dass der Wähler der Maschine oder zumindest dem Teil davon, der diese Eingabe aus der Wahl des Wählers berechnet, vertrauen muss.

Wir werden mit Oblivious Bingo Voting in Kapitel 5.3 ein Verfahren für eine Wahlmaschine vorstellen, bei der wir den Bestandteil der Maschine, dem der Wähler vertrauen muss, auf ein Minimum reduziert haben.

Vorberechnungen Bei vielen kryptographischen Wahlverfahren ist zusätzlich zur Berechnung des Auszählungsergebnisses noch eine zentrale Vorberechnung notwendig, deren Ausgabe (teilweise) in die Stimmabgabe jedes einzelnen Wählers einfließt. Die Maschinen, die diese Vorberechnungen durchführen, und deren Administratoren müssen im Allgemeinen ebenfalls als vertrauenswürdig angenommen werden, da sie meist das Wahlgeheimnis anhand der ihnen bekannten Daten brechen können. Wie die Vorberechnungen das Wahlgeheimnis gefährden können, werden wir beispielhaft in Kapitel 5.3 anhand von Bingo Voting näher betrachten. Wir werden dort für das spezielle Verfahren auch eine Lösung vorschlagen, bei der die Vorberechnung und die Berechnung des Auszählungsergebnisses als Mehrparteienberechnung realisiert ist, bei der das Vertrauen auf der Vertrauenswürdigkeit eines beliebigen Mitglieds der Wahlleitung beruht.

[1] Mit *direkt* ist hier gemeint, dass er keinen seine Wahl repräsentierenden Code oder ein Chiffrat an die Maschine übergibt. Möchte er für A stimmen, so klickt er auf ein mit A beschriftetes Feld.

Wahlleitung Die Wahlleitung, die die Mehrparteienberechnungen zur Bestimmung des Auszählungsergebnisses dann letztlich ausführt, muss idealerweise so zusammengesetzt sein, dass jeder Wähler mindestens so vielen Mitgliedern vertraut, wie für die Garantie der Korrektheit notwendig ist. Eine etwas schwächere, aber vielleicht leichter zu erfüllende Forderung wäre, dass jeder Wähler davon überzeugt sein muss, dass sich keine Teilgruppe innerhalb der Wahlleitung bilden kann, die eine ausreichende Größe besitzt, um die Wahl zu manipulieren oder das Wahlgeheimnis zu brechen. Treten etwa zwei Kandidaten gegeneinander an, so ist davon auszugehen, dass sie nicht zusammenarbeiten, damit einer von ihnen gewinnt. Wäre dem so, ließe sich der Sinn der Wahl in Frage stellen. Der Einfachheit halber soll bei dieser Formulierung die Erpressungsfreiheit, die dem Wähler garantiert, sich frei für einen Kandidaten entscheiden können, für den Moment Teil der Korrektheit sein. Wir werden auf die Begriffe der Erpressungsfreiheit und Korrektheit im Weiteren noch genauer eingehen.

Wir werden Verfahren vorstellen, die auf drei Prinzipien beruhen:

1. Protokolle, deren korrekte Ausführung auch für Beobachter nachvollziehbar sind.
2. Eine Wahlleitung, bei der im Zweifel das Vertrauen in ein Mitglied ausreichend ist.
3. Einfache Komponenten, die nicht frei programmierbar sind oder sogar gänzlich ohne Rechenleistung auskommen.

Nach einem Überblick über die wichtigsten Grundbegriffe werden wir uns zwei wahlrelevanten Themen widmen:

Geheimhalten der Wahl vor der Wahlmaschine: Wahlmaschinen, die die Eingaben von Wählern lernen, ermöglichen nicht nur Erpressung, sondern auch das Erstellen von Zwischenergebnissen. Wir stellen in Kapitel 5.3 *Oblivious Bingo Voting* eine Erweiterung von Bingo Voting vor, in der die Komponenten der Wahlmaschine, die über Rechenleistung verfügen, nichts über die Wahl des Wählers lernen.

Öffentlich nachvollziehbares Aussortieren überschriebener Stimmen: In Onlinewahlverfahren ist es mangels der Wahlkabine notwendig, eine Ausweichstrategie gegen *Wählen unter Beobachtung* anzubieten. Dem Wähler die Option zu eröffnen, seine Stimme zu überschreiben, ist eine davon. Dies funktioniert jedoch nur, wenn der Angreifer (der auch Teil der Wahlleitung sein kann) keine Möglichkeit hat herauszufinden, ob der Wähler eine zweite Stimme abgegeben hat. Wir stellen in Kapitel 5.4 *Coercion Resistant Revoting* ein Verfahren vor, das das Aussortieren der überschriebenen Stimmen nachvollziehbar macht, ohne dass Informationen über das Wählerverhalten bekannt werden.

5.2.1. Fokus der Arbeit

An einen Wahlprozess werden verschiedene Anforderungen gestellt. Artikel 38 des Grundgesetzes fordert, dass „die Abgeordneten des Deutschen Bundestages [...] in allgemeiner,

unmittelbarer, freier, gleicher und geheimer Wahl gewählt“ werden. Die Bundeswahlordnung [75] und das Bundeswahlgesetz [29] beschreiben, wie eine solche Wahl organisiert und durchgeführt werden muss: Zentrale Bestandteile des Wahlablaufs sind die Wahlkabine und die Wahlurne. Sie sollen vor allem die Freiheit der Wahlentscheidung und die Geheimhaltung der Stimmen gewährleisten. Über die Durchführung der Stimmabgabe und -auszählung hinaus kann auch die Feststellung, wer wahlberechtigt ist, bereits zum Streitpunkt werden [113].

In dieser Arbeit betrachten wir nur die Vorgänge der Stimmabgabe (Oblivious Bingo Voting) und der Aussortierung ungültiger Stimmen (Coercion Resistant Revoting). Die korrekte *Initialisierung* der Wahl und die Interpretation der ausgezählten Stimmen haben eine große Bedeutung für den gewünschten Ablauf, stehen aber nicht im Fokus dieser Arbeit. Zur Initialisierung zählt unter anderem auch die Frage, wer wahlberechtigt ist, welche Kandidaten zur Wahl stehen, die Darstellung der Wahlmöglichkeiten auf dem Stimmzettel (oder Entsprechendem) oder wer – wenn nötig – zu einer vertrauenswürdigen Wahlleitung gehört. Die Interpretation der Auszählung – die beispielsweise über den Sieger oder die Verteilung der Sitze in einem Parlament entscheidet – ist nicht zwangsläufig eindeutig.

5.2.2. Geheimhaltung der Stimme, Belegfreiheit und Erpressungsresistenz

Die Anforderungen der freien und geheimen Wahl, wie sie im Artikel 38 des Grundgesetzes stehen, finden sich in den Begriffen der Geheimhaltung der Stimme, der Belegfreiheit und der Erpressungsfreiheit wieder. Während sich die Anforderung *geheim* durch die Geheimhaltung der Stimme abbilden und einfach formal definieren lässt, ist die freie Äußerung des Wählerwillens nicht so einfach zu fassen.

Der erste Versuch, dies zu erreichen – die *Belegfreiheit* –, verfolgt den Gedanken, dass der Wähler nicht in der Lage sein darf, beweisen zu können, für welchen Kandidaten er seine Stimme abgegeben hat.

Juels, Catalano und Jakobsson [73] haben drei Angriffe auf die Möglichkeit der freien Willensbekundung gefunden, die von der Definition der Belegfreiheit nicht abgedeckt werden. Dazu gehören die bei Präsenzwahlen immer mögliche Erpressung, der Wahl fernzubleiben (*abstention attack*), und die Abgabe aller Autorisierungsmerkmale, so dass der Angreifer für das Wahlsystem von dem legitimen Wähler nicht unterscheidbar ist (*simulation attack*). Diese beiden Angriffe sind insbesondere bei Onlinewahlverfahren relevant. Der dritte Angriff, den Wähler zu einer selbst dem Erpresser unbekannten Stimmabgabe zu zwingen (*randomization attack*), ist auch bei Präsenzwahlen prinzipiell möglich. Für die Belegfreiheit darf der Wähler nicht beweisen können, für welchen Kandidaten er eine Stimme abgegeben hat; eine Randomisierungsattacke hat zur Folge, dass der Angreifer prüfen kann, dass sich der Wähler an seine Vorgaben gehalten hat, die es dem Wähler (mit hoher Wahrscheinlichkeit) unmöglich machen, für den von ihm gewünschten Kandidaten zu stimmen. Für welchen Kandidaten diese Stimme ist, kann der

Angreifer im Allgemeinen nicht beeinflussen. Ein Verfahren, bei dem das möglich ist, ist beispielsweise Punchscan [55]. Ein Wahlsystem, das belegfrei und außerdem resistent gegen die drei genannten Angriffe ist, nennen Juels, Catalano und Jakobsson erpressungsresistent (*coercion resistant*).

Diese Definition zugrundelegend muss das System, mit dem der Wähler seinen Wählerwunsch zum Ausdruck bringt, vertrauenswürdig sein. Das sind bei den papierbasierten Präsenzwahlen der Stimmzettel und der Stift; bei computergestützten Wahlmaschinen muss meist das gesamte Computersystem dazugezählt werden. Durch den Einsatz von Code-Voting [33] lässt sich diese Information auf zwei Systeme im Sinne eines Secret-Sharings [108] aufteilen.

Während bei Präsenzwahlen aus den drei Angriffen der Erpressungsresistenz nur die Randomisierungsattacke eine größere Rolle spielt, müssen die anderen Attacken im Kontext von Onlinewahlverfahren betrachtet werden. Die fehlende Wahlkabine und andere Authentifikationsmethoden ermöglichen neue Angriffe. Juels, Catalano und Jakobsson verwenden das Konzept, dass sich der Wähler mit ungültigem Schlüsselmaterial für die Authentifizierung gegen Erpressungen wehren kann.

In der Literatur gibt es einen zweiten Ansatz, der ein anderes Angreifermodell annimmt: die Möglichkeit, seine zuvor abgegebene Stimme durch einen neuen Stimmzettel zu überschreiben (*Revoting*). Wir diskutieren im Folgenden beide Ansätze. In Kapitel 5.4 betrachten wir den Unterschied ausführlicher und erläutern, welche Anforderungen erfüllt werden müssen, damit sich Revoting als Ausweichstrategie eignet.

Erpressungsresistenz bei Onlinewahlverfahren

Um die Erpressungsresistenz bei Onlinewahlverfahren zu erreichen, muss es dem Wähler möglich sein, eine Stimme abzugeben, ohne dass dies der Angreifer feststellen kann. Es muss außerdem gewährleistet sein, dass der Angreifer selbst durch Ausübung von Druck nicht alle notwendigen Informationen erhalten kann, die ihm es ermöglichen würden, im Namen des Wählers eine Stimme abzugeben.

Während das erste Ziel von Juels, Catalano und Jakobsson nur bei einem eingeschränkten Angreifer erreicht wird – der Angreifer darf die an der Auszählung beteiligten Mitglieder der Wahlleitung nicht korrumpieren –, erreichen sie durch die Einführung von *Fake Credentials*, dass ein Angreifer nicht herausfinden kann, ob das ihm ausgehändigte Schlüsselmaterial gültig ist.

Fake Credentials Juels, Catalano und Jakobsson haben mit der Verwendung von *Fake Credentials* ein Verfahren vorgeschlagen, das bei richtiger Umsetzung die Anforderungen an ein erpressungfreies Wahlverfahren erfüllt. Die Idee der Fake Credentials basiert auf der Annahme, dass der Angreifer echte von gefälschten Authentifizierungsinformationen nicht unterscheiden kann. Da mit gefälschten Authentifizierungsinformationen abgegebene

Stimmen nicht gewertet werden, kann ein Wähler – so der Gedanke der Autoren – in einem unbeobachteten Moment mit den gültigen Authentifizierungsinformationen seine gewünschte Stimme abgeben. Muss ein Wähler unter Beobachtung wählen – was aufgrund der fehlenden Wahlkabine bei Onlinewahlverfahren nie ausgeschlossen werden kann –, so verwendet er statt seiner gültigen von ihm erfundene, ungültige Authentifizierungsinformationen. Wird ein Wähler unter Druck gesetzt, seine Authentifizierungsinformationen preiszugeben (*simulation attack*), kann er ebenfalls erfundene, ungültige statt seiner echten Daten aushändigen.

Neben den Anforderungen an die Umsetzung, die direkt aus den Zielen des Verfahrens folgen, ergeben sich auch automatisch einige Einschränkungen. Schließt man systeminterne Angreifer nicht grundsätzlich aus, so darf der Authentifizierungsprozess nicht verraten, welcher Nutzer sich authentifiziert. Auch darf einer Stimme, wie sie abgegeben wurde, nicht ansehbar sein, ob ihr gültige Authentifizierungsinformationen zugrunde liegen oder ob sie im späteren Verlauf der Auszählung gelöscht werden muss. In keinem Falle kann ein Wähler eine Rückmeldung darüber erhalten, ob seine Stimme in das Wahlergebnis eingehen wird.

Diese Anforderung widerspricht dem *Basissatz von Sicherheitsanforderungen an Online-Wahlprodukte* der Common Criteria [13]. Darin wird gefordert, dass der Wähler „eine zutreffende Rückmeldung über die Erlaubnis bzw. Verweigerung und den Erfolg bzw. Misserfolg seiner Stimmabgabe“ erhält.

Revoting Im Gegensatz zu dem Ansatz, bei Erpressung gefälschte Authentifizierungsinformationen zur Abgabe einer ungültigen Stimme zu verwenden, kann im Falle von Revoting unter Beobachtung eine gültige Stimme abgegeben werden, die im Anschluss wieder überschrieben werden kann. Das ermöglicht es dem Wahlsystem, stets korrekte Rückmeldungen über die Gültigkeit der abgegebenen Stimme zu machen. Jedoch sind die (jeweils zuletzt) abgegebenen Stimmen erst final, wenn die Wahlphase beendet ist. Jede mit gültigen Authentifizierungsinformationen abgegebene Stimme kann, solange die Wahlphase noch nicht beendet ist, noch durch die erneute Abgabe einer neuen Stimme des gleichen Wählers überschrieben und damit ungültig werden. Ebenso ist ein Authentifizierungsverfahren zu wählen, das eine Weitergabe der Authentifizierungsinformationen unmöglich oder zumindest unwahrscheinlich macht. Welches Verfahren sich hierzu eignet, hängt sicherlich stark von den Gegebenheiten des jeweiligen Landes oder der jeweiligen Region ab.

Diese Art der Ausweichstrategie stellt ebenso wie das Verwenden von gefälschten Authentifizierungsinformationen eine Reihe von Anforderungen an die Umsetzung des Authentifizierungs- und Auszählungsverfahrens. So darf einem Stimmzettel, bevor die überschriebenen Stimmen gelöscht sind, nicht anzusehen sein, von welchem Wähler er stammt. Sonst ist es einem Erpresser möglich festzustellen, ob ein Wähler seine unter Beobachtung abgegebene Stimme überschrieben hat. Ebenso darf es nicht möglich sein herauszufinden, wie oft ein Wähler seine Stimme überschrieben hat, auch wenn seine Identität pseudonymisiert ist.

Des weiteren dürfen weder die Information, zu welcher Uhrzeit die gültigen Stimmen abgegeben wurden, noch andere Daten, aufgrund derer auf das Wahlverhalten geschlossen werden kann, bekannt werden.

Ein Verfahren, das diese Kriterien erfüllt, wird in Kapitel 5.4 vorgestellt.

5.2.3. Authentifizierung

Jedes Wahlverfahren benötigt eine Form der Authentifizierung des Wählers. Selbst die Wahlen, bei denen jeder (Anwesende) berechtigt ist, eine Stimme abzugeben, müssen garantieren können, dass maximal eine Stimme jedes Wählers in die Auszählung mit eingeht. Die Authentifizierung eines Wählers beweist im Idealfall nur, dass der Wähler berechtigt ist, diese (wohlgeformte) Stimme abgeben zu dürften. Eine Aussage über die Identität des Wählers oder den Inhalt seines Stimmzettels dürfen – je nach Definition – nicht aus den Authentifikationsdaten hervorgehen. Da Authentifizierungsverfahren nicht die Identität einer Person feststellen, sondern lediglich prüfen können, ob die Person über das Wissen, die Hardware oder die Eigenschaft der Person verfügt, die zuvor als notwendig definiert und für die Person festgelegt wurden, ist die Möglichkeit, diese Merkmale (nicht) weitergeben zu können, von zentraler Bedeutung für die Aussagekraft, die ein solches Verfahren haben kann. Ein Authentifizierungsvefahren kann sich prinzipiell auf die folgenden Mechanismen stützen:

- Kenntnis einer Information: zum Beispiel Kenntnis eines Passworts
- Besitz: zum Beispiel Besitz einer Smartcard, eines (neuen) Personalausweises,...
- Merkmal/Eigenschaft: zum Beispiel biometrische Eigenschaft (Fingerabdruck)

Nun scheint die Verwendung eines Merkmals eine gute Lösung zu sein, um ein Personifizieren durch einen Angreifer zu verhindern. Jedoch nimmt es einem Wähler die Möglichkeit, darüber zu lügen, was für Fake-Key-Mechanismen wichtig ist, und der Angreifer weiß, wo er das Authentifizierungsmerkmal findet.

Zwei Eigenschaften sollten erreicht werden:

- Der Angreifer darf sich nicht als Wähler ausgeben können. Das bedeutet, dass er die Authentifizierungsmerkmale nicht (alle) in Erfahrung bringen können darf.
- Der Angreifer darf es dem Wähler nicht unmöglich machen, sich zu authentifizieren. Das bedeutet, dass er dem Wähler keines der Authentifizierungsmerkmale entwenden können darf.

Während beim Entwurf kryptographischer Präsenzwahlverfahren die Kontrolle der Wahlberechtigung oft vom klassischen Papierwahlverfahren übernommen wird, müssen für Onlinewahlverfahren neue Methoden gefunden werden, die den Anforderungen entsprechen. Dabei können auch Anforderungen berücksichtigt werden, die bei Präsenzwahlen unmöglich zu erfüllen sind: Durch die Notwendigkeit, persönlich zu einer Präsenzwahl

zu erscheinen, ist es einem Angreifer immer möglich festzustellen, welcher Wähler eine Stimme abgibt. Der Wähler kann sich nicht anonym authentifizieren. Des weiteren schreibt die Bundeswahlordnung [75] in Paragraph 56, Absatz 4, vor, dass im Wählerverzeichnis vermerkt wird, wenn ein Wähler seinen Stimmzettel in die Urne eingeworfen hat. Das ist notwendig, um doppelte Stimmabgaben zu verhindern. Onlinewahlverfahren können hier über den Standard von Präsenzwahlen hinausgehen und doppelte Stimmabgaben verhindern, ohne dass bekannt wird, welcher Wähler seine Stimme abgegeben hat. Eine weitere Anforderung, die Juels, Catalano und Jakobsson an erpressungsfreie Wahlverfahren haben [73], ist eine Vermeidungsstrategie gegen Angriffe, bei denen die Authentifizierungsdaten des Wählers kopiert (gestohlen) werden. Angewendet auf die übliche Authentifizierungsmethode bei Präsenzwahlen bedeutet das, dass die Person, die die Authentifizierung prüft, den Wähler persönlich kennen muss, um überprüfen zu können, ob der vorgezeigte Ausweis tatsächlich dem Wähler gehört. Ist dem Prüfenden der Wähler nicht bekannt, ist es durchaus möglich, dass sich statt des legitimierten Wählers eine Person mit ähnlichem Aussehen autorisiert. Wie bereits erwähnt, wird bei den Authentifizierungsmethoden im Allgemeinen nicht die Identität der Person oder ihre Berechtigung festgestellt, sondern nur geprüft, ob die zuvor als hinreichend festgelegten Bedingungen erfüllt sind (zum Beispiel das Verfügen über ein hinreichend echt aussehendes Ausweisdokument, bei dem bestimmte Merkmale übereinstimmen).

5.2.4. Verifizierbarkeit

Die Korrektheit des Wahlergebnisses muss für den Wähler nachvollziebar sein. Kremer, Ryan und Smyth unterscheiden zwischen drei Stufen der Verifizierbarkeit [81]. Zunächst soll ein Wähler prüfen können, dass seine Stimme, so wie er sie abgeben wollte, auch in das Auszählungsergebnis eingegangen ist (*individuelle Verifizierbarkeit*). Zusätzlich soll prüfbar sein, dass sich das Wahlergebnis aus den öffentlichen Daten (Bulletin Board) ableitet (*universelle Verifizierbarkeit*). Kann ein Wähler außerdem prüfen, dass jede Stimme, die in das Wahlergebnis eingeht, von einer wahlberechtigten Person abgegeben wurde und jede wahlberechtigte Person maximal eine Stimme abgegeben hat, sprechen Kremer, Ryan und Smyth von Verifizierbarkeit der Stimmberechtigung (*eligibility verifiability*).

Hier eröffnen uns Onlinewahlen neue Möglichkeiten. Bei einer Präsenzwahl bleibt dem Wähler nur zu beobachten, ob jeder, der eine Stimme abgibt, von der Wahlleitung auch zu einem Eintrag in der Liste der gültigen Wähler zugeordnet wird. Bei Onlinewahlen ist es je nach Verfahren möglich, dass für jede Stimme geprüft wird, ob ihre Authentifizierungsdaten mit einem Eintrag aus der Wählerliste übereinstimmen. Das ist möglich, ohne dass das Wahlgeheimnis gefährdet wird. Je nach Umsetzung ist es sogar möglich, einen solchen Abgleich durchzuführen, ohne dass ein Wähler dazu gezwungen werden kann, der Wahl fernzubleiben (*Abstention Attack*). Wir werden auf diesen Punkt insbesondere in Kapitel 5.4 *Coercion Resistant Revoting*, wenn wir über das Aussortieren überschriebener Stimmen sprechen, noch genauer eingehen.

5.2.5. Vorbereitung der Wahl

Wie schon zuvor erwähnt, muss das Wahlverfahren korrekt initialisiert werden. Das bedeutet unter anderem, dass es eine Liste der gültigen Wähler geben muss (das *Wählerverzeichnis*), dass die Kandidaten feststehen und dass eine gegebenenfalls auszuführende Vorberechnung korrekt ausgeführt wurde. Diese Arbeit befasst sich lediglich mit der Vorberechnung, jedoch nicht mit den Prozessen zur Sicherstellung der Korrektheit der Wahllisten.

Für das in Kapitel 5.3 vorgestellte Verfahren *Oblivious Bingo Voting* werden wir uns ausführlich der korrekten Vorberechnung widmen: Wir schlagen ein Verfahren vor, das es einem Angreifer unmöglich macht, Informationen aus der Vorberechnung zu gewinnen oder deren Ergebnisse unbemerkt zu manipulieren, wenn die Wahlleitung auch nur ein ehrliches Mitglied hat.

5.2.6. Auszählungsergebnis und Wahlergebnis

Die Begriffe *Auszählungsergebnis* und *Wahlergebnis* werden oft synonym verwendet, unterscheiden sich jedoch genau genommen.

Das Auszählungsergebnis enthält die Information, wie viele Stimmzettel es für welchen Kandidaten gibt. Oder – etwas allgemeiner formuliert –, wie viele Wähler ihren Stimmzettel von allen zur Verfügung stehenden Möglichkeiten jeweils identisch ausgefüllt haben.

Das Wahlergebnis ist eine Interpretation des Auszählungsergebnisses. Aus dem Auszählungsergebnis wird beispielsweise der Gewinner der Wahl abgeleitet. Die Existenz unterschiedlicher Arten der Mehrheit (einfache Mehrheit, Zweidrittelmehrheit), damit eine Entscheidung Gültigkeit erlangt, zeigt die Notwendigkeit unterschiedlicher Auslegungen eines Auszählungsergebnisses. Dass diese Interpretations- oder Ableitungsvorschrift aber nicht immer offensichtlich oder eindeutig ist, zeigt das Urteil des Bundesverfassungsgerichts vom 25. Juli 2012 [27], nach dem die Regelung zu Überhangmandaten, wie sie bisher angewendet wurde und den Effekt des negativen Stimmgewichts ermöglichte, nicht zulässig ist. Dass die Vorschriften sehr komplex werden können, zeigen Präferenzwahlen, bei denen Wähler die Wahlmöglichkeiten in eine von ihnen präferierte Reihenfolge bringen.

Kryptographische Wahlverfahren interessieren sich in der Regel nur für die Auszählung. Die Interpretation obliegt im Zweifelsfall der Rechtsprechung und basiert auf einer Vereinbarung, die vor der Wahl getroffen wird. Die Interpretation ist jedoch insbesondere dann für das Wahlverfahren von Bedeutung, wenn Methoden zur Erfüllung der Erpressungsfreiheit über die Auszählung hinausgehen müssen. Die Auszählung von Rangfolgen- und Präferenzwahlen sind Beispiele dafür; ebenso sind Wahlen, bei denen der Wähler einen Namen frei auf den Stimmzettel eintragen kann, der nicht in der Kandidatenliste enthalten ist, ein Beispiel, wie Anforderungen an Wahlsysteme die Komplexität des Problems erhöhen können, Wahlen erpressungsfrei zu gestalten.

5.3. Oblivious Bingo Voting

Die digitale Umsetzung von ursprünglich analogen Prozessen senkt die Kosten und bei korrekter Funktionsweise auch die Fehlerquote. Das (begründete) Vertrauen in die digitale Umsetzung herzustellen, das für alle Systeme notwendig ist, die persönliche Daten verarbeiten oder sicherheitskritisch sind, stellt eine ebenso große Herausforderung dar, wie die Systeme robust gegen intelligente Angriffe zu gestalten. Schließlich skaliert nicht nur ihre Leistungsfähigkeit gut; auch das Potential, Schaden anzurichten, steigt mit dem Grad der Digitalisierung und Vernetzung.

Um das Vertrauen der Nutzer zu gewinnen, sind zwei Aspekte gleichermaßen wichtig. Einerseits müssen die Protokolle den Schutz bieten, der für den Einsatzzweck notwendig ist. Andererseits müssen diese Protokolle auch korrekt umgesetzt werden.

Die korrekte Umsetzung beinhaltet auch, dass die vorgesehene Soft- und Hardware verwendet wird; davon kann sich der Administrator, der das System aufsetzt und wartet, am ehesten überzeugen. Für den Wähler ist das im Allgemeinen nicht nachzuvollziehen. (Bei Hardware reicht selbst eine Analyse auf der Gatterebene nicht aus. Becker, Regazzoni, Paar und Burleson haben gezeigt [14], dass Manipulationen unterhalb der Gatterebene die Funktionsweise eines Bausteins gezielt beeinflussen können. Sie beschreiben, wie sie eine Modifikation durch die Änderung der Polarisierung der Dotierung erreichen konnten.)

Wir versuchen den Teil, dem der Nutzer Vertrauen entgegen bringen muss, auf möglichst einfache Hardware zu reduzieren, die nach der Wahl überprüft werden kann.

Umprogrammierbare Maschinen sind immer anfällig gegen Schadsoftware.
Komplexe Programme sind nie fehlerfrei.

Frei programmierbare und komplexe Teile
müssen als (passiv/aktiv) korrumpiert angenommen werden.

Die grundlegende Überlegung, von der wir ausgehen, ist, dass Manipulationen in Software rückgängig gemacht werden können; ein Virus kann sich nach getaner Arbeit selbst löschen. Daher können Manipulationen durch eine Kontrolle nach dem Einsatz der Maschine nicht mehr zuverlässig gefunden werden. Manipulationen in Hardware zu entfernen ist nur durch einen Austausch des Bauteils möglich – ein Prozess, der durch einen Menschen beobachtet werden kann. Aus diesem Grund betrachten wir in diesem Abschnitt beispielhaft ein System, in dem wir uns – um die Sicherheit zu steigern – auf nicht umprogrammierbare Komponenten stützen. In diesem Beispiel verfolgen wir das Ziel, eine Wahlmaschine so zu gestalten, dass für jeden frei programmierbaren Bestandteil der Maschine die Eingabe des Nutzers verborgen bleibt.

Nicht nur der Wähler muss der Maschine vertrauen können. Die Wahlleitung muss sich sicher sein, dass ein Wähler nicht zwei Stimmen abgeben oder bereits abgegebene Stimmen manipulieren kann. Das ergibt sich in unserem Fall jedoch bereits aus der Konstruktion.

Bevor wir in Abschnitt 5.3.4 erläutern, wie wir die Ziele erreichen, gehen wir zunächst in Abschnitt 5.3.1 auf verwandte Arbeiten ein. Die Grundlagen, die wir für unsere Konstruktion benötigen, erläutern wir in Abschnitt 5.3.2. Abschnitt 5.3.3 enthält einen Überblick über die verwendete Notation.

5.3.1. Verwandte Arbeiten

Arbeiten über kryptographische Wahlverfahren lassen sich grob in zwei Bereiche unterteilen: Präsenzwahlverfahren und Onlinewahlverfahren. Da diese Variante von Bingo Voting den Präsenzwahlverfahren zuzuordnen ist, gehen wir hier nur auf andere Vorschläge für kryptographische Präsenzwahlverfahren ein.

- In PEVS [12] erreichen Based et al. Erpressungsfreiheit, indem sie dem Wähler die Möglichkeit geben, eine unbegrenzte Zahl Schlüsselpaare zu erzeugen. Diese werden durch die Wahlleitung blind signiert. Der Angreifer kann nicht erkennen, welches Schlüsselpaar für die Wahl verwendet wurde. Dafür muss die Wahlmaschine jedoch vertrauenswürdig sein.
- Split-ballot voting [86] adressiert die Anforderung, nicht einer einzigen Wahlleitung vertrauen zu müssen. Das Schema verwendet keine Wahlmaschine. Der Wähler muss eine Addition auf Papier durchführen, um seine Stimme aufzuteilen. Der Wähler kann dementsprechend nicht durch einen einfachen Tastendruck – oder eine vergleichbare Operation – seine Stimme zum Ausdruck bringen; übernimmt eine Maschine diese Aufgabe, muss diese wieder als vertrauenswürdig eingestuft sein. Dennoch erreichen sie, ähnlich unserem Schema, langfristige Sicherheit durch die Verwendung von Commitments.
- BeleniosRF [41] ist eine Erweiterung von Helios [7], um Belegfreiheit zu erreichen. Dazu verwenden Cortier et al. *Signaturen auf randomisierbaren Chiffraten*, die von Blazy et al. [18] eingeführt wurden. Die Wahlmaschine ist dadurch in der Lage, das Chiffrat der Stimme und die dazugehörige Signatur zu rerandomisieren. So kann der Wähler seine Stimme nicht wiedererkennen. Eine passiv korrumpierte Wahlmaschine lernt dennoch die Wahlentscheidung. Die öffentlichen Daten sind zudem nicht langfristig sicher, da Chiffrate veröffentlicht werden.
- Im Wahlverfahren Scantegrity II [36, 37, 31], das bei der Kommunalwahl in Takoma Park 2009 eingesetzt wurde, kann der Wähler anhand eines Codes überprüfen, ob seine Stimme korrekt gezählt wurde. Der Wähler markiert mit einem speziellen Stift ein Feld hinter dem Kandidaten seiner Wahl. Dort erscheint für eine begrenzte Zeit der Code, den er sich für die Kontrolle notieren muss. Der Beleg wird anschließend gescannt. Alle für die Stimmzettel benötigten Zufallswerte werden anhand eines auf die Wahlleitung verteilten *Seeds* auf einem zentralen Rechner erzeugt, der die Stimmzettel erzeugt und zur Erstellung an einen Druckservice sendet. Die Angriffspunkte für das Brechen des Wahlgeheimnisses sind einerseits der zentrale Rechner, der alle verwendeten Zufallswerte kennt, der Druckservice und der Scanner, der die Wahlentscheidung des Wähler sieht. In unserem Verfahren ist weder ein

Vertrauen in eine zentrale Instanz notwendig noch in eine durch Software gesteuerte Komponente der Wahlmaschine.

Die in diesem Kapitel vorgestellten Arbeiten bauen auf dem Verfahren *Bingo Voting* [20] auf. Die Ergebnisse wurden gemeinsam von Dirk Achenbach, Anne Borcherding, Bernhard Löwe, Jörn Müller-Quade und Jochen Rill in den Papieren *Oblivious Voting—Hiding Votes from the Voting Machine in Bingo Voting* [2] und *Towards Realising Oblivious Voting* [3] veröffentlicht. Die Weiterentwicklung der Architektur und der Protokolle basieren auf Ideen des Autors dieses Dokuments. Die Aufteilung der Maschine, der Entwurf der vertrauenswürdigen Hardwarebausteine, die *blinden Commitments* und die Protokolle während der Vorberechnung, der Wahl und der Auszählung wurden von mir entworfen. Der Sicherheitsbeweis und die Umsetzung wurden hauptsächlich von meinen Co-Autoren und wissenschaftlichen Hilfskräften erstellt.

5.3.2. Grundlagen

In diesem Abschnitt stellen wir die Grundlagen vor, die für unsere Konstruktion notwendig sind. Da unser Ansatz auf Bingo Voting aufbaut, widmen wir uns zunächst der ursprünglichen Variante von Bingo Voting. Im Anschluss stellen wir mit *blinden Commitments* und *Physical Oblivious Transfer* zwei Bausteine vor, die eine zentrale Rolle spielen.

Unser Ansatz baut auf dem Protokoll *Bingo Voting* auf, weshalb wir hier einen kurzen Überblick über das ursprüngliche Verfahren geben.

Überblick über die ursprüngliche Version Bingo Voting

Bingo Voting ist ein Protokoll für Präsenzwahlverfahren. Es garantiert auch bei korrumpierter Wahlmaschine die Korrektheit des Wahlergebnisses. Für die Erpressungsfreiheit muss die Maschine im ursprünglichen Entwurf selbst unkorrumpiert sein.

Die Grundlagen des ursprünglichen Bingo-Voting-Entwurfs haben wir in unsere Umsetzung übernommen. Insbesondere bleiben die Daten, die veröffentlicht werden, gleich.

Die Wahlmaschine besteht aus drei Komponenten (siehe Abbildung 5.1):

- einer frei programmierbaren Maschine, die über eine Eingabemöglichkeit verfügt,
- einem vertrauenswürdigen, unkorrumpierten Zufallszahlengenerator,
- einem Drucker, der den Beleg für den Wähler druckt.

Da Bingo Voting als Eingabe lediglich die Wahl des Wählers erhält, kann es als Ergänzung zu jedem System genutzt werden, das die Wahl (in digitaler Form) lernt. Denkbar ist ein klassisches Papierwahlsystem, das die Stimme mit Hilfe eines Scanners digitalisiert; nach dem Scanvorgang fällt der Stimmzettel in eine Urne. Ein Wähler erhält als Ergebnis seiner Stimmabgabe einen Beleg, auf dem jeder Partei eine mehrstellige Zufallszahl zugeordnet ist (siehe Abbildung 5.2). Da die Zufallszahlen gleichverteilt gezogen werden und für sich

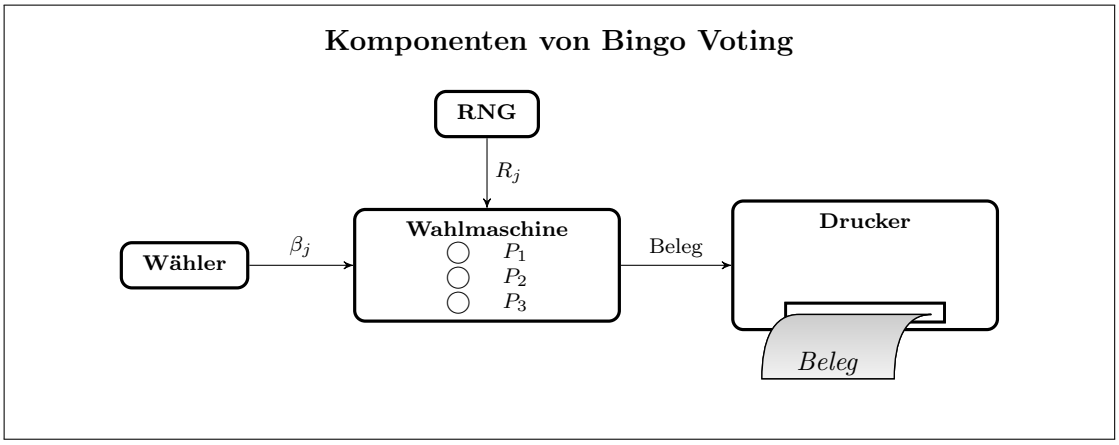

Abbildung 5.1.: Übersicht über die Komponenten in Bingo Voting: Der Wähler übergibt seine Stimme β_j an die Wahlmaschine. Nachdem der Zufallszahlengenerator eine frische Zufallszahl gezogen hat, zeigt er diese an und sendet sie an die Wahlmaschine. Diese erstellt einen Beleg und sendet diesen an den Drucker.

genommen keinen Bezug zu den ihnen zugeordneten Wahlmöglichkeiten haben, gibt der Beleg ohne Kenntnis der Vorberechnung und der internen Abläufe der Maschine keine Auskunft, welche Wahl der Wähler getroffen hat. Im Falle des gewählten Kandidaten stammt die Zufallszahl vom Zufallszahlengenerator, der für jede Stimmabgabe eine neue Zufallszahl generiert und dem Wähler anzeigt. Dass seine Stimme korrekt angenommen wurde, überprüft der Wähler, indem er die Zufallszahl *seines Kandidaten* auf dem Beleg mit der Zufallszahl auf dem Zufallszahlengenerator vergleicht. Für alle anderen Kandidaten steuert die Wahlmaschine die Zufallszahlen bei, die wir – im Unterschied zur *frischen Zufallszahl* des RNG – *Füllstimmen* nennen. Die Füllstimmen werden in der Vorbereitungsphase generiert und in der Maschine gespeichert. Für jeden Kandidaten gibt es gleich viele Füllstimmen; jede Füllstimme ist genau einem Kandidaten zugeordnet und wird nur einmal verwendet. Das Auszählungsergebnis lässt sich anhand der nicht benötigten Füllstimmen pro Kandidat ablesen. Damit der Beleg Aussagekraft hat, muss für jeden Beleg bewiesen werden, dass genau eine der Zufallszahlen frisch gezogen ist und es sich bei allen anderen um Füllstimmen handelt. Außerdem muss jede Füllstimme entweder auf einem Beleg oder als Stimme für den ihr zugeordneten Kandidaten Verwendung finden.

In den folgenden drei Abschnitten betrachten wir die drei Phasen der Vorbereitung, der Wahl und der Auszählung genauer. Dabei gehen wir von einer Wahl mit n_P Kandidaten und n_E Wählern aus.

Vorbereitung Vor der Wahl werden für jeden Kandidaten p_i genau n_E Zufallszahlen $\{N_{i,j}\}_{j=1}^{n_E}$ gezogen. Aus den Tupeln $(p_i, N_{i,j})$ werden (unconditionally hiding) Commitments $c_{i,j} = \texttt{COM}((N_{i,j}, p_i))$ berechnet. Die Commitments werden permutiert und

veröffentlicht. Im Anschluss daran wird bewiesen, dass alle Kandidaten dieselbe Anzahl Füllstimmen erhalten haben.

Die Berechnungen werden von der Wahlleitung oder der Wahlmaschine ausgeführt.

Wahlphase Um seine Stimme abzugeben, übermittelt der Wähler seine Wahl β_j an die Wahlmaschine. Auf welche Weise das geschieht (Tastendruck, papierbasiertes Verfahren mit Scanner, ...), spielt dabei keine Rolle. Der Zufallszahlengenerator erzeugt eine neue Zufallszahl R_j[2]. Der Zufallszahlengenerator zeigt dem Wähler die neue Zufallszahl an und sendet sie an die Wahlmaschine. Die Wahlmaschine erstellt ein Commitment auf das Paar (β_j, R_j) und speichert es. Zur Erstellung des Beleges (siehe Abbildung 5.2) verwendet die Wahlmaschine für den gewählten Kandidaten die vom Zufallszahlengenerator erzeugte frische Zufallszahl R_j; für alle anderen Kandidaten verwendet sie eine noch ungenutzte der vorberechneten Füllstimmen. Auf diese Weise wird für jeden nicht gewählten Kandidaten eine Füllstimme mehr verbraucht als für den gewählten Kandidaten. Der Wähler überprüft, bevor er die Wahlkabine verlässt, ob die Zahl auf dem Beleg neben dem gewählten Kandidaten mit der auf dem Zufallszahlengenerator übereinstimmt.

Außerhalb der Wahlkabine lässt sich anhand des Beleges nicht mehr nachvollziehen, welcher Kandidat gewählt wurde, da die Zufallszahl des Zufallszahlengenerators nicht von den Füllstimmen zu unterscheiden ist.[3] Für den Wähler, der die Zufallszahl auf dem Zufallszahlengenerator gesehen hat, ist der Beleg jedoch aussagekräftig.

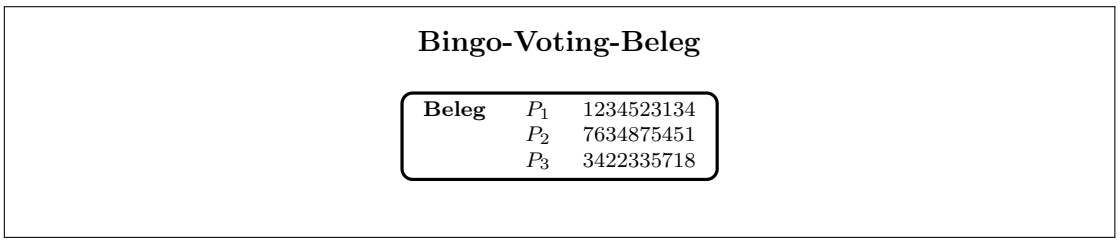

Abbildung 5.2.: Die Zahlen, die auf den Beleg neben die Kandidaten gedruckt sind, sind gleichverteilt zufällig gezogen. Eine von ihnen wurde in der Gegenwart des Wählers frisch gezogen und ihm angezeigt.

Auszählung Nach der Wahl veröffentlicht die Wahlmaschine gemeinsam mit den Korrektheitsbeweisen das Auszählungsergebnis. Das Auszählungsergebnis kann die Maschine aus der Zahl der vorbereiteten Füllstimmen pro Kandidat (n_E), der abgegebenen Stimmen (n_β) und den pro Kandidat p_i verbrauchten Füllstimmen (n_i) bestimmen: Unter Berücksichtigung der Wahlbeteiligung steht jede nicht verbrauchte Füllstimme des Kandidaten p_i

[2] Die Zufallszahlen, die der Zufallszahlengenerator erzeugt, sind ununterscheidbar von denen, die in der Vorberechnung erzeugt wurden.

[3] Das gilt nicht für den RNG und den Ersteller der Füllstimmen und der dazugehörigen Commitments. Diese können zwischen Füllstimmen und frischer Zufallszahl trivialerweise unterscheiden, da sie die frische Zufallszahl respektive alle Füllstimmen kennen. Diese Daten können auch nicht direkt nach der Erstellung gelöscht werden, da sie zum Öffnen der Commitments benötigt werden.

für eine Stimme für den Kandidaten p_i: $n_E - n_\beta - n_i$. Für die Beweise werden die folgenden Daten veröffentlicht:

- alle Belege,
- ungenutzte Füllstimmen mit den dazugehörigen Commitments,
- nicht-interaktive Zero-Knowledge-Beweise, dass die Füllstimmen der bisher nicht geöffneten Commitments für Belege verwendet wurden.

Aus der Zahl der Belege und der geöffneten Commitments der ungenutzten Füllstimmen kann das Auszählungsergebnis nachvollzogen werden. Nun muss nachgewiesen werden, dass die übrigen Füllstimmen, deren Commitments noch nicht geöffnet wurden, auf Belegen Verwendung fanden. Betrachten wir den Prozess für einen Beleg: Zu diesem Beleg gehören $n_P - 1$ in der Vorbereitung veröffentlichte Commitments. Zu welchen Kandidaten diese gehören, wird durch die Permutation vor der Veröffentlichung verdeckt. Diese $n_P - 1$ Commitments werden gemeinsam mit dem während der Wahlphase erstellten Commitment mit Hilfe eines verifizierbaren Shuffles rerandomisiert, permutiert und geöffnet.

Um die Auszählung durchführen zu können, muss der Ersteller der Beweise Kenntnis darüber haben, welche Commitments unverwendete Füllstimmen enthalten und welche Commitments zu Füllstimmen gehören, die auf den Belegen Verwendung gefunden haben. Das befähigt ihn auch, das Wahlgeheimnis zu brechen. Wenn die Instanz, die die Füllstimmen und die dazugehörenden Commitments erstellt hat und daher zwischen frischer Zufallszahl und (ihr bekannter) Füllstimme unterscheiden kann, auch die Auszählung und die Erstellung der Beweise übernimmt, führt das zu keinem neuen Angriffsvektor. Es zeigt aber, dass in dieser Form des Protokolls für die Auszählung Wissen notwendig ist, das das Wahlgeheimnis gefährdet.

Annahmen Damit Bingo Voting seine selbst gesteckten Ziele erreicht, müssen einige Annahmen erfüllt sein. Die Personen/Computer, die an der Vorberechnung beteiligt sind, müssen unkorrumpiert sein. Sie haben eine Vielzahl an Möglichkeiten, die Geheimhaltung der Wählerstimmen zu brechen.

Die Wahlmaschine selbst muss vertrauenswürdig sein, da sie sowohl die Füllstimmen kennt als auch die Wahl des Wählers lernt. Sie wäre also nicht nur in der Lage, die Geheimhaltung der Wählerstimmen zu brechen, sondern auch während der Wahl die aktuelle Statistik der bisher abgegebenen Stimmen anzuzeigen.

Der Zufallszahlengenerator muss ebenfalls vertrauenswürdig sein. Er kennt die frischen Zufallszahlen, womit er das Wahlgeheimnis brechen könnte. Zudem muss er die Zufallszahlen echt zufällig wählen, um einer Wiedererkennung vorzubeugen. Auch darf er keine Informationen von der Wahlmaschine entgegennehmen; in Zusammenarbeit mit der Wahlmaschine könnte die Stimme des Wählers geändert werden, ohne dass der Wähler dies feststellen kann.

Die Einhaltung der Ablaufreihenfolge ist für die Erpressungsfreiheit ebenso wichtig. Eine Randomisierungsattacke ist leicht möglich, wenn der Zufallszahlengenerator die frische Zufallszahl vor der Stimmabgabe anzeigt.

Oblivious Bingo Voting Die später vorgestellte Variante von Bingo Voting löst die zwei Schwächen des ursprünglichen Entwurfs. Daher geben wir hier bereits einen kurzen Ausblick auf die verwendeten Ansätze.

Zum einen verhindern wir, dass ein frei programmierbarer[4] Baustein die Wahl des Wählers in Erfahrung bringen kann. Zum anderen gestalten wir die Vorberechnung der Wahl so, dass kein Protokollteilnehmer die Füllstimmen kennt. Nur der Drucker erhält die Zahlen, die auf den Beleg gedruckt werden müssen; er kann aber zu diesem Zeitpunkt bereits nicht mehr zwischen Füllstimme und frischer Zufallszahl unterscheiden. Daraus folgt, dass während der Wahl keine Statistik über die bisher abgegebenen Stimmen erstellt werden kann und dass die Instanz, die die Auszählung vornimmt und deren Korrektheitsbeweis erstellt, keine Information über das Wahlergebnis besitzen muss.

Diese Ziele erreichen wir durch den Einsatz von Bausteinen, die nicht frei programmierbar sind und nur sehr wenig und nicht-permanenten Speicher besitzen, und solchen, die ganz ohne Recheneinheit auskommen. Wir sind der Meinung, dass die Sicherheit vieler Systeme deutlich gesteigert werden könnte, wenn an strategisch geeigneten Stellen Hardwarebausteine ohne Rechenleistung oder zustandslose Bausteine mit nicht veränderbarer, einfacher Programmierung verbaut würden.

Wahlleitung

Wie in herkömmlichen, papierbasierten Wahlverfahren wird auch bei kryptographischen Wahlen ein Gremium benötigt, das die Wahl organisiert. Eine schlecht zusammengesetzte Gruppe eröffnet Angriffsflächen. Die Größe oder Art der Zusammensetzung darf nicht durch das eingesetzte Wahlverfahren eingeschränkt sein. Gelingt es, eine Gruppe zusammenzustellen, bei der davon ausgegangen werden kann, dass mindestens eins der Mitglieder nicht mit den anderen gemeinsam einen Angriff durchführt, kann die Sicherheit eines Verfahrens darauf aufgebaut werden.

Die Sicherheit des Verfahrens darf nicht auf der Vertrauenswürdigkeit einer bestimmten Person oder Maschine beruhen.
Von der Zusammensetzung der Wahlleitung – deren Mitglieder alle dieselbe Aufgabe haben – ist das Vertrauen in die Wahl abhängig.

[4] Genauer: Der Baustein, der ohne Zusammenarbeit mit anderen Bausteinen etwas über die Wahl herausfinden kann, verfügt weder über Speicher noch über Rechenleistung.

Blinde Commitments

Für unsere Konstruktion benötigen wir einen Baustein, den wir *blindes Commitment* nennen. Bevor wir erklären, was blinde Commitments sind und wie sie technisch funktionieren, möchten wir zunächst motivieren, zu welchem Zweck wir sie einsetzen wollen.

Wir haben bereits das Problem angesprochen, dass die Wahlmaschine das Wahlgeheimnis gefährden kann, da sie die Füllstimmen kennt. In unserer Konstruktion soll sie nur noch mit Chiffraten in Berührung kommen.

Daten, die das Wahlgeheimnis brechen, dürfen keinem einzelnen Menschen und keiner einzelnen Maschine bekannt sein.
Die Füllstimmen dürfen niemandem vor dem Zeitpunkt ihrer Veröffentlichung laut Protokoll bekannt sein.

Gleichzeitig sollen sich die Daten, die veröffentlicht werden, nicht vom ursprünglichen Entwurf unterscheiden; wir stützen uns für die Korrektheit weiterhin auf vor der Wahl veröffentlichte Commitments der Füllstimmen. Eine dritte Instanz zu bestimmen, die die Füllstimmen erzeugt, die Commitments erstellt und die Füllstimmen verschlüsselt an die Wahlmaschine weitergibt, würde das Problem nur verschieben. Unser Ziel ist, dass – unter der Annahme, dass mindestens $n_A - k + 1$ Mitglieder der Wahlleitung vertrauenswürdig sind – zu keinem Zeitpunkt jemand Kenntnis der Füllstimmen hat. (Dabei ist k ein Sicherheitsparameter, der angibt, wie viele der n_A Mitglieder der Wahlleitung zur Auszählung notwendig sind.) Dabei ist *zu keinem Zeitpunkt* insofern nicht ganz haltbar, da zum einen die verwendeten Füllstimmen im Klartext auf die Belege gedruckt werden und zum anderen die nicht benötigten Füllstimmen am Ende der Wahl auch entschlüsselt werden. Der springende Punkt dabei ist, dass die nicht verwendeten Füllstimmen für sich genommen kein Wahlgeheimnis brechen und die Füllstimmen auf den Belegen ununterscheidbar von den frisch gewählten Zufallszahlen sind. Daher wäre es korrekter zu sagen, dass die Füllstimmen bis zu ihrer im Protokoll vorgesehenen Veröffentlichung auf einem Beleg oder während der Auszählung niemand vorhersagen kann.

Um das zu erreichen, erstellen die Mitglieder der Wahlleitung im Sinne eines *Secret-Sharings* [108] für jede zu erzeugende Füllstimme einen Share auf die Füllstimme. Jedes Wahlleitungsmitglied erzeugt ein Commitment auf den Share der Füllstimme und verschlüsselt die Unveil-Informationen. Durch die Verwendung von homomorphen Commitment-Schemata kann aus den Commitments auf die Shares ein Commitment auf die Füllstimme erzeugt werden. Ein homomorphes Verschlüsselungsverfahren ermöglicht dasselbe für die Unveil-Informationen.

Kryptographische Commitments erlauben es dem Ersteller, sich auf einen von ihm gewählten Klartext m gegenüber einem zweiten Protokollteilnehmer festzulegen, ohne

dass dieser Kenntnis über m erlangen kann. Später kann der Ersteller des Commitments m (und den verwendeten Zufall) offenlegen (*unveil*), um glaubhaft zu zeigen, dass m die für die Erstellung des Commitments verwendete Nachricht war. Commitments müssen die Eigenschaften *hiding* und *binding* erfüllen. Hiding bedeutet, dass der Empfänger vor dem Offenlegen durch den Sender keinen Rückschluss auf die Nachricht m erhalten kann. Binding bedeutet, dass der Sender keine andere Nachricht $m' \neq m$ veröffentlichen kann, ohne dass dem Empfänger dies auffällt.

Wir benötigen eine dritte Eigenschaft: Das Commitment soll *blind* sein. Das bedeutet, dass der bzw. die Ersteller keine Information über die Nachricht m haben, zu der das Commitment erstellt wurde. Die Information über m und alle weiteren Informationen die für das *unveil* notwendig sind, liegen zu jedem Zeitpunkt der Erstellung nur verschlüsselt vor. Für das Offenlegen ist somit der geheime Schlüssel für die Chiffrate notwendig, von dem wir annehmen, dass er auf alle n_A Mitglieder der Wahlleitung verteilt ist (Threshold-Encryption). Wir nehmen weiterhin an, dass sich mindestens $n_A - k + 1$ Mitglieder der Wahlleitung weigern, ihren Schlüssel – außer an den vorgesehenen Stellen im Protokollablauf – einzusetzen. Basierend auf der Sicherheit der Chiffre und des Commitmentverfahrens ist es dann niemandem möglich, Informationen über die Nachricht m oder die Unveil-Information zu erhalten.

Da nur die Commitments, nicht aber die Chiffrate veröffentlicht werden, kann durch die Verwendung von Commitments, die die Nachricht informationstheoretisch perfekt verbergen, *everlasting security* der öffentlichen Daten erreicht werden. Die Inhalte sollen im Sinne einer Mehrparteienberechnung von den Mitgliedern der Wahlleitung verteilt erstellt werden.

Die folgenden Bausteine benötigen wir für unsere Realisierung blinder Commitments, einer ähnlichen Konstruktion, wie sie Moran et al. [86] diskutiert haben.

Definition 3 (Homomorphe Verschlüsselung) *Sei* $\texttt{ENC}(m, r)$ *ein* IND-CPA*-sicheres homomorphes asymmetrisches Verschlüsselungsverfahren, m die verschlüsselte Nachricht und r der dazu verwendete Zufall. Sei* $\texttt{ENC}(m)$ *dasselbe Verfahren ohne die explizite Nennung des verwendeten Zufalls. Für die Nachrichten m und m' und den Zufällen r, r' benötigen wir die Operationen „$+$“ und „$\cdot$“, so dass gilt:*

$$\texttt{ENC}(m, r) \cdot \texttt{ENC}(m', r') = \texttt{ENC}(m + m', r + r').$$

Definition 4 (Homomorphe Commitments) *Sei* $\texttt{COM}(m, r)$ *ein perfekt verbergendes und komplexitätstheoretisch bindendes Commitmentverfahren, m die Nachricht und r der dazu verwendete Zufall. Sei* $\texttt{COM}(m)$ *dasselbe Verfahren ohne die explizite Nennung des verwendeten Zufalls. Für die Nachrichten m und m' und den Zufällen r, r' benötigen wir die Operationen „$+$“ und „$\cdot$“, so dass gilt:*

$$\texttt{COM}(m, r) \cdot \texttt{COM}(m', r') = \texttt{COM}(m + m', r + r').$$

Definition 5 (Blindes Commitment) *Wir nennen*

$$C := \texttt{bCOM}(m,\, r_1, r_2, r_3) = (c, u, v)$$

ein blindes Commitment auf eine Nachricht m mit den Zufallswerten r_1, r_2, r_3, wobei...

- *... $c := \texttt{COM}(m,\, r_1)$ ein Commitment auf die Nachricht m mit dem Zufall r_1,*
- *... $u := \texttt{ENC}(m,\, r_2)$ eine Verschlüsselung der Nachricht m mit dem Zufall r_2,*
- *... $v := \texttt{ENC}(r_1,\, r_3)$ eine Verschlüsselung des Zufalls r_1 mit dem Zufall r_3 ist.*

Seien $\texttt{bCOM}(m,\, r_1)$ und $\texttt{bCOM}(m)$ jeweils dasselbe Verfahren ohne die explizite Angabe der Zufälle für die Verschlüsselung bzw. aller verwendeten Zufälle.

Ein Tupel aus zwei blinden Commitments auf einen Kandidaten p_i und eine Zufallszahl[5] $N_{i,\,j}$ nennen wir Commitmentpaar:

$$(C,\ C') := \left(\texttt{bCOM}\left(p_i,\, r_1, r_2, r_3\right),\, \texttt{bCOM}\left(N_{i,\,j},\, r_1', r_2', r_3'\right)\right)$$

Blinde Commitments benötigen die Eigenschaft, rerandomisierbar zu sein, da sie kompatibel mit Mix-Nets [7] sein müssen, wie sie häufig in Wahlverfahren Anwendung finden [7, 98, 21, 57]. Müssen nur die Chiffrate rerandomisiert werden, wie es während der Stimmabgabe der Fall sein wird, kann auf die homomorphe Eigenschaft des Verschlüsselungsverfahrens zurückgegriffen werden, da dadurch keines der anderen beiden Elemente beeinflusst wird. Für die Rerandomisierung des ganzen blinden Commitments ist eine Rerandomisierung der einzelnen Elemente nicht ausreichend:

Definition 6 (Rerandomisierung) *Sei $C = (c, u, v) = \texttt{bCOM}(m,\, r_1, r_2, r_3)$ ein blindes Commitment auf die Nachricht m, das mit den Zufallszahlen r_1,r_2 und r_3 erstellt wurde. Seien r_4, r_5 und r_6 frisch erzeugte Zufallszahlen und 0 das neutrale Element bezüglich der Verknüpfung $+$.*

Wir definieren

$$\textsf{re-rand}(C,\, r_4, r_5, r_6) := (c', u', v')$$

mit

$$\begin{aligned} c' &:= c \cdot \texttt{COM}(0,\, r_4) = \texttt{COM}(m,\, r_1 + r_4) \\ u' &:= u \cdot \texttt{ENC}(0,\, r_5) = \texttt{ENC}(m,\, r_2 + r_5) \\ v' &:= v \cdot \texttt{ENC}(r_4,\, r_6) = \texttt{ENC}(r_1 + r_4,\, r_3 + r_6) \end{aligned}$$

Das bedeutet, dass $\textsf{re-rand}(C,\, r_4, r_5, r_6) = \texttt{bCOM}(m,\, r_1 + r_4, r_2 + r_5, r_3 + r_6)$. Als Kurzschreibweise verwenden wir auch hier ohne explizite Angabe der Zufallswerte $\textsf{re-rand}(C)$.

Für die Sicherheit des Protokolls ist es notwendig, dass ein rerandomisiertes blindes Commitment auf m ununterscheidbar von einem blinden Commitment auf einen anderen Wert m' ist.

[5] Statt der Füllstimme $N_{i,\,j}$ kann an dieser Stelle auch eine frische Zufallszahl R_j Verwendung finden.

Definition 7 (Ununterscheidbarkeit der Rerandomisierung) *Seien m, m' und r_i wie oben beschrieben.* $\forall m, m', r_1, r_2, r_3, r_4, r_5, r_6\ \exists r_7, r_8, r_9$:
$\mathsf{re\text{-}rand}(\mathtt{bCOM}(m,\, r_1, r_2, r_3), r_4, r_5, r_6)$ *ist ununterscheidbar von* $\mathtt{bCOM}(m',\, r_7, r_8, r_9)$.

Die Korrektheit eines Mixes kann weiterhin wie üblich mit Hilfe von *Randomized Partial Checking* [7] oder *Shadow Mixes* [35] bewiesen werden. Wir empfehlen wegen der Schwächen der Randomized-Partial-Checking-Technik die Verwendung von Shadow Mixes mit einem entsprechend hohen Sicherheitsparameter.

Erstellung der blinden Commitments Die zentrale Idee eines blinden Commitments ist, dass bei der Erstellung eines Commitments auf eine zufällige Nachricht m selbst die Ersteller keine Kenntnis über m oder die dazugehörige Unveil-Information haben. Bei der Erstellung eines blinden Commitments auf einen bewusst gewählten Wert – wie beispielsweise den Namen eines Kandidaten – muss der Zufallswert der Unveil-Information so gewählt werden, dass ihn keiner der an der Erstellung beteiligten Parteien kennt.

Damit die Inhalte der Chiffrate (und damit der zu dem Commitment gehörenden Nachricht) niemandem bekannt sind, erstellen wir diese verteilt. Dazu nutzen wir die Homomorphieeigenschaft des Verschlüsselungs- und des Commitmentverfahrens: Jedes Mitglied der Wahlleitung erstellt einen Share auf das blinde Commitment bestehend aus einem Commitment auf einen zufälligen Wert und den verschlüsselten Unveil-Informationen. Das Commitment und die Chiffrate aller Mitglieder der Wahlleitung werden zu dem finalen blinden Commitment zusammengefügt. Eine genauere Beschreibung ist in Abbildung 5.4 dargestellt. Werden die blinden Commitments auf die Füllstimmen mit einer solchen Konstruktion erstellt, kann keine Untergruppe der Wahlleitung, die weniger als k Mitglieder umfasst, Informationen über die Füllstimmen erlangen.

Blinde Commitments
realisierbar mit Pedersen-Commitment und Paillier-Kryptosystem

Sei $\mathtt{ENC}(m,\, r)$ ein IND-CPA-sicheres homomorphes Verschlüsselungsverfahren und $\mathtt{COM}(m,\, r)$ ein perfekt verbergendes und komplexitätstheoretisch bindendes Commitmentverfahren, m die Nachricht und r frisch gewählter Zufall. Für die Nachrichten m und m' und die Zufälle r, r' benötigen wir die Operationen „+“ und „·“, so dass (beliebige, aber konstante) $\mu_m^{(enc)}$, $\mu_r^{(enc)}$, $\mu_m^{(com)}$ und $\mu_r^{(com)}$ existieren, für die gilt:

$$\mathtt{ENC}(m,\, r) \cdot \mathtt{ENC}(m',\, r') \equiv \mathtt{ENC}\left((m + m') \mod \mu_m^{(enc)},\, (r + r') \mod \mu_r^{(enc)}\right) \mod \mu^{(enc)}$$

$$\mathtt{COM}(m,\, r) \cdot \mathtt{COM}(m',\, r') \equiv \mathtt{COM}\left((m + m') \mod \mu_m^{(com)},\, (r + r') \mod \mu_r^{(com)}\right) \mod \mu^{(com)}$$

Es gelte weiter, dass $\mu_m^{(enc)} = \mu_m^{(com)} = \mu_r^{(com)}$. Alternativ könnte ein zweites Verschlüsselungsverfahren mit $\mu_m^{(enc\text{-}2)}$, $\mu_r^{(enc\text{-}2)}$ verwendet werden und die folgenden Bedingungen gelten: $\mu_m^{(enc)} = \mu_m^{(com)}$ und $\mu_m^{(enc\text{-}2)} = \mu_r^{(com)}$.

Abbildung 5.3.: Beschreibung, wie sich die Klartexte und die Zufälle der homomorphen Commitments gegenseitig verhalten müssen.

Als Commitment-Schema bietet sich das Pedersen-Commitment [95] an. Als Basis für ein dazu passendes Threshold-Verschlüsselungsverfahren verwenden wir das Paillier-Kryptosystem. Cramer, Damgård, Jurik und Nielsen [44, 42] beschreiben, wie das Verfahren für unsere Bedürfnisse angepasst werden kann. Für die Kompatibilität ist nicht nur die Verknüpfung entscheidend – beide Verfahren sind additiv homomorph –, sondern auch die der Nachrichtenräume (siehe Abbildung 5.3). Die angepasste Variante des Paillier-Kryptosystems erfüllt beide Bedingungen.

Wir verwenden die Konstruktion der blinden Commitments nicht nur für die Füllstimmen, sondern auch für die Kandidaten. Nun ist aber durch die Struktur der veröffentlichten Daten klar, auf welchen Kandidaten ein bestimmtes blindes Commitment erstellt wurde. Daher ist es bei der Erstellung der blinden Commitments auf die Kandidaten auch nicht notwendig, das geheim zu halten. Da die blinden Commitments vor der weiteren Verarbeitung rerandomisiert werden, gilt dasselbe auch für die Unveil-Informationen.

Daher können wir den Erstellungsprozess für die blinden Commitments auf die Kandidaten vereinfachen: Ein Wahlleitungsmitglied wählt die für die Erstellung des blinden Commitments notwendigen Zufälle und veröffentlicht diese zusammen mit dem daraus erstellten blinden Commitment. Jeder ist in der Lage, die korrekte Erstellung des blinden Commitment nachzurechnen. Die Korrektheit der Rerandomisierung wird durch die Verwendung eines Shuffles gezeigt.

Physical Oblivious Transfer

In papierbasierten Wahlverfahren beruht die Geheimhaltung der vom Wähler auf dem Stimmzettel markierten Stimme darauf, dass die Gegenstände, die an der Markierung der Stimme beteiligt sind, diese nicht weitergeben können. In der klassischen Umsetzung, die in Deutschland beispielsweise bei Bundestagswahlen verwendet wird, ist das der Stimmzettel, der Stift und die Wahlkabine. Die Wahlkabine außen vorgelassen, da sie auch bei einer auf elektronischen Wahlmaschinen basierenden Stimmabgabe (und -auszählung) Verwendung finden muss, suchen wir für die Abgabe der Stimme einen Ersatz für Stift und Papier. Klassische Stifte und gewöhnliches Papier garantieren auf eine nachvollziehbare Art und Weise, dass das Geheimnis, für welchen Kandidaten ein Wähler stimmt, nicht unerwartet an Dritte übermittelt wird. Diese Einschränkung der Möglichkeit zu kommunizieren soll bei einem neuen System ebenfalls möglichst leicht zu kontrollieren sein. Wie wir einleitend schon betont haben, erfüllen frei programmierbare Bausteine unserer Ansicht nach diese Anforderung nicht.

Daher schlagen wir für unser Verfahren einen Baustein vor, der ein physisches *Oblivious Transfer* (pOT) mit mechanischen Mitteln realisiert. Zusammen mit drei einfachen elektronischen Bausteinen, deren Programmierung einerseits fest und andererseits sehr einfach ist, können wir zwar kein System erstellen, das annähernd so einfach zu überschauen ist wie Papier und Bleistift, das jedoch die Komplexität verglichen mit den bisher üblicherweise

Verteilte Erstellung blinder Commitmemts

Seien g und h die Erzeuger der zyklischen Gruppe G für das Pedersen-Commitment und $\mathrm{pk}_{\mathcal{T}}$ der öffentliche Schlüssel der Wahlleitung. Die Wahlleitung bestehe aus n_A Mitgliedern. Jedes Mitglied a_l der Wahlleitung erstellt Commitments mit selbst erzeugten zufälligen $N_{i,j}^{(a_l)}$ und $r_{i,j}^{(a_l)}$

$$c_{i,j}^{(a_l)} := \mathtt{COM}(N_{i,j}^{(a_l)}, r_{i,j}^{(a_l)}) = g^{N_{i,j}^{(a_l)}} h^{r_{i,j}^{(a_l)}},$$

mit $1 \leq i \leq n_P$, $1 \leq j \leq n_E$, und $1 \leq l \leq n_A$, und die entsprechenden Chiffrate

$$u_{i,j}^{(a_l)} := \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}\left(N_{i,j}^{(a_l)}\right)$$

$$v_{i,j}^{(a_l)} := \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}\left(r_{i,j}^{(a_l)}\right).$$

Das Produkt

$$\begin{aligned} c_{i,j} &:= \mathtt{COM}(N_{i,j}, r_{i,j}) \\ &= \mathtt{COM}\left(\sum_{l=1}^{n_A} N_{i,j}^{(a_l)}, \sum_{l=1}^{n_A} r_{i,j}^{(a_l)}\right) \\ &= \prod_{l=1}^{n_A} c_{i,j}^{(a_l)} \end{aligned}$$

ist das Commitment auf die Füllstimme $N_{i,j}$. Die zwei Produkte

$$u_{i,j} := \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}(N_{i,j}) = \mathtt{ENC}\left(\sum_{l=1}^{n_A} N_{i,j}^{(a_l)}\right) = \prod_{l=1}^{n_A} u_{i,j}^{(a_l)}$$

$$v_{i,j} := \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}(r_{i,j}) = \mathtt{ENC}\left(\sum_{l=1}^{n_A} r_{i,j}^{(a_l)}\right) = \prod_{l=1}^{n_A} v_{i,j}^{(a_l)}$$

sind die Chiffrate der dazu gehörenden Unveil-Information.

Abbildung 5.4.: Verteiltes Erstellen der blinden Commitments auf zufällige Füllstimmen. So lange mindestens ein Mitglied a_l der Wahlleitung die Wahl von $N_{i,j}^{(a_l)}$ und $r_{i,j}^{(a_l)}$ geheim hält und sich mindestens $n_A - k + 1$ der Mitglieder nicht an einer Entschlüsselung beteiligen, können die Werte $N_{i,j}$ und $r_{i,j}$ nicht bestimmt werden.

verwendeten Eingabemöglichkeiten drastisch reduziert. Die Annahme vertrauenswürdiger Hardwarekomponenten ist in der Kryptographie keine Seltenheit [76, 87].

Oblivious Transfer (OT) ist eine kryptographische Primitive, die 1981 von Rabin [99] vorgestellt wurde. Ein Oblivious Transfer erhält zwei Eingaben m_0 und m_1 von einem Protokollteilnehmer A; der zweite Protokollteilnehmer B gibt ein Bit b ein und erhält m_b. Welches Bit B gewählt hat, bleibt A verborgen und damit auch, welche der beiden Nachrichten B nach Abschluss des Protokolls kennt. Der zweite Protokollteilnehmer erfährt ausschließlich den Wert der von ihm ausgewählten Eingabe m_b; über m_{1-b} kann er keine Informationen in Erfahrung bringen.

Die Wahlmaschine darf die Stimme des Wählers nicht in Erfahrung bringen können.
Für die Umgebung des Eingabegerätes (pOT) muss ununterscheidbar sein, ob der Knopf des pOT gedrückt ist.

Letztendlich handelt es sich um einen Schalter, der nur so bedient werden kann, dass eine von zwei Eingaben auf einen Ausgabekanal übertragen wird. Der Nachteil einer physischen Trennung eines elektrischen Übertragungskanals ist, dass der Sender dies anhand des anliegenden Widerstands feststellen kann. Daher schlagen wir ein System vor, in dem die Übermittlung über einen optischen Kanal realisiert wird (siehe Abbildung 5.5): Beide Eingabekanäle steuern je eine LED – unabhängig davon, ob die Nachricht an den Ausgabekanal übertragen wird. Die Ausbreitung des Lichts einer der beiden LEDs wird durch einen Schirm blockiert. Durch das Drücken eines Knopfs kann das Schild von der einen LED zu der anderen LED verschoben werden. Das Licht der anderen LED wird von einem Photowiderstand wieder in ein elektrisches Signal übertragen.

Das Prinzip ähnelt einem Optokoppler mit zwei sendenden Dioden, von denen nur eine von der empfangenden Diode *gesehen* werden kann. Dadurch wird eine galvanische Trennung zwischen Sender und Empfänger erreicht. Da beide Dioden *ihre* Nachricht senden und dementsprechend Strom verbrauchen, ist ihr Verhalten unabhängig davon, welche Diode blockiert wird; so ist es dem Sender nicht möglich, anhand des Stromverbrauchs oder des Widerstandes festzustellen, welche der beiden Nachrichten ausgeliefert wird.

Mögliche Umsetzung eins pOT-Bausteins

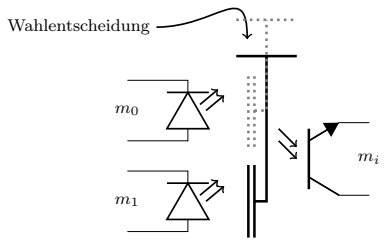

Abbildung 5.5.: Der pOT-Baustein: Der Baustein ähnelt der Funktionsweise eines Optokopplers. Auf der linken Seite senden zwei Dioden permanent die Nachrichten m_0 und m_1. Auf der rechten Seite wandelt ein Phototransistor die Lichtsignale wieder in elektronische Daten um. Eine der beiden Dioden ist abgeschirmt. Wenn der Wähler den zu dem pOT gehörenden Kandidaten wählt, verschiebt er den Schirm von einer Dioden zu der anderen.

5.3.3. Zusammenfassung der Notation

Die im Folgenden betrachteten Wahlen werden n_P Kandidaten (Parteien) p_i und n_E Wähler e_j haben: $P := \{p_i\}_{i=1}^{n_P}$ und $E := \{e_j\}_{j=1}^{n_E}$. Jeder Wähler hat eine Stimme, die er genau einem Kandidaten geben kann (1-aus-n-Wahlen[6]). Die Kandidaten und die Wahlentscheidung des Wählers betreffende Variablen werden mit dem Index i versehen, während zwischen den Wählern durch den Index j unterschieden wird. Für die korrekte Initialisierung der Wahl und die Auszählung stützen wir uns auf die Wahlleitung, bestehend aus einer Gruppe mit n_A Mitgliedern a_l, von der wir annehmen, dass sich mindestens $n_A - k + 1$ der Mitglieder an das Protokoll halten und insbesondere keine Chiffrate entschlüsseln, die laut Protokoll nicht entschlüsselt werden dürfen. Für die Mitglieder der Wahlleitung verwenden wir den Index l. Jedes Mitglied verfügt über einen eigenen geheimen Schlüssel $\mathrm{sk}_{\mathcal{T},l}$ zur Entschlüsselung von Chiffraten sowie über einen öffentlichen Schlüssel für die Verifikation von Signaturen $\mathrm{s}_l^{(ver)}$ (und den dazugehörigen geheimen Schlüssel $\mathrm{s}_l^{(sig)}$). Den öffentlichen Schlüssel zur Erstellung von Chiffraten, die von k Mitgliedern der Wahlleitung entschlüsselt werden können, bezeichnen wir mit $\mathrm{pk}_{\mathcal{T}}$. Abbildung 5.6 zeigt eine Übersicht über das benötigte Schlüsselmaterial.

Die Variablen N und R verwenden wir für Zufallszahlen. Die öffentlichen Parameter für ein Pedersen-Commitment-Schema [95] sind g und h. Des weiteren werden wir Commitments eines der blinden Commitments mit c bezeichnen; u und v sind die dazugehörigen Chiffrate. C, C', D und D' sind Tripel aus je einem Commitment c und zwei Chiffraten u und v.

Schlüsselmaterial in Oblivious Bingo Voting

Beschreibung	Bezeichner	Ersteller	Kenntnis
Signaturverfahren			
Signaturschlüssel	$\mathrm{s}_l^{(sig)}$	a_l	a_l
Verifikationsschlüssel	$\mathrm{s}_l^{(ver)}$	a_l	öffentlich
Threshold-Verschlüsselungsverfahren			
geheime Schlüssel	$\mathrm{sk}_{\mathcal{T},l}$		a_l
öffentlicher Schlüssel	$\mathrm{pk}_{\mathcal{T}}$		intern
Commitmentverfahren			
öffentliche Parameter	g, h		öffentlich

Abbildung 5.6.: Für diese Umsetzung von Bingo Voting werden Schlüssel für das Threshold-Verschlüsselungsverfahren und für Signaturen benötigt. Den Signaturschlüssel $\mathrm{s}_l^{(sig)}$ erstellt jedes Mitglied der Wahlleitung a_l selbst. Von wem die Schlüssel für das Threshold-Verfahren erstellt werden, hängt von dem eingesetzten Verfahren ab. Von *öffentlichen* Daten soll jeder Kenntnis haben (können). *Interne* Daten dürfen nur der Wahlleitung bekannt sein. Die Signaturschlüssel sowie die persönlichen Teile des geheimen Schlüssels des Verschlüsselungsverfahrens kennen nur die jeweiligen Mitglieder der Wahlleitung.

[6] Systeme, in denen Wähler mehrere Stimmen zur Verfügung haben, lassen sich im Allgemeinen auf 1-aus-n-Wahlen reduzieren. Die Variable n steht in diesem Begriff für die Zahl der Kandidaten, die wir mit n_P bezeichnen.

5.3.4. Unsere Konstruktion

In diesem Abschnitt stellen wir unsere Konstruktion vor. Sie basiert auf dem Originalentwurf von Bingo Voting. Zunächst skizzieren wir die Idee, bevor wir auf die einzelnen Bausteine im Detail eingehen. Die Daten, die während des Protokollablaufs veröffentlicht werden, unterscheiden sich nicht vom Originalentwurf. Das heißt, dass wir die Korrektheits- und Sicherheitseigenschaften von Bingo Voting erben. Unser Ziel ist es, dass die Sicherheitseigenschaften auch dann erreicht werden, wenn die Wahlmaschine passiv korrumpiert ist. Das bedeutet insbesondere, dass die Maschine weder die Füllstimmen noch die frischen Zufallszahlen noch die Wahl des Wählers kennen darf.

Grundlegende Idee

Im Originalentwurf von Bingo Voting muss die Wahlmaschine vertrauenswürdig sein, damit das Wahlgeheimnis gewahrt bleibt. Denn sie kennt die Füllstimmen $N_{i,j}$ für die nicht gewählten Kandidaten und den frischen Zufall R_j, der an die Stelle des gewählten Kandidaten gesetzt wird. Außerdem weiß die Maschine, für welchen Kandidaten der Wähler stimmt. Damit könnte nicht nur das Wahlgeheimnis gebrochen werden; die Maschine könnte auch Zwischenergebnisse berechnen, was im Widerspruch zum Prinzip der Gleichheit steht.

Ziel muss es sein, die Wahl des Wählers vor der Maschine geheim zu halten und sie nur auf Chiffraten operieren zu lassen. Die öffentlichen Daten sind – ebenso wie im Original – informationstheoretisch verbergend. Daten, die nur die Wahlleitung und die Maschine kennen, sind komplexitätstheoretisch verbergend.

Um das Ziel zu erreichen, verwenden wir drei Techniken:

- Die in der Vorberechnung erstellten Commitments sind verteilt berechnete blinde Commitments.
- Die Unveil-Informationen werden verschlüsselt und ebenfalls verteilt erzeugt. Als Verschlüsselungsverfahren wird ein Threshold-Verschlüsselungsverfahren verwendet, so dass k Schlüsselinhaber für die Entschlüsselung der Chiffrate notwendig sind.
- Für die Stimmabgabe wird ein Mechanismus verwendet (pOT), der die Information, welcher Kandidat gewählt wurde, vor dem Rest der Maschine abschirmt.

Durch diese drei Techniken können wir sicherstellen, dass nur durch die Beteiligung von mindestens k von n_A Mitgliedern der Wahlleitung Informationen über die Füllstimmen gewonnen werden können. In dem Kapitel 5.3.2 ist eine genauere Beschreibung der Techniken zu finden. Die Wahlmaschine sendet statt der Füllstimmen nun für jeden Kandidaten genau eine verschlüsselte Füllstimme in Richtung des Druckers. Die Ersetzung einer der verschlüsselten Füllstimmen durch eine verschlüsselte, frische Zufallszahl wird auf dem Weg zum Drucker durch die pOTs vorgenommen. Diese senden, wenn der Knopf nicht betätigt wurde, die Eingabe der Wahlmaschine an den Drucker weiter; sonst wird die

vom Zufallszahlengenerator frisch gezogene und verschlüsselte Zufallszahl gesendet. An dieser Stelle sei noch einmal hervorgehoben, dass die pOTs über keinerlei Rechenleistung verfügen. Der Drucker erhält nun eine Reihe von Chiffraten, die er sich von der Wahlleitung entschlüsseln lässt. Im Anschluss kann er den Beleg drucken. Der Drucker erhält zwar die Füllstimmen und den frischen Zufall im Klartext, kann diese jedoch nicht voneinander unterscheiden; für ihn sind diese ebenso wenig aussagekräftig wie der Beleg für einen Angreifer.

Modularisierung des Protokolls von Bingo Voting

Das Originalprotokoll von Bingo Voting sieht eine Trennung der Wahlmaschine und des Zufallszahlengenerators vor. Damit konnte erreicht werden, dass die Auszählung bei unkorrumpiertem Zufallszahlengenerator korrekt ist. Wir bauen auf dieser Idee auf und teilen die Wahlmaschine – wie in Abbildung 5.7 dargestellt – weiter auf, um das Wahlgeheimnis vor allen Komponenten mit Rechenleistung zu wahren.

Der wichtigste Schritt ist dabei die Separierung des Eingabegerätes, das wir im weiteren als Input-Device bezeichnen. Dadurch können wir auch bei korrumpierter Wahlmaschine garantieren, dass die Maschine keine Zwischenergebnisse berechnen kann. Die Erpressungsfreiheit können wir für passiv korrumpierte Maschinen garantieren.

Das Input-Device besteht aus mehreren Instanzen einer Komposition von einfachen Bausteinen. Zentrales Element ist ein Baustein, den wir pOT nennen. Er realisiert einen Oblivious Transfer, basierend auf galvanischer Trennung zwischen Ein- und Ausgabe.

Des weiteren nutzen wir verschiedene kleine Komponenten für sehr spezifische Aufgaben: eine *Append-Box*, eine *Split-Box* und eine *Rerandomisierungsbox*. Die Append-Box gibt ihre Eingabe wieder aus, nachdem sie ihr einen ihr fest einprogrammierten String angehängt hat. Die Split-Box ist das entsprechende Gegenstück; sie trennt eine aus zwei Teilen stammende Eingabe auf und gibt sie auf zwei unterschiedlichen Kanälen aus. Die Rerandomisierungsbox erhält als Eingabe ein Tupel von Chiffraten, rerandomisiert diese und gibt sie wieder aus.

Diese drei Komponenten sollen als dedizierte Hardwarebausteine umgesetzt werden. Sie benötigen keinen persistenten Speicher, und für ihre Programmierung, die nur wenige Programmzeilen umfasst, kann ein *Read-Only*-Speicherbaustein (ROM) verwendet werden. Da sie einfach zu verifizieren sind, nehmen wir sie als vertrauenswürdig an. Die Verwendung von einfachen, vertrauenswürdigen Bausteinen erinnert an *tamper-proof* Hardware, die in der Kryptographie oft Anwendung findet [76, 87].

Die Aufteilung der Maschine in einzelne Komponenten reduziert nicht nur den Umfang der Maschine, sondern auch die Komplexität. Die Aufgaben, die nach der Modularisierung für den frei programmierbaren Teil der Wahlmaschine verbleiben, beschränken sich auf das Speichern und Organisieren von Daten. Wir nennen diesen Teil daher im weiteren *Storage-Device*. Zwischen allen Komponenten nehmen wir einen dedizierten Kanal an.

Im Folgenden diskutieren wir die Funktion der einzelnen Komponenten. Der Zusammenhang dieser Bausteine ist in Abbildung 5.7 dargestellt. Auf den Ablauf der Stimmabgabe, der Auszählung und der Beweise gehen wir ab Abschnitt 5.3.4 ein.

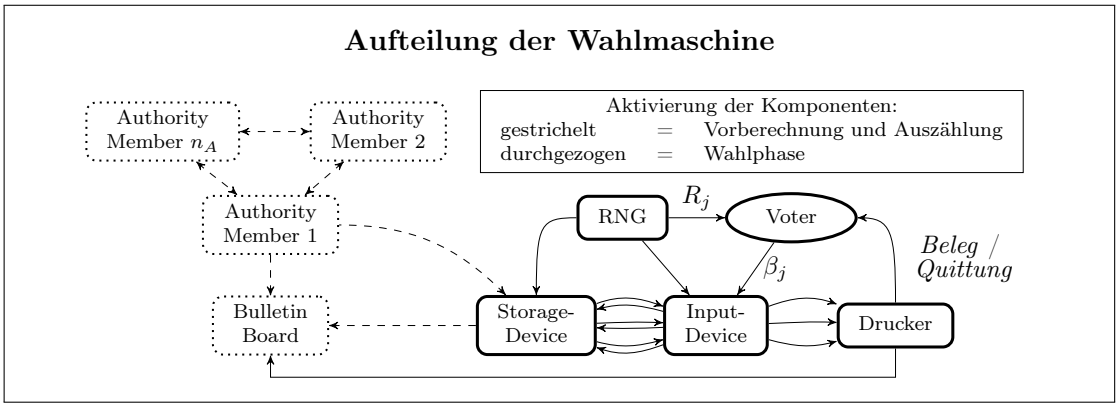

Abbildung 5.7.: Wir teilen die Wahlmaschine in vier Hauptkomponenten ein: ein Storage-Device, das Chiffrate auf die Füllstimmen und während der Wahl anfallende Daten speichert; einen RNG (Abbildung 5.9), der die für den gewählten Kandidaten verwendete frische Zufallszahl R_j zieht und dem Wähler anzeigt; ein Input-Device (Abbildung 5.10 und 5.12), über das der Wähler den gewählten Kandidaten β_j aussucht; und einen Drucker (Abbildung 5.13), der den Beleg für den Wähler erstellt. Die Zahl der Pfeile zwischen Storage-Device, Input-Device und Drucker zeigt, dass es sich um eine Wahl mit drei Auswahlmöglichkeiten handelt.

Das Storage-Device Das zentrale und komplexeste Element unserer Wahlmaschine ist das *Storage-Device* – die Speichereinheit, in der die verschlüsselten Füllstimmen und alle während der Wahl anfallenden Daten gespeichert werden. Alle anderen Teile der Wahlmaschine speichern keine Daten und können nach jeder Stimmabgabe zurückgesetzt werden.

Die Abbildung 5.8 gibt einen Überblick über die vor der Wahl erzeugten und während der Wahl anfallenden Daten. Das Commitmentpaar $(D_{i,j},\ D'_{i,j})$ wird vor der Wahl erstellt. Der Ablauf der Erstellung wird später in Algorithmus 4 und 5 beschrieben.

Das Storage-Device erhält vor der Wahl die verschlüsselten Füllstimmen. Diese sind in Abbildung 5.8 für einen Wähler grau umrahmt.

Die Aufgabe des Storage-Devices ist es, für jede Stimmabgabe einen Satz verschlüsselter Füllstimmen an das Input-Device zu senden (in Abbildung 5.8 grau umrandet). Im Anschluss erhält es ein blindes Commitment vom RNG und für jeden Kandidaten ein Chiffrat vom Input-Device (beides in Abbildung 5.8 schwarz umrahmt). Diese Daten speichert das Storage-Device.

Der Zufallszahlengenerator Die Funktion des RNG mussten wir gegenüber der des im ursprünglichen Entwurf verwendeten RNG um einen Schritt erweitern. Der RNG zieht

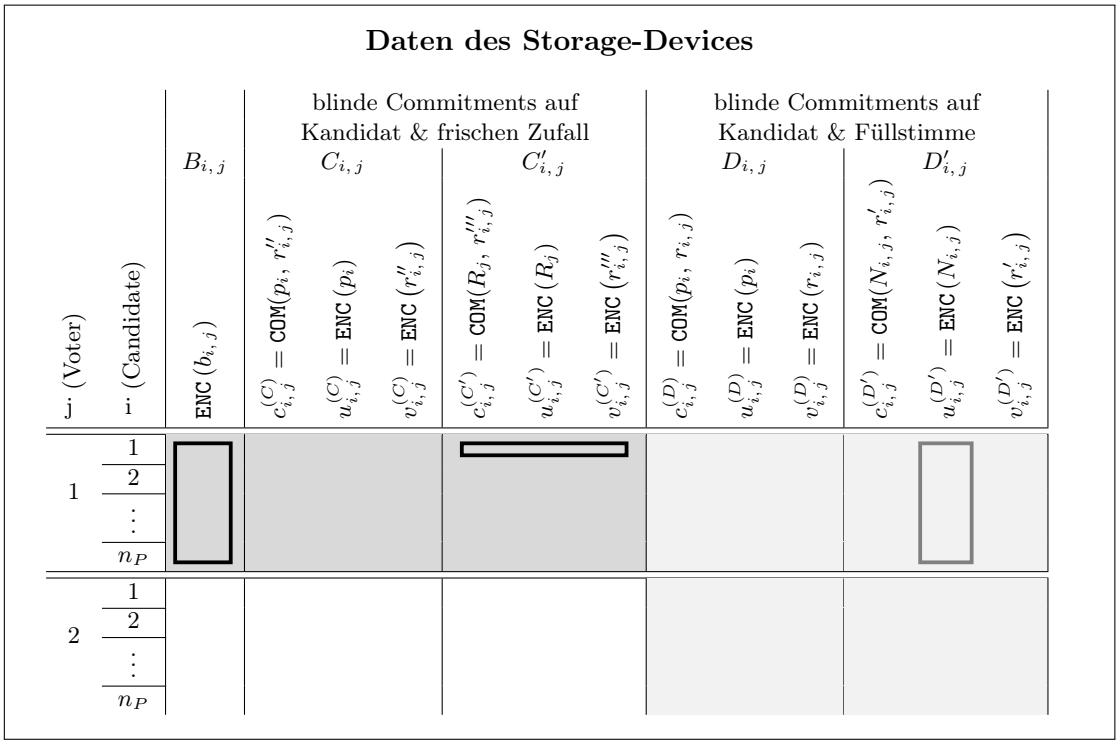

Abbildung 5.8.: Die Tabelle zeigt die vor der Wahl berechneten Daten (Spalten $D_{i,j}$ und $D'_{i,j}$ – hellgrau hinterlegt) sowie die Daten, die während der Wahl anfallen (Spalten $B_{i,j}$, $C_{i,j}$ und $C'_{i,j}$ – für einen Wähler dunkelgrau hinterlegt). Die Tabelle enthält für jeden der n_E Wähler eine Zeile für jeden der n_P Kandidaten. Auf dem Storage-Device werden die verschlüsselten Füllstimmen ($u_{i,j}^{(D')}$) gespeichert (für einen Wähler grau umrahmt). Die schwarz umrahmten Daten erhält das Storage-Device während der Stimmabgabe des (ersten) Wählers.

eine frische Zufallszahl R_j und zeigt sie dem Wähler an. Diese Zufallszahl wird anstelle der in der Vorbereitungsphase generierten Füllstimme verwendet. Statt – wie im ursprünglichen Entwurf – R_j im Klartext an die Wahlmaschine (bzw. an das Storage-Device) zu senden, erzeugt der RNG ein blindes Commitment auf R_j: $C'_{1,j} := \texttt{bCOM}(R_j, r) = \big(\texttt{COM}(R_j, r), \texttt{ENC}(R_j), \texttt{ENC}(r)\big)$ (siehe Abbildung 5.9). Einerseits sendet es dies an das Storage-Device; andererseits sendet es das Chiffrat $\texttt{ENC}(R_j)$ an jeden einzelnen pOT (siehe Abbildung 5.10). Zur Erstellung des blinden Commitments verwendet der RNG den gleichen öffentlichen Schlüssel $\text{pk}_{\mathcal{T}}$, wie er zur Erstellung der blinden Commitments in der Vorbereitungsphase verwendet wurde (siehe Abbildung 5.4). Zur Übertragung der Chiffrate an die pOTs nehmen wir dedizierte Kanäle zwischen RNG und den einzelnen pOTs an. An dieser Stelle möchten wir noch einmal betonen, dass der RNG vertrauenswürdig sein muss. Insbesondere dürfen die Zahlen, die er generiert, nicht unterscheidbar von den Zufallszahlen (Füllstimmen) der Vorbereitungsphase sein. Wir nehmen den RNG als vertrauenswürdig an; gleichzeitig verweisen wir aber darauf, dass – ebenso wie die Erstellung der blinden Commitments – durch eine Mehrparteienberechnung das Vertrauen verteilt werden kann. Das ist jedoch nicht Teil dieser Arbeit.

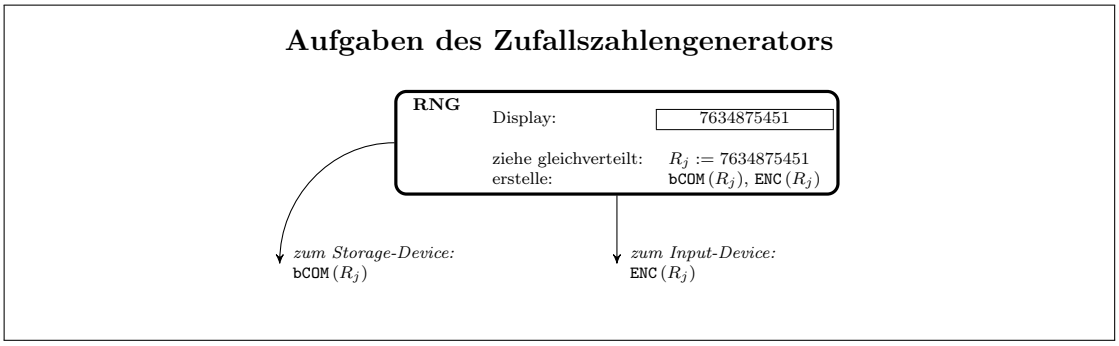

Abbildung 5.9.: Der Zufallszahlengenerator erstellt pro Wähler eine frische Zufallzahl. An das Input-Device sendet er diese mit dem öffentlichen Schlüssel der Wahlleitung verschlüsselt. An das Storage-Device sendet er die frische Zufallszahl als blindes Commitmemt.

Append-x-Box, Rerandomisierungsbox und Split-Box Zwischen Ausgabe aus dem Storage-Device und dem Eingang in die pOTs sowie nach der Ausgabe der pOTs müssen die Daten modifiziert werden. Dazu ist kein Wissen darüber notwendig, welcher Kandidat die Stimme des Wählers erhalten hat. Für diesen Zweck verwenden wir drei kleine Hardwarebausteine, deren technischer Aufbau und Programmierung leicht überschaubar und verifizierbar ist (siehe Abbildung 5.12).

Append-Box Die Append-Box hat je einen Ein- und einen Ausgang. Sie erwartet an ihrem Eingang eine Zeichenkette fester Länge. An diese hängt sie eine ihr fest einprogrammierte Zeichenkette und gibt das Ergebnis auf ihrem Ausgang aus. Konkret handelt es sich bei der angehängten Zeichenkette entweder um das Chiffrat einer Eins oder einer Null. Die Box erhält demzufolge ein einzelnes Chiffrat und gibt ein Tupel aus zwei Chiffraten aus. Je nach Implementierung kann es nötig sein, dass ein Trennzeichen – eine Folge aus Nullen und Einsen – die beiden Elemente in der Bitdarstellung trennt.

$$enc(a) \xrightarrow{\texttt{append}_1} enc(a), enc(1) \qquad \text{bzw.} \qquad enc(a) \xrightarrow{\texttt{append}_0} enc(a), enc(0)$$

Split-Box Die Split-Box hat einen Eingang und zwei Ausgänge. Wie die Append-Box erwartet sie eine Eingabe fester Länge – die Länge, die die Ausgaben der Append-Box haben. Die Split-Box trennt ein von der Append-Box erstelltes Tupel wieder und gibt auf ihren zwei Ausgängen je eines der beiden Teile des Tupels wieder aus.

$$enc(a), enc(b) \xrightarrow{\texttt{split}} \begin{cases} enc(a) \\ enc(b) \end{cases}$$

Rerandomisierungsbox Die Rerandomisierungsbox hat einen Ein- und einen Ausgang. Sie erwartet Eingaben der gleichen Länge wie die Split-Box und interpretiert die Eingabe als Tupel aus zwei Chiffraten. Sie wählt zwei Zufallszahlen $\bar{r_1}$ und $\bar{r_2}$ gleichverteilt, mit denen sie die beiden Chiffrate rerandomisiert und anschließend

das Tupel der beiden rerandomisierten Chiffrate auf ihrem Ausgang wieder ausgibt. Optional kann dieser Baustein über einen weiteren Eingang verfügen, über den er den Zufall erhält, der für die Rerandomisierung benötigt wird.

$$\begin{aligned}\texttt{ENC}(m_1, r_1), \texttt{ENC}(m_2, r_2) \xrightarrow{\texttt{rerand}} & \texttt{ENC}(m_1, r_1) \cdot \texttt{ENC}(0, \bar{r_1}), \texttt{ENC}(m_2, r_2) \cdot \texttt{ENC}(0, \bar{r_2}) \\ & = \texttt{ENC}(m_1, r_1'), \ \texttt{ENC}(m_2, r_2')\end{aligned}$$

Diese Bausteine haben einen festen und sehr begrenzten Funktionsumfang. Daher nehmen wir an, dass es möglich ist, sie mit geringem Aufwand so zu gestalten, dass ihre Funktionalität weder erweiter- noch änderbar ist und nachprüfbar bleibt, dass sie korrekt funktionieren.

Input-Device mit Physical Oblivious Transfer Der ursprüngliche Entwurf macht keine Angabe darüber, in welcher Form der Wahlmaschine die Stimme des Wählers mitgeteilt wird. Prinzipiell kann Bingo Voting an jedes Verfahren angeschlossen werden, das dem Verifizierungsmechanismus Bingo Voting die Stimme des Wählers mitteilen kann. Aus unserer Sicht ist das jedoch schützenswerte Information, die in digitaler Form erst am Ende der Auszählungsphase entschlüsselt vorliegen darf. Auf diese Information ist der ursprüngliche Entwurf jedoch angewiesen. Unsere Modifikation des Verfahrens erlaubt es uns, die Wahl des Wählers in einer einzelnen Komponente zu kapseln, die selbst über keine Rechenleistung verfügt.

Für jeden Kandidaten muss entschieden werden, ob ihm die frische Zufallszahl des RNG oder eine seiner vorberechneten Füllstimmen zugeordnet wird (siehe Abbildung 5.10). Diese Entscheidung trifft der Wähler mit Hilfe eines Bausteins, der einen Oblivious Transfer realisiert.

Eine mögliche Realisierung für einen Physical Oblivious Transfer(pOT) beschreiben wir in Abschnitt 5.3.2. Für jeden Kandidaten gibt es ein pOT. Der pOT bekommt zwei Eingaben; eine vom Storage-Device, eine vom RNG.

Beide Eingaben werden zunächst mit Hilfe einer Append-0- und einer Append-1-Box durch ein zusätzliches Chiffrat ergänzt. Anschließend werden die Chiffrate der daraus entstehenden Tupel durch eine Rerandomisierungsbox rerandomisiert (siehe Abbildung 5.11). Die rerandomisierten Tupel sind die Eingaben der pOTs.

Der Wähler entscheidet per Tastendruck, ob das Tupel mit der Füllstimme oder mit der frischen Zufallszahl weiterverarbeitet wird. Ohne sein Zutun wird die Eingabe des Storage-Device ausgewählt, die für einen nicht gewählten Kandidaten steht. Die Eingabe des RNG wird weitergeleitet, wenn er die Taste des pOT betätigt hat. In diesem Fall gibt er eine Stimme für den dem pOT zugeordneten Kandidaten ab.

Einen Mechanismus, der verhindert, dass der Wähler zwei Kandidaten eine Stimme gibt, ließe sich mechanisch realisieren, wird aber in dieser Arbeit nicht näher betrachtet. Das Drücken zweier Tasten ist vergleichbar mit dem Ankreuzen zweier Kandidaten bei der

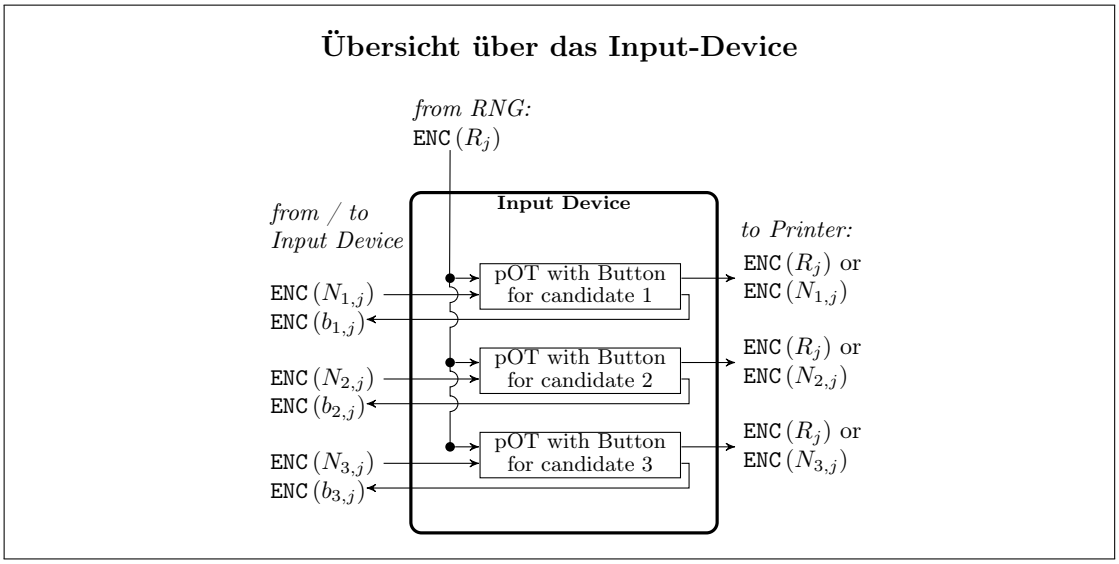

Abbildung 5.10.: Unser Input-Device besteht aus einem pOT-Baustein pro Kandidat. Jeder Baustein erhält zwei Eingaben: die Chiffrate der frischen Zufallszahl und der Füllstimme. Siehe Abbildung 5.12 für Details.

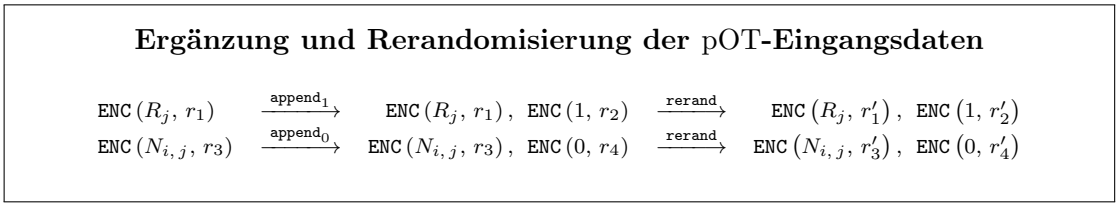

Abbildung 5.11.: Die Daten, die das Storage-Device und der RNG an das Input-Device senden, werden – bevor sie an die jeweiligen pOTs weitergeleitet werden – durch ein Chiffrat zu einem Tupel ergänzt, das im Anschluss rerandomisiert wird.

Papierwahl. Der zugehörige Stimmzettel wird dadurch ungültig. Die Auszählung kann dementsprechend korrigiert werden.

Das Tupel, das der pOT ausgibt, wird durch eine Split-Box aufgeteilt. Der erste Teil wird an den Drucker, der zweite an das Storage-Device gesendet. Einen Überblick über den Datenfluss für einen Kandidaten gibt die Abbildung 5.12.

Das Storage-Device erhält für jeden Kandidaten ein Chiffrat, das die Information enthält, ob der Kandidat gewählt wurde (1) oder nicht (0). Wäre das Storage-Device mächtig genug, um die Chiffrate zu brechen, so könnte es damit auch das Wahlgeheimnis brechen. Obwohl die Append-Boxen immer das gleiche Chiffrat anhängen, kann das Storage-Device durch die Rerandomisierung den Chiffraten nicht ansehen, ob sie eine Eins oder eine Null enthalten. Der Drucker erhält ein Chiffrat mit je einer Füllstimme $N_{i,j}$ bzw. dem frischen Zufall R_j pro Kandidat. Er wird sich die Chiffrate entschlüsseln lassen, um den Beleg drucken zu können. Der Drucker erhält damit jedoch nicht mehr Informationen, als die, die ohnehin veröffentlicht werden.

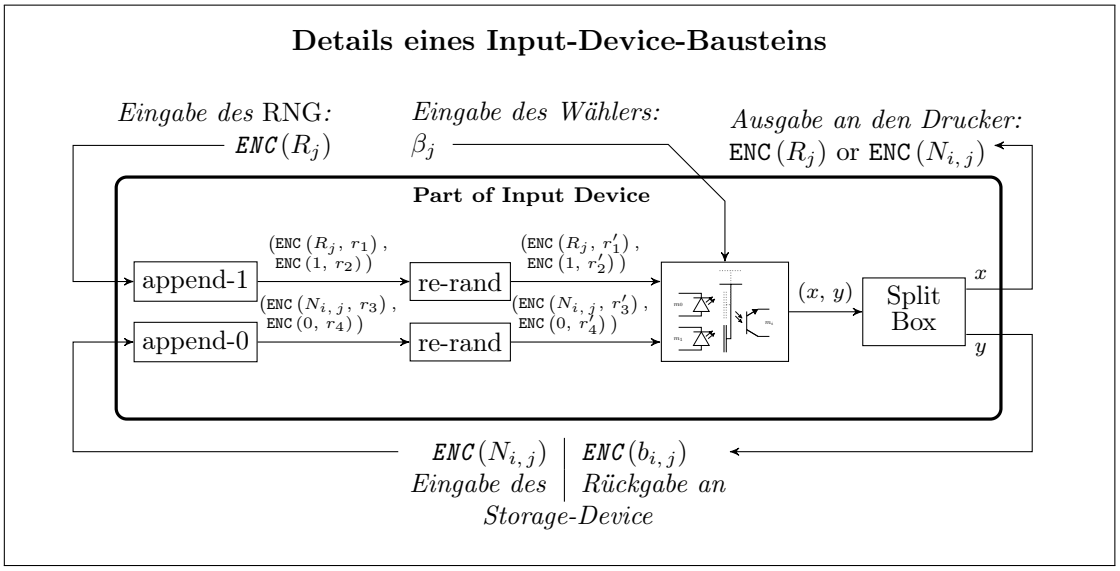

Abbildung 5.12.: Physical Oblivious Transfer (pOT)-Baustein: Beide Eingaben werden mit einer verschlüsselten 0 beziehungsweise 1 ergänzt. Anschließend wird das daraus entstandene Tupel rerandomisiert.

Drucker Der Drucker, der den Beleg für den Wähler druckt, erhält – wie im Abschnitt über den pOT schon beschrieben – verschlüsselte Füllstimmen und die verschlüsselte frische Zufallszahl. Die Reihenfolge entspricht der der zugehörigen Kandidaten. Da der Drucker Füllstimmen nicht von frischen Zufallszahlen unterscheiden kann, kann er aus der Reihenfolge nicht darauf schließen, welcher Kandidat gewählt wurde.

Der Drucker hat Orakelzugriff auf die geheimen Schlüssel der Wahlleitung (siehe Abbildung 5.13). Damit kann er für jedes Chiffrat die k notwendigen Teilentschlüsselungen vornehmen, die er benötigt, um den Klartext des Chiffrats zu berechnen. Der Beleg, den er druckt, entspricht dem des ursprünglichen Bingo-Voting-Entwurfs.

Der Orakelzugriff auf die geheimen Schlüssel der Wahlleitung kann auf viele unterschiedliche Arten gestaltet werden. Für die weitere Beschreibung des Verfahrens nehmen wir an, dass der Drucker Zugriff auf ein Hardware-Token jedes Mitglieds der Wahlleitung hat. Diesem Token kann er das Chiffrat schicken und erhält die Teilentschlüsselung als Antwort.

Ein Mechanismus, der die Benutzung der Tokens einschränkt, ist sicherlich sinnvoll. Hierfür sollte ein Protokoll entworfen werden, das es den Besitzern der Token – den Mitgliedern der Wahlleitung – ermöglicht festzustellen, ob die Token auch nur für die für den Ablauf der Wahl notwendigen Entschlüsselungen verwendet wurden. Die einfachste Möglichkeit, die jedoch noch keinen vollständigen Schutz bietet, ist die Protokollierung, an wie vielen Entschlüsselungen das Token mitgewirkt hat. Dann ist es im Nachhinein möglich festzustellen, ob die Zahl der Entschlüsselungen zu der der abgegebenen Stimmen passt. Die Token sollten sich – wie die Urne bei Papierwahlverfahren – zugriffsgeschützt im Blickfeld der Wahlaufsicht befinden.

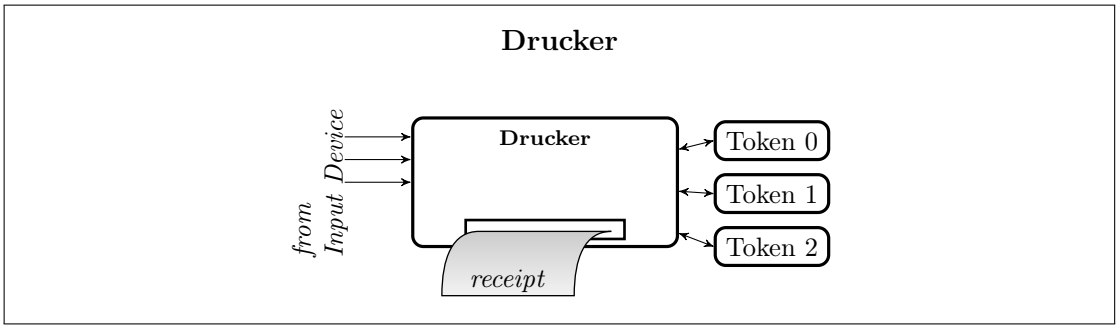

Abbildung 5.13.: Der Drucker verwendet Hardwaretoken der Mitglieder der Wahlleitung zur Entschlüsselung der Füllstimmen respektive der Zufallszahl, die das pOT ausgegeben hat. Die Hardwaretoken speichern den geheimen Schlüssel sk_l der Wahlleitung a_l.

Bulletin Board Wie die meisten kryptographischen Wahlverfahren benötigen auch wir eine Möglichkeit, Daten so zu veröffentlichen, dass sie nicht mehr verändert werden können (siehe Abschnitt 2.7). Dazu wird üblicherweise ein sogenanntes Bulletin Board ($\mathcal{BB}$) verwendet. In unserem Fall benötigen wir jedoch nicht nur eine Möglichkeit, Daten zu veröffentlichen, so dass sie jeder lesen kann. Wir benötigen außerdem die Möglichkeit, Daten so zu veröffentlichen, dass sie ausschließlich die Mitglieder der Wahlleitung lesen können. Da die Daten, die für jeden zugänglich sein sollen, und die Daten, die nur für die Mitglieder der Wahlleitung lesbar sein dürfen, eng miteinander verknüpft sind, wollen wir sie in der Darstellung nicht trennen. Wir bezeichnen sie stattdessen als *interne* und als *öffentliche* Daten. Die Umsetzung des $\mathcal{BB}$s kann mit Hilfe zweier unabhängiger Systeme gelöst werden.

Daten, die wir auf dem $\mathcal{BB}$ für alle sichtbar veröffentlichen, sind unter anderem der erste Teil der blinden Commitments: das Commitment selbst. Auch die Korrektheitsbeweise werden hier veröffentlicht. Auf dem internen Teil werden der zweite und dritte Teil veröffentlicht: die Chiffrate, die die Unveil-Informationen enthalten. Eine Übersicht ist in Abbildung 5.14 dargestellt.

Vorbereitungsphase

Die Vorbereitungsphase lässt sich in vier Abschnitte unterteilen. Zunächst muss das notwendige Schlüsselmaterial generiert werden. In einem zweiten Schritt werden die blinden Commitments auf die Füllstimmen und die Kandidaten erzeugt, die im darauffolgenden Schritt veröffentlicht werden. Im vierten Schritt wird die Wahlmaschine mit den für die Wahl erforderlichen Daten versorgt.

Öffentliche Parameter und Schlüsselerzeugung Im ersten Teil der Vorbereitungsphase werden die öffentlichen Parameter festgelegt und das Schlüsselmaterial erzeugt. Da die Authentifizierung – losgelöst von der kryptographischen Lösung – wie bei klassischen Papierwahlverfahren durchgeführt wird, werden nur Schlüssel für die Wahlleitung benötigt,

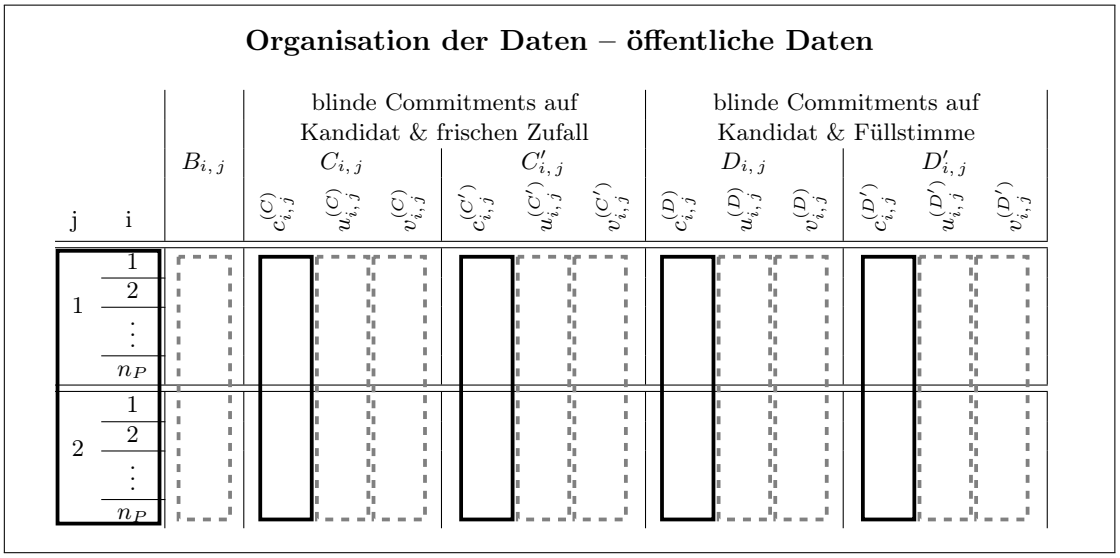

Abbildung 5.14.: Die Tabelle zeigt die Struktur der vor der Wahl berechneten und während der Wahl anfallenden Daten. Nicht alle Daten werden für alle (Wahlberechtigte und nicht Wahlberechtigte) sichtbar veröffentlicht. Die Daten, die das Wahlgeheimnis informationstheoretisch schützen (Commitments), können für jeden sichtbar zugänglich gemacht werden (schwarz umrahmt); sie sind für den Beweis der korrekten Auszählung notwendig. Die Daten, die das Wahlgeheimnis nur komplexitätstheoretisch schützen (Chiffrate), werden nur zwischen den Mitgliedern der Wahlleitung geteilt (grau gestrichelt umrahmt); sie sind notwendig, um die Auszählung und den Korrektheitbeweis erstellen zu können.

die teilweise auch der späteren Verifikation dienen. Dazu gehören die Schlüssel für das Threshold-Verschlüsselungsverfahren, die Signaturschlüssel und die öffentlichen Parameter für das Commitmentverfahren (siehe Abbildung 5.6). Da die Chiffrate nur für die interne Kommunikation der Mitglieder der Wahlleitung und der Wahlmaschine dienen, muss der öffentliche Schlüssel des Verfahrens $\mathrm{pk}_{\mathcal{T}}$ auch nur diesen Parteien bekannt sein. Konkret sind das die Mitglieder der Wahlleitung, da sie $\mathrm{pk}_{\mathcal{T}}$ für die Erstellung der blinden Commitments benötigen, der RNG, der ein blindes Commitment auf die frisch gezogene Zufallszahl erstellen muss, und die Rerandomisierungsbox, die für die Rerandomisierung ein Chiffrat der Null mit frisch gezogenem Zufall benötigt. Die Verifikationsschlüssel für die von der Wahlleitung erstellten Signaturen werden hingegen für die universell prüfbare Korrektheit ebenso benötigt wie die öffentlichen Parameter des Commitmentverfahrens. Daher werden sie auf dem $\mathcal{BB}$ veröffentlicht.

Der Prozess der Schlüsselerzeugung folgt den Bedingungen der verwendeten Verfahren. Da die Verfahren für das Commitment und für die Threshold-Verschlüsselung eng miteinander verwoben sind, machen wir in Abbildung 5.6 keine näheren Angaben über den Ersteller des Schlüsselmaterials. Insbesondere bei der Erstellung der geheimen Schlüssel des Threshold-Verschlüsselungsverfahrens ist es wünschenswert, dass diese ebenso wie die blinden Commitments verteilt erstellt werden und nur das entsprechende Mitglied

Überblick über die Vorbereitungsphase

Öffentliche Parameter und Schlüsselmaterial:

- Die Mitglieder der Wahlleitung erzeugen gemeinsam g und h für das Commitmentverfahren.
- Jedes Mitglied der Wahlleitung erzeugt ein Schlüsselpaar zum Signieren.
- Das Schlüsselmaterial für das Threshold-Verschlüsselungsverfahren wird berechnet und an die Mitglieder der Wahlleitung verteilt.

Erstellung der blinden Commitments:

- Die Mitglieder der Wahlleitung erzeugen blinde Commitments auf die Füllstimmen $N_{i,j}$.
- Ein Mitglied der Wahlleitung erstellt ein blindes Commitment je Kandidatennamen p_i,
- und erstellt die Tupel aus Füllstimme und Kandidat ($\texttt{bCOM}(N_{i,j})$, $\texttt{bCOM}(p_i)$)

Veröffentlichung:

- Die Mitglieder der Wahlleitung veröffentlichen die Daten (siehe Abbildung 5.16).

Konfiguration der Maschine:

- Das Storage-Device erhält die Chiffrate der Füllstimmen.
- Die Append-x-Box, Rerandomize-Box und Split-Box erhalten ihre Parameter.
- Direkt vor Beginn der Wahl: Drucker erhält von den Mitgliedern der Wahlleitung die Token.

Abbildung 5.15.: Übersicht über den Ablauf der Vorbereitungsphase

der Wahlleitung seinen geheimen Schlüssel kennt. Ähnliches gilt für die Parameter des Commitmentverfahrens.

Erstellung der blinden Commitments und Veröffentlichung der Daten Im zweiten Teil der Vorbereitungsphase werden die Füllstimmen erzeugt. Dies geschieht nach dem Prinzip des Secret-Sharings [108]. Die grundlegenden Techniken sind in Kapitel 5.3.2 beschrieben.

Wir geben für einen besseren Überblick während der Beschreibung der Erstellung der blinden Commitments auch gleich an, wo welche Daten veröffentlicht werden. Einen Überblick über die veröffentlichten Daten gibt die Abbildung 5.16. Zu Beginn gehen wir davon aus, dass die öffentlichen Parameter und die Verifikationsschlüssel, die in der letzten Phase generiert wurden, bereits veröffentlicht sind.

Bevor wir den Ablauf der Berechnungen und der Kommunikation zwischen den einzelnen Mitgliedern der Wahlleitung erläutern, fassen wir die grundlegende Idee zusammen.

Idee Für jede benötigte Füllstimme erzeugt jedes Mitglied der Wahlleitung a_l einen Share r_0, erstellt ein Commitment auf diesen und verschlüsselt die Unveil-Informationen (r_0, r_1) des Commitments:

$$c_{i,j}^{(a_l)} := \texttt{COM}(r_0, r_1), \qquad u_{i,j}^{(a_l)} := \texttt{ENC}(r_0, r_2), \qquad v_{i,j}^{(a_l)} := \texttt{ENC}(r_1, r_3).$$

Dank der homomorphen Eigenschaft des Commitmentverfahrens können die Commitments auf die Shares aller Mitglieder der Wahlleitung homomorph zu einem Commitment auf die Füllstimme zusammengefügt werden: Gleiches gilt für die Chiffrate, die die Unveil-Informationen enthalten.

$$c_{i,j}^{(D')} := \prod_{l=1}^{n_A} c_{i,j}^{(a_l)} \qquad u_{i,j}^{(D')} := \prod_{l=1}^{n_A} u_{i,j}^{(a_l)} \qquad v_{i,j}^{(D')} := \prod_{l=1}^{n_A} v_{i,j}^{(a_l)}$$

Ablauf für blinde Commitments auf Füllstimmen Für eine Garantie, dass die Füllstimmen von niemandem beeinflusst werden können, ist die Reihenfolge der einzelnen Schritte während der Erstellung von Bedeutung. Wir erläutern im Folgenden die Abschnitte des Pseudocodes aus Algorithmus 4 aus der Perspektive eines Wahlleitungsmitglieds a_l.

1. a_l **erstellt die Shares auf die Füllstimmen** (Zeile 3 – 10)
   Erstelle ein blindes Commitment für jeden einzelnen Share.

2. a_l **legt sich auf die Wahl der Shares fest** (Zeile 11 – 15)
   Erstelle ein Commitment $\zeta^{(a_l)}$ auf alle eben erstellten blinden Commitments und veröffentliche dieses intern. *Dabei ist zu beachten, dass die Reihenfolge, in welcher die Mitglieder ihre Daten intern veröffentlichen, festgelegt ist; beispielsweise bezüglich l in aufsteigender Reihenfolge (siehe Zeile 13).*

3. a_l **sendet die Unveil-Informationen der Commitments** $\zeta^{(a_l)}$ (Zeile 16 – 22)
   Sende die Unveil-Informationen zu $\zeta^{(a_l)}$ an die anderen Mitglieder der Wahlleitung, so dass diese $\zeta^{(a_l)}$ überprüfen können, wenn die zuvor erhaltenen Unveil-Informationen (a_{n_A} bis a_{l+1}) der anderen Mitglieder korrekt sind. Prüfe im weiteren Verlauf die Unveil-Informationen der nachfolgenden Mitglieder. Brich die Protokollausführung im Fehlerfall ab. *In den Unveil-Informationen sind auch die blinden Commitments auf die Shares enthalten. Das Versenden der Unveil-Informationen erfolgt verglichen mit dem Versenden des Commitments in umgekehrter Reihenfolge (vergleiche dazu die Zeilen 13 und 17).*

4. a_l **berechnet blinde Commitments auf Füllstimmen** (Zeile 23 – 31)
   Aus den blinden Commitments auf die Shares der anderen Mitglieder der Wahlleitung und der in Schritt eins erstellten blinden Commitments auf die eigenen Shares können nun die blinden Commitments auf die Füllstimmen berechnet werden. Erstelle als Beleg, mit den bisher berechneten Daten einverstanden zu sein, zwei Signaturen: eine Signatur auf die Daten, die nach eigenen Berechnungen zufolge für alle sichtbar veröffentlicht werden müssen, und eine Signatur auf die Daten, die intern sichtbar sein werden.

5. a_l **veröffentlicht die Daten** (Zeile 32 – 42)
   Veröffentliche die im letzten Schritt berechneten Signaturen entsprechend der Sichtbarkeit der signierten Daten. *Ein Mitglied der Wahlleitung veröffentlicht das Ergebnis der Berechnung der blinden Commitments auf die Füllstimmen: Die Commitments*

Algorithmus 4 Algorithmus zur Erzeugung der blinden Commitments (für alle a_l)

const $1 \leq l \leq n_A$ ▷ ID von a_l
procedure CreateBlindCommitmentsOnDummyVotes(n_P, n_E)
▷ Erstelle Shares:
  for $1 \leq i \leq n_P$ **do** ▷ für jeden Kandidaten und...
    for $1 \leq j \leq n_E$ **do** ▷ ... für alle Wähler...
      $r_1, r_2, r_3, r_4 \leftarrow$ random value ▷ ... wähle vier Zufallszahlen und:
      $c_{i,j}^{(a_l)} \leftarrow \texttt{COM}(r_1, r_2)$ ▷ erzeuge Commitment
      $u_{i,j}^{(a_l)} \leftarrow \texttt{ENC}(r_1, r_3)$ ▷ erzeuge Unveil-Information
      $v_{i,j}^{(a_l)} \leftarrow \texttt{ENC}(r_2, r_4)$ ▷ erzeuge Unveil-Information
    end for
  end for
▷ committe auf eigene Shares ohne Kenntnis der anderen:
  $\zeta^{(a_l)} \leftarrow \texttt{COM}\left(\{\{c_{i,j}^{(a_l)}, u_{i,j}^{(a_l)}, v_{i,j}^{(a_l)}\}_{i=1}^{n_P}\}_{j=1}^{n_E}\right)$
  if $l \neq 1$ **then** ▷ veröffentliche Commitments in aufsteigender Reihenfolge
    warte, bis $\zeta^{(a_{l-1})}$ intern veröffentlicht wurde
  end if
  veröffentliche intern: $\zeta^{(a_l)}$
▷ Unveile die Commitments auf die Shares:
  if $l \neq n_A$ **then** ▷ unveile in umgekehrter Reihenfolge
    warte, bis $\zeta^{(a_{l+1})}$ unveilt ist
  end if
  veröffentliche intern: Unveil Informationen von $\zeta^{(a_l)}$
  if wenn Unveilinformationen $\zeta^{(a_1)}$, $\zeta^{(a_2)}$, ..., or $\zeta^{(a_{n_A})}$ nicht stimmen **then**
    melde den Fehler und brich ab
  end if
▷ zur Erinnerung: $\{\{c_{i,j}^{(a_l)}, u_{i,j}^{(a_l)}, v_{i,j}^{(a_l)}\}_{i=1}^{n_P}\}_{j=1}^{n_E}$ sind Teil der Unveilinformationen $\zeta^{(a_l)}$
▷ Berechne blinde Commitments auf Füllstimme:
  for $1 \leq i \leq n_P$ **do** ▷ für jeden Kandidaten und...
    for $1 \leq j \leq n_E$ **do** ▷ ... alle Wähler:
      $c_{i,j}^{(D')} \leftarrow \prod_{k=1}^{n_A} c_{i,j}^{(a_k)}$ ▷ erstelle das Commitment des blinden Commitments
      $u_{i,j}^{(D')} \leftarrow \prod_{k=1}^{n_A} u_{i,j}^{(a_k)}$ ▷ erstelle Unveil-Information des blinden Commitments
      $v_{i,j}^{(D')} \leftarrow \prod_{k=1}^{n_A} v_{i,j}^{(a_k)}$ ▷ erstelle Unveil-Information des blinden Commitments
    end for
  end for
  $\sigma_{\text{pub}}^{(a_l)} \leftarrow \texttt{SIG}_{s_l^{(sig)}}\left(\left\{\left\{c_{i,j}^{(D')}\right\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}\right)$ ▷ bestätige korrektes Ergebnis
  $\sigma_{\text{int}}^{(a_l)} \leftarrow \texttt{SIG}_{s_l^{(sig)}}\left(\left\{\left\{(u_{i,j}^{(D')}, v_{i,j}^{(D')})\right\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}\right)$ ▷ bestätige korrektes Ergebnis
▷ Veröffentliche Daten:
  if $l = 1$ **then** ▷ a_1 veröffentlicht Daten
    öffentlich: $\{\{c_{i,j}^{(D')}\}_{i=1}^{n_P}\}_{j=1}^{n_E}$ ▷ Commitments öffentlich
    intern: $\{\{u_{i,j}^{(D')}, v_{i,j}^{(D')}\}_{i=1}^{n_P}\}_{j=1}^{n_E}$ ▷ verschlüsselte Unveil-Information intern
  end if
  öffentlich: $\sigma_{\text{pub}}^{(a_l)}$ ▷ veröffentliche Signatur öffentlich
  intern: $\sigma_{\text{int}}^{(a_l)}$ ▷ veröffentliche Signatur intern
  for $1 \leq l \leq n_A$ **do**
    verifiziere $\sigma_{\text{pub}}^{(a_l)}$ und $\sigma_{\text{int}}^{(a_l)}$ ▷ wenn nicht, melde Fehler und brich ab
  end for
  signiere „alles ok“ zusammen mit einer eindeutigen Bezeichnung der Wahl
  bestätige durch Veröffentlichung dieser Signatur den korrekten Ablauf
end procedure

aus den blinden Commitments sind für jeden sichtbar, die Chiffrate nur für Mitglieder der Wahlleitung.

Für den Fall, dass das erwartete Verhalten[7] der anderen Mitglieder der Wahlleitung mit dem tatsächlichen Geschehen nicht übereinstimmt, muss das Protokoll geordnet abgebrochen werden. Wie das geschieht, ist nicht Fokus der Arbeit und hängt von den rechtlichen Rahmenbedingungen ab. Entscheidend ist, dass – sollte eines der Mitglieder der Wahlleitung Bedenken bzgl. der Korrektheit des Ablaufs haben – jeder, der die Wahl beobachtet, über die Mängel im Ablauf des Protokolls unterrichtet ist.

Technische Bemerkung: Unveil-Reihenfolge
Die Commitments auf die Shares werden – verglichen mit der Veröffentlichung – in umgekehrter Reihenfolge geöffnet (siehe Zeilen 13 und 17 aus Algorithmus 4). Dies geschieht, um Manipulationen auf Basis der veröffentlichten Commitments zu verhindern. Diesen Zusammenhang, den wir bereits in Abschnitt 2.4.2 beschrieben haben, fassen wir hier noch einmal kurz zusammen.

Manuel Blum hat 1983 ein Protokoll veröffentlicht [19], wie zwei Parteien, die sich nicht im gleichen Raum befinden, mit Hilfe eines Commitments ein zufälliges Bit b berechnen können; sie führen einen fairen Münzwurf aus. Dazu legt sich (nach der Parameterwahl) Protokollteilnehmer A auf ein Bit b_A fest, berechnet ein Commitment c auf dieses Bit und sendet c an Protokollteilnehmer B. B wählt jetzt seinerseits ein Bit b_B und sendet dieses im Klartext an A. Nun kann A die Unveil-Information zu c an B senden. Anschließend können beide Protokollteilnehmer das Ergebnisbit $b = b_A \oplus b_B$ berechnen.

Sendet B nicht das gewählte Bit, sondern ebenfalls ein Commitment, ist die Unveil-Reihenfolge entscheidend, ob das Protokoll weiterhin sicher ist:

Protokollteilnehmer A öffnet das Commitment zuerst Sendet Protokollteilnehmer A zuerst die Unveil-Information, ist es – je nach Beschaffenheit des Commitments – Protokollteilnehmer B möglich, eine Variante von c statt eines eigenen Commitments zu senden. Die Unveil-Informationen zu c erhält B von A, bevor B das modifizierte Commitment öffnen muss. Dadurch ist es B möglich, ein bestimmtes Ergebnis zu erzwingen.

Protokollteilnehmer B öffnet das Commitment zuerst Wird die Unveil-Reihenfolge umgedreht, so kennt B zwar das Commitment von A, jedoch nicht die zugehörigen Unveil-Informationen. Jede Partei kennt vor der Berechnung des eigenen Commitments entweder das Commitment der anderen Partei (im Fall von B) oder vor dem Öffnen des eigenen Commitments die Unveil-Informationen der anderen Partei (im Fall von A); jedoch niemals beides. Das Protokoll bleibt sicher.

Es kann gezeigt werden, dass diese Eigenschaft auch dann gilt, wenn das Protokoll auf mehrere Parteien ausgeweitet wird.

Ablauf für blinde Commitments auf Kandidatennamen Die Erstellung der blinden Commitments auf die Kandidaten erfolgt – da die Nachrichten nicht zufällig sind – nach einem leicht abgewandelten Schema, das in Algorithmus 5 als Pseudocode dargestellt ist. Bei der Erstellung der blinden Commitments für die Kandidaten kann mit offenen Karten gespielt werden. Ein Mitglied der Wahlleitung wählt für jeden Kandidaten einmal die drei notwendigen Zufallszahlen und erzeugt damit die Commitments und die Chiffrate,

[7] fehlerhafte Signaturen, fehlende oder falsche Daten, Abweichungen zwischen den veröffentlichten Ergebnissen und den eigenen Berechnungen, ...

die in einem blinden Commitment enthalten sind. Das Commitment wird zusammen mit dem Namen des Kandidaten und des zur Erstellung verwendeten Zufalls veröffentlicht. Die Chiffrate werden – zusammen mit dem für die Verschlüsselung verwendeten Zufall – intern kommuniziert.

Algorithmus 5 Erstellung des blinden Commitments auf alle p_i (nur a_1)

1: **procedure** CreateBlindCommitmentsOnCandidates($n_P, \{p_i\}_{i=1}^{n_P}$)
2: **for** $1 \leq i \leq n_P$ authority member a_1 **do** ▷ für jeden Kandidaten...
3: $r_1, r_2, r_3 \leftarrow$ random value ▷ ...wähle drei Zufallszahlen und:
4: $c_{i,1}^{(D)} \leftarrow \texttt{COM}(p_i, r_1)$ ▷ erzeuge Commitment
5: $u_{i,1}^{(D)} \leftarrow \texttt{ENC}(p_i, r_2)$ ▷ erzeuge Unveil-Information
6: $v_{i,1}^{(D)} \leftarrow \texttt{ENC}(r_1, r_3)$ ▷ erzeuge Unveil-Information
7: öffentlich: $c_{i,1}^{(D)}, p_i, r_1$ ▷ veröffentliche Commitment
8: intern: $u_{i,1}^{(D)}, v_{i,1}^{(D)}, r_2, r_3$
9: **end for**
10: **end procedure**

Auf diesem Weg ist die korrekte Erstellung der (blinden) Commitments nachvollziehbar. Die blinden Commitments – die die Eigenschaft *blind* und *hiding* bisher nicht erfüllen – werden durch die Rerandomisierung während des Shuffles tatsächlich zu blinden Commitments, deren Inhalt und Unveil-Information nicht von weniger als k Mitgliedern der Wahlleitung rekonstruiert werden können.

Zuordnung der blinden Commitments auf Füllstimme und Kandidat Im Folgenden wird den blinden Commitments auf die Füllstimmen von Kandidat p_i, von denen zuvor mindestens n_E Stück erzeugt wurden, eine Kopie des blinden Commitments auf p_i zugeordnet. Dadurch ist öffentlich jedem Commitment auf eine Füllstimme ein Commitment auf den Kandidaten zugeordnet. Der Inhalt des Commitments auf den Kandidaten ist jedem bekannt; der Inhalt des Commitments auf die Füllstimme ist keiner einzelnen Person/Maschine bekannt. Während der Auszählung werden die blinden Commitments auf die Kandidaten zusammen mit dem ihnen zugeordneten blinden Commitment auf die Füllstimme als Paar durch einen Shuffle geführt. Dadurch werden aus den blinden Commitments auf den Kandidaten *echte* blinde Commitments.

Wir weichen hier vom ursprünglichen Entwurf von Bingo Voting etwas ab. Wir veröffentlichen mehr Informationen. Das ist nicht notwendig, vereinfacht aber die Kontrolle der vorberechneten Daten.

Während der Originalentwurf nur die Information preisgibt, dass es gleich viele Commitments auf alle Kandidaten gibt, zeigen wir auch, welches Commitment welchen Kandidaten enthält. Im Folgenden möchten wir kurz darauf eingehen, warum das möglich ist.

Im Originalentwurf kennt der, der den Beweis führt, ebenfalls die Inhalte der Commitments. Die Commitments liegen in zufällig permutierter Reihenfolge vor. Um die Korrektheit eines Belegs zu zeigen, werden die $n_P - 1$ Commitments der für den Beleg verwendeten Füllstimmen genommen und um das Commitment ergänzt, das während der Wahl auf die frische Zufallszahl und den Kandidaten erzeugt wurde. Wäre bekannt, welche Inhalte die Commitments haben, ließen sich hier Rückschlüsse auf den Inhalt des Stimmzettels ziehen.

In unserem Entwurf werden alle n_P Commitments auf die Füllstimmen eines Belegs um Commitments auf die frische Zufallszahl ergänzt. Die Reihenfolge der Commitments entspricht noch genau der Reihenfolge der Kandidaten auf dem Wahlzettel (oder einer anderen beliebigen, aber festen Reihenfolge). Erst nach einem Shuffle wird je Kandidat entweder das Commitment auf die Füllstimme und das auf den Kandidaten oder auf die frische Zufallszahl und den Kandidaten verwendet.

Der Originalentwurf ließe sich durch einen zusätzlichen Shuffle vor dem Aussortieren der Füllstimme des gewählten Kandidaten ebenfalls dahingehend modifizieren.

Durch die Signaturen kann sich jeder Wähler davon überzeugen, dass kein Mitglied der Wahlleitung Bedenken bezüglich der Vorberechnungen hat, sofern es nicht so unter Druck gesetzt wurde, dass es wider seine Überzeugungen gehandelt hat.

Veröffentlichte Daten auf dem Bulletin Board

Vorbereitungsphase: Daten, die während der Vorbereitungsphase veröffentlicht werden:

- öffentliche Parameter des Commitmentverfahrens: g und h.
- Verifikationsschlüssel der Mitglieder der Wahlleitung: $\{\mathrm{s}_l^{(ver)}\}_{l=1}^{n_A}$
- Commitments der blinden Commitments auf Füllstimmen und Kandidaten: $\left\{\{c_{i,j}^{(D)}, c_{i,j}^{(D')}\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}$
- Die Unveil-Informationen zu $\left\{\{c_{i,j}^{(D)}\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}$ (Commitments auf die Kandidaten).
- Signaturen über alle Commitments von jedem Wahlleitungsmitglied: $\{\sigma_{\text{öffentlich}}^{(a_l)}\}_{l=1}^{n_A}$
- Bestätigung jedes Wahlleitungsmitglieds, dass die Vorberechnung erfolgreich und ehrlich abgeschlossen ist.

Abbildung 5.16.: Die Tabelle zeigt einen Überblick über die in der Vorbereitungsphase veröffentlichten Daten.

Konfiguration der Maschine Zum Schluss wird die Wahlmaschine konfiguriert und mit den notwendigen Daten versorgt:

- Das Storage-Device bekommt die Chiffrate auf die Füllstimmen $\{\{u_{i,j}^{(D')}\}_{i=1}^{n_P}\}_{j=1}^{n_E}$ übergeben. (Siehe Abbildung 5.8.)

- Die Append-x-Box bekommt den String, den sie an Nachrichten anhängen muss, in einem nicht beschreibbaren Speicherbaustein (ROM) übergeben.
- Die Rerandomize-Box erhält den öffentlichen Schlüssel $\mathrm{pk}_{\mathcal{T}}$, den sie für die Rerandomisierung der Chiffrate benötigt (ebenfalls auf einem ROM).
- Der Split-Box muss eventuell noch die Länge der Chiffrate oder das Trennzeichen zwischen den Chiffraten übergeben werden, damit diese korrekt getrennt werden können. Da sich diese während der Wahl nicht ändern, ist es ratsam, auch hier schreibgeschützten Speicher einzusetzen.

Durchführung des Wahlprotokolls

Im Folgenden beschreiben wir den Vorgang der Stimmabgabe eines Wählers. Dazu wurde im Vorfeld – vor Beginn der Wahlphase – die Wahlmaschine mit Füllstimmen in Form von einem Set verschlüsselter Füllstimmen pro Wähler bestückt. Die Wahlberechtigung des Wählers wird geprüft, bevor der Wähler die Wahlkabine betritt, in der die Wahlmaschine steht. Die Prüfung der Wahlberechtigung ist in diesem Fall unabhängig vom Wahlverfahren und daher außerhalb des Fokus dieser Arbeit. Im Anschluss gehen wir detaillierter auf die internen Abläufe im Storage-Device und den Gesamtzusammenhang der dort abgelegten Daten und der Auszählung ein.

Stimmabgabe Ist die Autorisierung des Wählers j geprüft, betritt er die Wahlkabine. In der Wahlkabine drückt er den Knopf des Kandidaten, den er wählen möchte, und bestätigt seine Eingabe durch das Betätigen eines weiteren Knopfes. Dieser Bestätigungsknopf setzt die internen Abläufe der Wahlmaschine in Gang. Abbildung 5.17 gibt einen Überblick über die Komponenten der Maschine und die Teilnehmer des Protokolls und welche Daten während der Stimmabgabe zwischen ihnen ausgetauscht werden. Im Folgenden erläutern wir die Aktionen und den Ablauf.

Für jeden Kandidaten i sendet das Storage-Device das Chiffrat $u_{i,\,j}^{(D')}$, das die Füllstimme $N_{i,\,j}$ enthält, an den pOT im Input-Device, der dem Kandidaten i zugeordnet ist. Gleichzeitig zieht der RNG eine frische Zufallszahl R_j, die die Füllstimme des gewählten Kandidaten ersetzt. Mit Hilfe dreier weiterer Zufallszahlen erstellt der RNG ein blindes Commitment $C'_{1,\,j}$ auf R_j. Den zweiten Teil des blinden Commitments – die verschlüsselte Zufallszahl R_j – sendet der RNG an jeden pOT des Input-Device. Das blinde Commitment $C'_{1,\,j}$ als Ganzes erhält das Storage-Device. Auf seinem Display zeigt der RNG R_j an.

Im Input-Device werden die Daten, die das Storage-Device und der RNG gesendet haben, weiterverarbeitet. Zunächst werden die Chiffrate durch eine Append-x-Box ergänzt: Die Eingaben, die das Storage-Device sendet, werden durch ein Chiffrat ergänzt, das eine Null enthält. Die Eingabe, die vom RNG gesendet wird, wird vor jedem pOT mit dem Chiffrat einer Eins ergänzt. Im Anschluss werden die Tupel rerandomisiert und an den pOT weitergeleitet (Siehe Abbildung 5.11). Ein pOT überträgt eine der beiden Eingänge

Datenfluss während der Stimmabgabe

Während der Stimmabgabe fließen innerhalb der Wahlmaschine einige Daten. Wir geben hier einen Überblick über die Daten, die während der Stimmabgabe des j-ten Wählers fließen.

Wähler	→	**Input-Device:**	β_j
Storage-Device	→	**Input-Device:**	$\{u_{i,j}^{D'}\}_{i=1}^{n_P}$
RNG	→	**Storage-Device:**	$C'_{1,j} = \mathtt{bCOM}(R_j) = \left(\mathtt{COM}(R_i, r_j), \mathtt{ENC}(R_j), \mathtt{ENC}(r_j)\right)$
RNG	→	**Input-Device:**	$\mathtt{ENC}(R_j)$
RNG	→	**Wähler:**	R_j
Input-Device	→	**Storage-Device:**	$\{\mathtt{ENC}(b_{i,j})\}_{i=1}^{n_P}$
Input-Device	→	**Drucker:**	$\{\mathtt{ENC}(x)\}_{i=1}^{n_P}$ wobei $x = R_j$ wenn $\beta_j \stackrel{\wedge}{=} i$, $x = N_{i,j}$ sonst.
Drucker (Beleg)	→	**Wähler:**	$\{(p_i, x)\}_{i=1}^{n_P}$ wobei $x = R_j$ wenn $\beta_j \stackrel{\wedge}{=} i$, $x = N_{i,j}$ sonst.

Abbildung 5.17.: Während der Abgabe einer Stimme werden Daten zwischen den einzelnen Komponenten ausgetauscht. Das Storage-Device sendet n_P Chiffrate an das Input-Device und erhält die gleiche Zahl an Chiffraten zurück. Der RNG sendet ein Chiffrat an das Input-Device und ein blindes Commitment an das Storage-Device. Der Drucker erhält n_P Chiffrate.

an seinen Ausgang: Wenn der Knopf des pOT vom Wähler betätigt wurde, leitet er die Eingabe des RNG weiter, im anderen Fall die Eingabe des Storage-Device. Die Ausgaben der pOTs werden von je einer Split-Box getrennt. Der erste Teil ($\mathtt{ENC}(N_{i,j})$ bzw. $\mathtt{ENC}(R_j)$) wird an den Drucker weitergeleitet, der zweite Teil ($\mathtt{ENC}(0)$ bzw. $\mathtt{ENC}(1)$) an das Storage-Device (siehe Abbildung 5.12).

Das Storage-Device erhält für jeden Kandidaten i ein Chiffrat, das wir mit $B_{i,j}$ bezeichnen, und speichert es zu dem entsprechenden Chiffrat $u_{i,j}^{(D')}$.

Der Drucker kann sich die Chiffrate, die er von den Split-Boxen erhält, mit Hilfe der als Entschlüsselungsorakel fungierenden Hardwaretoken der Mitglieder der Wahlleitung teilentschlüsseln lassen. Aus den Rückgabewerten der Hardwaretoken kann der Drucker den Klartext berechnen. Hat er alle Chiffrate entschlüsselt, druckt er den Beleg für den Wähler. Dieser kann nun die Zufallszahl, die hinter seinem gewählten Kandidaten steht, mit der auf dem Display des RNG vergleichen.

Sowohl bei der Übergabe der Chiffrate an den Drucker als auch an das Storage-Device ist wichtig, auf welchem Kanal die Komponenten die an sie gesendeten Daten erhalten. Daraus können sie ablesen, zu welchem Kandidaten der Wert, den sie erhalten, gehört. Da wir die Wahlmaschine als passiv korrumpiert annehmen, muss eine Manipulation an der Stelle nicht befürchtet werden. Ein Vertauschen der Elemente würde lediglich zu einem nicht funktionierenden Beweis führen. Eine unauffällige Manipulation ist auf diesem Weg nicht möglich.

Organisation der Daten In Abbildung 5.8 haben wir die Struktur der vor der Wahl berechneten und während der Wahl anfallenden Daten bereits dargestellt. Abbildung 5.18 enthält die gleiche Tabelle; sie zeigt, welche Daten identisch sind und daher nur einmal berechnet und anschließend kopiert werden.

Die Zeilen der Tabelle werden mit Hilfe zweier Zähler angesprochen: Der Zähler $1 \leq j \leq n_E$ steht für den Wähler, der Zähler $1 \leq i \leq n_P$ für die Wahlmöglichkeit (Kandidat) des Wählers. Für jede Kombination aus i und j gibt es eine Zeile. Wir sprechen von einem Set, wenn alle Zeilen eines Wählers gemeint sind: $\{(i, j)\}_{i=1}^{n_P}$.

Jede Zeile (i, j) der Tabelle enthält fünf Elemente:

$D_{i,j}$:	ein blindes Commitment auf den Kandidatennamen p_i	$(i = i' \Rightarrow D_{i,j} = D_{i'j'})$
$D'_{i,j}$:	ein blindes Commitment Füllstimme $N_{i,j}$	
$C_{i,j}$:	ein blindes Commitment auf den Kandidatennamen p_i	$(C_{i,j} = D_{i,j})$
$C'_{i,j}$:	ein blindes Commitment auf den frisch gezogenen Zufall R_j	$(j = j' \Rightarrow C'_{i,j} = C'_{i'j'})$
$B_{i,j}$:	ein Chiffrat, das eine Null oder eine Eins enthält	

Die Spalten $D_{i,j}$ und $D'_{i,j}$ werden vor der Wahl von der Wahlleitung mit den in der Vorbereitungsphase erstellten blinden Commitments gefüllt (in Abbildung 5.8 hellgrau hinterlegt). Die Elemente $C_{i,j}$ sind Kopien der Einträge $D_{i,j}$.

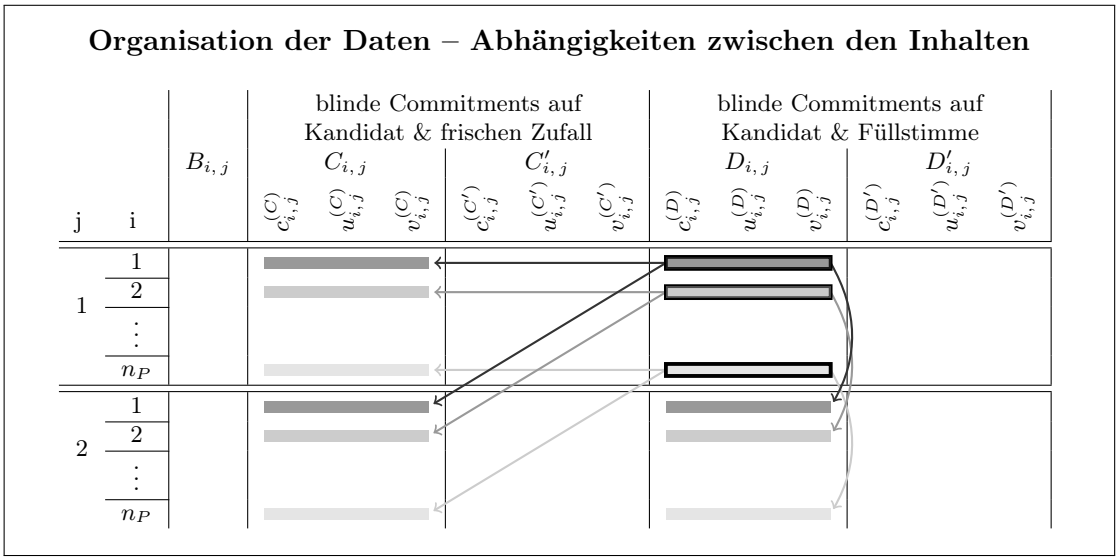

Abbildung 5.18.: Die Tabelle zeigt die Struktur der vor der Wahl berechneten und während der Wahl anfallenden Daten. Manche Tabellenzellen enthalten Kopien anderer Zellen: Die Daten aus den grau hinterlegten Zellen, die schwarz umrandet sind, werden in die grau hinterlegten Zellen ohne Umrandung kopiert. Die Daten, die kopiert werden, sind blinde Commitments auf die Kandidaten, die zu diesem Zeitpunkt noch öffentlich verifizierbar sind (Klartext und Unveil-Information sind öffentlich bekannt).

Während der Stimmabgabe des Wählers e_j werden die Daten erstellt, mit denen sein Set vervollständigt wird; die entsprechenden Zellen der Spalten $B_{i,j}$, $C'_{i,j}$ und $C_{i,j}$ (für $j = 1$ in Abbildung 5.8 dunkelgrau hinterlegt) werden mit den Ausgaben der Split-Boxen und des Zufallszahlengenerators sowie durch Kopien vorhandener Daten gefüllt. Wie aus der oberen Ablaufbeschreibung einer Stimmabgabe hervorgeht, sendet das Storage-Device Daten an das Input-Device und erhält ein blindes Commitment von dem RNG. Im Anschluss erhält es n_P Chiffrate vom Input-Device (aus den Split-Boxen). Abbildung 5.19 zeigt eine Übersicht über die Abläufe innerhalb des Storage-Devices.

Organisation der Daten innerhalb des Storage-Device

Vorbereitungsphase: Das Storage-Device erhält die verschlüsselten Füllstimmen.

1. Das Storage-Device erhält für jeden Wähler j einen Satz mit n_P Chiffraten:

$$\text{Wahlleitung} \xrightarrow{\left\{\left\{u_{i,\,j}^{(D')}\right\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}} \text{Storage-Device}$$

Wahlphase: Für jede Stimmabgabe des Wählers j:

2. Das Storage-Device sendet einen Satz Chiffrate an das Input-Device:

$$\text{Storage-Device} \xrightarrow{\left\{u_{i,\,j}^{(D')}\right\}_{i=1}^{n_P}} \text{Input-Device}$$

3. Im Anschluss erhält das Storage-Device ein blindes Commitment und n_P Chiffrate:

$$\text{RNG} \xrightarrow{C'_{1,\,j}:=\texttt{bCOM}\left(R_j\right)} \text{Storage-Device}$$

$$\text{Input-Device} \xrightarrow{\left\{B_{i,\,j}:=\texttt{ENC}\left(b_{i,\,j}\right)\right\}_{i=1}^{n_P}} \text{Storage-Device}$$

4. Diese werden den in Schritt 2 – dem ersten Schritt der Wahlphase – gesendeten Chiffraten zugeordnet gespeichert:

$$\left\{\left(B_{i,\,j},\, C'_{1,\,j},\, u_{i,\,j}^{(D')}\right)\right\}_{i=1}^{n_P}$$

Auszählungsphase: Gespeicherte Daten werden ausgelesen (für Auszählung siehe Abbildung 5.22).

5. Die Wahlleitung speichert die neuen Daten der n_β Stimmabgaben für die Auszählung:

$$\text{Storage-Device} \xrightarrow{\left\{\left\{\left(B_{i,\,j},\, C'_{1,\,j}\right)\right\}_{i=1}^{n_P}\right\}_{j=1}^{n_\beta}} \text{Wahlleitung}$$

Abbildung 5.19.: Während der Abgabe einer Stimme sendet und erhält das Storage unterschiedliche Daten. Zusätzlich werden Daten innerhalb des Speichers dupliziert. Die Übersicht zeigt die Zusammenhänge.

Bei den Daten, die das Storage-Device für die Stimmabgabe des Wählers e_j an das Input-Device sendet, handelt es sich um die Chiffrate $\{u_{i,\,j}^{(D')}\}_{i=1}^{n_P}$. Von dem Commitment, das das Storage-Device vom RNG erhält, wird je eine Kopie in den Zellen der Spalte $C'_{i,\,j}$ und den Zeilen $(i,\, j)$ mit $1 \leq i \leq n_P$ gespeichert. Die Rückgabe des Input-Device enthält genau n_P Chiffrate. Diese werden in den Zellen der Spalte $B_{i,\,j}$ und den Zeilen $(i,\, j)$ mit $1 \leq i \leq n_P$ gespeichert. Die verbleibenden Zellen der Spalte $C_{i,\,j}$ mit den Zeilennummern $(i,\, j)$ mit $1 \leq i \leq n_P$ werden mit einer Kopie des Zelleninhalts der entsprechenden Zelle aus der Spalte $D_{i,\,j}$ gefüllt.

Auszählung

Nach Ende der Wahlphase werden alle Belege veröffentlicht. Jeder Wähler kann sich davon überzeugen, dass sein Stimmzettel unter den veröffentlichten ist und sich daher – wenn der Beweis korrekt ist – auch sicher sein, dass seine Stimme in das Auszählungsergebnis eingegangen ist.

Die Auszählung mit dem dazugehörigen Korrektheitsbeweis kann in drei Schritte eingeteilt werden. Die veröffentlichten Commitments werden in zwei Listen anhand der verschlüsselten Bits $b_{i,\,j}$ aufgeteilt. Danach kann die Korrektheit der Belege bewiesen werden. Im Anschluss wird das Auszählungsergebnis anhand der ungenutzten Füllstimmen ermittelt.

Überblick über die Auszählungsphase

Aufteilung der veröffentlichten Commitments: Das Commitment auf die nicht verwendete Füllstimme repräsentiert die Stimme eines Stimmzettels. Die verwendeten Füllstimmen zusammen mit der frischen Zufallszahl werden für den Korrektheitsbeweis des Belegs verwendet.

Beweis der Korrektheit der Belege: Es wird gezeigt, dass $n_P - 1$ der Commitments auf Füllstimmen und ein Commitment auf die frische Zufallszahl zusammen mit den ihnen zugeordneten Commitments auf die Kandidatennamen den Beleg widerspiegeln.

Auszählung der ungenutzten Füllstimmen: Alle Commitments auf die *nicht* verwendeten Füllstimmen werden zusammen mit den ihnen zugeordneten Commitments auf die Kandidatennamen geöffnet. Anhand der hier erscheinenden Kandidatennamen kann das Auszählungsergebnis abgelesen werden.

Abbildung 5.20.: Übersicht über den Ablauf der Auszählungsphase

Zur Auszählung darf es nicht notwendig sein, das Wahlgeheimnis brechen zu können.

Es ist $k - 1$ (korrumpierten) Mitgliedern der Wahlleitung (= Auszählende) nicht möglich, Informationen über das Wahlergebnis und die Korrektheit hinaus zu gewinnen.

Zu den bereits veröffentlichen blinden Commitments D und D' werden nun noch die dazu korrespondierenden blinden Commitments C und C' und die Chiffrate $B_{i,\,j}$ auf dem $\mathcal{BB}$ veröffentlicht. Alle Chiffrate darunter dürfen nur in dem internen Bereich des $\mathcal{BB}$ sichtbar sein. Die Commitments sind im öffentlichen Bereich sichtbar (siehe Abbildung 5.14). Dadurch entsteht im öffentlichen Bereich eine nach $(i,\,j)$ sortierte Liste mit je n_P Quadrupel aus Commitments für jeden der n_β abgegebenen Stimmzettel:

$$\left\{ \left\{ (c_{i,\,j}^{(C)},\, c_{i,\,j}^{(C')},\, c_{i,\,j}^{(D)},\, c_{i,\,j}^{(D')}) \right\}_{i=1}^{n_P} \right\}_{j=1}^{n_\beta}$$

Im internen Bereich, der eine Obermenge des öffentlichen Bereichs ist, sind die folgenden Elemente sichtbar: (Abbildung 5.8 zeigt eine Übersicht.)

$$\left\{\left\{(B_{i,j},\, C_{i,j},\, C'_{i,j},\, D_{i,j},\, D'_{i,j})\right\}_{i=1}^{n_P}\right\}_{j=1}^{n_\beta}$$

Veröffentlichte Daten auf dem Bulletin Board

Vorbereitungsphase: Daten, die während der Vorbereitungsphase veröffentlicht werden (siehe auch Abbildung 5.16):

- öffentliche Parameter des Commitmentverfahrens: g und h.
- Verifikationsschlüssel der Mitglieder der Wahlleitung: $\{\mathrm{s}_l^{(ver)}\}_{l=1}^{n_A}$
- Commitments der blinden Commitments auf Füllstimmen und Kandidaten: $\left\{\{c_{i,j}^{(D)}, c_{i,j}^{(D')}\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}$
- Die Unveil-Informationen zu $\left\{\{c_{i,j}^{(D)}\}_{i=1}^{n_P}\right\}_{j=1}^{n_E}$ (Commitments auf die Kandidaten).
- Signaturen über alle Commitments von jedem Wahlleitungsmitglied: $\{\sigma_{\text{öffentlich}}^{(a_l)}\}_{l=1}^{n_A}$
- Bestätigung jedes Wahlleitungsmitglieds, dass die Vorberechnung erfolgreich und ehrlich abgeschlossen ist.

Wahlphase: Daten, die während der Wahlphase anfallen, werden nach deren Ende veröffentlicht:

- Kopien der Belege, die die Wähler erhalten haben.
- Commitment $\{c_{1,j}^{(C')}\}_j^{n_\beta}$ auf die frischen Zufallszahlen, die für die abgegebenen Stimmen verwendet wurden.

Auszählungsphase: Die korrekte Auszählung und die korrekte Erstellung der Belege wird bewiesen:

- Unveil-Informationen aller Commitments der Sets, die nach der Wahlphase nicht durch ein Commitment ergänzt wurden.
- Beweise für korrektes Mischen...
  - ... aller Sets sowie korrektes Entschlüsseln des Bits, das angibt, welche der permutierten Zeilen den gewählten Kandidaten enthält.
  - ... des Korrektheitsbeweises der Belege sowie die zugehörigen Unveil-Informationen.
  - ... der Auszählung sowie die zugehörigen Unveil-Informationen.

Abbildung 5.21.: Bereits vor der Wahl werden die ersten Daten veröffentlicht. Anhand dieser kann die Korrektheit der Auszählung gezeigt werden. Dafür müssen weitere, während der Wahl angefallene Daten veröffentlicht werden. Die Abbildung gibt einen Überblick über die für den Beweis relevanten Daten.

Aufteilung der veröffentlichten Commitments Das Auszählungsergebnis kann nun aus den Daten auf dem $\mathcal{BB}$ berechnet und mit Hilfe der veröffentlichten Belege verifiziert werden. Bei der folgenden Beschreibung berücksichtigen wir die ungenutzten Sätze Füllstimmen zunächst nicht; die dazu gehörenden Commitments können (am Ende der Auszählung) geöffnet werden, ohne das Wahlgeheimnis zu gefährden. Jedes Set wird nun durch einen verifizierbaren Mix geführt, wobei die Einträge $C_{i,j}$, $C'_{i,j}$, $D_{i,j}$, $D'_{i,j}$ und $B_{i,j}$

einer Zeile als Tupel betrachtet und daher nicht voneinander getrennt werden. Öffentlich verifizierbar ist ausschließlich die korrekte Rerandomisierung und Permutation der Paare aus Commitments $c_{i,j}^{(D)}$, $c_{i,j}^{(D')}$, $c_{i,j}^{(C)}$ und $c_{i,j}^{(C')}$. Die dazugehörigen Chiffrate sowie das Chiffrat $B_{i,j}$ sind nur für die Mitglieder der Wahlleitung sichtbar; die korrekte Behandlung der Chiffrate wird daher nur ihnen bewiesen. Die Korrektheit der Auszählung hängt jedoch nur von den Commitments ab. Nun ist noch ersichtlich, welche Commitments zu demselben Stimmzettel/Beleg gehören, jedoch nicht, welches Tupel zu welchem Kandidaten gehört. Zum besseren Verständnis des weiteren Ablaufs sei hier noch einmal gesagt, dass jedes Tupel neben dem Chiffrat $B_{i,j}$ sowohl ein blindes Commitmentpaar auf einen Kandidaten p_i und eine Füllstimme $N_{i,j}$ als auch auf einen Kandidaten p_i und die frische Zufallszahl R_j dieser Stimmabgabe enthält. Für die Weiterverarbeitung legt die Wahlleitung eine leere Liste U (*unbenutzt*) an, in der die Commitmentpaare gespeichert werden, deren Füllstimmen auf keinen Beleg gedruckt wurden.

Für jedes Set j führt die Wahlleitung nun die folgenden Aktionen aus:

- Erstelle eine leere Liste Q_j (Q= Quittung/Beleg)
- Für jedes Tupel $(\pi(i), j)$[8] führt die Wahlleitung die folgenden Schritte aus:
  - Entschlüssele das Chiffrat $B_{\pi(i),j} = \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}\left(b_{\pi(i),j}\right)$.
    Das Bit $b_{\pi(i),j}$ besagt, ob die Füllstimme $N_{\pi(i),j}$ dieses Tupels verwendet wurde ($b_{\pi(i),j} = 0$) oder ob statt der Füllstimme die frische Zufallszahl R_j des RNG *auf den Beleg gedruckt wurde ($b_{\pi(i),j} = 1$).*
  - Wenn $b_{\pi(i),j} = 1$ füge $(D_{\pi(i),j},\ D'_{\pi(i),j})$ zu der Liste U hinzu
    füge $(C_{\pi(i),j},\ C'_{\pi(i),j})$ zu der Liste Q_j hinzu
  - Wenn $b_{\pi(i),j} = 0$ füge $(D_{\pi(i),j},\ D'_{\pi(i),j})$ zu der Liste Q_j hinzu
    lösche das Tupel $(C_{\pi(i),j},\ C'_{\pi(i),j})$

Beweis der Korrektheit der Belege Für den Beweis der Korrektheit der Belege wird jede der Listen Q_j durch einen verifizierbaren Mix geführt. Sind die Mitglieder der Wahlleitung von den Korrektheitsbeweisen des Mix überzeugt, entschlüsseln[9] sie gemeinsam die Chiffrate der blinden Commitments aus den Listen Q_j. Dadurch erhalten sie die Unveil-Informationen zu den Commitments. Diese veröffentlichen sie zu den entsprechenden Commitments im für jeden lesbaren Bereich des $\mathcal{BB}$. Der Inhalt der Commitments muss – abgesehen von der Reihenfolge der Zeilen – mit den Inhalten des veröffentlichten Belegs übereinstimmen, von dem der Wähler, dessen Stimme der Beleg repräsentiert, einen

[8] Wir schreiben $\pi(i)$, um zu verdeutlichen, dass die Zeilen permutiert wurden. Die Gesamtpermutation π, die aus dem verifizierbaren Mix stammt, ist niemandem bekannt und wäre nur durch die Beteiligung aller Mitglieder der Wahlleitung zu berechnen.

[9] Durch den Einsatz eines k-aus-n_A-Threshold-Verschlüsselungsverfahren für die Chiffrate der blinden Commitments (siehe Abschnitt 5.3.2) ist für das Öffnen der Commitments die Beteiligung von mindestens k Mitgliedern der Wahlleitung notwendig.

Ausdruck erhalten hat. Der Beweis der Korrektheit der Belege unterscheidet sich nicht vom ursprünglichen Entwurf von Bingo Voting.

Auszählung der ungenutzten Füllstimmen Die Liste U enthält alle ungenutzten blinden Commitments der für die Wahl benötigten Sets. Die Liste wird durch einen verifizierbaren `Shuffle` rerandomisiert und permutiert, bevor die Commitments geöffnet werden. Dadurch wird ersichtlich, wie viele Stimmen die einzelnen Kandidaten erhalten haben und dass die hier verwendeten Füllstimmen auf keinem der Belege Verwendung finden. Wenn alle Kandidaten die gleiche Zahl an vorberechneten Füllstimmen hatten, kann durch diese restlichen, für die Beweise der Korrektheit der Belege nicht verwendeten Füllstimmen, das Auszählungsergebnis berechnet (bzw. abgezählt) werden. Dass für alle Kandidaten gleich viele Füllstimmen erzeugt wurden, ist durch das öffentliche Erstellen der blinden Commitments auf die Kandidatennamen und das Kopieren der Commitments nachvollziehbar.

Die blinden Commitments (bzw. Füllstimmen) der für die Wahl nicht benötigten Sets können nun geöffnet werden. Sie enthalten keine Daten, die das Wahlgeheimnis gefährden. Jedes Mitglied der Wahlleitung – das zur Öffnung der verbleibenden Commitments (implizit durch seine Beteiligung) sein Einverständnis geben muss – kann zu diesem Zeitpunkt nachvollziehen, dass keines der Commitments für die bisherigen Beweise benötigt wurde und die korrekte Erstellung aller Belege bewiesen ist. Es sollte – bevor die Commitments geöffnet werden – eine Frist verstrichen sein, in der Wähler fehlende Belege reklamieren können.

Verifikation

Die Verifikation der Auszählung unterscheidet sich kaum von der des Originalvorschlags von Bingo Voting: Es wird – wie im vorherigen Abschnitt *Auszählung* beschrieben – bewiesen, welche Füllstimmen ungenutzt sind und dass alle anderen auf den veröffentlichten Belegen verwendet wurden. Zusätzlich müssen die Mitglieder der Wahlleitung in unserem Verfahren noch prüfen, wie oft ihr Hardwaretoken, das an den Drucker der Wahlmaschine angeschlossen ist, während der Stimmabgabe verwendet wurde. Ein k-aus-n_A-Threshold-Verschlüsselungsverfahren zugrundelegend, muss ein Hardwaretoken bei n_P Kandidaten und n_β abgegebener Stimmen im Erwartungswert $\lceil \frac{k \cdot n_\beta \cdot n_P}{n_A} \rceil$ oft verwendet werden. Sollten die Hardwaretoken öfter verwendet worden sein, kann ein Versuch, das Wahlgeheimnis zu brechen, nicht ausgeschlossen werden.

Wahlleitung

Ohne einen Vertrauensanker durch Annahmen über die Vertrauenswürdigkeit einzelner Teilnehmer kann die Korrektheit einer Wahl, die nicht durch eine Mehrparteienberechnung realisiert ist, an der jeder Wähler teilnimmt (und weiß, dass er selbst nicht korrumpiert

Auszählung der Stimmen (Zusammenfassung)

Sei $\left\{\{(D_{i,j}, D'_{i,j})\}_{i=1}^{n_P}\right\}_{j=1}^{n_\beta}$ mit $D_{i,j} = (c_{i,j}^{(D)}, u_{i,j}^{(D)}, v_{i,j}^{(D)})$ und $D'_{i,j} = (c_{i,j}^{(D')}, u_{i,j}^{(D')}, v_{i,j}^{(D')})$ die Menge der vor der Wahl erzeugten Commitmentpaare auf die Füllstimmen und den Kandidatennamen p_i.
Weiterhin seien $C_{i,j} = (c_{1,j}^{(C)}, u_{1,j}^{(C)}, v_{1,j}^{(C)}) := D_{i,j}$ Kopien des blinden Commitments auf den Kandidatennamen p_i, und $C'_{1,j} = (c_{1,j}^{(C')}, u_{1,j}^{(C')}, v_{1,j}^{(C')})$ sei das blinde Commitment, das der RNG während der Stimmabgabe des Wählers e_j an das Storage-Device gesendet hat. Für $2 \leq i \leq n_P$ seien $C'_{i,j}$ Kopien von $C'_{1,j}$. Des Weiteren seien $\{B_{i,j}\}_{i=1}^{n_P}$ die rerandomisierten Chiffrate der Einsen oder Nullen, die durch die Append-x-Boxen angehängt und durch die Split-Boxen an das Storage-Device weitergeleitet wurden.
Daraus ergeben sich für jede der n_β abgegebenen Stimmen n_P Quintupel:

$$\{\{(B_{i,j}, C_{i,j}, C'_{i,j}, D_{i,j}, D'_{i,j})\}_{i=1}^{n_P}\}_{j=1}^{n_\beta}$$

Die Aufteilung der veröffentlichten Commitments:

- Erstelle eine leere, globale Liste U für die ungenutzten Füllstimmen.
- Für jede abgegebene Stimme j führt die Wahlleitung die folgenden Schritte aus:
  - Mische verifizierbar korrekt die Menge der Tripel $\{(B_{i,j}, D_{i,j}, D'_{i,j})\}_{i=1}^{n_P}$ der vor der Wahl veröffentlichen Commitments. Dabei wird für die öffentliche Verifikation das korrekte Mischen der Menge $\{(c_{i,j}^{(D)}, c_{i,j}^{(D')})\}_{i=1}^{n_P}$ bewiesen.
  - Entschlüssle alle $B_{i,j}$ zu $b_{i,j}$ und veröffentliche die Klartexte. Die Mitglieder der Wahlleitung beweisen sich gegenseitig die korrekte Dechiffrierung.
  - Erstelle eine leere Liste Q_j; für jedes Tripel aus $\{(b_{i,j}, D_{i,j}, D'_{i,j})\}_{i=1}^{n_P}$:
    → Wenn $b_{i,j} = 1$, dann füge $(D_{\pi(i),j}, D'_{\pi(i),j})$ zu Liste U und $(C_{\pi(i),j}, C'_{\pi(i),j})$ zu Liste Q_j hinzu.
    → Wenn $b_{i,j} = 0$, dann füge $(D_{\pi(i),j}, D'_{\pi(i),j})$ zu Liste Q_j hinzu.

Beweis der Korrektheit der Belege:

- Mische und öffne die Tupel der Listen Q_j.

Auszählung der ungenutzten Füllstimmen:

- Mische und öffne die Tupel der Liste U.

Abbildung 5.22.: Die Auszählung findet über die ungenutzten Füllstimmen statt. Die Korrektheit der Auszählung ist von der Korrektheit der Belege abhängig.

ist), nicht überzeugend geschehen. Der Aspekt, wann eine Bevölkerung einer Regierung und ihren Handlungen ihr Vertrauen ausspricht, liegt außerhalb des Fokus dieser Arbeit. Wir sind jedoch davon überzeugt, dass der beste Ansatz, dieses Vertrauen zu erzeugen, die Verteilung auf eine Gruppe ist. Die Zusammensetzung dieser Gruppe sollte nicht durch das Wahlverfahren vorgegeben oder eingeschränkt sein. Von der Größe und Zusammensetzung hängt schließlich ab, wie stark die Wähler (und alle Beobachter) dem Auszählungsergebnis trauen (können). In unserem Verfahren ist sie frei wählbar und alle Mitglieder haben die gleiche Aufgabe: die Teilnahme an der Vorberechnung in der Vorbereitungsphase, Teilentschlüsselung der Chiffrate für den Drucker und die Erstellung der Auszählung und der zugehörigen Beweise. Das soll zu Füllstimmen führen, die unsere Anforderungen erfüllen, dazu, dass nur Chiffrate entschlüsselt und Commitments geöffnet werden, wenn es das Protokoll vorsieht und alle Beweise korrekt geführt werden. Um ein möglichst großes

Vertrauen zu erzielen, ist es sicherlich empfehlenswert, dass von jeder Interessengruppe und zusätzlich unabhängigen Beobachtergruppen mindestens je ein Mitglied vertreten ist. Die Parameter des Verfahrens können so gewählt werden, dass für die Vertrauenswürdigkeit der gesamten Gruppe ein einzelnes vertrauenswürdiges Mitglied ausreichend ist. Für die Erstellung der Zufälle in den Commitmets und den Chiffraten ist das bereits der Fall. Für die Entschlüsselung hingegen ist dann jedes Mitglied der Gruppe notwendig (k-aus-n_A-Threshold-Verschlüsselungsverfahren mit $k = n_A$). Das hat jedoch zur Folge, dass – sollte sich eines der n_A Mitglieder der Wahlleitung weigern, an der Auszählung mitzuwirken – kein Wahlergebnis berechnet werden kann. Daher ist eine sorgfältige Abwägung und Betrachtung aller möglichen Konstellationen, welche Mitglieder in welcher Gruppe der Wählerschaft Vertrauen genießen, notwendig: $n_A - k$ darf nie kleiner sein als die kleinste Gruppe der Vertrauensträger, damit die gesamte Wahlleitung von allen Wählern als vertrauenswürdig angesehen wird.

Sicherheit

Das Ziel der Weiterentwicklung von Bingo Voting ist es, die Wahlentscheidung des Wählers vor allen Komponenten mit Rechenleistung geheim zu halten. In diesem Abschnitt skizzieren wir die Sicherheitsanalyse.

Wir kommen zu dem Schluss, dass eine passiv korrumpierte Wahlmaschine keine Informationen über die vom Wähler abgegebene Stimme erhält, obwohl der Wähler seine Wahl durch das Drücken eines dem Kandidaten fest zugeordneten Knopfes zum Ausdruck bringt. Die Auswahl wird vor dem Storage-Device durch das pOT verborgen. Wir beweisen, dass das Storage-Device, solange die anderen Komponenten die an sie gestellten Anforderungen erfüllen, außer der Zahl der abgegebenen Stimmen anhand der Daten, die es bekommt, nichts lernen kann. Dazu nutzen wir einen spielbasierten Ununterscheidbarkeitsbeweis. In diesem Spiel kann der Angreifer zunächst eine Stimme für einige Wähler abgeben und alle Datenflüsse innerhalb des Storage-Devices beobachten. Im Anschluss gibt er zwei Kandidaten p_{i_0} und p_{i_1} vor. Nachdem vom Spiel ein Zufallsbit r gezogen wurde, erhält er die Daten, die während der Abgabe einer Stimme für den Kandidaten p_{i_r} anfallen. Wir zeigen mit Hilfe einer Reduktion auf IND-CPA, dass er nicht unterscheiden kann, für welchen der beiden Kandidaten gestimmt wurde. Das ist ausreichend, solange nur eine Minderheit der Mitglieder der Wahlleitung korrumpiert ist, da das Storage-Device in diesem Fall die Chiffrate durch die Verwendung des Entschlüsselungsalgorithmus nicht entschlüsseln kann. Daraus kann abgeleitet werden, dass das Storage-Device keine Information über die Wahl des Wählers lernt.

Definition 8 (View) *Wir definieren* $\mathsf{view}(j, \beta_j)$ *als das Set von Nachrichten, das das Storage-Device beobachten kann, wenn der Wähler* e_j *die Stimme* β_j *abgibt:*

$$\mathit{view}(j, \beta_j) := \left\{ \left(\mathit{ENC}\left(b_{i,j} = \left\{ \begin{array}{lcl} 1 & \Leftrightarrow & i = \beta_j \\ 0 & \Leftrightarrow & i \neq \beta_j \end{array} \right. \right), \mathit{ENC}(N_{i,j}), \mathit{bCOM}(R_j, r) \right) \right\}_{i=1}^{n_P}$$

Sicherheitsspiel 1 ($\mathsf{IND\text{-}CV}^{\mathcal{A}}_{(\mathsf{enc},\mathsf{com})}(k)$)

1. Das Experiment führt die Setup-Phase der Threshold-Verschlüsselung durch, erhält (pk, sk) und zieht ein zufälliges bit $r \leftarrow \{0,1\}$
2. Der Angreifer gibt die Zahl der Kandidaten n_P und die der Wähler n_E vor. Das Experiment führt die Vorberechnungen für ein Wahlverfahren entsprechend diesen Parametern durch. Es übergibt die berechneten Werte an den Angreifer.
3. Der Angreifer wählt einen Wähler $1 \leq j \leq n_E$, der noch nicht gewählt hat, und eine Stimme β_j für den Wähler j. Er übergibt beides dem Experiment. Das Experiment führt das Protokoll für den Wähler j mit dem durch β_j festgelegten Kandidaten aus und gibt die View $\texttt{view}(j, \beta_j)$ an den Angreifer aus.
4. Der Angreifer kann diesen Schritt so oft wiederholen, wie es seine Laufzeitbeschränkung zulässt. Im Anschluss sendet er „`end`“ an das Experiment.
5. Der Angreifer wählt nun erneut einen Wähler $1 \leq j_c \leq n_E$, der seine Stimme noch nicht abgegeben hat, und *zwei* Kandidaten p_{i_0} und p_{i_1} ($p_{i_0} \neq p_{i_1}$). Er übergibt seine Wahl (j_c, p_{i_0}, p_{i_1}) an das Experiment. Das Experiment antwortet dem Angreifer mit $\texttt{view}(j_c, p_{i_r})$.
6. Der Angreifer erhält zusätzlich alle öffentlichen Daten und gibt r' als Vermutung für die Belegung von r aus.

Definition 9 (Ununterscheidbarkeit bei gewählter Wählerentscheidung) *Eine Wahl ist* ununterscheidbar bei gewählter Wählerentscheidung – IND-CV*, CV steht für* chosen votes *–, wenn*

$$\forall \mathcal{A}, c \in \mathbb{N} \exists k_0 \forall k > k_0 : |\Pr[\mathsf{IND\text{-}CV}^{\mathcal{A}}_{(\mathsf{enc},\mathsf{com})} = 1]| \leq \frac{1}{2} + k^{-c}$$

Für einen besseren Überblick skizzieren wir hier die Standarddefinition für IND-CPA-Sicherheit [77].

Sicherheitsspiel 2 ($\mathsf{IND\text{-}CPA}^{\mathcal{A}}_{\mathsf{enc}}(k)$)

1. das Experiment führt das Setup für `enc` aus und erhält das Schlüsselpaar (pk, sk).
2. Das Experiment zieht $r \leftarrow \{0,1\}$ zufällig.
3. Der Angreifer erhält als Eingabe 1^k und pk und führt beliebige Berechnungen aus.
4. Der Angreifer gibt zwei Nachrichten m_0 und m_1 mit $|m_0| = |m_1|$ aus.
5. Das Experiment gibt $\texttt{ENC}(m_r)$ an den Angreifer zurück.
6. Der Angreifer gibt seine Vermutung r' über die Belegung von r aus.

Theorem 1 *Oblivious Bingo Voting hat die Eigenschaft ‚Ununterscheidbarkeit bei gewählter Wählerentscheidung‘, wenn das Verschlüsselungsverfahren* `enc` IND-CPA*-sicher und das Commitmentverfahren* `com` *perfekt verbergend ist.*

Beweis 1 *Wir beweisen die Aussage in zwei Schritten. Zunächst modifizieren wir das Spiel, indem wir die Commitments nicht berücksichtigen: Da die Commitments per Annahme perfekt verbergend sind, sind sie informationstheoretisch ununterscheidbar von Zufall. Der Angreifer kann daraus keine Information gewinnen. Der verbleibende Teil der View sieht wie folgt aus:*

$$\texttt{view'}(j, \beta_j) := \{(\texttt{ENC}(b_{i,j}), \texttt{ENC}(N_{i,j}), \texttt{ENC}(R_j), \texttt{ENC}(r))\}_{i=1}^{n_P}$$

Als nächstes verwenden wir – um den Widerspruch zu zeigen – einen erfolgreichen Angreifer auf IND-CV *für die Konstruktion eines erfolgreichen Angreifers auf* IND-CPA*. Die Reduktion funktioniert wie folgt:*

- *Verwende – nach der Wahl für n_P und n_E durch den Angreifer – den öffentlichen Schlüssel pk des* IND-CPA*-Experiments, um die blinden Commitments zu erstellen, und übergib sie dem Angreifer.*
- *Generiere und sende nach Erhalt eines Wählernamens $1 \leq j \leq n_E$ und einer Stimme β_j $\texttt{view'}(j, \beta_j)$ mit Hilfe der Funktion* `enc` *und dem öffentlichen Schlüssel pk.*
- *Generiere nach Erhalt der Nachricht „*`end`*“, j_c und zweier Kandidaten p_{j_0} und p_{j_1} zwei Sätze von Nachrichten m_0 und m_1, die die Klartexte von $\texttt{view'}(j_c, \beta_{j_0})$ und $\texttt{view'}(j_c, \beta_{j_1})$ enthalten: $\{(b_{i,j}, p_i, r, R_j, r')\}_{i=1}^{n_P}$.*
- *Sende m_0 und m_1 an das* IND-CPA*-Experiment.*
- *Leite die Nachricht m_b vom* IND-CPA*-Experiment an den Angreifer innerhalb des* IND-CV*-Experiments weiter und verwende seine Ausgabe r' als Antwort an das* IND-CPA*-Experiment.*

Die Reduktion simuliert das IND-CV*-Experiment perfekt für den Angreifer. Daher können wir den Vorteil des Angreifers auf das* IND-CPA*-Experiment übertragen. Da wir dazu im Widerspruch stehend annehmen, dass das Verschlüsselungsschema* IND-CPA*-sicher ist, zeigt das die Aussage.*

In dem Beweis verwenden wir statt des üblichen IND-CPA-Spiels eine modifizierte Variante; dem Experiment werden von dem Simulator statt zweier Nachrichten zwei Vektoren von Nachrichten übergeben. Eine davon verschlüsselt das Experiment elementweise mit dem öffentlichen Schlüssel *pk*. Katz und Lindell zeigen in ihrem Buch *Introduction to modern cryptography: principles and protocols* [77] in dem Beweis zu Theorem 10.10, dass ein Verfahren, das IND-CPA-sicher ist, auch den Sicherheitsbegriff erfüllt, in dem statt einzelner Nachrichten Vektoren von Nachrichten verwendet werden.

5.3.5. Realisierung

Um die Machbarkeit der Idee zu überprüfen, haben wir einen Prototypen entwickelt. Die Umsetzung erfolgte in mehreren Schritten. Zunächst galt es, die blinden Commitmemts zu implementieren und deren korrekte Funktionsweise zu testen; sowohl die homomorphen

Eigenschaften als auch die Rerandomisierbarkeit sind für Oblivious Bingo Voting notwendig. Auf dieser Basis können die Algorithmen zur Vorberechnung (siehe Algorithmen 4 und 5) und zur Auszählung umgesetzt werden.

Für die Umsetzung des Physical Oblivious Transfer (pOT) haben wir eine etwas andere Lösung als die hier beschriebene verwendet. Statt des modifizierten Optokopplers haben wir ein Relais verwendet, dessen Aus- und Eingänge mit Hilfe von Optokopplern galvanisch von den Sendern und dem Empfänger getrennt sind. So war es uns möglich, den pOT mit handelsüblichen Bausteinen prototypisch umzusetzen. Für einen realen Einsatz sind andere Mittel zu wählen, da die Relais durch ihre elektromagnetische Abstrahlung voraussichtlich Seitenkanalangriffe ermöglichen.

Durch die Verwendung von Relais war es uns auch möglich, ein Zurücksetzen des ausgewählten Kandidaten sowohl bei Abgabe der Stimme als auch bei Änderung der Auswahl zu realisieren. Dadurch ist es nicht möglich, zwei Kandidaten die frische Zufallszahl zuzuordnen.

5.3.6. Zusammenfassung und Ausblick

Die hier vorgestellte Umsetzung eines kryptographischen Wahlverfahren ist nach unserem Wissen die erste Wahlmaschine, die die Wählerentscheidung vor den Bauteilen mit Rechen- und Speicherkapazitäten verbirgt, obwohl sie der Wähler durch das Drücken eines *seinem* Kandidaten zugeordneten Knopfes zum Ausdruck bringt. Einen passiven Angreifer vorausgesetzt, ist kein Teil der Wahlmaschine alleine in der Lage, die Erpressungsfreiheit zu umgehen. Um das zu erreichen, verschieben wir das Vertrauen, das ursprünglich in die komplexe Wahlmaschine gesetzt werden musste, auf kleine Komponenten, die einen sehr reduzierten Funktionsumfang haben; im Fall des pOT verfügt der Baustein nicht einmal über eine Schaltlogik, die über die eines Lichtschalters hinausgeht. Auch der andere neue Baustein – die Rerandomisierungsbox –, in die wir Vertrauen setzen müssen, führt lediglich wenige, einfache mathematische Berechnungen durch, die in einer überschaubaren Menge Codezeilen darstellbar sind und sich evtl. auch durch einige fest verdrahtete logische Bausteine umsetzen lassen. Wir glauben, dass sie sich aus handelsüblichen Komponenten zusammensetzen lässt und eine Verifikation möglich ist. Der in Bingo Voting bereits verwendete vertrauenswürdige Zufallszahlengenerator bleibt auch bestehen.

Christian Henrich beschreibt in seiner Dissertation [69] verschiedene Optimierungen auf das ursprüngliche Verfahren. Die Modifikationen lassen sich zu großen Teilen unverändert auf oblivious Bingo Voting anwenden. Insbesondere die Anwendung von Hashketten, um das Entfernen einzelner Stimmzettel zu vermeiden, ist empfehlenswert. Die Möglichkeit, im Vergleich zum Originalentwurf kürzere Zufallszahlen zu verwenden, ergibt sich durch die geänderte Konstruktion automatisch: Bereits vor der Wahl steht fest, welche Füllstimmen zueinander gehören. Kollisionen über verschiedene Stimmzettel hinweg stellen kein Problem dar. Da eine Kollision zwischen zwei Füllstimmen ununterscheidbar von der Kollision zwischen einer Füllstimme und der frischen Zufallszahl ist, gefährdet sie das Wahlgeheimnis

nicht. Einem Angreifer wäre es im zweiten Fall jedoch möglich, die Stimme von einem zum anderen Kandidaten zu transferieren. Er kann dabei aber dennoch nicht wissen, welchem Kandidaten die Stimme ursprünglich gegeben wurde. Wurde sie keinem der beiden Kandidaten gegeben, bei denen die Kollision aufgetreten ist, ändert die Manipulation des Angreifers die Stimme nicht.

Die Annahme von Bingo Voting, dass ein vertrauenswürdiger Zufallszahlengenerator Verwendung findet, dessen korrekte Funktionsweise durch den Wähler prüfbar ist und durch den das Verfahren seinen Namen erhalten hat, ließe sich möglicherweise ebenfalls auf eine insgesamt vertrauenswürdige Gruppe aufteilen. Eine interessante Fragestellung darüber hinaus wäre, unter welchen Annahmen und mit welchen zusätzlichen (vertrauenswürdigen) Komponenten sich das Verfahren in ein Onlinewahlverfahren erweitern ließe. Selbstverständlich kommen bei Onlinewahlen – alleine durch den Wegfall der Wahlkabine – weitere Angriffsvektoren hinzu. Eine davon – durch Erpressung seine Stimme unter Beobachtung abzugeben – diskutieren wir im nächsten Kapitel 5.4.

Hardwaretoken, wie wir sie zum Entschlüsseln der Chiffrate für den Drucker vorsehen, sind bereits kommerziell erhältlich. In Kapitel 4, in dem ein Kopierschutz – basierend auf Hardwaretoken der Firma Wibu-Systems – vorgestellt wird, kommen solche Hardwaretoken in Form von USB-Dongles zum Einsatz.

Ein aktiver Angreifer kann statt des Chiffrats der Füllstimme des Kandidaten, für den der erpresste Wähler wählen soll, einen zufälligen Wert an das Input-Device senden. Diese Änderung hat keine Folgen, wenn der Wähler tut, was ihm vom Angreifer vorgegeben wurde; wählt er anders, schlägt der Korrektheitsbeweis in der Auszählungsphase fehl. Dieser Angriff ließe sich beispielsweise durch eine Kontrolle der Ausgabe des Storage-Device unterbinden.

5.4. Coercion Resistant Revoting

Bei Präsenzwahlverfahren kann durch organisatorische Maßnahmen garantiert werden, dass ein Wähler seinen Stimmzettel unbeobachtet kennzeichnen kann. Eine solche Infrastruktur ist in Deutschland in § 50, Abschnitt 1 der Bundeswahlordnung [75] vorgeschrieben:

> „In jedem Wahlraum richtet die Gemeindebehörde eine Wahlzelle oder mehrere Wahlzellen mit Tischen ein, in denen der Wähler seinen Stimmzettel unbeobachtet kennzeichnen und falten kann. Die Wahlzellen müssen vom Tisch des Wahlvorstandes aus überblickt werden können. Als Wahlzelle kann auch ein nur durch den Wahlraum zugänglicher Nebenraum dienen, wenn dessen Eingang vom Tisch des Wahlvorstandes aus überblickt werden kann."

Füllt ein Wähler seinen Stimmzettel zuhause aus, kann ihm der Schutz, dass er in dem Moment seiner Stimmabgabe nicht direkt beeinflusst wird und er keinen Nachweis[10] anfertigen muss, wie er gewählt hat, nicht mehr garantiert werden. Das gilt nicht nur für die hier im weiteren behandelten kryptographischen Onlinewahlverfahren, sondern auch für die Briefwahl und Wahlen, bei denen der Wähler die Wahlunterlagen aufgrund ihrer Komplexität zuvor nach Hause geschickt bekommt.

Während die Briefwahl eine Ausnahme im normalen Wahlprozess darstellt, wird die Stimmabgabe außerhalb eines leicht kontrollierbaren Rahmens bei Onlinewahlverfahren zur Regel. Maßnahmen, die es einem Wähler ermöglichen, einer dadurch begünstigten Erpressung auszuweichen, verstehen sich daher als obligatorisch. Zwei unterschiedliche Verfahren haben sich in der Literatur etabliert: Die Möglichkeit, eine Stimme mit ungültigen Authentifizierungsdaten abzugeben (*Fake Keys*) oder die Stimme durch eine erneute Stimmabgabe zu überschreiben (*revoting*).

Obwohl beide Verfahren eigene Vor- und Nachteile haben und ihnen nicht dasselbe Angreifermodell zugrunde liegt, schließt sich die Anwendung beider Prinzipien nicht gegenseitig aus. In keinem der Fälle darf es einem Angreifer möglich sein, herauszufinden, ob ein Wähler wie vom Erpresser verlangt handelt oder eine Ausweichstrategie wählt. Die angewendeten Verfahren, Algorithmen und Implementierungen müssen daher – über die üblichen Anforderungen hinausgehend – weitere Anforderungen erfüllen. Im folgenden Abschnitt werden die Anforderungen an ein Onlinewahlverfahren besprochen, das mit Hilfe der Möglichkeit, eine bereits abgegebene Stimme zu überschreiben, die Erpressungsfreiheit erreichen möchte, und Angriffe skizziert, die sonst möglich wären. Im Anschluss daran stellen wir ein Verfahren vor, das diese Anforderungen erfüllt.

[10] Streng genommen gilt die Annahme, dass der Wähler keinen Beleg dafür anfertigen kann, wie er gewählt hat, seit der Miniaturisierung von Kameras nicht mehr. Inzwischen führt vermutlich die Mehrheit der Wähler eine Video- und Fotokamera in Form eines Mobiltelefons/Smartphones während der Stimmabgabe mit sich. Auch eine in der Wahlkabine versteckt angebrachte Kamera ist kaum auszuschließen.

5.4.1. Anforderungen und Ziele

In diesem Abschnitt erläutern wir Anforderungen an ein Verfahren, das das Überschreiben von Stimmen als Ausweichstrategie für Erpressungen nutzt, und die daraus resultierenden Herausforderungen. Um als Ausweichstrategie tauglich zu sein, muss der Wähler abstreiten können, seine Stimme durch eine erneute, ggf. andere Stimmabgabe überschrieben zu haben. Das bedeutet implizit auch, dass er nicht beweisen kann, keine weitere Stimme abgegeben zu haben. Dabei muss die Auszählung weiterhin korrekt und verifizierbar sein: Die zählende Stimme jedes Wählers, der mindestens eine Stimme abgegeben hat, ist genau die letzte seiner vor dem Ende der Wahlphase abgegebenen gültigen Stimmen. Eine Stimme ist im oben genannten Sinne genau dann gültig, wenn sie wohlgeformt[11] ist und über valide Authentifizierungsinformationen verfügt. Das heißt, dass bei einem solchen Verfahren bis zum Wahlschluss nicht feststeht, ob eine Stimme zählen wird. In diesem Punkt unterscheiden sich Revoting-Verfahren grundsätzlich von herkömmlichen Verfahren, bei denen in dem Moment, in dem der Wähler seinen Stimmzettel abgegeben hat, seine Wahl unveränderlich feststeht. Bei Revoting-Verfahren geschieht das automatisch – ohne Zutun des Wählers – mit Ablauf der Wahlphase. Das hat zur Folge, dass ein Erpresser bis zum Ende der Wahlphase Druck auf einen Wähler ausüben kann. Somit stellt die Stärke des Verfahrens, die es einem Wähler im Idealfall ermöglicht, trotz einer Erpressung seine gewünschte Stimme abzugeben, gleichzeitig auch eine Schwäche dar, da ein Angreifer einen Wähler noch erpressen kann, obwohl dieser bereits eine Stimme abgegeben hat.

Dem Wähler als Ausweichstrategie gegen Angriffe, die durch das Fehlen einer Wahlkabine bei Onlinewahlen unvermeidbar sind, die Möglichkeit zu bieten, eine zuvor abgegebene Stimme zu überschreiben, kann nur dann funktionieren, wenn sowohl die Stimmabgabe als auch die Auszählung alle Metadaten verbergen, die auf das Wählerverhalten schließen lassen. Als zusätzliche Anforderung muss für jedes Wahlverfahren gelten, so einfach wie möglich zu sein, damit es jeder Wähler nutzen kann [6]. Aus unserer Sicht schließt das jedes Verfahren aus, in dem der Wähler zur Stimmabgabe beispielsweise rechnen oder sich – abgesehen von dem Wunsch, seine Stimme abzugeben – etwas für das Überschreiben einer zuvor abgegebenen Stimme merken muss [82].

Wir stellen in diesem Kapitel ein Auszählungsverfahren vor, das die oben genannten Anforderungen erfüllt: eine Stimmabgabe ist stets unabhängig von den zuvor ausgeführten Aktionen des Wählers, und Angreifer können anhand der Auszählung nicht auf das Wählerverhalten schließen. Der Prozess der Stimmabgabe selbst wird dabei nicht betrachtet; die Art der Authentifizierung ist jedoch durch den Prozess und die an ihn gestellten Anforderungen notwendigerweise Teil des Lösungsvorschlags.

[11] Eine Garantie, dass die verschlüsselten Stimmen wohlgeformt sind, ist Teil des Abgabeprozesses und von der Art der Stimme abhängig. Daher betrachten wir diesen Aspekt in dieser Arbeit nicht.

Erpressungsstrategien durch Metadaten

Metadaten, die bei der Verarbeitung der Daten entstehen, können Angriffe ermöglichen. Wir betrachten im Folgenden einige Klassen von Metadaten und die daraus resultierenden Angriffsmöglichkeiten.

Geht eine Stimme in das Wahlergebnis ein: Obwohl der Wähler überzeugt davon sein muss, dass seine (zuletzt abgegebene) Stimme Teil des Wahlergebnisses ist, darf ein Angreifer nicht in Erfahrung bringen können, ob eine Stimme, die von ihm erpresst wurde, in das Wahlergebnis eingeht oder ob die Stimme aussortiert wird. Sähe er, ob „seine Stimme“ gelöscht oder als ungültig markiert wird, wüsste er auch, dass sein Angriff nicht erfolgreich war und der Wähler eine weitere Stimme abgegeben hat. Damit wäre dem Wähler offensichtlich die Möglichkeit genommen, das Überschreiben der Stimme abzustreiten.

Abgabezeitpunkt der gültigen Stimme: Wird einem Teilnehmer im Prozess der Stimmabgabe und -auszählung bekannt, zu welchem Zeitpunkt eine Stimme abgegeben wurde, die als gültige, nicht überschriebene Stimme in das Wahlergebnis eingeht, so kann dieser leicht herausfinden, ob die Stimme, die ein Wähler in seiner Anwesenheit abgegeben hat, überschrieben wurde. Dazu muss er sich lediglich den Zeitstempel merken, den der Stimmzettel erhalten hat, den er vom Wähler erpresst hat. Taucht dieser nicht in der Auszählung auf, so kann er daraus ableiten, dass die Stimme überschrieben wurde und sich der Wähler der Erpressung nicht gebeugt hat.

Zahl abgegebener Stimmen eines bestimmten Wählers: Kann ein Angreifer herausfinden, wie oft ein Wähler eine Stimme abgegeben hat, ist es ihm möglich, diesen zu erpressen, die in seinem Beisein abgegebene Stimme nicht mehr zu überschreiben. Würde er es dennoch tun, verriete die Differenz zwischen der erwarteten und tatsächlichen Zahl der Stimmzettel, wie sich der Wähler verhalten hat. Diese Metadaten fallen insbesondere bei einer Authentifikation durch *Benutzername und Passwort* oder durch vom Wähler angefertigte Signaturen an. Im ersten Fall ist für den Authentifikationsserver offensichtlich, welcher Nutzer eine Stimme abgeben möchte. Der zweite Fall wird gerne verwendet, um die Legitimation einer Stimme nachweisen zu können und Angriffe zu unterbinden, bei denen die Gleichheit der Wahl ausgehebelt werden soll, indem zusätzliche Stimmzettel hinzugefügt werden. Dazu ist ein öffentliches Verzeichnis der legitimen öffentlichen Schlüssel der Wähler notwendig. Um herauszufinden, wie oft der Nutzer mit dem öffentlichen Schlüssel pk seine Stimme überschrieben hat, muss ein Angreifer nur alle Signaturen überprüfen, ob sie mit pk erfolgreich verifizierbar sind.

Anzahl der abgegebenen Stimmen je Wähler bei pseudonymisierten Wählernamen: Wenn ein Stimmzettel nicht einem Wähler direkt, jedoch allen anderen seiner Stimmzettel zugeordnet werden kann, ist es einem Angreifer möglich, den Wähler so oft Stimmen überschreiben zu lassen, dass die Länge dieser Kette eindeutig und aussagekräftig ist. Der Angreifer erwartet dann für jeden Wähler, den er erpresst hat,

eine Kette der für ihn eindeutigen Länge. Diesen Angriff hat bereits Warren Smith als *1009-Angriff* beschrieben [111].

Quotienten der Wähleridentitäten: Um festzustellen, ob ein neuerer Stimmzettel einen älteren überschreibt, müssen die damit verknüpften (verschlüsselten) Identitäten miteinander verglichen werden. Da das Entschlüsseln der Identitäten den zweiten Angriff ermöglichen würde, müssen die Identitäten verdeckt verglichen werden. Doch auch die Quotienten der Identitäten der beiden Stimmzettel, die im Falle des gleichen Wählers Eins (respektive Null, bei einer Subtraktion) wäre, darf von niemandem in Erfahrung gebracht werden. Die Quotienten, die zwischen zwei unterschiedlichen Identitäten zwar ungleich Eins (bzw. Null), aber immer identisch sind, ermöglichen die Durchführung des dritten Angriffs.

Alle überschriebenen Stimmen durch einen Wähler oder je eine pro Wähler Das hier skizzierte Extrem, dass alle überschriebenen Stimmen von einem einzelnen Wähler stammen, ist in der Praxis sicherlich nicht wahrscheinlich. Dennoch sollte die Auszählung verbergen, welche Verteilung den neu abgegebenen Stimmen zugrunde liegt.

Herkömmliche Lösungsansätze

Einige Lösungen scheinen naheliegend, sind jedoch nicht zielführend, da die Anwendung für den Wähler zu aufwendig ist oder Metadaten anfallen, die den Wähler verraten.

Annullieren der letzten Entscheidung: Gibt der Wähler mit seiner Stimme eine inverse Stimme der zu überschreibenden ab, muss sich der Wähler dafür die letzte Entscheidung merken. Das widerspricht unseren Anforderungen.

Zuordnung zur überschriebenen Stimme: Muss der Wähler angeben, welche Stimme er überschreibt, muss sich der Wähler merken, welches seine letzte Stimme war [82]. Das widerspricht unseren Anforderungen.

Rundenbasiertes Aussortieren: Der Ansatz, in mehreren Runden Paare abgegebener Stimmen mit gleicher Wähler-ID zu identifizieren und die jeweils ältere Stimme zu entfernen, bis keine Paare mehr gefunden werden können, lässt Rückschlüsse auf die Menge der Stimmen zu, die ein einzelner Wähler abgegeben hat. Dadurch ist ein Angriff, wie er oben beschrieben wurde, möglich.

5.4.2. Verwandte Arbeiten

Die Wichtigkeit, Vorteile und Herausforderungen des Konzeptes Revoting werden von Volkamer und Grimm [118] und Post [117] diskutiert. Dennoch wird keine Lösung für die Herausforderungen gegeben. Viele Onlinewahlverfahren bieten dem Wähler die Möglichkeit, seine Stimme zu überschreiben. Als Möglichkeit für den Wähler, sich gegen Erpressung zu wehren, sehen es nicht alle. Keines der uns bekannten Wahlverfahren verwendet eine

Revoting-Strategie, die keinen Rückschluss auf das Wahlverhalten eines Wählers zulässt, ohne dass er sich Informationen aus einer Stimmabgabe für eine erneute Stimmabgabe merken muss und die gleichzeitig öffentlich nachvollziehbar ist. Kutyłowski und Zagórski stellen ein Schema vor [82], bei dem der Wähler Informationen aus seinem letzten Wahlgang für eine erneute Stimmabgabe benötigt. Spycher et al. schlagen ein hybrides System vor [112], in dem der Wähler in einer Präsenzwahl seine online abgegebene Stimme überschreiben kann, wenn er eine rerandomisierte Version seines online abgegebenen Stimmzettels vorlegen kann.

Für die Wahl des Präsidenten der *Université Catholique de Louvain* wurde eine modifizierte Version von Helios [8] verwendet. Revoting wurde wegen des als gering eingestuften Erpressungsrisikos hauptsächlich aus Bequemlichkeitsgründen angeboten. Überschriebene Stimmzettel wurden auf dem Bulletin Board durch neuere ersetzt. Dadurch kann ein Wähler nicht abstreiten, erneut eine Stimme abgegeben zu haben. Die Umsetzung von Revoting im Wahlverfahren, das für eine Wahl 2011 in Norwegen entworfen wurde [58], bietet ebenfalls keine Abstreitbarkeit. Die Auditoren sehen, wie viele Stimmen die einzelnen Wähler abgegeben haben. Der *Riigikogu Election Act* fordert für estnische Wahlen [102] die Möglichkeit, seine Stimme überschreiben zu können, jedoch – soweit uns bekannt ist – nicht, dass ein Wähler dies abstreiten können muss [84].

Juels, Catalano und Jakobsson schlagen ein Verfahren vor [74], das Revoting ermöglicht, als Ausweichstrategie jedoch Fake Keys nutzt. Daher ist die Umsetzung der Aussortierung der überschriebenen Stimmzettel nicht so ausgelegt, dass der Wähler sein tatsächliches Verhalten abstreiten kann. Civitas [89] baut direkt auf der Arbeit von Juels, Catalano und Jakobsson auf; Selections [40, 39] und Cobra [53] folgen auch der Idee, ungültige Passwörter zu verwenden, die für den Menschen gegebenenfalls leicht zu erzeugen sein müssen. Das Aussortieren geschieht dabei zwar nachvollziehbar, dabei wird aber das Verhalten der Wähler nicht verborgen. Effizientere Implementierungen der Ideen von Juels, Catalano und Jakobsson [73] wurden von Araújo et al. [11] und Smith [111] vorgeschlagen. Das Prinzip des Revotings wurde dabei jedoch nicht optimiert. Das Problem bei der Verwendung von Fake Keys liegt darin, dass der Wähler selbst keine Rückmeldung erhält, ob seine Stimme angenommen wurde. Zusätzlich müssen die Fake Keys auch vor den Augen des Erpressers spontan erzeugt werden können, ohne dass er dies feststellen kann.

Die in diesem Kapitel vorgestellten Arbeiten basieren auf Überlegungen von Jeroen van de Graaf. Der initialen Idee folgten lange ausführliche Diskussionen zwischen Herrn Müller-Quade, Frau Kempka, Herrn Achenbach und mir, welche Eigenschaften ein Verfahren erfüllen muss, welche Konzepte prinzipiell nicht zielführend sein können und welche Anforderungen an einen Wähler gestellt werden können. In diesen Diskussionen hat sich auch herausgestellt, dass die ursprüngliche Idee zu viele Informationen über das Wählerverhalten preisgibt.

Der Algorithmus, der diese Probleme löst und den wir vorstellen, basiert auf meinem Konzept. Das Sicherheitsmodell ist das Ergebnis vieler Diskussionen zwischen Herrn Achenbach und mir. Der Sicherheitsbeweis wurde hauptsächlich von meinem Kollegen

Dirk Achenbach erstellt. Die Ergebnisse wurden gemeinsam von Dirk Achenbach, Carmen Kempka, mir und Jörn Müller-Quade in dem Papier *Improved Coercion-Resistant Electronic Elections through Deniable Re-Voting* [1] veröffentlicht.

5.4.3. Modelle

Vor der Vorstellung eines Auszählungsprotokolls, das die oben beschriebenen Angriffe verhindert, stellen wir noch das zugehörige Angreifermodell vor. Im Unterschied zu der von Juels, Catalano und Jakobsson [73] vorgeschlagenen Verwendung ungültiger Schlüssel (*Fake Keys*) im Falle einer Erpressung, kann der Wähler bei Revoting zeitweise der vollständigen Kontrolle des Erpresser unterstehen, solange dieser – wie in den Anforderungen an die Authentifizierung in Abschnitt 5.2.3 schon formuliert – die Authentifizierungsdaten weder kopieren noch entwenden kann. Er erhält demzufolge nur die Möglichkeit, sie zu nutzen. Im Falle einer Kopplung an eine Hardware wie einem Personalausweis (z. B. dem nPA) könnte das eine plausible Annahme sein. Ein weiterer Unterschied besteht darin, dass ein Wähler im Falle der Verwendung von Fake Keys zu einem beliebigen Zeitpunkt während der Wahlphase – vor oder nach der Erpressung – unbeobachtet sein muss, während der Wähler im Falle von Revoting nach der Erpressung noch einmal die Gelegenheit haben muss, seine Stimme unbeobachtet abzugeben.

Abstreitbarkeit erneut abgegebener Stimmen

Auch wenn alle oben formulierten Anforderungen erfüllt sind, gibt es noch Angriffe, die systembedingt nicht auszuschließen sind. So weiß ein Angreifer bei einer Wahl, bei dem alle Wähler für den gleichen Kandidaten gestimmt haben, zweifelsfrei, für welchen Kandidaten der Wähler, den er erpresst hat, gestimmt hat. Solche Angriffe können auch in einer idealisierten Wahl nicht ausgeschlossen werden, solange das Auszählungsergebnis – wie generell üblich – öffentlich ist und aus den akkumulierten Stimmen besteht. Hierzu wäre eine Verfremdung der Auszählung notwendig, ohne dass sich das daraus abgeleitete Wahlergebnis[12] verändert. Solche Techniken sollen aber – insbesondere weil es eine Interpretationsvorschrift der Auszählung zugrunde legt – nicht Teil dieser Arbeit sein.

Der Erfolg eines Angreifers muss dementsprechend in Relation zum Erfolg eines Angreifers eines *idealen Protokolls* definiert werden, das per Definition keine Angriffsstelle bietet. Das ideale Protokoll erhält die abgegebenen (verschlüsselten) Stimmen und gibt lediglich das Auszählungsergebnis aus. Nur die Angriffsflächen, die ein Angreifer im realen, nicht aber im idealen Spiel hat, können minimiert werden und sind daher für uns von Bedeutung. Zur Bestimmung des Vorteils des Angreifers wird die Erfolgswahrscheinlichkeit eines Angreifers im realen Experiment gegenüber der Erfolgswahrscheinlichkeit im idealen Experiment betrachtet. Das ideale Experiment enthält alle nicht vermeidbaren

[12] z. B. eine Sitzverteilung im Parlament

Informationsflüsse. Der Vorteil des Angreifers muss sich dementsprechend aus dem zusätzlichen Informationsgewinn einer realen Protokollausführung ableiten lassen. Relevant ist nicht die Unterscheidung zwischen realer und idealer Welt, wie es im UC-Modell der Fall ist, sondern die Differenz der Erfolgswahrscheinlichkeiten in den unterschiedlichen Experimenten.

Sicherheitsmodell für die Fake-Key-Ausweichstrategie

Juels, Catalano und Jakobsson formulieren dafür zwei Experimente. Während das eine Experiment das reale Protokoll abbildet (siehe Abbildung 5.23), steht das andere für die ideale Ausführung der Funktionalität (siehe Abbildung 5.24), die das Protokoll realisieren soll. In der idealen Ausführung wird eine Funktion $\texttt{ideal} - \texttt{tally}$ verwendet, die per Definition keine verwertbaren Anhaltspunkte außer der Ausgabe – der Auszählung – selbst enthält. Beide Experimente sind so formuliert, dass sie als Ausweichstrategie die Verwendung ungültigen Schlüsselmaterials vorsehen. Wird die Möglichkeit, die unter Erpressung abgegebene Stimme zu überschreiben, als Ausweichstrategie gewählt, müssen die Experimente anders formuliert werden. Unsere modifizierten Experimente werden in Abbildung 5.25 für das reale Protokoll und in Abbildung 5.26 für das ideale Protokoll dargestellt.

Die Parameter k_i sind die Sicherheitsparameter einzelner Bausteine. Die Zahl der unkorrumpierten Wähler ist durch n_U gegeben, die Zahl der Wahlmöglichkeiten durch n_P. Die Verteilung D_{n_U,n_P} gibt für jeden Wähler eine Wahlentscheidung aus. Welche Verteilung anzuwenden ist, legt der Angreifer fest.

Experiment für reales Protokoll mit Verwendung ungültigen Schlüsselmaterials Um die Unterschiede zwischen den beiden Ansätzen (ungültiges Schlüsselmaterial und die Möglichkeit, seine Stimme abstreitbar zu überschreiben) auch in den Sicherheitsmodellen besser darstellen zu können, gehen wir zunächst auf die von Juels, Catalano und Jakobsson [73] definierten Experimente ein. Die Änderungen, die für die Abstreitbarkeit der erneuten Stimmabgabe notwendig sind, werden im Anschluss diskutiert.

Im Experiment eines realen Protokollablaufs – dargestellt in Abbildung 5.23 –, in dem ungültiges Schlüsselmaterial als Ausweichstrategie verwendet wird, wählt der Angreifer zunächst $n_{\mathcal{A}}$ der n_E Protokollteilnehmer, die er vollständig kontrolliert (Schritt 1). Von diesen Wählern kennt er insbesondere das Schlüsselmaterial und die Authentifizierungsdaten aus Schritt 2. Nach der Auswahl des Angreifers, welchen Protokollteilnehmer j_C er – basierend auf dem Schlüsselmaterial der von ihm kontrollierten Angreifer – versuchen möchte zu erpressen (Schritt 3), wird die Auswahl, die der Angreifer getroffen hat, kontrolliert (Schritt 4). Im Anschluss wird ein Bit b gezogen (Schritt 5), anhand dessen entschieden wird, ob der Wähler die Ausweichstrategie nutzt, indem er einen falschen Schlüssel generiert und dem Angreifer diesen statt seines gültigen überlässt (Schritt 6),

Experiment $\mathbf{Exp}^{\textit{fake-key-c-resist}}_{ES,\mathcal{A},H}(k_1,\, k_2,\, k_3,\, n_E,\, n_{\mathcal{A}},\, n_P)$

1	V	$\leftarrow$	$\mathcal{A}$(voter names, "control voters");
2	$\{(\mathrm{sk}_j, \mathrm{pk}_j)$	$\leftarrow$	$\mathtt{register}(SK_{\mathcal{R}}, j, k_1)\}_{j=1}^{n_E}$;
3	(j_C, β)	$\leftarrow$	$\mathcal{A}(\{\mathrm{sk}_j\}_{j\in V}$, "set target voter and vote");
4	**if** $\lvert V\rvert \neq n_{\mathcal{A}}$ **or** $j_C \notin \{1,2,\ldots,n_E\} \setminus V$ **or** $\beta \notin \{1,2,\ldots,n_P\} \cup \{\emptyset\}$ **then**		
	output 0;		
5	$b \in_U \{0,1\}$;		
	if $b = 0$ **then**		
6	$\widetilde{sk}$	$\leftarrow$	$\mathtt{fakekey}(\mathrm{pk}_{\mathcal{T}}, \mathrm{sk}_{j_C}, \mathrm{pk}_j)$;
7	$\mathcal{BB}$	$\Leftarrow$	$\mathtt{vote}(\mathrm{sk}_{j_C}, \mathrm{pk}_{\mathcal{T}}, n_P, \beta, k_2)$;
	else		
8	$\widetilde{sk}$	$\leftarrow$	sk_{j_C};
9	$\mathcal{BB}$	$\Leftarrow$	$\mathtt{vote}(\{\mathrm{sk}_j\}_{j\neq j_C,\ j\notin V}, \mathrm{pk}_{\mathcal{T}}, n_P, D_{n_U,n_P}, k_2)$;
10	$\mathcal{BB}$	$\Leftarrow$	$\mathcal{A}(\widetilde{sk}, \mathcal{BB}$, "cast ballots");
11	$(\mathbf{X}, P)$	$\leftarrow$	$\mathtt{tally}(\mathrm{sk}_{\mathcal{T},l}, \mathcal{BB}, n_P, \{\mathrm{pk}_i\}_{i=1}^{n_V}, k_3)$;
12	b'	$\leftarrow$	$\mathcal{A}(\mathbf{X}, P$, "guess b");
13	**if** $b' = b$ **then**		
	output 1;		
	else		
	output 0;		

Abbildung 5.23.: Experiment $\mathbf{Exp}^{\textit{fake-key-c-resist}}_{ES,\mathcal{A},H}$ für die Verwendung von Fake Keys von Juels, Catalano und Jakobsson [73]. Dieses Experiment beschreibt die Möglichkeiten für einen Angreifer $\mathcal{A}$ in der *realen Welt*. Dabei steht $\emptyset$ für eine Enthaltung.

um in Schritt 7 selbst eine Stimme abzugeben, oder ob sich der Protokollteilnehmer j_C erpressen lässt und dem Angreifer seinen geheimen Schlüssel übergibt (Schritt 8). In Schritt 9 geben alle Protokollteilnehmer, die vom Angreifer nicht in Schritt 1 korrumpiert wurden, ihre Stimme ab. Der „Entscheidung" jedes unkorrumpierten Wählers liegt eine Verteilung D_{n_U,n_P} zugrunde, die beschreibt, welcher Kandidat wie häufig gewählt wird. D_{n_U,n_P} ist eine Verteilung über den Vektor $(\beta_1, \beta_2, \ldots, \beta_{n_U}) \in (1, \ldots, n_P \cup \{\emptyset\} \cup \{\lambda\})^{n_U}$, wobei $\emptyset$ für eine Enthaltung steht und λ für einen Stimmzettel mit ungültigen Authentifizierungsdaten; n_P ist die Zahl der Kandidaten. Für n_U unkorrumpierte Wähler ($n_U = n_E - n_{\mathcal{A}} - 1$) wird ein Vektor $(\beta_1, \beta_2, \ldots, \beta_{n_U})$ entsprechend der Verteilung D_{n_U,n_P} gezogen. Der j-te Wähler gibt seine Stimme gemäß β_j mit den Authentifizierungsdaten sk_j ab. Diese Verteilung muss im realen und idealen Experiment identisch gewählt sein. Anschließend gibt der Angreifer „seine" Stimmen ab: die der vom ihm kontrollierten Wähler als auch von dem Wähler, den er zu erpressen versucht. Die anschließende Auszählung gibt das Ergebnis und einen Korrektheitsbeweis zurück (Schritt 11), anhand derer der Angreifer das Bit b zu erraten versucht (Schritt 12).

Experiment für das ideale Protokoll, mit Verwendung ungültigen Schlüsselmaterials

Das in Abbildung 5.24 dargestellte Experiment für den idealen Protokollablauf unterscheidet sich – abgesehen davon, dass es eine ideale Auszählung verwendet – an zwei zentralen Stellen: Der Angreifer erhält im idealen Experiment sowohl im Fall $b = 0$ als auch im Fall $b = 1$ den echten Schlüssel des erpressten Wählers (Schritt 7); außerdem muss er sich nur anhand des Auszählungsergebnisses für die Belegung von b' entscheiden

Experiment $\mathbf{Exp}^{\textit{fake-key-c-resist-ideal}}_{ES,\mathcal{A}',H}(k_1,\ k_2,\ k_3,\ n_E,\ n_{\mathcal{A}'},\ n_P)$

1 $V \leftarrow \mathcal{A}'$(voter names, "control voters");
2 $\{(\mathrm{sk}_j, \mathrm{pk}_j) \leftarrow \texttt{register}(SK_{\mathcal{R}}, j, k_1)\}_{j=1}^{n_E}$;
3 $(j_C, \beta) \leftarrow \mathcal{A}'$("set target voter and vote");
4 **if** $|V| \neq n_{\mathcal{A}'}$ **or** $j_C \notin \{1, 2, \ldots, n_E\} - V$ **or** $\beta \notin \{1, 2, \ldots, n_P\} \cup \{\emptyset\}$ **then**
output 0;
5 $b \in_U \{0, 1\}$;
if $b = 0$ **then**
6 $\mathcal{BB} \Leftarrow \texttt{vote}(\mathrm{sk}_{j_C}, \mathrm{pk}_{\mathcal{T}}, n_P, \beta, k_2)$;
7 $\widetilde{sk} \leftarrow \mathrm{sk}_{j_C}$;
8 $\mathcal{BB} \Leftarrow \texttt{vote}(\{\mathrm{sk}_j\}_{j \neq j_C,\ j \notin V}, \mathrm{pk}_{\mathcal{T}}, n_P, D_{n_U, n_P}, k_2)$;
9 $\mathcal{BB} \Leftarrow \mathcal{A}'(\widetilde{sk}, \{\mathrm{sk}_j\}_{j \in V}$, "cast ballots");
10 $(\mathbf{X}, P) \leftarrow \texttt{ideal-tally}(\mathrm{sk}_{\mathcal{T},\,l}, \mathcal{BB}, n_P, \{\mathrm{pk}_i\}_{i=1}^{n_V}, k_3)$;
11 $b' \leftarrow \mathcal{A}'(\mathbf{X}$, "guess b");
12 **if** $b' = b$ **then**
output 1;
else
output 0;

Abbildung 5.24.: Experiment $\mathbf{Exp}^{\textit{fake-key-c-resist-ideal}}_{ES,\mathcal{A}',H}$ für Fake Keys von Juels, Catalano und Jakobsson [73]. Dieses Experiment beschreibt die Möglichkeiten des Angreifers $\mathcal{A}'$ in der *realen Welt*.

(Schritt 11). Den Beweis, dass die Auszählung korrekt ist, erhält er im idealen Experiment nicht. Das nimmt ihm – in der idealen Welt – die Möglichkeit, anhand des Schlüssels oder des Korrekheitsbeweises festzustellen, ob sein Erpressungsversuch erfolgreich ist. Dadurch sind in diesem Modell Angriffe abgedeckt, die durch eine Unterscheidung zwischen echtem und falschem Schlüssel möglich sind, sowie durch Informationen, die der Beweis enthält. Angriffe, die sich auf das Auszählungsergebnis stützen, sind sowohl im realen als auch im idealen Experiment möglich und daher für den Sicherheitsbegriff nicht relevant. Die ideale Auszählfunktion `ideal-tally` wählt in Abhängigkeit von b die Stimme des Wählers oder des Angreifers für die Auszählung aus.

Sicherheitsmodell für die Revoting-Ausweichstrategie

Ähnlich wie Juels, Catalano und Jakobsson Erpressungsfreiheit (coercion resistance) mit der Verwendung ungültigen Schlüsselmaterials modelliert haben, modellieren wir sie mit Hilfe der Abstreitbarkeit einer erneuten Stimmabgabe durch zwei Modelle: ein Modell für das reale Protokoll (Abbildung 5.25) und eines für ein ideales Protokoll (Abbildung 5.26). Das ideale Protokoll veröffentlicht neben dem Wahlergebnis insbesondere auch die Gesamtzahl der abgegebenen Stimmen und – je nach Designentscheidung – auch welcher Wähler (mindestens) eine Stimme abgegeben hat. Die Information über die Wahlbeteiligung kann, wenn das gefordert ist, in dem von uns entworfenen Protokoll aber eliminiert werden, ohne dass das Protokoll an Beweiskraft verliert. Juels, Catalano und Jakobsson [73] haben diese Eigenschaft in ihrer Definition von Erpressungsfreiheit

gefordert: Der Angriff einer erzwungenen Enthaltung erwirkt – wie der Name schon sagt –, dass ein Wähler nicht vom Angreifer unbemerkt an der Wahl teilnehmen kann.

Experiment für das reale Protokoll, mit der Möglichkeit seine Stimme zu überschreiben Wir definieren analog zu Juels, Catalano und Jakobsson ein reales und ein ideales Spiel (Abbildung 5.25 und 5.26).

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(k_1,\ k_2,\ k_3,\ n_E,\ n_{\mathcal{A}},\ n_P)$

1 $V \leftarrow \mathcal{A}$(voter names,"control voters");
2 $\{(\mathrm{sk}_j, \mathrm{pk}_j) \leftarrow \texttt{register}(SK_{\mathcal{R}}, j, k_1)\}_{j=1}^{n_E}$;
3 $(j_C, \beta) \leftarrow \mathcal{A}(\{\mathrm{sk}_j\}_{j\in V}$,"set target voter and vote");
4 **if** $|V| \neq n_{\mathcal{A}}$ **or** $j_C \notin \{1,2,\ldots,n_E\} - V$ **or** $\beta \notin \{1,2,\ldots,n_P\} \cup \{\emptyset\}$ **then**
output 0;
5 $\mathcal{BB} \Leftarrow \texttt{vote}(\{\mathrm{sk}_j\}_{j\notin V},\ PK_{\mathcal{T}},\ n_P,\ D_{n_U,n_P},\ k_2)$;
6 $\mathcal{BB} \Leftarrow \mathcal{A}(\mathrm{sk}_{j_C},\ \mathcal{BB}$, "cast ballots");
7 $b \in_U \{0,1\}$;
if $b = 0$ **then**
8 $\mathcal{BB} \Leftarrow \texttt{vote}(\{\mathrm{sk}_{j_C}\},\ PK_{\mathcal{T}},\ n_P,\ D_{n_U,n_P}, k_2)$;
9 $(\mathbf{X}, P) \leftarrow \texttt{tally}(SK_{\mathcal{T}},\ \mathcal{BB},\ n_P,\ \{\mathrm{pk}_i\}_{i=1}^{n_V}, k_3)$;
10 $b' \leftarrow \mathcal{A}(\mathbf{X},\ P,\ \mathcal{BB}$, "guess b");
11 **if** $b' = b$ **then**
output 1;
else
output 0;

Abbildung 5.25.: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}$. Das Experiment beschreibt die Möglichkeiten des Angreifers $\mathcal{A}$ in der *realen Welt*. D_{n_U,n_P} gibt nicht wie im Fall von Fake Keys einzelne Wahlentscheidungen aus; β_i kann nun aus einer Reihe von Wahlentscheidungen bestehen.

Die ersten vier Schritte des realen Experiments entsprechen denen im realen Experiment $\mathbf{Exp}^{\textit{fake-key-c-resist}}_{ES,\mathcal{A},H}$ von Juels, Catalano und Jakobsson (sieh Abbildung 5.23): Der Angreifer wählt, bevor das Schlüsselmaterial generiert wird, eine Menge korrumpierter Wähler und im Anschluss einen, den er versucht zu erpressen. Danach wird die Gültigkeit der Auswahl des Angreifers kontrolliert. Da keine Generierung falscher Schlüssel notwendig ist, werden im nächsten Schritt die Stimmen der unkorrumpierten Wähler abgegeben. Anders als im Experiment von Juels, Catalano und Jakobsson kann – entsprechend der Verteilung D_{n_U,n_P} – auch der vom Angreifer erpresste Wähler eine oder mehrere Stimmen abgeben. Der darauf folgende Schritt 6, in dem der Angreifer seine Stimmen abgibt, entspricht dem Schritt 10 des Experiments $\mathbf{Exp}^{\textit{fake-key-c-resist}}_{ES,\mathcal{A},H}$ von Juels, Catalano und Jakobsson. Dass erst in Schritt 7 das Zufallsbit gezogen wird, anhand dessen sich der erpresste Wähler entscheidet, ob er sich durch eine erneute Abgabe der Stimme der Erpressung widersetzt, bringt zum Ausdruck, dass die Entscheidung, ob sich der Wähler der Erpressung widersetzt, zu dem Zeitpunkt der Erpressung – anders als bei der Nutzung ungültigen Schlüsselmaterials – noch nicht gefallen sein muss und daher für den Angreifer keine Evidenz vorhanden sein kann, ob seine Erpressung erfolgreich sein wird. In dem Fall, dass das gezogene Bit $b = 0$ ist, wird die vom Angreifer für den erpressten Wähler gesendete Stimme überschrieben.

Auch hier macht die Reihenfolge den Unterschied der zwei Ansätze deutlich: Der Wähler muss in diesem Fall nach der Erpressung die Gelegenheit haben, erneut eine Stimme abgeben zu können. Ab der im Anschluss stattfindenden Auszählung gleichen sich die Experimente wieder. In $\mathbf{Exp}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist}}$ erhält der Angreifer lediglich das Bulletin Board explizit als Eingabe, das in $\mathbf{Exp}_{ES,\mathcal{A},H}^{\textit{fake-key-c-resist}}$ implizit im Beweis P enthalten sein muss.

Im Gegensatz zu den Experimenten von Juels, Catalano und Jakobsson gibt der Angreifer in den zwei Experimenten $\mathbf{Exp}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist}}$ und $\mathbf{Exp}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist-ideal}}$ jeweils gültige Stimmen ab. In der Zeit der Kontrolle durch den Angreifer wehrt sich der Wähler nicht gegen die Erpressung und folgt den Anweisungen des Angreifers, was zunächst zu einer gültigen Stimmabgabe in dessen Interesse führt. Erst durch das Überschreiben der Stimme in Schritt 8 des Experiments der realen Protokollausführung – dem Moment, in dem sich der Erpresste der Erpressung widersetzt – wird die erzwungene Stimme ungültig. Zu beachten ist, dass eine grundlegende Annahme der Authentifizierungsmethode (siehe Abschnitt 5.2.3) ist, dass der Angreifer die Authentifizierungsinformationen nicht entwenden kann beziehungsweise nicht die notwendigen Maßnahmen ergreifen möchte, um dies zu tun. Er zwingt den Wähler lediglich, sie in seinem Beisein korrekt zu verwenden.

Experiment für ideales Protokoll mit der Möglichkeit, seine Stimme zu überschreiben
Im idealen Experiment nutzen wird eine Funktion, die wir `ideal-tally` nennen (siehe Abbildung 5.26). Diese Funktion führt eine idealisierte Auszählung durch, deren Ausgabe sich auf die Anzahl der eingegebenen Stimmen und das Auszählungsergebnis $\mathbf{X}$ beschränkt. In Abhängigkeit vom Bit b nutzt es dafür die vom Wähler oder die vom Angreifer angegebene Stimme. Der Angreifer im idealen Modell erhält daher auch neben der Zahl der abgegebenen Stimmen und der Auszählung keine weiteren Daten vom Bulletin Board. Die Zahl der abgegebenen Stimmen ist im Fall $b = 0$ um eins höher als im Fall $b = 1$.

Im idealen Experiment ist es für den erpressten Wähler nicht notwendig, die Stimme des Angreifers zu überschreiben, da die `ideal-tally` die vom Angreifer für ihn abgegebene Stimme nur dann wertet ($b = 1$), wenn der Wähler im realen Experiment die erpresste Stimme nicht überschreiben würde. Der Angreifer erhält jedoch die Anzahl der insgesamt abgegebenen Stimmen (Zeile 11). Im Falle $b = 0$, in dem der Wähler den Erpressungsversuch abwehrt und im realen Experiment eine weitere Stimme durch das Überschreiben der erpressten hinzukommen würde, erhält der Angreifer eine Zahl, die um eins größer ist. Sofern er die Zahl der Stimmen, die ohne Erpressungsversuch abgegeben worden wären, nicht kennt, da er keine genaue Kenntnis über die gezogene Verteilung D_{n_U,n_P} hat, kann er nicht entscheiden, ob er das ursprüngliche, aus D_{n_U,n_P} abgeleitete l (Schritt 7), oder das Inkrement aus Schritt 9 erhalten hat.

Wir definieren den Vorteil des Angreifers $\mathcal{A}$ als $\mathbf{Adv}_{\mathcal{A}}^{\textit{revoting}}(k_1, k_2, k_3, n_E, n_{\mathcal{A}}, n_P) :=$

$$\mathbf{Succ}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist}}(k_1, k_2, k_3, n_V, n_A, n_C) - \mathbf{Succ}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist-ideal}}(k_1, k_2, k_3, n_V, n_A, n_C).$$

Ein Wahlverfahren bietet *abstreitbares Revoting*, wenn für alle Angreifer gilt, dass sein Vorteil $\mathbf{Adv}_{\mathcal{A}}^{\text{revoting}}$ vernachlässigbar in k_1, k_2, k_3 ist.

Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}(k_1,\ k_2,\ k_3,\ n_E,\ n_{\mathcal{A}'},\ n_P)$

```
1   V                 ←   A'(voter names, "control voters");
2   {(sk_j, pk_j)     ←   register(SK_R, j, k_1)}_{j=1}^{n_E};
3   (j, β)            ←   A'("set target voter and vote");
4   if |V| ≠ n_A' or j ∉ {1,2,...,n_E} − V or β ∉ {1,2,...,n_P} ∪ {∅} then
        output 0;
5   BB                ⇐   vote({sk_j}_{j∉V}, PK_T, n_P, D_{n_U,n_P}, k_2);
6   BB                ⇐   A'(sk_{j_C}, |BB|, "cast ballots");
7   l                 ⇐   |BB|;
8   b ∈_U {0,1};
    if b = 0 then
9       l             ⇐   |l| + 1;
10  (X, P)            ←   ideal-tally(SK_T, BB, n_P, {pk_i}_{i=1}^{n_V}, k_3, b);
11  b'                ←   A'(X, l, "guess b");
12  if b' = b then
        output 1;
    else
        output 0;
```

Abbildung 5.26.: Experiment $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A}',H}$. Dieses Experiment beschreibt die Möglichkeiten des Angreifers $\mathcal{A}'$, basierend auf dem Auszählungsergebnis. Mit dem Erfolg des Angreifers $\mathcal{A}'$ wird der Erfolg von $\mathcal{A}'$ normalisiert.

5.4.4. Erpressungsresistentes Aussortierungsverfahren

Im Folgenden stellen wir unseren Ansatz für ein Auszählverfahren vor, das die notwendigen Anforderungen erfüllt, wenn Wähler ihre Stimme abstreitbar überschreiben können sollen. Der Ablauf der Wahl lässt sich dabei in einzelne Phasen einteilen, die mit den Phasen der Lösung von Juels, Catalano und Jakobsson (siehe Abbildung 5.27) vergleichbar sind. Zu beachten ist, dass Juels, Catalano und Jakobsson auch die Möglichkeit vorsehen, die eigene Stimme durch eine neue überschreiben zu können. Da ihre Lösung jedoch die Verwendung von ungültigen Schlüsseln als Ausweichstrategie vorsieht, ist es nicht notwendig, die Abstreitbarkeit einer erneut abgegebenen Stimme zu garantieren. Ziel unseres Ansatzes ist es, die Aussortierung der überschriebenen Stimmen so zu gestalten, dass sie beweisbar korrekt ist und keine Informationen verrät, außer...

- ... wie viele Stimmen abgegeben wurden.
- ... wie viele Stimmen nach der Aussortierung verblieben sind.
- ... dass von jedem Wähler, der eine Stimme abgegeben hat, genau eine Stimme übrig geblieben ist.
- ... dass keine der verbliebenen Stimmen überschrieben wurde.
- ... dass jede der verbliebenen Stimmen von einem wahlberechtigten Wähler abgegeben wurde.

Wir untersuchen Revoting als Ausweichstrategie für einen erpressten Wähler. Die Ergebnisse sollen die Machbarkeit zeigen und zielen nicht darauf ab, eine effiziente, einsatzbereite Lösung zu sein.

Überblick

Bevor wir auf die Details des Protokolls eingehen, möchten wir zunächst einen Überblick über das Verfahren und die Parallelen zum Entwurf von Juels, Catalano und Jakobsson geben: Um eine Stimme abzugeben, senden die Wähler ihren Stimmzettel gemeinsam mit einem Beweis an einen Server, der diese sofort neben allen anderen Stimmzetteln auf einer Webseite (Bulletin Board) veröffentlicht. Bei Juels, Catalano und Jakobsson enthält der Beweis eine Aussage über die Kenntnis des Inhalts des Stimmzettels sowie über dessen Gültigkeit und die verschlüsselte Identität. In unserem Verfahren dient der öffentliche Schlüssel des Wählers (pk_i) als Identität, der verschlüsselt neben der ebenfalls verschlüsselten Stimme und dem unverschlüsselten Zeitstempel zum Stimmzettel gehört. Teil des Stimmzettels ist außerdem ein *nicht-interaktiver Zero-Knowledge-Beweis über die Kenntnis* (NIZKPoK) einer Signatur über die verschlüsselte Identität, die verschlüsselte Stimme und den Zeitstempel sowie ein Beweis, dass die Signatur mit dem öffentlichen Schlüssel (pk_i) zu verifizieren ist. Nach der Wahlphase, während der die Wähler ihre Stimmen abgeben konnten, werden die Stimmen in fünf Phasen ausgezählt. Die Phasen, wie sie von Juels, Catalano und Jakobsson beschrieben werden, sind in Abbildung 5.27 dargestellt.

Neben der Form des digitalen Äquivalents zum Stimmzettel, dessen Form oben bereits angedeutet ist und im folgenden Abschnitt genauer beschrieben wird, unterscheidet sich unser Protokoll im Wesentlichen im zweiten Schritt, in dem die Duplikate entfernt werden. Um diesen Vorgang zu realisieren, so dass keine außer den oben genannten Informationen anfallen, berechnen wir zu jedem Stimmzettel ein verschlüsseltes Label, das die Information enthält, ob die Stimme überschrieben wurde. Nach der Erstellung der verschlüsselten Label werden die Stimmzettel (ohne den Zeitstempel) gemeinsam mit ihren Labeln durch ein Mix-Net geführt. Danach können die Label entschlüsselt werden, ohne dass die Gefahr besteht, dass Rückschlüsse auf das Wählerverhalten gezogen werden können.

Im folgenden Abschnitt beleuchten wir die gewählte Authentifizierungsmethode, bevor wir dann die Protokollidee etwas ausführlicher skizzieren und im Anschluss die Details der Lösung betrachten.

Stimmzettel und Authentifizierungsmethode Für unser Auszählungsverfahren muss der Stimmzettel die folgenden Informationen enthalten: die eigentliche Stimme, die Identität des Wählers und den Zeitpunkt, zu dem die Stimme abgegeben wurde. Die Erstellung des Stimmzettels steht nicht im Fokus der Arbeit; es bedarf aber mindestens genauso viel Sorgfalt, ein geeignetes Protokoll zur Erstellung und Abgabe des Stimmzettels zu finden, sofern der PC, an dem der Stimmzettel erstellt wird, als korrumpiert angenommen wird. Im Folgenden gehen wir daher davon aus, dass die Hilfsmittel, die der Wähler während der Erstellung des Stimmzettels verwendet, vertrauenswürdig sind.

Zur Erstellung des Stimmzettels verschlüsselt der Wähler die Stimme β_j, die er abgeben möchte, sowie seine Identität – repräsentiert durch seinen eigenen öffentlichen Schlüs-

Phasen der Auszählung nach Juels, Catalano und Jakobsson

Beweise überprüfen: Die Wahlleitung überprüft alle mit den Stimmzetteln mitgelieferten Beweise und sortiert diejenigen aus, die mit einem ungültigen Beweis versehen sind.

Doppelte Stimmzettel entfernen: Da Juels, Catalano und Jakobsson das erneute Abgeben einer Stimme vorsehen, müssen überschriebene Stimmzettel entfernt werden. Um doppelte Stimmzettel ausfindig zu machen, überprüfen sie paarweise die den Stimmzetteln zugeordneten verschlüsselten IDs der Wähler mit Hilfe von *plaintext equivalency test*s (PETs). Von jedem Wähler werden alle bis auf einen Stimmzettel aussortiert; welcher Stimmzettel nicht aussortiert wird, wird anhand einer zuvor festgelegten Regel entschieden.

Shuffle: Die übrig gebliebenen Stimmzettel werden durch ein Mix-Net geleitet, so dass die Verbindung zu den ursprünglich abgegebenen Stimmzetteln verloren geht.

Authentifizierung überprüfen: Die Wahlleitung sortiert Stimmen ohne gültige Authentifizierung aus.

Auszählung: Die verbleibenden Stimmen werden entschlüsselt und gezählt.

Abbildung 5.27.: Die fünf Phasen im Ablauf des von Juels, Catalano und Jakobsson [73] vorgestellten Protokolls.

sel pk_j – mit dem öffentlichen Schlüssel der Wahlleitung. Der Zeitstempel ts_j bleibt unverschlüsselt. Der Stimmzettel sieht – wenn $\mathrm{pk}_{\mathcal{T}}$ der Schlüssel der Wahlleitung ist – dementsprechend bisher wie folgt aus:

$$\left(\mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}\left(\beta_j\right),\ \mathtt{ENC}_{\mathrm{pk}_{\mathcal{T}}}\left(\mathrm{pk}_j\right),\ \mathrm{ts}_j\right)$$

Um die Abgabe dieser Daten unter seiner Identität zu autorisieren, erstellt er zunächst eine Signatur σ über die drei teilweise verschlüsselten Werte und die folgenden Beweise (NIZKPoK):

1. Kenntnis einer Signatur σ, für die gilt $\mathtt{VERIF}_{\mathrm{pk}'_j}\left(\text{Stimmzettel},\ \sigma\right) = \text{True}$
2. $\mathrm{pk}'_j = \mathrm{pk}_j$

Sowohl σ als auch pk_j werden dabei nicht veröffentlicht. Diese Beweise fügt der Wähler nun zu seinem Stimmzettel als Autorisierungsinformation hinzu und gibt diesen ab. Zu beachten ist, dass das Verfahren zur Erstellung und Abgabe der Stimme sicherstellen muss, dass der Zeitstempel stimmt. Das lässt sich jedoch durch eine einfache serverseitige Kontrolle sicherstellen, da dieser nicht verschlüsselt ist. Die Form der Stimme ist für unser Auszählverfahren irrelevant, da diese Daten nicht verarbeitet werden.

Mit Hilfe des Authentifizierungsverfahrens wird sichergestellt, dass nur Stimmen von wahlberechtigten Wählern in das Wahlergebnis einfließen und diese nur in ihrem Namen Stimmen abgeben können: Unter der Annahme, dass den geheimen Schlüssel eines Wählers nur der Wähler selbst kennt und dass ein Angreifer ohne Kenntnis des geheimen Schlüssels nur mit vernachlässigbarer Wahrscheinlichkeit eine Signatur erstellen kann, die sich mit dem öffentlichen Schlüssel des Wählers verifizieren lässt, ist nur der Wähler selbst in

der Lage, einen gültigen Stimmzettel unter seinem Namen zu autorisieren. Jede Stimme muss, damit die Auszählung universell verifizierbar ist, auch für jeden Außenstehenden nachvollziehbar durch einen Wahlberechtigten autorisiert sein. Die Erstellung der Liste, wer zu dem Kreis der Wahlberechtigten gehört, ist eine organisatorische Herausforderung und kein Problem, das üblicherweise mit kryptographischen Mitteln gelöst wird. Daher nehmen wir eine öffentliche Liste mit allen Wahlberechtigten sowie ihren öffentlichen Schlüsseln als gegeben an. Die Implementierung, wie der Wähler seinen geheimen Schlüssel verwahrt und nutzt, ist zwar für unser Auszählungsprotokoll irrelevant, für das Gesamtkonzept einer Onlinewahl aber sehr wichtig. Das kann ebenso über eine Smartcard geschehen wie direkt über eine Software. Die Überlegungen für eine geeignete Umsetzung müssen einerseits berücksichtigen, dass relevante Angreifer daran gehindert werden müssen, den geheimen Schlüssel in Erfahrung zu bringen. Andererseits darf es auch nicht möglich sein, den Wähler daran zu hindern, seinen Schlüssel (unbemerkt) zu nutzen (siehe Abschnitt 5.2.3).

Protokollidee Zu dem Zeitpunkt, zu dem überschriebene Stimmzettel als solche erkennbar sind und aussortiert werden, darf keine Verbindung mehr zu ihrer Position auf dem Bulletin Board erkennbar sein. Dazu werden die Verbindungen vor dem Aussortieren der überschriebenen Stimmen und der Entschlüsselung der Stimmzettel mit Hilfe eines Mix-Nets aufgehoben. Dabei gehen insbesondere die Zeitpunkte, zu denen die Stimmzettel abgegeben wurden, verloren. Daher berechnen wir, bevor die Stimmzettel durch das Mix-Net geleitet werden, für jeden Stimmzettel ein verschlüsseltes Label, dessen Klartext bis zum Zeitpunkt der Entschlüsselung keinem Protokollteilnehmer bekannt ist, danach aber Auskunft darüber gibt, ob die Stimme überschrieben wurde. Entschlüsselt wird das Label nach dem Mix-Net gemeinsam durch die Mitglieder der Wahlleitung.

Bevor wir die Details der Lösung betrachten, geben wir zunächst einen Überblick, mit welchen Schritten wir das verschlüsselte Label erstellen. Zentrales Element ist dabei der mit einem multiplikativ-homomorphen Verschlüsselungsverfahren verschlüsselte öffentliche Schlüssel pk_j des Wählers, der Teil des Stimmzettels ist. Für jeden abgegebenen Stimmzettel b_{j_2} wird der verschlüsselte öffentliche Schlüssel mit denen der zuvor abgegebenen Stimmzettel b_{j_1} $(j_1 < j_2)$ verglichen $\texttt{ENC}\left(\mathrm{pk}_{j_1}\right)/\texttt{ENC}\left(\mathrm{pk}_{j_2}\right) = \texttt{ENC}\left(\mathrm{pk}_{j_1}/\mathrm{pk}_{j_2}\right)$ und das Ergebnis zu dem älteren Stimmzettel b_{j_1} gespeichert. Der Quotient der öffentlichen Schlüssel der beiden Stimmzettel ist *Eins*, wenn die Stimmzettel vom selben Wähler stammen, und *ungleich Eins* sonst. Für jeden neu abgegebenen Stimmzettel erhalten alle bereits abgegebenen einen neuen verschlüsselten Quotienten. Ist einer der Quotienten eines Stimmzettels Eins, so wurde dieser durch einen neueren des gleichen Wählers überschrieben. Stimmzettel, bei denen ein Angreifer einen öffentlichen Schlüssel angibt, zu dem er keinen geheimen Schlüssel kennt, werden aufgrund ihrer ungültigen Beweise aussortiert. Dadurch wird das Überschreiben fremder Stimmen verhindert. Aus den verschlüsselten Quotienten, die nach Abschluss der Wahlphase zu einer Stimme gespeichert sind, muss nun das Label berechnet werden. Wir wollen dazu die Homomorphie der Verschlüsselung ausnutzen. Da aber eine Eins, auf deren Existenz die Reihe der verschlüsselten Quotienten geprüft werden soll, das Produkt nicht verändern und daher nicht auffallen würde, wenden wir einen Trick

an. Bevor das Produkt aus den verschlüsselten Quotienten berechnet wird, ändern wir jede verschlüsselte Eins zu einer verschlüsselten Zufallszahl und jedes Chiffrat, das eine Zahl ungleich Eins enthält, zu einer verschlüsselten Eins (siehe Abbildung 5.28). Das Produkt über die wie eben beschriebenen konvertieren Quotienten ist unser gesuchtes verschlüsseltes Label.

Funktionsweise der Berechnung der Labels o_j

$$\begin{array}{rcccccccl} 12 \equiv & 81 & \cdot & 53 & \cdot & 87 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 1 \equiv & 1 & \cdot & 1 & \cdot & 1 & \cdot & 1 & \bmod N \end{array}$$

(a) Stimmzettel ist nicht überschrieben; keiner der Faktoren ist 1.

$$\begin{array}{rcccccccl} 25 \equiv & 81 & \cdot & 53 & \cdot & 1 & \cdot & 48 & \bmod N \\ & \downarrow & & \downarrow & & \downarrow & & \downarrow & \\ 139 \equiv & 1 & \cdot & 1 & \cdot & 139 & \cdot & 1 & \bmod N \end{array}$$

(b) Stimmzettel ist genau einmal überschrieben; einer der Faktoren ist 1.

Abbildung 5.28.: Vor der Konvertierung ist anhand des Produktes nicht zu entscheiden, ob einer der Faktoren 1 ist (obere Zeile). Um die Homomorphie des Verschlüsselungsverfahrens auszunutzen, ersetzen wir Zahlen $\neq 1$ durch 1 und umgekehrt. Nun kann anhand des Produkts beurteilt werden, ob ein Faktor $\neq 1$ ist (untere Zeile).

Protokoll

Bei dem von uns vorgestellten Verfahren handelt es sich um ein Aussortierungsverfahren, das überschriebene Stimmen so entfernt, dass ein Wähler abstreiten kann, dass durch seine Handlung eine Stimme überschrieben wurde. Die Einordnung dieses Verfahrens in den Ablauf des gesamten Wahlprozesses stellt sich wie folgt dar:

Eine Wahl lässt sich im Allgemeinen in drei Phasen einteilen: Eine Vorbereitungsphase (Abbildung 5.29), die Wahlphase (Abbildung 5.30) und die Auszählungs- bzw. Auswertungsphase (Abbildung 5.31). Die Aussortierung findet zu Beginn der letzten der drei Phasen statt. In den folgenden Abschnitten werden wir unser Protokoll für den zweiten Schritt der dritten Phase, der *Aussortierung der überschriebenen Stimmen*, im Detail erklären. Wir beginnen jedoch mit der dazu notwendigen Authentifizierungsmethode.

Stimmzettel und Authentifizierung Wie bereits erwähnt, steht der Stimmabgabeprozess nicht im Fokus unserer Arbeit. Wir kommen aber nicht umhin, das Format, das ein Stimmzettel bei Abgabe haben muss, in einem gewissen Rahmen festzulegen. Das von uns vorgegebene Format enthält neben den Authentifizierungsinformationen drei Teile:

$$b_i = (\mathtt{ENC}(\beta_j), \mathtt{ENC}(\mathrm{pk}_i), \mathrm{ts}_j),\ NIZKPoK$$

Stimme: $\mathtt{ENC}(\beta_j)$
Der erste Teil des Stimmzettels ist das Chiffrat (oder die Chiffrate) der Stimme β_j.

Phase eins – Vorberechnung

Vorbereitung Vorberechnung vor der Wahlphase

1. Schlüsselerzeugung:
   In dieser Phase wird das zur Auszählung notwendige Schlüsselmaterial erstellt: Der öffentliche Schlüssel $\mathrm{pk}_{\mathcal{T}}$, der in unserem Fall für die Verschlüsselung der Stimme und der Identität des Wählers benötigt wird, wird gemeinsam von der Gruppe der Auszählenden erstellt. Im gleichen Zug wird auch ein geheimer Schlüssel $\mathrm{sk}_{\mathcal{T},\,l}$ für jedes Mitglied a_l der Gruppe für die verteilte Entschlüsselung erzeugt (siehe Abschnitt 2.1.2).
2. Registrierung:
   Jeder Wähler e_j $(1 \leq j \leq n_E)$ erhält das Schlüsselmaterial, um sich für die Stimmabgabe ausweisen zu können. In unserem Fall handelt es sich um ein Schlüsselpaar $(\mathrm{pk}_j, \mathrm{sk}_j)$ eines EUF-CMA-sicheren Signaturschemas (siehe Abschnitt 2.2). Wir nehmen an, dass es einen effizienten, nicht-interaktiven *Zero-Knowledge-Beweis der Kenntnis* für diese Signaturen gibt. Der öffentliche Schlüssel pk_j fungiert im folgenden Schema ebenfalls als Identifikationsnummer des Wählers. Der Name des Wählers wird gemeinsam mit dem öffentlichen Schlüssel in einer später veröffentlichten Liste L_{Voter} gespeichert. Sollen Angriffe, die auf die Teilnahme an der Wahl zielen (siehe Abschnitt 5.2.2), verhindert werden, kann der öffentliche Schlüssel hier bereits verschlüsselt abgelegt werden (siehe Abschnitt 5.4.4, Seite 137).
3. Veröffentlichung:
   Teile der vorberechneten Daten und andere, die Wahl betreffende Daten werden veröffentlicht: eine Kandidatenliste $\mathcal{C} = (p_1, \ldots, p_{n_P})$, der öffentliche Schlüssel $\mathrm{pk}_{\mathcal{T}}$ und die Liste der Wahlberechtigten L_{Voter}.

Abbildung 5.29.: Erste Phase im Ablauf des Wahlablaufs für das von uns vorgestellte Protokoll.

Dabei ist es für uns irrelevant, welche Form die Stimme hat. Es ist lediglich von Bedeutung, dass das Chiffrat für die später angewendeten Mix-Netze rerandomisierbar ist. Wir werden im Folgenden der Einfachheit halber davon ausgehen, dass es sich um ein einzelnes Chiffrat handelt.

Technische Bemerkung: Form der ersten Veröffentlichung
Kann der Wähler die Form der Stimme, wie sie (das erste Mal) veröffentlicht wird, bestimmen, kann ein Angreifer den Wähler immer zwingen, ein von ihm vorgegebenes Element abzugeben, das er auf dem Bulletin Board wiedererkennt. Im Allgemeinen muss durch ein Protokoll zwischen Bulletin Board und Wähler eine Rerandomisierung berechnet werden, die im Anschluss veröffentlicht wird. Durch die Möglichkeit, erneut eine Stimme abzugeben und dies abstreiten zu können, müssen wir diesen Angriff in unserem Protokoll nicht berücksichtigen. Um dennoch frei wählen zu können, folgt der erpresste Wähler den Anweisungen des Angreifers und überschreibt die erpresste Stimme im Anschluss mit seiner gewünschten Stimme.

Wähler-ID: $\mathtt{ENC}\,(\mathrm{pk}_i)$

Der zweite Teil des Stimmzettels ist die verschlüsselte ID des Wählers. Aufgrund der von uns verwendeten Authentifizierungstechnik muss die ID der öffentliche Schlüssel eines Signaturschlüsselpaares sein. Das verwendete Verschlüsselungsverfahren muss für einen PET geeignet sein (siehe Kapitel 2.1.4).

Phase zwei – Abgabe der Stimmen

Wahlphase Abgabe der Stimmen durch den Wähler

In der Wahlphase geben die Wähler e_j jeweils eine oder mehrere Stimmen ab. Dazu erstellen sie für jede Stimme einen Stimmzettel b_j. Dieser besteht in unserem Verfahren aus der verschlüsselten Stimme $\texttt{ENC}_{\text{pk}_\mathcal{T}}(\beta_j)$ mit $\beta_j \in \{p_1, \ldots, p_{n_P}\}$, seinem verschlüsselten öffentlichen Schlüssel $\texttt{ENC}_{\text{pk}_\mathcal{T}}(\text{pk}_j)$, einem Zeitstempel ts_j und einem Beweis $NIZKPoK$ über die Kenntnis einer Signatur über $\texttt{ENC}_{\text{pk}_\mathcal{T}}(\beta_j)$, $\texttt{ENC}_{\text{pk}_\mathcal{T}}(\text{pk}_j)$ und ts_j sowie der Verwendung des richtigen pk_j. Weitere Details zum Stimmzettel finden sich in Kapitel 5.4.4 auf Seite 124.

Die Stimmzettel werden in unserem Falle direkt nach der Abgabe veröffentlicht. Ein dazu verwendetes *Schwarzes Brett* wird üblicherweise als *Bulletin Board* ($\mathcal{BB}$) bezeichnet. Die Zahl der abgegebenen Stimmzettel benennen wir mit n_β.

Abbildung 5.30.: Zweite Phase im Ablauf des Wahlablaufs für das von uns vorgestellte Protokoll.

Zeitstempel: ts_j

Der dritte Teil des Stimmzettels ist der (unverschlüsselte) Zeitstempel, wann die Stimme erstellt und abgegeben wurde. Der Zeitstempel legt die Reihenfolge der Stimmen auf dem Bulletin Board fest. Die Reihenfolge ist entscheidend, da neuere Stimmen ältere überschreiben.

Technische Bemerkung: Korrektheit des Zeitstempels
Um Erpressungen zu vermeiden, die dem Wähler die Möglichkeit nehmen, die bereits abgegebene Stimme zu überschreiben, muss vermieden werden, dass Stimmen im Voraus erstellt werden können. Die Erstellung muss daher zwingend mit der vom Angreifer nicht gänzlich korrumpierten Wahlleitung geschehen. Um einen korrekten Zeitstempel zu garantieren, könnte der Stimmzettel beispielsweise (inklusive des Zeitstempels) von der Wahlleitung signiert werden. Ein ehrliches Wahlleitungsmitglied wird nie eine Signatur auf einen Zeitstempel in der Zukunft erstellen. Da es sich dabei aber eher um Details der Stimmabgabe handelt, beachten wir der Einfachheit halber diese Erweiterung nicht.

Authentifizierung: NIZKPoK

Für die Authentifizierung darf die Kenntnis der Identität des Wählers nicht notwendig sein. Wir nehmen, um das zu erreichen, einen nicht-interaktiven-Zero-Knowledge-Kenntnisbeweis an, der das Folgende nachweist:

1. Der Wähler kennt eine Signatur σ, für die gilt

$$\texttt{VERIF}_{\text{pk}'_i}\left(\left(\texttt{ENC}\left(\beta_j\right), \texttt{ENC}\left(\text{pk}_i\right), \text{ts}_j\right),\ \sigma\right) = 1$$

2. Der Schlüssel, mit dem die Signatur erfolgreich überprüft werden kann, entspricht dem aus dem Chiffrat.

$$\text{pk}'_i = \text{pk}_i$$

Dabei ist zu beachten, dass weder σ noch pk_i preisgegeben werden.

Phase drei – Bestimmung des Auszählungsergebnisses

Auszählung Bestimmung und Auszählung der gültigen Stimmen

1. Beweise prüfen:
   Die Überprüfung, ob der Stimmzettel wohlgeformt ist, kann prinzipiell auch direkt nach der Abgabe während der Wahlphase geschehen. Sie stellt sicher, dass die Anforderungen, die für eine korrekte Auszählung an den Stimmzettel gestellt werden müssen, eingehalten werden. Juels, Catalano und Jakobsson fordern beispielsweise einen nicht-interaktiven Beweis der Kenntnis der abgegebenen Stimme und der Wähleridentität sowie dass die Wahl gültig ist. Für unser Verfahren muss geprüft werden, ob der Wähler eine Signatur des Stimmzettels kennt, die zu dem öffentlichen Schlüssel passt, der in verschlüsselter Form Teil des Stimmzettels ist.
2. Aussortieren überschriebener Stimmen:
   Vor der Entschlüsselung der Stimmen müssen die Stimmen, die überschrieben wurden, gelöscht werden. Dabei darf es für einen Angreifer nicht nachvollziehbar sein, ob ein von ihm beobachteter Wähler eine seiner zuvor abgegebenen Stimmen überschrieben hat. Ein solcher Aussortierungsprozess wird im Folgenden im Detail besprochen.
3. Wahlberechtigung prüfen:
   Nachdem von jedem Wähler nur noch ein Stimmzettel vorhanden ist, wird geprüft, ob dieser Wähler auch berechtigt war, eine Stimme abzugeben. Dies vor der Aussortierung der überschriebenen Stimmen zu tun, würde möglicherweise Angriffe ermöglichen, wie sie in Abschnitt 5.4.1 beschrieben sind.
4. Auszählung der Stimmen:
   Die Menge der (verschlüsselten) Stimmen wird zu einem Auszählungsergebnis $\mathbf{X}$ verrechnet. Der einfachste Fall ist die Entschlüsselung der Chiffrate, so dass die Stimmen auf ihre Gültigkeit hin überprüft und alle gültigen aufaddiert werden können.
5. Interpretation des Ergebnisses:
   Nach der Auszählung muss das Ergebnis noch interpretiert werden. Das heißt, aus der Anzahl der Stimmen für die einzelnen Kandidaten müssen Sitze o. ä. berechnet werden.

Abbildung 5.31.: Dritte Phase im Ablauf des Wahlablaufs für das von uns vorgestellte Protokoll.

Aussortieren überschriebener Stimmen Das *Aussortieren der überschriebenen Stimmen* teilen wir in mehrere Schritte ein: Die Eingabe des ersten Schritts ist der Inhalt des Bulletin Boards, das als Liste der abgegebenen Stimmzettel gesehen werden kann. Der letzte Schritt – die Überprüfung der Wahlberechtigung – gibt eine Liste aus, die alle validen, nicht überschriebenen Stimmzettel ohne ihre Zeitstempel und Wählerinformation, rerandomisiert und in einer zufälligen Reihenfolge enthält. Ein Stimmzettel ist genau dann überschrieben, wenn es einen gültigen Stimmzettel mit gleichem verschlüsselten öffentlichen Schlüssel und neuerem Zeitstempel auf dem Bulletin Board gibt.

1. **Berechnen verschlüsselter Information, ob zwei Stimmzettel vom gleichen Wähler stammen:** Wir gehen von einer Liste aller abgegebenen, gültigen Stimmzettel aus. Dazu wurden die zugehörigen NIZKPoKs geprüft und diejenigen mit ungültigen Beweisen aussortiert. Die NIZKPoKs werden nicht in den nächsten Schritt übernommen. Die Liste wird – wenn nicht schon geschehen – nach Abgabezeit

(ts_j) des Stimmzettels sortiert. Nun werden die verschlüsselten öffentlichen Schlüssel $\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_j)$ eines Stimmzettels j mit allen $\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_{j'})$ der neueren Stimmzettel j' verglichen: $(d_{j,j'}, \Pi_{j,j'}) := \mathtt{EPET}(\mathtt{ENC}(\mathrm{pk}_j), \mathtt{ENC}(\mathrm{pk}_{j'}))$. Das Protokoll (EPET), mit dessen Hilfe diese verschlüsselten Informationen berechnet werden, ist in Abschnitt 5.4.4 auf Seite 131 näher erklärt. Das daraus resultierende Chiffrat $d_{j,j'}$ enthält eine Eins, wenn die öffentlichen Schlüssel identisch sind, eine zufällige Zahl sonst. Zugeordnet wird $d_{j,j'}$ dem älteren Stimmzettel j.

2. **Berechnen der Labels, ob Stimmzettel überschrieben wurden** Aus den zuvor berechneten $d_{j,j'}$ werden die verschlüsselten Labels o_j erstellt, die die Information enthalten, ob der Stimmzettel überschrieben wurde. Dazu müssen die Inhalte der $d_{j,j'}$ geändert werden: Enthält $d_{j,j'}$ eine Eins, wird es durch ein Chiffrat ersetzt, das eine Zufallszahl enthält. In allen anderen Fällen wird es durch ein Chiffrat mit einer Eins ersetzt. Das neue Chiffrat bezeichnen wir mit $\hat{d}_{j,j'}$. Dieser Vorgang findet losgelöst von den Stimmzetteln statt und wird in Abschnitt 5.4.4 auf Seite 133 genauer beschrieben. Für jeden Stimmzettel b_j ergibt die Aggregierung der Elemente $\hat{d}_{j,j'}$ mit $j+1 \leq j' \leq n_\beta$ das gesuchte verschlüsselte Label o_j, wobei n_β die Anzahl der Stimmzettel ist.

3. **Entferne alle überschriebenen Stimmzettel** Nach einem Shuffle der Stimmzettel ohne den Zeitstempel und mit dem dazugehörigen Label o_j können die Stimmzettel, die keine Label mit einer Eins enthalten, gelöscht werden.

4. **Stimmzettel mit ungültigen IDs werden entfernt** Im letzten Schritt werden die Stimmzettel, die mit einer ungültigen Identität versehen sind, aus dem Prozess entfernt. In Abhängigkeit davon, ob es ein Urnenbuch geben soll, kann dieser Prozess durch das Öffnen der Chiffrate stattfinden oder durch einen Zero-Knowledge-Beweis, dass sich der entsprechende öffentliche Schlüssel in der Liste der gültigen Wähler befindet.

Im Weiteren werden wir auf die Details der vier Schritte eingehen.

Berechnen verschlüsselter Information, ob zwei Stimmzettel vom gleichen Wähler stammen In diesem Schritt findet der Vergleich statt, der aussagt, ob ein neu abgegebener Stimmzettel einen bereits bestehenden Stimmzettel überschreibt. Obwohl es sich dabei um den ersten Schritt der Nachberechnung handelt, die nach der Wahlphase stattfindet, können die Berechnungen schon während der Wahl stattfinden. Ein Stimmzettel kann Auswirkungen auf die bereits abgegebenen Stimmzettel haben, jedoch nie auf Stimmzettel, die später eingefügt werden.

Ausgehend von zwei leeren Listen L_0 und L_{trash}, von denen L_0 ein leeres Bulletin Board repräsentiert, gelten stets die folgenden Bedingungen:

- Alle Stimmzettel in der Liste L_0 haben einen gültigen Beweis (siehe Abschnitt 5.4.4).

- Der Zeitstempel ts_a des Stimmzettels L_0[a] ist kleiner als der Zeitstempel ts_b des Stimmzettels L_0[b] für alle $a < b$.
- Allen Stimmzetteln in der Liste L_0, denen mindestens ein neuerer Stimmzettel mit der gleichen ID folgt, ist mindestens eine verschlüsselte Eins angehängt.
- Allen Stimmzetteln in der Liste L_0, denen noch kein neuerer Stimmzettel mit gleicher ID folgt, ist keine verschlüsselte Eins angehängt.

Für eine leere Liste gelten diese vier Invarianten trivialerweise. Wird eine neue Stimme abgegeben, so müssen die folgenden Bedingungen gelten, bevor sie in die Liste aufgenommen wird:

- Der Stimmzettel muss über einen gültigen Beweis verfügen. Damit wird die Einhaltung der ersten Invariante sichergestellt.
- Der Zeitstempel ts_j des Stimmzettels muss größer sein als die aller vorangegangenen Stimmzettel. Eine Methode, die sicherstellt, dass Stimmzettel mit der aktuellen Zeit als Zeitstempel erstellt werden, muss Teil des Prozesses der Stimmabgabe sein. Ist diese gegeben, ist die Bedingung zur Einhaltung der zweiten Invariante erfüllt.

Sind diese Bedingungen nicht erfüllt, so wird der Stimmzettel zu Dokumentationszwecken an die Liste L_{trash} angehängt. Dem Wähler kann eine entsprechende Rückmeldung gegeben werden.

Sind diese Bedingungen erfüllt, so werden die folgenden Aktionen ausgeführt:

- Sei L_0 eine Liste mit Stimmzetteln, für die die oben genannten Invarianten gelten. L_0 enthalte o. B. d. A. bereits n-1 Elemente mit $n \geq 1$. Sei b_n der neue Stimmzettel mit $b_n = (\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\beta_n), \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_n), \mathrm{ts}_n)$.
- Berechne für jeden Stimmzettel $b_j = (\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\beta_j), \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_j), \mathrm{ts}_j) \in \mathrm{L}_0$:

  $$d_{j,n} := \mathtt{EPET}(\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_j), \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_n))$$

  und hänge das Chiffrat $d_{j,n}$ an den Stimmzettel b_j an.
- Hänge den neuen Stimmzettel b_n an die Liste L_0 als n-ten Eintrag an.

Für jeden neu abgegebenen Stimmzettel b_n wird allen bereits abgegebenen b_j ein neues $d_{j,n}$ angehängt (siehe Abbildung 5.32). Bereits vorhandene $d_{j,n'}$ $(n' < n)$ werden nicht verändert. Um den Invarianten drei und vier zu genügen, ist es ausreichend, wenn allen bereits abgegebenen Stimmzetteln, die von dem gleichen Wähler stammen, von dem auch der hinzuzufügende Stimmzettel erstellt wurde, eine verschlüsselte Eins und allen anderen ein anderer verschlüsselter Wert angehängt wird. Diese Bedingungen erfüllen die $d_{j,n}$, die durch EPET erstellt wurden:

$$d_{j,n} \begin{cases} = \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(1) & \text{wenn } \mathrm{pk}_j = \mathrm{pk}_n \\ = \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(r) & \text{mit } r \neq 1, \text{ wenn } \mathrm{pk}_j \neq \mathrm{pk}_n \end{cases}$$

Genau genommen garantiert der Prozess EPET nicht, dass $r \neq 1$. Die Wahrscheinlichkeit dafür kann aber vernachlässigt werden. Zu jedem Zeitpunkt gilt, dass an einen Stimmzettel j genau $n - j$ $d_{j,n'}$ angehängt sind, wobei b_j der j-te von insgesamt n Stimmzetteln ist. Für n' gilt $j < n' \leq n$.

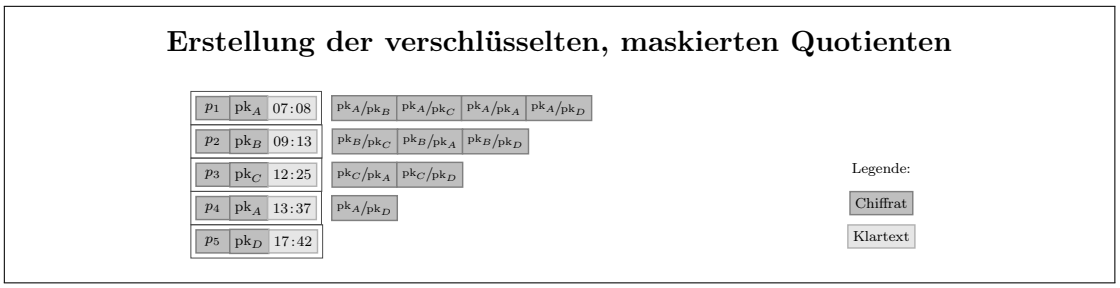

Abbildung 5.32.: Encrypted Plaintext Equality Tests (EPETs) werden zwischen verschlüsselten öffentlichen Schlüsseln aller bereits abgegebenen Stimmzettel $(\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\beta_j), \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_j), \mathrm{ts}_j)$ und des neu hinzugekommenen Stimmzettels $(\mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\beta_n), \mathtt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{pk}_n), \mathrm{ts}_n)$ ausgeführt. Wenn $\mathrm{pk}_j = \mathrm{pk}_n$, ist das Ergebnis eine verschlüsselte Eins, in allen anderen Fällen ein Chiffrat eines zufälligen Wertes. Verschlüsselte Werte werden durch eine Box mit einem dunkleren Hintergrund dargestellt. Boxen mit hellerem Hintergrund stellen eine unverschlüsselte Information dar: in dieser Graphik ausschließlich die Zeitstempel. Die Randomisierung der Quotienten $(\mathrm{pk}_A/\mathrm{pk}_B)$ durch einen zufälligen Exponenten $\hat{r}$ $((\mathrm{pk}_A/\mathrm{pk}_B)^{\hat{r}})$ haben wir der Übersichtlichkeit halber nicht explizit dazugeschrieben.

Berechnen der Labels, ob Stimmzettel überschrieben wurden Um die Homomorphie der Chiffrate $d_{j,j'}$ für die Zusammenfassung zu einem Wert auszunutzen, der besagt, ob der Stimmzettel überschrieben wurde, müssen die Inhalte *konvertiert* werden. Im aktuellen Zustand müsste herausgefunden werden, ob es sich bei mindestens einem der $d_{j,j'}$ für jedes j um eine verschlüsselte Eins handelt. Im Produkt aus zumeist Zufallszahlen lässt sich nicht feststellen, ob sich unter den Faktoren eine Eins befand. Umgekehrt ließe sich bei der Betrachtung eines Produktes feststellen, ob sich in der Reihe der Faktoren eine oder mehrere Zufallszahlen (Zahlen $\neq 1$) befinden.

Die bisher gültigen Invarianten sollen dementsprechend wie folgt geändert werden:

- Alle Stimmzettel in der Liste L_0 haben einen gültigen Beweis.
- Der Zeitstempel ts_a des Stimmzettels $\mathrm{L}_0[\mathrm{a}]$ ist kleiner als der Zeitstempel ts_b des Stimmzettels $\mathrm{L}_0[\mathrm{b}]$ für alle $a < b$.
- Allen Stimmzetteln in der Liste L_0, denen mindestens ein neuerer Stimmzettel mit der gleichen ID folgte, ist mindestens eine verschlüsselte $\underline{\text{Zahl} \neq 1}$ angehängt.
- Allen Stimmzetteln in der Liste L_0, denen kein neuerer Stimmzettel mit gleicher ID folgte, haben keine verschlüsselte $\underline{\text{Zahl} \neq 1}$ angehängt.

Dazu müssen alle Chiffrate $d_{j,j'}$ so geändert werden, dass

$$\texttt{ENC}(x) \mapsto \begin{cases} \texttt{ENC}(r) & \text{wenn } x = 1 \\ \texttt{ENC}(1) & \text{wenn } x \neq 1 \end{cases}$$

gilt. (Der weitere Aussortierungsprozess kann so gestaltet werden, dass statt der Zufallszahl r auch ein fester Wert verwendet werden könnte. Elementar ist dann dabei, dass später ein PET verwendet wird, um den Inhalt von o_j auf Gleichheit mit *Eins* zu prüfen, da das Produkt verraten würde, wie viele Elemente keine *Eins* enthalten.)

Für die Umwandlung verschlüsselt die Wahlleitung die Zeitstempel ($\texttt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{ts}_j)$) und erstellt Tupel aus den $d_{j,j'}$ und dem zugehörigen verschlüsselten Zeitstempel:

$$\left(\texttt{ENC}_{\mathrm{pk}_\mathcal{T}}(\mathrm{ts}_j),\, d_{j,j'}\right)$$

Dabei kann für jedes Tupel, das zu demselben Stimmzettel gehört, auch dasselbe, bitweise identische Chiffrat des Zeitstempels verwendet werden. Alle Tupel werden zu einer zuvor leeren Liste L_1 hinzugefügt (siehe Abbildung 5.33). Alle Tupel in L_1 werden im anschließenden `Shuffle` rerandomisiert.

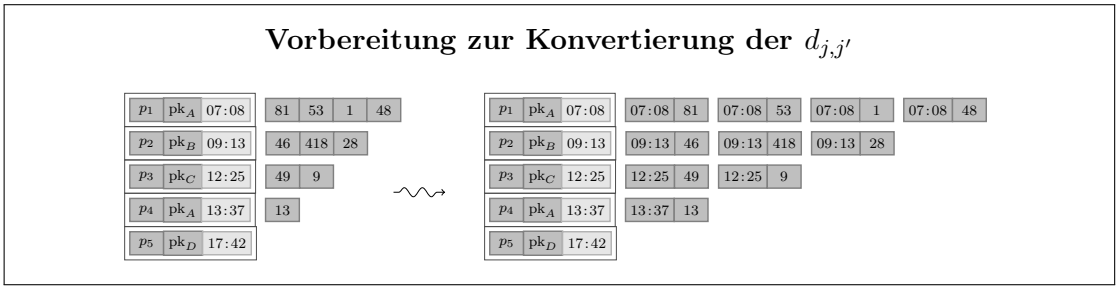

Abbildung 5.33.: Die maskierten Quotienten aus dem vorherigen Schritt werden durch die verschlüsselten Zeitstempel ergänzt und bilden das Tupel $(\texttt{ENC}(ts_i), d_{j,j'})$. Für eine bessere Verständlichkeit verwenden wir exemplarische Zahlenwerte. Der erste Stimmzettel wurde überschrieben. Daher enthält die ihm zugeordnete Liste eine verschlüsselte Eins.

Die Liste der Tupel L_1 wurde durch einen `Shuffle` rerandomisiert und permutiert. Der Zusammenhang mit den ursprünglichen Elementen auf dem Bulletin Board ist aufgelöst. Für die später notwendige Zuordnung werden die jetzt noch verschlüsselten Zeitstempel verwendet.

In einem mehrstufigen Prozess, der im Folgenden beschrieben wird, werden die $d_{j,j'}$ geöffnet und durch ein Chiffrat $\hat{d}_{j,j'}$, das den konvertierten Wert enthält, ersetzt (siehe dazu Abbildung 5.34).

Der Prozess, in dem die Chiffrate geöffnet werden, um zu bestimmen, welchen Inhalt $\hat{d}_{j,j'}$ haben müssen, muss selbst nicht verifizierbar sein. Die Korrektheit des Ergebnisses für sich ist beweisbar. Dennoch würde die Zahl der Einsen dem, der die Chiffrate entschlüsselt, zu viel über das Abstimmungsverhalten der Wähler verraten. Um darüber keine Informationen

offenzulegen, muss die Zahl der Einsen (in Relation zu der Zahl der abgegebenen Stimmen) bereits vor der Durchführung der Konvertierung feststehen. Danach dürfen lediglich aus der Reihenfolge keine Rückschlüsse mehr gezogen werden können. Die Nivellierung der Einsen kann dadurch geschehen, dass jedes Chiffrat zweimal konvertiert wird, wobei nur das Ergebnis der ersten Konvertierung für die weiteren Berechnungen Verwendung findet. Um diese Bedingung zu erfüllen, hilft der folgende Aufbau:

Der Einfachheit halber betrachten wir zunächst eine Architektur, bei der wir annehmen, dass die Komponenten nicht miteinander kooperieren, um das Geheimnis zu erfahren. Die Komponenten lassen sich leicht so erweitern, dass jedes Mitglied der Wahlleitung eine Rolle übernehmen kann und das Schema genau dann sicher ist, wenn mindestens ein Mitglied vertrauenswürdig ist. Eine Komponente nennen wir den *Koordinator*, die andere ist für die Entschlüsselung und Konvertierung zuständig. Der Koordinator erhält als Eingabe die Liste L_1 der zu konvertierenden $d_{j,j'}$. Er rerandomisiert alle $d_{j,j'}$ in der Liste. Des weiteren fügt er Chiffrate an zufälligen Stellen in die Liste ein, die entweder eine Eins oder einen zufälligen Wert enthalten. Die Position dieser merkt er sich, um sie später entfernen zu können. Jedes Element erhält ein Flag, das drei Zustände annehmen kann: 0, 1, 2. Das Flag spiegelt wider, wie oft das Chiffrat bereits zur Konvertierung geschickt wurde. Die folgende Prozedur wiederholt der Koordinator, bis alle Flags in dem Zustand 2 sind.

1. Ziehe ein zufälliges Element aus der Liste aller Elemente, deren Flag $\neq 2$ ist.
2. Sende das Element zum Konvertieren.
3. Wenn das Flag des Elements $= 0$ ist, rerandomisiere den Rückgabewert, ersetze damit das ursprüngliche Element und setze das Flag auf 1.
4. Wenn das Flag des Elements $= 1$ ist, verwirf den Rückgabewert und setze das Flag auf 2.

Zum Schluss entfernt der Koordinator alle zuvor hinzugefügten Chiffrate aus der Liste. Die verbliebenen Chiffrate rerandomisiert er. Das Resultat (L_2) wird zurückgegeben, wenn erfolgreich bewiesen wurde, dass die Konvertierung korrekt durchgeführt wurde.

Um zu beweisen, dass die Konvertierung korrekt vonstattengegangen ist, werden die folgenden Vergleiche berechnet:

$$\mathtt{PET}(a \cdot b,\, a)$$

und

$$\mathtt{PET}(a \cdot b,\, b)$$

Dabei sind a und b die erneut rerandomisierten und zufällig permutierten Ein- und Ausgabepaare.[13] Es wird demzufolge das Produkt zwischen Ein- und Ausgang einmal mit dem Eingang und einmal mit dem Ausgang verglichen. Das Produkt muss genau ein Element $\neq 1$ enthalten. Sonst wären sowohl Eingang als auch Ausgang $\mathtt{ENC}\,(1)$. Enthält sowohl der Ein- als auch der Ausgang einen Wert $\neq 1$, ist das Produkt aus beiden weder

[13] Es gibt zwei mögliche Permutationen: die Identität und die Vertauschung.

gleich dem Eingang noch gleich dem Ausgang. Bei einem Vergleich mit dem Ein- und dem Ausgang muss also bei genau einem Vergleich das Ergebnis „ist gleich“ sein. Ist in beiden Fällen das Ergebnis „ist gleich“, ist sowohl der Ein- als auch der Ausgang $= \texttt{ENC}(1)$. Ist in beiden Fällen das Ergebnis „ist ungleich“, so enthalten sowohl Ein- als auch Ausgang Werte $\neq 1$.

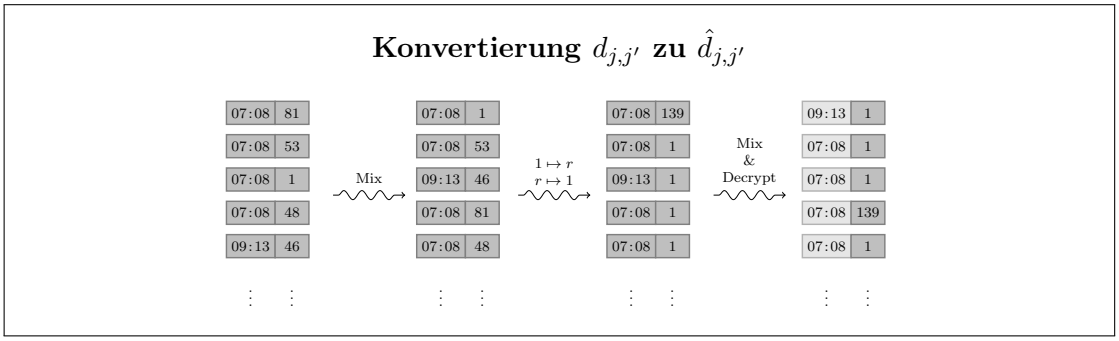

Abbildung 5.34.: Konvertiere $d_{j,j'}$ zu $\hat{d}'_{j,j}$: wenn $\texttt{DEC}(d_{j,j'}) = 1$, ersetze $d_{j,j'}$ mit $\texttt{ENC}(r)$ $(r \neq 1)$ und mit $\texttt{ENC}(1)$ sonst.

Nach der Kontrolle, dass die $\hat{d}_{j,j'}$ korrekt erstellt wurden, wird die Liste L_2 mit den Tupeln $(\hat{d}_{j,j'}, \texttt{ENC}(\mathrm{ts}_j))$ erneut durch ein `Shuffle` geführt. Im Anschluss werden die verschlüsselten Zeitstempel entschlüsselt, so dass die $\hat{d}_{j,j'}$ den entsprechenden Stimmzetteln zugeordnet werden können. Abgesehen von der Reihenfolge der $\hat{d}_{j,j'}$ je Stimmzettel ist die ursprüngliche Zusammengehörigkeit nun wieder nachvollziehbar hergestellt. Es wurde lediglich die gewünschte Vertauschung zwischen verschlüsselten Werten gleich und ungleich Eins vorgenommen. Die Reihenfolge spielt keine Rolle, da nur von Bedeutung ist, ob sich in der Liste ein verschlüsselter Wert ungleich Eins befindet.

Die neuen Invarianten sind demzufolge erfüllt. Zur Erstellung der gesuchten verschlüsselten Markierungen o_j können jetzt alle $\hat{d}_{j,j'}$ multipliziert werden:

$$o_j := \prod_{j'=\min(j+1,n)}^{n} \hat{d}_{j,j'}.$$

Das resultierende Chiffrat o_j, das die Information enthält, ob der Stimmzettel überschrieben wurde, wird an den Eintrag j, dem Stimmzettel b_j, in der Liste L_0 angehängt. Das Chiffrat o_j enthält genau dann eine Eins, wenn alle Chiffrate $\hat{d}_{j,j'}$ für $j < j' \leq n$ ebenfalls eine Eins enthielten (siehe dazu Abbildung 5.35).

Entferne alle überschriebenen Stimmzettel Die verschlüsselten Markierungen, welche Stimmzettel überschrieben wurden, sind erstellt. Aus Teilen der um die o_j erweiterten Liste L_0 wird eine neue Liste L_4 erstellt. In sie werden die verschlüsselten Stimme $\texttt{ENC}(\beta_j)$, der verschlüsselte öffentliche Schlüssel des Wählers $\texttt{ENC}(\mathrm{pk}_j)$ und die verschlüsselte Markierung o_j, ob der Stimmzettel überschrieben wurde, übernommen. Die Liste L_4 enthält von jedem Stimmzettel mit gültigem Beweis die verschlüsselten Informationen

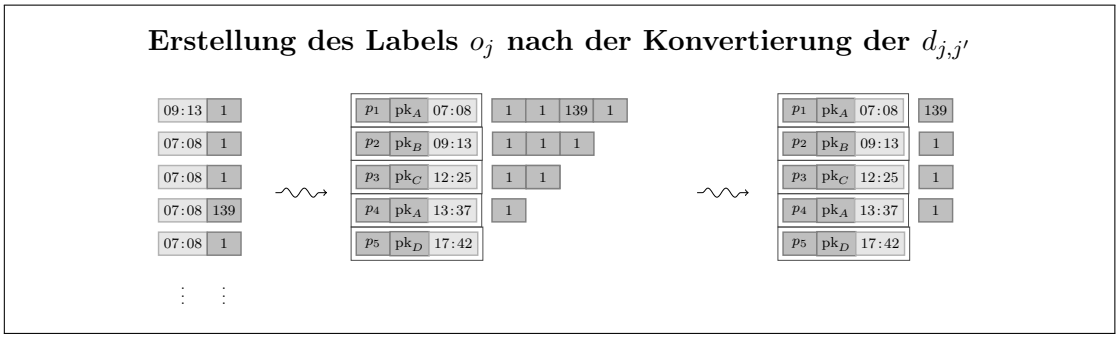

Abbildung 5.35.: Die verschlüsselten, konvertierten Quotienten werden zu ihrem Ursprungsstimmzettel zurücksortiert. Danach kann o_j als homomorphes Produkt über alle dem Stimmzettel zugeordneten $\hat{d}_{j,j}$ berechnet werden. In der Abbildung ist die Reihenfolge des besseren Verständnisses wegen so gewählt, dass sie identisch mit der ursprünglichen Reihenfolge ist. Das Label o_j enthält nun die Information, ob der Stimmzettel von einem neueren überschrieben wurde.

über die Stimme, die er repräsentiert, die Identität und ob der Stimmzettel überschrieben wurde. Die Verbindung zwischen den Elementen der Listen L_0 und L_4 wird durch ein `Shuffle` getrennt. Insbesondere ist die Information, zu welcher Zeit ein Stimmzettel abgegeben wurde, in der Liste L_4 nicht mehr enthalten. In dieser Phase kann nun entweder durch die Entschlüsselung der o_j oder durch einen PET ($\mathtt{PET}(o_j, \mathtt{ENC}(1))$) herausgefunden werden, welche der Stimmen überschrieben wurden. Ein PET muss dann verwendet werden, wenn während der Konvertierung die verschlüsselten Einsen durch Chiffrate eines festen Wertes $\neq 1$ ersetzt wurden. Die Stimmzettel, die nicht überschrieben wurden, werden in die Eingangsliste L_5 der nächsten Phase übernommen:

$$\mathrm{L}_4[j] \in \mathrm{L}_5 \Leftrightarrow \mathtt{PET}(o_j, \mathtt{ENC}(1)) = \mathit{true}$$

Streng genommen kann aus L_5 die Spalte der o_j entfernt werden. Sie enthält keine Information mehr.

Stimmzettel mit ungültigen IDs werden entfernt Im letzten Schritt wird die Gültigkeit des öffentlichen Schlüssels, der in verschlüsselter Form Teil jeden Stimmzettels ist, geprüft. Bisher können Stimmzettel enthalten sein, für die ein beliebiger öffentlicher Schlüssel verwendet wurde. Der Beweis, der bei Abgabe des Stimmzettels geprüft wird, garantiert lediglich, dass der Wähler, der den Stimmzettel abgegeben hat, eine Signatur kannte, die mit diesem Schlüssel verifizierbar ist. Daraus lässt sich schließen, dass er oder eine mit ihm in Kontakt stehende Person den geheimen Schlüssel, der zu dem öffentlichen Schlüssel gehört, kennt, nicht aber, ob der öffentliche Schlüssel einem Wähler zugeordnet ist, der in der offiziellen Wahlliste steht.

Die einfachste Form der Prüfung ist die Entschlüsselung der verwendeten öffentlichen Schlüssel. Dadurch können die Stimmen mit ungültigen Schlüsseln nachvollziehbar aussortiert werden. Das dadurch automatisch entstehende Urnenbuch, anhand dessen nachvoll-

ziehbar ist, welcher Wähler eine Stimme abgegeben hat, steht allerdings im Widerspruch zu einer Anforderung des Sicherheitsbegriffes *Erpressungsfreiheit*: Ein Wähler kann gezwungen werden, keine Stimme abzugeben. Um diesen Angriff zu verhindern, werden die öffentlichen Schlüssel in verschlüsselter Form an die Liste der autorisierten Wähler L_{Voter} angehängt. Durch einen Beweis für jeden Stimmzettel, dass der darin enthaltene verschlüsselte öffentliche Schlüssel auch in L_{Voter} enthalten ist, ohne zu nennen welcher, lässt sich die Autorisierung prüfen, ohne offenlegen zu müssen, um welchen Kandidaten es sich handelt.

Sicherheit

In diesem Abschnitt skizzieren wir die Sicherheitsanalyse des oben angebenen Protokolls. Dazu geben wir eine Reduktion auf das *Decisional Diffie-Hellman*-Problem (DDH) an: Aus einem gegebenen (erfolgreichen) Angreifer $\mathcal{A}$ auf das Experiment $\mathbf{Exp}_{ES,\mathcal{A},H}^{\textit{revoting-c-resist}}$ konstruieren wir einen Simulator $\mathcal{S}$, der in der mit nicht vernachlässigbarer Wahrscheinlichkeit entscheiden kann, ob in einem gegebenen Quadrupel (g, g^a, g^b, g^c) die Bedingung $c = a \cdot b$ in der zyklischen Gruppe $G = \langle g \rangle$ gilt. Wie in der Definition nehmen wir einen korrumpierten Wähler an. Der Beweis lässt sich durch mehrere parallele Instanzen des DDH-Problems entsprechend erweitern. Unser Ansatz entspricht dem von Juels, Catalano und Jakobsson [74].

Das DDH-Experiment $\mathbf{Exp}_G^{\mathsf{ddh}}$ zieht ein geheimes Zufallsbit d. Wenn $d = 1$, setzt das Experiment $g^c = g^{ab}$. Bei $d = 0$ wird g^c zufällig aus G gezogen. Aus dem DDH-Quadrupel leitet der Simulator $\mathcal{S}$ zwei öffentliche Schlüssel ab:

$$(g_1 := g,\ g_2 := g^a,\ y_g := g_1^{x_1} g_2^{x_2} = g^{x_1} g^{a x_2}) \text{ und}$$

$$(h_1 := g^b,\ h_2 := g^c,\ y_h := h_1^{x_1} h_2^{x_2} = g^{b x_1} g^{c x_2}).$$

Im dritten Schritt der Vorbereitungsphase (Veröffentlichung) schickt $\mathcal{S}$ den ersten an den Angreifer, während er den zweiten für die Ausführung des weiteren Protokolls verwendet.

Um einen Klartext m mit dem modifizierten Elgamal-Schema unter Verwendung des öffentlichen Schlüssels (g_1, g_2, y) zu verschlüsseln, wähle eine Zufallszahl r und berechne $(g_1^r, g_2^r, y^r m)$. Wenn in unserem äußeren Experiment $d = 1$ gilt, ist $c = ab$, und damit gilt für jedes m, dass es r und r' gibt, so dass $(g_1^{r'}, g_2^{r'}, y_g^{r'} m) = (h_1^r, h_2^r, y_h^r m)$. Die Belegung von r' kann mit $r' = br$ auch genau angegeben werden. Das bedeutet auch, dass die Chiffrate unter den beiden öffentlichen Schlüsseln die gleiche Verteilung besitzen.

Im anderen Fall, wenn $d = 0$ und damit $c \neq ab$, ist m perfekt in der Verschlüsselung versteckt. Eine Nachricht m, die mit dem ersten öffentlichen Schlüssel ver- und dem zweiten entschlüsselt wird, führt zu folgendem Ergebnis:

$$(h_1^r = g^{br} = g_1^{br}, h_2^r = g^{cr} = g_2^{(c/a)r},$$

$$y_h^r m = g^{rbx_1} g^{rcx_2} m = g^{rbx_1} g^{abrx_2} g^{(c-ab)rx_2} m = y_g^{br} g^{(c-ab)rx_2} m)$$

Das bedeutet, dass in dem Experiment $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist}$ die Wahl des Wählers und sein öffentlicher Schlüssel perfekt versteckt sind, wenn $d = 0$. In diesem Fall sind die Möglichkeiten des Angreifers auf die im idealen Experiment $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}$ reduziert. Um das zu zeigen, muss gezeigt werden, wie die Eingaben des Angreifers im Falle $d = 0$ mit den Informationen von $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}$ vorgetäuscht werden können. Dann ist offensichtlich, warum der Angreifer der realen Auszählung keinen Vorteil gegenüber der idealen Auszählung haben kann.

Im idealen Experiment $\mathbf{Exp}_{ES,\mathcal{A},H}^{revoting\text{-}c\text{-}resist\text{-}ideal}$ erhält der Angreifer sowohl die Zahl der abgegebenen Stimmzettel als auch die der in das Wahlergebnis eingegangenen. Der Inhalt der m-Elgamal-Chiffrate ist – für $d = 0$ – perfekt vor dem Angreifer verborgen. Da diese nicht von einem zufällig gewählten Gruppenelement zu unterscheiden sind, können für die vorgetäuschten Chiffrate zufällig gewählte Gruppenelemente verwendet werden. Lediglich die Menge der Elemente und deren Struktur muss übereinstimmen. Das lässt sich aus der Ausgabe des idealen Wahlprozesses (abgegebene Stimmzettel, gültige Stimmzettel) jedoch ableiten.

Der Simulator $\mathcal{S}$ erhält die Verteilung $W \in D_{n_U,n_P}$ und die Challenge (g, g^a, g^b, g^c) mit $c = ab$, wenn $d = 1$ oder $c \neq ab$, wenn $d = 0$.

Schlüsselerzeugung

Der Simulator $\mathcal{S}$ wählt den geheimen Schlüssel $\mathrm{sk}_{\mathcal{T}} := (x_1, x_2)$ zufällig. $\mathcal{S}$ berechnet daraus den öffentlichen Schlüssel $\mathrm{pk}_{\mathcal{T}}$ und die Menge C wie folgt:

$$\mathrm{pk}_{\mathcal{T}} := (g^{x_1}, g^{x_2}, h = (g_1^{x_1} g_2^{x_2})) \mod p$$

$$C = \{p_i\}_{i=1}^{n_P}$$

Registrierung

Der Simulator $\mathcal{S}$ simuliert den Registrierungsprozess: $\mathcal{S}$ generiert die Schlüsselpaare für die Wähler: $\{(\mathrm{pk}_i, \mathrm{sk}_i)\}_{i=1}^{n_E}$

Veröffentlichung

$\mathcal{S}$ veröffentlicht den öffentlichen Schlüssel der Auszählung $\mathrm{pk}_{\mathcal{T}}$, die Kandidatenliste C und $L_{\text{Voter}} = \{\text{„Wählername}_i\text{“}, \ \mathtt{ENC}\,(\mathrm{pk}_i)\}_{i=1}^{n_E}$.

Korruption

Der Angreifer $\mathcal{A}$ wählt $n_{\mathcal{A}}$ Wähler, die er korrumpiert. Sei V die Menge der korrumpierten Wähler. $\mathcal{A}$ wählt außerdem einen Wähler j_c, den er versucht zu erpressen. Für die korrumpierten und den erpressten Wähler entscheidet sich $\mathcal{A}$ für je einen Kandidaten β, für den der Angreifer im Namen des korrumpierten Wählers eine Stimme abgibt und für den der erpresste Wähler wählen soll. Der Simulator bricht ab, wenn die Wahl von $\mathcal{A}$ ungültig ist.

Zufallsbit $\bar{d}$

Das Bit $\bar{d}$ wird zufällig gewählt: $\bar{d} \in_U \{0,\ 1\}$

Simulation der ehrlichen Wähler

Für jeden ehrlichen Wähler[14] erstellt $\mathcal{S}$ die Stimmzettel mit den dazugehörigen Beweisen (NIZKPoK).

$$A_0 := \{(\mathtt{ENC}(\beta_j), \mathtt{ENC}(\mathrm{pk}_i), \mathrm{ts}_j),\ NIZKPoK\}$$
$$= \{b_i := ((h_1^{r_i}, h_2^{r_i}, h_1^{r_i x_1} h_2^{r_i x_2} p_j), (h_1^{k_i}, h_2^{k_i}, h_1^{k_i x_1} h_2^{k_i x_2} pk_i), ts), NIZKPoK\}_{i=1}^{n}$$

Dabei werden alle r_i und k_i zufällig und gleichverteilt aus Z_q gewählt. Die Kandidaten, für die der Simulator Stimmzettel erzeugt, werden anhand der Verteilung W gewählt. Für die Erzeugung der Beweise nutzt der Simulator die geheimen Schlüssel der Wähler, die er im Schritt *Schlüsselerzeugung* erzeugt hat.

Wahlvorgang des Angreifers

Der Angreifer $\mathcal{A}$ erstellt nun die Stimmzettel der Wähler $v \in V$ und den Wähler j_c auf die gleiche Weise wie der Simulator. Wir nennen die Liste dieser Stimmzettel B_0. $\mathcal{A}$ sendet B_0 an $\mathcal{BB}$.

Simulation der Auswertung

Der Simulator $\mathcal{S}$ simuliert eine ehrliche Auszählung, wobei er den geheimen Schlüssel $\mathrm{sk}_{\mathcal{T}}$ aus der Phase der *Schlüsselerzeugung* nutzt. Modifikationen des Angreifers in dem Prozess können ignoriert werden, da die Korrektheit jeden Schritts verifiziert werden kann.

Prüfen der Beweise

Die Beweise aller Stimmzettel aus A_0 und B_0 werden überprüft. Alle Stimmzettel mit gültigen Beweisen werden in die Liste E_1 übernommen.

Erstellen der $d_{j,j'}$

$\mathcal{S}$ erstellt die $d_{j,j'}$ entsprechend dem Schema durch das Berechnen der dafür notwendigen EPETs. Die Beweise für die korrekte Erstellung erzeugt $\mathcal{S}$ auf ehrliche Art und Weise.

Konvertierung der $d_{j,j'}$

$\mathcal{S}$ führt die Konvertierung korrekt aus und erstellt die dazugehörigen Beweise ebenfalls ehrlich. Während dieses Prozesses nutzt er den geheimen Schlüssel $\mathrm{sk}_{\mathcal{T}}$.

Erstellung des Labels o_j

$\mathcal{S}$ fasst die $\hat{d}_{j,j'}$ entsprechend dem Protokoll zu o_j zusammen.

Erstellen der endgültigen Liste der Stimmzettel

Zur Erstellung der Liste der endgültigen Stimmzettel folgt $\mathcal{S}$ dem Protokoll: $E_2 := \{(\mathtt{ENC}(\beta_j), \mathtt{ENC}(\mathrm{pk}_i)) : \mathtt{PET}(o_j, \mathtt{ENC}(1), =)1\}$

[14] nicht korrumpiert oder erpresst

Überprüfen der Wahlberechtigung
$\mathcal{S}$ erstellt eine Liste aller gültigen öffentlichen Wählerschlüssel K_0. Nach einem Mix von K_0 zu K_1 werden die verschlüsselten öffentlichen Schlüssel aus der Liste E_2 mit den Einträgen aus K_1 mit Hilfe eines PETs verglichen. Zur Entschlüsselung der PETs verwendet der Simulator den geheimen Schlüssel $\mathrm{sk}_{\mathcal{T}}$. Für jeden Eintrag j aus der Liste E_2, zu dem ein passender Eintrag in K_1 gefunden wird, wird der erste Teil in eine neue Liste E_3 kopiert. E_3 enthält die verschlüsselten gültigen Stimmen.

Entschlüsselung des Wahlergebnisses
Der Simulator $\mathcal{S}$ entschlüsselt die gültigen Stimmen (E_4) mit Hilfe des geheimen Schlüssels $\mathrm{sk}_{\mathcal{T}}$.

Vermutung des Angreifers über die Belegung des Bits $\bar{d}$
Der Angreifer $\mathcal{A}$ rät die Belegung von $\bar{d}$ und gibt seine Vermutung $\bar{d}'$ aus. Der Simulator $\mathcal{S}$ gibt d' aus, wobei er d' als 1 wählt, wenn $\bar{d}' = \bar{d}$ und 0 sonst.

Da der Simulator das Protokoll wie spezifiziert ausführt, ist die Simulation bei $d = 1$ für den Angreifer ununterscheidbar von einem realen Protokollablauf.

Sei $\mathcal{V}$ die Sicht des Angreifers $\mathcal{A}$. Dann ist

$$\Pr[S = 1 \mid d = 1] = Pr[\mathbf{Exp}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(\mathcal{V}) = 1] = \mathbf{Succ}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(\mathcal{V}).$$

Auf der anderen Seite hat der Angreifer keinen Vorteil in $\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V})$, wenn $d = 0$. Das bedeutet, dass

$$\Pr[S = 1 \mid d = 0] = Pr[\mathbf{Exp}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V}) = 1] = \mathbf{Succ}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V}).$$

Daraus folgt, dass

$$\begin{aligned}\mathbf{Adv}^{\mathsf{ddh}}_{\mathcal{S}} &= \Pr[S = 1 \mid d = 1] - Pr[S = 1 \mid d = 0] \\ &= \mathbf{Succ}^{\textit{revoting-c-resist}}_{ES,\mathcal{A},H}(\mathcal{V}) - \mathbf{Succ}^{\textit{revoting-c-resist-ideal}}_{ES,\mathcal{A},H}(\mathcal{V}) \\ &= \mathbf{Adv}^{\textit{revoting}}_{\mathcal{A}}.\end{aligned}$$

□

Fazit und Ausblick

Unser Verfahren zeigt, dass eine Aussortierung der überschriebenen Stimmen möglich ist, ohne dass dieser Vorgang oder dessen Verifikation die Abstreitbarkeit des Wählerverhaltens gefährdet. Der Aufwand unseres Verfahrens ist quadratisch in der Anzahl der abgegebenen Stimmzettel; das ergibt sich dadurch, dass der Wähler seinem Stimmzettel keine Informationen zu seiner letzten Stimme anhängen muss. Für die Abstreitbarkeit ist

es ebenfalls wichtig, dass die Zahl (und Art) der auszuführenden Schritte nur von der Zahl der abgegebenen Stimmen abhängt. Verfahren, deren Verlauf vom Verhalten der Wähler abhängig sind, geben dadurch gegebenenfalls Informationen über das Wählerverhalten preis, was die Abstreitbarkeit gefährden kann.

6. Zusammenfassung und Ausblick

An den drei Beispielen haben wir gezeigt, dass mit Hilfe vertrauenswürdiger Hardwarebausteine effiziente kryptographische Protokolle entworfen werden können, die gleichzeitig höhere Sicherheitsziele erreichen als bisher bekannte Lösungen. Insbesondere mit dem Entwurf und Bau einer modularisierten Wahlmaschine, deren mit Rechenleistung oder Speicher versehenen Bausteine die Wahlentscheidung des Wählers nicht in Erfahrung bringen können, ist es uns gelungen, einen weiteren Schritt hin zu vertrauenswürdigen Wahlmaschinen zu gehen. Dennoch müssen – nicht nur bei kryptographischen Wahlverfahren – vor ihrer Anwendung weitere Ziele erfüllt werden.

Die Möglichkeiten, effiziente Protokolle mit Hilfe von vertrauenswürdigen Hardwarebausteinen zu entwerfen, sind vielfältig. In Abhängigkeit von der Anwendung gilt es, die Hardware geeignet zu wählen. In vielen Fällen ist es ausreichend, wenn die Hardware einfache Funktionen erfüllt. Durch leichte Modifikationen des Funktionsumfangs können jedoch oft ganz andere, zusätzliche Sicherheitseigenschaften erreicht werden.

6.1. Nächste Schritte

Die in dieser Arbeit beschriebenen Protokolle erreichen Ziele, die bisher von keinem Protokoll erfüllt wurden, doch noch höhere Ziele können erreicht werden. Im Folgenden beschreiben wir einige Schritte, die wir im Anschluss an die in dieser Arbeit vorgestellten Ergebnisse gehen wollen.

6.1.1. Wahlverfahren

Das Bundesverfassungsgericht hat am 03. März 2009 [28] entschieden, dass „jeder Bürger [...] die zentralen Schritte der Wahl ohne besondere technische Vorkenntnisse zuverlässig nachvollziehen und verstehen können“ muss. Er muss ohne spezielle Vorkenntnisse – so verstehen wir das Urteil – beurteilen können, ob seine Stimme richtig gezählt wurde. Davon, dass er dazu verstehen muss, wie das vonstatten geht, ist auszugehen. Nach unserem heutigen Verständnis schließt das alle kryptographischen Lösungen aus, die auf mathematischen Konstruktionen aufbauen.

Dennoch sehen wir die Forschung auf diesem Gebiet aus unterschiedlichen Gründen als sinnvoll und notwendig an. Zunächst bieten uns die Forschungsergebnisse einen tieferen Einblick in die zugrunde liegende Problemstellung. Sie ermöglicht auch auszuloten, was

eingesetzte Verfahren grundsätzlich leisten können und demzufolge auch leisten müssen, wenn sie zum Einsatz kommen sollen. Die zweite Motivation, die uns antreibt, ist, dass sich die Umstände schnell ändern können; damit meinen wir nicht, dass das Urteil des Verfassungsgerichts aufgehoben wird. Jedoch könnte die Briefwahl, die die Anforderungen an die Urnenwahl nicht erfüllt und der *Allgemeinheit* wegen akzeptiert ist, durch eine Onlinewahl ergänzt werden. Diese muss unseres Erachtens jedoch höhere Anforderungen erfüllen als die bisher – beispielsweise in Norwegen [10] – eingesetzten Verfahren.

Oblivious Bingo Voting

Durch die Modularisierung, die Realisierung des *Oblivious Transfer* als vertrauenswürdigen Hardwarebaustein und die Gestaltung der Vorberechnung als Mehrparteienberechnung ist es uns gelungen, die Annahmen, auf denen das Vertrauen beruht, sehr klar zu formulieren. Unserer Einschätzung nach wird in der Vorbereitungsphase das Vertrauen einem Dritten gegenüber weiterhin notwendig bleiben; nicht zuletzt müssen die Wählerlisten korrekt sein, um die Wahlgrundsätze der Gleichheit und Allgemeinheit erfüllen zu können. Das Vertrauen kann jedoch auf einer *beliebigen ehrlichen Person aus der Gruppe der Wahlleitung* basieren. Um welches Mitglied es sich dabei handelt, ist irrelevant. Die nächsten Schritte müssen sein, Schutz auch gegen aktive Angreifer zu bieten, eine gute Realisierung des Zufallszahlengenerators zu finden, dem der Wähler vertrauen kann, und dem Wähler auch auf einer nicht modularisierten Wahlmaschine, auf der beliebige Software aufgespielt sein darf, nachvollziehbare Garantien geben zu können.

Bingo Voting für Onlinewahlverfahren

Bingo Voting bietet nicht nur für Präsenzwahlen eine gute Grundlage. Basierend auf sicherer Hardware, wie sie heute bereits verfügbar ist, kann das Konzept für ein Onlinewahlverfahren erweitert werden. Der PC des Wählers wird dabei nur für die bequeme Bedienung genutzt, ohne dass er die Stimme des Wählers in Erfahrung bringen oder manipulieren kann.

Für die Realisierung müssen insbesondere zwei Aspekte beachtet werden: Durch das Fehlen der Wahlkabine muss – wie es bei der Briefwahl auch notwendig wäre – dem Wähler die Möglichkeit gegeben werden, einer möglichen Erpressung auszuweichen; zudem steht dem Wähler Spezialhardware nur sehr eingeschränkt zur Verfügung.

Für den zuerst genannten Aspekt haben wir bereits mit *Revoting* eine Technik diskutiert, die auch in diesem Fall die Position des Wählers gegebenenfalls stärken würde. Der zweite Aspekt – der Mangel an Spezialhardware – stellt hier die größere Herausforderung dar: Einerseits muss der Wähler überzeugt werden, dass die frische Zufallszahl nicht mit einer Füllstimme vertauscht worden sein kann, und andererseits muss das Oblivious Transfer so realisiert werden, dass es für jeden Wähler, der seine Stimme online abgeben möchte, verfügbar ist.

Das Oblivious Transfer, das bei der in dieser Arbeit vorgestellten Variante von *oblivious Bingo Voting* physisch realisiert wurde, kann als Zweiparteienberechnung zwischen Server und einer vertrauenswürdigen Hardware – ähnlich einer Girocard – ausgeführt werden. Idealerweise würde der Baustein *Oblivious Transfer* so erweitert, dass – die Ein- und Ausgabe kennend – nicht abgeleitet werden kann, welche der Eingaben der Ausgabe zugrunde liegt. Dies kann, wenn es sich bei der Eingabe um rerandomisierbare Chiffrate handelt, beispielsweise durch eine blinde Rerandomisierung der Eingaben geschehen.

Gegeben eine Smartcard mit einem Display[1] als vertrauenswürdige Hardware ist es zudem möglich, *Code-Voting* [32] sicher zu realisieren. Beim Code-Voting gibt der Wähler seine Wahl nicht direkt in das System (beispielsweise seinem PC) ein, sondern übergibt einen zufälligen Wert, der der Partei zugeordnet ist. Für jeden Wähler muss diese Zuordnung neu und zufällig geschehen, damit aus dem Code keine Rückschlüsse auf die getroffene Wahl stattfinden können. Die einfachste Variante, Code-Voting zu implementieren, ist, dass eine zentrale Stelle für jeden Wähler eine Wahlkarte erstellt, auf der sie jeder Partei einen zufälligen Wert zuordnet. Aus dem abgegebenen Code kann sie dann die gewählte Partei ableiten. Dabei bricht sie aber gegebenenfalls das Wahlgeheimnis. Die Zuordnung der Codes dürfen nur dem Wähler (und seiner sicheren Hardware) bekannt sein; insbesondere darf es ihm nicht möglich sein, diese Information nachvollziehbar korrekt weiterzugeben. Durch die Smartcard mit Display können wir Erstellung und Auswertung des Codes geschickt trennen.

6.1.2. Bezahlprotokolle und *Point-of-Sale*-Terminals

Durch die Girocard kann die Kommunikation zwischen Bank, Geldkassette und Karte abgesichert werden. Ein zentraler Punkt in dem Protokoll, die Autorisierung der Transaktion, kann aber nicht von der Karte übernommen werden. Die Autorisierung ist in unserem Vorschlag mit der Authentifizierung des Nutzers gegenüber der Karte verknüpft. In unserem Protokoll geschieht das durch die Eingabe einer PIN und einer von der Bank gewählten Zufallszahl. Wir nehmen dazu – ähnlich der aktuellen Realisierung – vertrauenswürdige PIN-Pads an den Automaten an.

Das Protokoll ist so entworfen, dass es dem Angreifer, obwohl er die Kommunikation zwischen den Komponenten kontrolliert, nicht möglich ist, aus den ausgetauschten Nachrichten die PIN zu extrahieren. Gegen eine mechanische Manipulation des PIN-Pads oder eine optische Überwachung während der Eingabe können die kryptographischen Protokolle jedoch keinen Schutz bieten.

Durch eine Erweiterung der Girocard, einem Display, gäbe es einen neuen Freiheitsgrad, der für die Kommunikation mit dem Kunden genutzt werden kann. Herauszufinden, welche Angriffe über die bisher abgedeckten hinaus mit dieser Modifikation verhindert werden können, ist Ziel der weiteren Forschung.

[1] mehrstelliges 7-Segment-Display oder e-Paper

6.1.3. Blurry-Box automatisch anwendbar gestalten

Wir haben mit Blurry-Box gezeigt, dass ein Softwareschutzverfahren, dessen Methoden öffentlich bekannt sind, den Anforderungen der Industrie genügen kann. Der Hackers-Contest hat eindrucksvoll gezeigt, dass der Schutz den Erwartungen entspricht.

Um den Schutz einfach und kostengünstig einsetzen zu können, ist eine Automatisierung der Anwendung des Schutzes auf Softwareprodukte notwendig. Welche Strukturen in Software sich dafür eignen, gilt es in der weiteren Forschung herauszufinden.

6.2. Fazit

Sichere Protokolle in digitalisierten Prozessen sind die Grundlage für das Vertrauen der Nutzer. Die Sicherheit muss für sie nachvollziehbar sein. Das kann grundsätzlich auf unterschiedliche Arten geschehen. Die Teilnahme an den Protokollen im Sinne einer Mehrparteienberechnung ist eine Möglichkeit. In dieser Arbeit haben wir gezeigt, dass die nachvollziehbare Verwendung von Hardware, der der Nutzer vertrauen kann, als zentrales Element im Ablauf eines Protokolls ebenfalls funktioniert. Die Hardware kann dabei unterschiedliche Rollen einnehmen. Im Szenario des sicheren Geldabhebens übernimmt die Girocard einerseits die Rolle einer Instanz, der sowohl der Kunde als auch die Bank vertraut; andererseits nimmt die Karte dem Kunden Berechnungen ab, die für eine sichere Protokollausführung notwendig sind und von einem Menschen (in dieser Geschwindigkeit) nicht geleistet werden können. Dass Kunden auch Vertrauen in eine komplexe Lösung haben können, zeigt das Blurry-Box-Verfahren. Im Szenario der Wahlmaschine muss die Konstruktion der vertrauenswürdigen Hardware jedoch für den Wähler nachvollziehbar sein. Das Physical Oblivious Transfer garantiert auf einfache Art und Weise die sichere Ausführung eines zentralen Schrittes im Protokoll.

Die Verwendung einfacher Hardwarebausteine kann dazu beitragen, dass der Nutzer weniger oder kein Vertrauen mehr in die Software haben muss.

Eigene Veröffentlichungen

[1] Dirk Achenbach, Carmen Kempka, Bernhard Löwe und Jörn Müller-Quade. „Improved coercion-resistant electronic elections through deniable re-voting“. In: *USENIX Journal of Election Technology and Systems (JETS)* (2015), S. 26–45.

[2] Dirk Achenbach, Bernhard Löwe, Jörn Müller-Quade und Jochen Rill. „Oblivious Voting: Hiding Votes from the Voting Machine in Bingo Voting“. In: *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016*. 2016, S. 85–96. DOI: 10.5220/0005964300850096. URL: http://dx.doi.org/10.5220/0005964300850096.

[3] Dirk Achenbach, Anne Borcherding, Bernhard Löwe, Jörn Müller-Quade und Jochen Rill. „Towards Realising Oblivious Voting“. In: *E-Business and Telecommunications. 13th International Joint Conference, ICETE 2016, Lisbon, Portugal, July 26-28, 2016, Revised Selected Papers*. 2017. ISBN: 978-3-319-67875-7. DOI: 10.1007/978-3-319-67876-4.

[4] Brandon Broadnax, Matthias Huber, Bernhard Löwe, Jörn Müller-Quade und Patrik Scheidecker. „Towards Efficient Software Protection Obeying Kerckhoffs's Principle using Tamper-proof Hardware“. In: *Proceedings of the 15th International Joint Conference on e-Business and Telecommunications, ICETE 2018 - Volume 2: SECRYPT, Porto, Portugal, July 26-28, 2018*. 2018, S. 719–724.

Andere Publikationen

[5] Masayuki Abe und Fumitaka Hoshino. „Remarks on Mix-Network Based on Permutation Networks“. In: *Public Key Cryptography*. Hrsg. von Kwangjo Kim. Bd. 1992. Lecture Notes in Computer Science. Springer, 2001, S. 317–324. ISBN: 3-540-41658-7.

[6] Claudia Z. Acemyan, Philip Kortum, Michael D. Byrne und Dan S. Wallach. „From Error to Error: Why Voters Could not Cast a Ballot and Verify Their Vote With Helios, Prêt à Voter, and Scantegrity II“. In: *USENIX Journal of Election Technology and Systems (JETS)* 2 (2015), S. 1–25. ISSN: 2328-2797. URL: https://www.usenix.org/jets/issues/0302/acemyan.

[7] Ben Adida. „Helios: Web-based Open-audit Voting“. In: *Proceedings of the 17th Conference on Security Symposium*. Bd. 17. SS'08. San Jose, CA: USENIX Association, 2008, S. 335–348. URL: http://dl.acm.org/citation.cfm?id=1496711.1496734.

[8] Ben Adida, Olivier De Marneffe, Olivier Pereira und Jean-Jacques Quisquater. „Electing a University President Using Open-audit Voting: Analysis of Real-world Use of Helios“. In: *Proceedings of the 2009 Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*. EVT/WOTE'09. Montreal, Canada: USENIX Association, 2009, S. 10–10. URL: http://dl.acm.org/citation.cfm?id=1855491.1855501.

[9] Ross Anderson. „Why cryptosystems fail“. In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM. 1993, S. 215–227.

[10] Arne Ansper, Sven Heiberg, Helger Lipmaa, Tom André Øverland und Filip van Laenen. „Security and Trust for the Norwegian E-Voting Pilot Project *E-valg 2011*“. In: *NordSec*. 2009, S. 207–222.

[11] Roberto Araújo, Sébastien Foulle und Jacques Traoré. „A Practical and Secure Coercion-Resistant Scheme for Internet Voting“. English. In: *Towards Trustworthy Elections*. Hrsg. von David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski und Ben Adida. Bd. 6000. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, S. 330–342. ISBN: 978-3-642-12979-7. DOI: 10.1007/978-3-642-12980-3_20. URL: http://dx.doi.org/10.1007/978-3-642-12980-3_20.

[12] Md Abdul Based, Joe-Kai Tsay und Stig Fr Mjølsnes. „PEVS: A Secure Electronic Voting Scheme Using Polling Booths“. In: *Data and Knowledge Engineering*. Springer, 2012, S. 189–205.

[13] *Basissatz von Sicherheitsanforderungen an Online-Wahlprodukte*. 2008. URL: https://www.commoncriteriaportal.org/files/ppfiles/pp0037b.pdf.

[14] Georg T. Becker, Francesco Regazzoni, Christof Paar und Wayne P. Burleson. „Stealthy Dopant-Level Hardware Trojans“. In: *Cryptographic Hardware and Embedded Systems - CHES 2013: 15th International Workshop, Santa Barbara, CA, USA, August 20-23, 2013. Proceedings*. Hrsg. von Guido Bertoni und Jean-Sébastien Coron. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, S. 197–214. ISBN: 978-3-642-40349-1. DOI: 10.1007/978-3-642-40349-1_12. URL: https://doi.org/10.1007/978-3-642-40349-1_12.

[15] Polizeipräsident in Berlin. *Jackpotting. Polizeimeldung vom 20.10.2015*. 20. Okt. 2015. URL: https://www.berlin.de/polizei/polizeimeldungen/pressemitteilung.386807.php (besucht am 25. 06. 2017).

[16] Olivier Billet, Henri Gilbert und Charaf Ech-Chatbi. „Cryptanalysis of a white box AES implementation“. In: *Selected Areas in Cryptography*. Springer. 2005, S. 227–240.

[17] Nir Bitansky, Ran Canetti, Shafi Goldwasser Shai, Halevi, Yael Tauman Kalai und Guy N. Rothblum. „Program obfuscation with leaky hardware“. In: *Advances in Cryptology–ASIACRYPT 2011*. Springer, 2011, S. 722–739.

[18] Olivier Blazy, Georg Fuchsbauer, David Pointcheval und Damien Vergnaud. „Signatures on randomizable ciphertexts“. In: *Public Key Cryptography–PKC 2011*. Springer, 2011, S. 403–422.

[19] Manuel Blum. „Coin Flipping by Telephone a Protocol for Solving Impossible Problems“. In: *SIGACT News* 15.1 (Jan. 1983), S. 23–27. ISSN: 0163-5700. DOI: 10.1145/1008908.1008911. URL: http://doi.acm.org/10.1145/1008908.1008911.

[20] Jens-Matthias Bohli, Jörn Müller-Quade und Stefan Röhrich. „Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator“. English. In: *E-Voting and Identity*. Hrsg. von Ammar Alkassar und Melanie Volkamer. Bd. 4896. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, S. 111–124. ISBN: 978-3-540-77492-1. DOI: 10.1007/978-3-540-77493-8_10. URL: http://dx.doi.org/10.1007/978-3-540-77493-8_10.

[21] Jens-Matthias Bohli, Jörn Müller-Quade und Stefan Röhrich. „Bingo Voting: Secure and Coercion-Free Voting Using a Trusted Random Number Generator“. In: *VOTE-ID 2007*. Hrsg. von A. Alkassar und M. Volkamer. Bd. 4896. Lecture Notes in Computer Science. Springer-Verlag, 2007, S. 111–124.

[22] Mike Bond, Marios O Choudary, Steven J Murdoch, Sergei Skorobogatov und Ross Anderson. „Be prepared: The EMV preplay attack“. In: *IEEE Security & Privacy* 13.2 (2015), S. 56–64.

[23] Mike Bond, Omar Choudary, Steven J Murdoch, Sergei Skorobogatov und Ross Anderson. „Chip and Skim: cloning EMV cards with the pre-play attack“. In: *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE. 2014, S. 49–64.

[24] Dan Boneh, David Mazieres und Raluca Ada Popa. „Remote oblivious storage: Making oblivious RAM practical“. In: *MIT – CSAIL Technical Reports* (2011).

[25] Dan Boneh und Mark Zhandry. „Multiparty key exchange, efficient traitor tracing, and more from indistinguishability obfuscation“. In: *Advances in Cryptology–CRYPTO 2014*. Springer, 2014, S. 480–499.

[26] Anne Broadbent, Gus Gutoski und Douglas Stebila. „Quantum one-time programs“. In: *Advances in Cryptology–CRYPTO 2013*. Springer, 2013, S. 344–360.

[27] Bundesverfassungsgericht: Andreas Voßkuhle, Gertrude Lübbe-Wolff, Michael Gerhardt, Herbert Landau, Peter Huber, Monika Hermanns, Peter Müller, Sibylle Kessal-Wulf. *Urteil des Zweiten Senats vom 25. Juli 2012. 2 BvF 3/11*. 25. Juli 2012. URL: `http://www.bverfg.de/e/fs20120725_2bvf000311.html`.

[28] Bundesverfassungsgericht: Andreas Voßkuhle, Siegfried Broß, Lerke Osterloh, Udo Di Fabio, Rudolf Mellinghoff, Gertrude Lübbe-Wolff, Michael Gerhardt, Herbert Landau. *Urteil des Zweiten Senats vom 03. März 2009. 2 BvC 3/07 - Rn. (1-163)*. 3. März 2009. URL: `http://www.bverfg.de/entscheidungen/cs20090303_2bvc000307.html`.

[29] *Bundeswahlgesetz*. 3. Mai 2013. URL: `https://www.bundestag.de/blob/189210/.../bwahlg_pdf-data.pdf`.

[30] Jan Camenisch und Markus Stadler. „Efficient group signature schemes for large groups“. English. In: *Advances in Cryptology — CRYPTO '97*. Hrsg. von Jr. Kaliski BurtonS. Bd. 1294. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 1997, S. 410–424. ISBN: 978-3-540-63384-6. DOI: `10.1007/BFb0052252`. URL: `http://dx.doi.org/10.1007/BFb0052252`.

[31] Richard T Carback u. a. *Scantegrity II Municipal Election at Takoma Park: The First E2E Binding Governmental Election with Ballot Privacy*. Proceedings of the 19th USENIX Security Symposium. 2010.

[32] David Chaum. „SureVote: Technical Overview“. In: WOTE 2001. 2001.

[33] David Chaum. „Surevote: technical overview“. In: *Proceedings of the workshop on trustworthy elections (WOTE'01)*. 2001.

[34] David Chaum und Torben P. Pedersen. „Wallet Databases with Observers“. In: *Proceedings of the 12th Annual International Cryptology Conference on Advances in Cryptology*. CRYPTO '92. London, UK, UK: Springer-Verlag, 1993, S. 89–105. ISBN: 3-540-57340-2. URL: `http://dl.acm.org/citation.cfm?id=646757.705670`.

[35] David Chaum, Peter Y. A. Ryan und Steve Schneider. „A Practical Voter-Verifiable Election Scheme“. In: *Computer Security – ESORICS 2005: 10th European Symposium on Research in Computer Security, Milan, Italy, September 12-14, 2005. Proceedings*. Hrsg. von Sabrina de Capitani di Vimercati, Paul Syverson und Dieter Gollmann. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, S. 118–139. ISBN: 978-3-540-31981-8. DOI: `10.1007/11555827_8`. URL: `http://dx.doi.org/10.1007/11555827_8`.

[36] David Chaum u. a. „Scantegrity II: End-to-end verifiability by voters of optical scan elections through confirmation codes“. In: *Information Forensics and Security, IEEE Transactions on* 4.4 (2009), S. 611–627.

[37] David Chaum u. a. „Scantegrity II municipal election at Takoma Park: the first E2E binding governmental election with ballot privacy“. In: *19th USENIX Security Symposium*. 2010.

[38] Stanley Chow, Philip Eisen, Harold Johnson und Paul C. Van Oorschot. „White-box cryptography and an AES implementation“. In: *Selected Areas in Cryptography*. Springer. 2003, S. 250–270.

[39] Jeremy Clark und Urs Hengartner. *Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance*. Cryptology ePrint Archive, Report 2011/166. `http://eprint.iacr.org/`. 2011.

[40] Jeremy Clark und Urs Hengartner. „Selections: Internet Voting with Over-the-Shoulder Coercion-Resistance“. In: *Financial Cryptography*. Hrsg. von George Danezis. Bd. 7035. Lecture Notes in Computer Science. Springer, 2011, S. 47–61. ISBN: 978-3-642-27575-3.

[41] Véronique Cortier, Georg Fuchsbauer und David Galindo. *BeleniosRF: A Strongly Receipt-Free Electronic Voting Scheme*. Cryptology ePrint Archive, Report 2015/629. `http://eprint.iacr.org/2015/629`. 2015.

[42] Ronald Cramer, Ivan Damgård und Jesper B. Nielsen. „Multiparty computation from threshold homomorphic encryption“. In: *Advances in cryptology—EUROCRYPT 2001* (2001), S. 280–300.

[43] Joan Daemen und Vincent Rijmen. „AES proposal: Rijndael“. In: (1999).

[44] Ivan Damgård und Mads Jurik. „A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System“. In: *Proceedings of the 4th International Workshop on Practice and Theory in Public Key Cryptosystems* (2001).

[45] Ivan Damgård, Sigurd Meldgaard und Jesper Buus Nielsen. „Perfectly secure oblivious RAM without random oracles“. In: *Theory of Cryptography*. Springer, 2011, S. 144–163.

[46] Nico Döttling, Thilo Mie, Jörn Müller-Quade und Tobias Nilges. „Basing Obfuscation on Simple Tamper-Proof Hardware Assumptions.“ In: *IACR Cryptology ePrint Archive* 2011 (2011), S. 675.

[47] Nico Döttling, Thilo Mie, Jörn Müller-Quade und Tobias Nilges. „Implementing resettable UC-Functionalities with untrusted tamper-proof hardware-tokens“. In: *Theory of Cryptography*. Springer, 2013, S. 642–661.

[48] Georg Ismar (dpa). *Wahlcomputer in Venezuela: Maduro kontert Betrugsvorwürfe*. 3. Aug. 2017. URL: `https://heise.de/-3791237` (besucht am 08. 08. 2017).

[49] Morris Dworkin. *Recommendation for block cipher modes of operation. methods and techniques*. Techn. Ber. DTIC Document, 2001.

[50] Taher Elgamal. „A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms“. In: *Advances in Cryptology Proceedings of CRYPTO 84*. Hrsg. von David Chaum George Robert Blakley. Berlin, Heidelberg: Springer-Verlag, 1985, S. 10–18. ISBN: 978-3-540-15658-1.

[51] Taher 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.

[52] EMVCo. *Specifications*. `http://www.emvco.com/specifications.aspx`. [Online; accessed 06-June-2017]. 2017.

[53] Aleksander Essex, Jeremy Clark und Urs Hengartner. „Cobra: Toward Concurrent Ballot Authorization for Internet Voting“. In: *Proceedings of the 2012 International Conference on Electronic Voting Technology/Workshop on Trustworthy Elections*. EVT/WOTE'12. Bellevue, WA: USENIX Association, 2012, S. 3–3. URL: http://dl.acm.org/citation.cfm?id=2372353.2372356.

[54] Norman E Fenton und Martin Neil. „A critique of software defect prediction models“. In: *IEEE Transactions on software engineering* 25.5 (1999), S. 675–689.

[55] Kevin Fisher, Richard Carback und Alan T Sherman. „Punchscan: Introduction and system definition of a high-integrity election system“. In: *Proceedings of Workshop on Trustworthy Elections*. 2006.

[56] Sanjam Garg, Craig Gentry, Shai Halevi, Mariana Raykova, Amit Sahai und Brent Waters. „Candidate indistinguishability obfuscation and functional encryption for all circuits“. In: *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*. IEEE. 2013, S. 40–49.

[57] Ida Sofie Gebhardt Stenerud und Christian Bull. *When Reality Comes Knocking - Norwegian Experiences with Verifiable Electronic Voting*. EVOTE2012.

[58] Kristian Gjøsteen. „Analysis of an internet voting protocol.“ In: *IACR Cryptology ePrint Archive* 2010 (2010), S. 380.

[59] Oded Goldreich. *Foundations Of Cryptography – Basic Tools*. Cambridge University Press, 2001.

[60] Oded Goldreich. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.

[61] Oded Goldreich. „Towards a theory of software protection and simulation by oblivious RAMs“. In: *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM. 1987, S. 182–194.

[62] Oded Goldreich und Rafail Ostrovsky. „Software protection and simulation on oblivious RAMs“. In: *Journal of the ACM (JACM)* 43.3 (1996), S. 431–473.

[63] Shafi Goldwasser, Silvio Micali und Ronald L. Rivest. „A Digital Signature Scheme Secure Against Adaptive Chosen-message Attacks“. In: *SIAM J. Comput.* 17.2 (Apr. 1988), S. 281–308. ISSN: 0097-5397. DOI: 10.1137/0217017. URL: http://dx.doi.org/10.1137/0217017.

[64] Shafi Goldwasser und Guy N. Rothblum. „On best-possible obfuscation“. In: *Theory of Cryptography*. Springer, 2007, S. 194–213.

[65] Shafi Goldwasser, Yael T., Kalai und Guy N. Rothblum. „One-time programs“. In: *Advances in Cryptology – CRYPTO 2008*. Bd. 5157. Springer, 2008, S. 39–56.

[66] Philippe Golle, Markus Jakobsson, Ari Juels und Paul Syverson. „Universal Re-encryption for Mixnets“. In: *Topics in Cryptology – CT-RSA 2004*. Hrsg. von Tatsuaki Okamoto. Bd. 2964. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, S. 163–178. ISBN: 978-3-540-20996-6. DOI: 10.1007/978-3-540-24660-2_14. URL: http://dx.doi.org/10.1007/978-3-540-24660-2_14.

[67] Michael T. Goodrich, Michael Mitzenmacher, Olga Ohrimenko und Roberto Tamassia. „Oblivious RAM simulation with efficient worst-case access overhead“. In: *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*. ACM. 2011, S. 95–100.

[68] Jens Groth. „A Verifiable Secret Shuffe of Homomorphic Encryptions“. In: *Public Key Cryptography - PKC 2003*. Hrsg. von Yvo Desmedt. Bd. 2567. Lecture Notes in Computer Science. 10.1007/3-540-36288-6_11. Springer Berlin / Heidelberg, 2002, S. 145–160. URL: http://dx.doi.org/10.1007/3-540-36288-6_11.

[69] Christian Henrich. „Improving and analysing bingo voting“. Diss. Karlsruher Institut für Technologie (KIT), 2012.

[70] Markus Jakobsson und Ari Juels. „Mix and Match: Secure Function Evaluation via Ciphertexts“. In: *Advances in Cryptology — ASIACRYPT 2000*. Hrsg. von Tatsuaki Okamoto. Bd. 1976. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2000, S. 162–177. ISBN: 978-3-540-41404-9. DOI: 10.1007/3-540-44448-3_13. URL: http://dx.doi.org/10.1007/3-540-44448-3_13.

[71] Kimmo Järvinen, Vladimir Kolesnikov, Ahmad-Reza Sadeghi und Thomas Schneider. „Garbled circuits for leakage-resilience: Hardware implementation and evaluation of one-time programs“. In: *Cryptographic Hardware and Embedded Systems, CHES 2010*. Springer, 2010, S. 383–397.

[72] Marc Joye. „On white-box cryptography“. In: *Proceedings of the 1st International Conference Security of Information and Networks*. 2008, S. 7.

[73] Ari Juels, Dario Catalano und Markus Jakobsson. „Coercion-Resistant Electronic Elections“. English. In: *Towards Trustworthy Elections*. Hrsg. von David Chaum, Markus Jakobsson, RonaldL. Rivest, PeterY.A. Ryan, Josh Benaloh, Miroslaw Kutylowski und Ben Adida. Bd. 6000. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2010, S. 37–63. ISBN: 978-3-642-12979-7. DOI: 10.1007/978-3-642-12980-3_2. URL: http://dx.doi.org/10.1007/978-3-642-12980-3_2.

[74] Ari Juels, Dario Catalano und Markus Jakobsson. „Coercion-resistant electronic elections“. In: *WPES '05: Proceedings of the 2005 ACM workshop on Privacy in the electronic society*. Alexandria, VA, USA: ACM, 2005, S. 61–70. URL: http://markus-jakobsson.com/papers/jakobsson-wpes05.pdf.

[75] Bundesministerium der Justiz. *Bundeswahlordnung (BWO)*. http://www.bundestag.de/blob/189208/d12bf66e0f0c75b5d194eb238b3e3d1f/bwahlo_1pdf-data.pdf (in German). 1985, last revised 2013.

[76] Jonathan Katz. „Universally composable multi-party computation using tamper-proof hardware“. In: *Advances in Cryptology-EUROCRYPT 2007*. Springer, 2007, S. 115–128.

[77] Jonathan Katz und Yehuda Lindell. *Introduction to modern cryptography: principles and protocols*. CRC press, 2007.

[78] Shahram Khazaei, Tal Moran und Douglas Wikström. „A Mix-Net from Any CCA2 Secure Cryptosystem“. English. In: *Advances in Cryptology – ASIACRYPT 2012*. Hrsg. von Xiaoyun Wang und Kazue Sako. Bd. 7658. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, S. 607–625. ISBN: 978-3-642-34960-7. DOI: 10.1007/978-3-642-34961-4_37. URL: http://dx.doi.org/10.1007/978-3-642-34961-4_37.

[79] Olga Kochetova. *Jackpot am Geldautomaten. Wie man mit oder ohne Malware zu Bargeld kommen kann*. 26. Apr. 2016. URL: https://de.securelist.com/analysis/veroffentlichungen/71316/malware-and-non-malware-ways-for-atm-jackpotting-extended-cut/ (besucht am 25. 06. 2017).

[80] Vladimir Kolesnikov und Thomas Schneider. „Improved garbled circuit: Free XOR gates and applications“. In: *Automata, Languages and Programming*. Springer, 2008, S. 486–498.

[81] Steve Kremer, Mark Ryan und Ben Smyth. „Election verifiability in electronic voting protocols“. In: *European Symposium on Research in Computer Security*. Springer. 2010, S. 389–404.

[82] Mirosław Kutyłowski und Filip Zagórski. „Verifiable Internet Voting Solving Secure Platform Problem“. English. In: *Advances in Information and Computer Security*. Hrsg. von Atsuko Miyaji, Hiroaki Kikuchi und Kai Rannenberg. Bd. 4752. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2007, S. 199–213. ISBN: 978-3-540-75650-7. DOI: 10.1007/978-3-540-75651-4_14. URL: http://dx.doi.org/10.1007/978-3-540-75651-4_14.

[83] John Leyden. *ATM security devs rush out patch after boffins deliver knockout blow. Researchers had full control and were able to make unauthorised withdrawals*. Englisch. 3. Mai 2017. URL: https://www.theregister.co.uk/2017/05/03/atm_security_software_vuln/ (besucht am 25. 06. 2017).

[84] Epp Maaten. „Towards Remote E-Voting: Estonian case.“ In: *Electronic Voting in Europe*. Hrsg. von Alexander Prosser und Robert Krimmer. Bd. 47. LNI. GI, 2004, S. 83–100. ISBN: 3-88579-376-8. URL: http://dblp.uni-trier.de/db/conf/evoting/evoting2004.html#Maaten04.

[85] Declan McCullagh. *Security researcher demonstrates ATM hacking. IOActive's Barnaby Jack reveals at Black Hat how he found ways to remotely log into ATMs without a password and force them to spit out cash*. Englisch. 28. Juli 2010. URL: https://www.cnet.com/news/security-researcher-demonstrates-atm-hacking/ (besucht am 25. 06. 2017).

[86] Tal Moran und Moni Naor. „Split-ballot voting: everlasting privacy with distributed trust“. In: *ACM Transactions on Information and System Security (TISSEC)* 13.2 (2010), S. 16.

[87] Tal Moran und Gil Segev. „David and Goliath commitments: UC computation for asymmetric parties using tamper-proof hardware“. In: *Advances in Cryptology–EUROCRYPT 2008*. Springer, 2008, S. 527–544.

[88] Steven J Murdoch, Saar Drimer, Ross Anderson und Mike Bond. „Chip and PIN is Broken“. In: *Security and Privacy (SP), 2010 IEEE Symposium on*. IEEE. 2010, S. 433–446.

[89] Andrew C. Myers, Michael Clarkson und Stephen Chong. „Civitas: Toward a Secure Voting System“. In: *IEEE Symposium on Security and Privacy*. IEEE. Mai 2008, S. 354–368. DOI: 10.1109/SP.2008.32. URL: http://www.truststc.org/pubs/450.html.

[90] C. Andrew Neff. *A verifiable secret shuffle and its application to e-voting*. CCS '01 Proceedings of the 8th ACM conference on Computer and Communications Security. 2001.

[91] Diebold Nixdorf. *Diebold Nixdorf First ATM Manufacturer To Support Windows 10*. Englisch. 6. Apr. 2017. URL: http://news.dieboldnixdorf.com/press-releases/diebold-nixdorf-first-atm-manufacturer-to-support-windows-10.htm (besucht am 25. 06. 2017).

[92] Rafail Ostrovsky. „Efficient computation on oblivious RAMs“. In: *Proceedings of the twenty-second annual ACM symposium on Theory of computing*. ACM. 1990, S. 514–523.

[93] Pascal Paillier. „Public-Key Cryptosystems Based on Composite Degree Residuosity Classes“. In: *Advances in Cryptology – EUROCRYPT '99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings*. Hrsg. von Jacques Stern. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, S. 223–238. ISBN: 978-3-540-48910-8. DOI: 10.1007/3-540-48910-X_16. URL: http://dx.doi.org/10.1007/3-540-48910-X_16.

[94] Pascal Paillier. „Public-Key Cryptosystems Based on Composite Degree Residuosity Classes“. In: *EUROCRYPT*. Hrsg. von Jacques Stern. Bd. 1592. Lecture Notes in Computer Science. Springer, 1999, S. 223–238. ISBN: 3-540-65889-0.

[95] Torben Pryds Pedersen. „Advances in Cryptology – CRYPTO '91: Proceedings“. In: Hrsg. von Joan Feigenbaum. Berlin, Heidelberg: Springer Berlin Heidelberg, 1992. Kap. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing, S. 129–140. ISBN: 978-3-540-46766-3. DOI: 10.1007/3-540-46766-1_9. URL: http://dx.doi.org/10.1007/3-540-46766-1_9.

[96] Torben Pryds Pedersen. „Non-interactive and Information-Theoretic Secure Verifiable Secret Sharing“. In: *Advances in Cryptology – CRYPTO '91: Proceedings*. Hrsg. von Joan Feigenbaum. Bd. 576. Lecture Notes in Computer Science. Springer, 1991, S. 129–140.

[97] Benny Pinkas und Tzachy Reinman. „Oblivious RAM revisited“. In: *Advances in Cryptology–CRYPTO 2010*. Springer, 2010, S. 502–519.

[98] Stefan Popoveniuc und Benjamin Hosp. „An Introduction to PunchScan“. In: *Towards Trustworthy Elections*. Hrsg. von David Chaum, Markus Jakobsson, Ronald L. Rivest, Peter Y. A. Ryan, Josh Benaloh, Miroslaw Kutylowski und Ben Adida. Bd. 6000. Lecture Notes in Computer Science. Springer, 2010, S. 242–259. ISBN: 978-3-642-12979-7.

[99] Michael O Rabin. „How To Exchange Secrets with Oblivious Transfer.“ In: *Technical Report TR-81, Aiken Computation Lab, Harvard University* (1981).

[100] rad./dpa. *Ermittlungen wegen Wahlbetrugs – Venezuela*. 3. Aug. 2017. URL: http://www.faz.net/aktuell/politik/ausland/venezuela-verdacht-auf-millionenfachen-wahlbetrug-15134499.html (besucht am 08. 08. 2017).

[101] Daniel Regalado. *SUCEFUL: Next Generation ATM Malware*. Englisch. 11. Sep. 2015. URL: https://www.fireeye.com/blog/threat-research/2015/09/suceful_next_genera.html (besucht am 25. 06. 2017).

[102] Riigikogu. *Riigikogu Election Act*. Riigi Teataja. https://www.riigiteataja.ee/en/eli/ee/501092014005/consolide/current. 2002.

[103] Jens Rosbach. *Whistleblower belastet Vorstandschef – Jüdische Gemeinde Berlin*. 11. Aug. 2016. URL: http://www.deutschlandfunk.de/juedische-gemeinde-berlin-whistleblower-belastet.862.de.html?dram:article_id=362859 (besucht am 08. 08. 2017).

[104] Jens Rosbach. *Wie sauber war die Wahl zum Gemeindeparlament? – Streit in Berlins Jüdischer Gemeinde*. 23. Jan. 2016. URL: http://www.deutschlandfunkkultur.de/streit-in-berlins-juedischer-gemeinde-wie-sauber-war-die.2165.de.html?dram:article_id=343374 (besucht am 08. 08. 2017).

[105] Amit Sahai und Brent Waters. „How to use indistinguishability obfuscation: Deniable encryption, and more“. In: *Proceedings of the 46th Annual ACM Symposium on Theory of Computing*. ACM. 2014, S. 475–484.

[106] Kazue Sako und Joe Kilian. „Receipt-free mix-type voting scheme: a practical solution to the implementation of a voting booth“. In: *Proceedings of the 14th annual international conference on Theory and application of cryptographic techniques*. EUROCRYPT'95. Saint-Malo, France: Springer-Verlag, 1995, S. 393–403. ISBN: 3-540-59409-4. URL: http://dl.acm.org/citation.cfm?id=1755009.1755052.

[107] Claus-Peter Schnorr. „Efficient signature generation by smart cards“. English. In: *Journal of Cryptology* 4.3 (1991), S. 161–174. ISSN: 0933-2790. DOI: 10.1007/BF00196725. URL: http://dx.doi.org/10.1007/BF00196725.

[108] Adi Shamir. „How to share a secret“. In: *Communications of the ACM* 22.11 (1979), S. 612–613.

[109] Elaine Shi, T-H Hubert Chan, Emil Stefanov und Mingfei Li. „Oblivious RAM with $O((\log n)3)$ worst-case cost“. In: *Advances in Cryptology–ASIACRYPT 2011*. Springer, 2011, S. 197–214.

[110] Bundesamt für Sicherheit in der Informationstechnik (BSI). „Technische Richtlinie BSI TR-03116 – Kryptographische Vorgaben für Projekte der Bundesregierung“. In: (2017). URL: https://www.bsi.bund.de/DE/Publikationen/TechnischeRichtlinien/tr03116/index_htm.html (besucht am 07. 08. 2017).

[111] Warren D. Smith. *New cryptographic election protocol with best-known theoretical properties*. Frontiers in Electronic Elections (FEE 2005). 2005.

[112] Oliver Spycher, Rolf Haenni und Eric Dubuis. „Coercion-resistant hybrid voting systems“. In: *Electronic Voting*. 2010, S. 269–282.

[113] Linda Staude. *Faustkampf im Parlament – Streit um Wahlgesetzreform in Kenia*. 5. Jan. 2017. URL: http://www.deutschlandfunk.de/streit-um-wahlgesetzreform-in-kenia-faustkampf-im-parlament.1773.de.html?dram:article_id=375561 (besucht am 08. 08. 2017).

[114] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu und Srinivas Devadas. „Path oram: An extremely simple oblivious ram protocol“. In: *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM. 2013, S. 299–310.

[115] Nick Summers. *ATMs Face Deadline to Upgrade From Windows XP. The vast majority of automated teller machines have an April deadline to get off Windows XP*. Englisch. 18. Jan. 2014. URL: https://www.bloomberg.com/news/articles/2014-01-16/atms-face-deadline-to-upgrade-from-windows-xp (besucht am 25. 06. 2017).

[116] Jo Goll und Torsten Mandalka. *Wahlskandal um die Jüdische Gemeinde in Berlin – Neue Zeugenaussagen erhärten Verdacht der Wahlfälschung*. 28. Sep. 2016. URL: http://www.rbb-online.de/politik/beitrag/2016/09/vorwuerfe-wahlfaelschung-juedische-gemeinde.html (besucht am 08. 08. 2017).

[117] Gerald V. Post. „Using re-voting to reduce the threat of coercion in elections“. In: *Electronic Government, an International Journal*. Volume 7, Number 2/2010. Inderscience Publishers, 2010, S. 168–182. URL: https://inderscience.metapress.com/content/e841t68786434728/resource-secured/?target=fulltext.pdf.

[118] Melanie Volkamer und Rüdiger Grimm. „Multiple Casts in Online Voting: Analyzing Chances“. In: *Electronic Voting 2006: 2nd International Workshop, Co-organized by Council of Europe, ESF TED, IFIP WG 8.6 and E-Voting.CC, August, 2nd - 4th, 2006 in Castle Hofen, Bregenz, Austria*. Hrsg. von Robert Krimmer. Bd. 86. LNI. GI, 2006, S. 97–106. ISBN: 978-3-88579-180-5. URL: http://subs.emis.de/LNI/Proceedings/Proceedings86/article4554.html.

[119] *Wahlcomputer-Firma will Belege für Manipulation haben*. 2. Aug. 2017. URL: http://www.deutschlandfunk.de/venezuela-wahlcomputer-firma-will-belege-fuer-manipulation.1939.de.html?drn:news_id=775739 (besucht am 08. 08. 2017).

A. Übersicht über Bezeichnungen und Variablen

Im Folgenden geben wir einen Überblick über die meistverwendeten Bezeichnungen aus den zentralen Kapiteln der Arbeit. Bezeichnungen, die nur lokal verwendet werden, sind hier nicht aufgeführt und überschreiben gegebenenfalls die hier beschriebene Bedeutung. Wir haben uns bemüht, die Bezeichnungen zwischen den Kapiteln unverändert zu lassen; aufgrund der unterschiedlichen Ausrichtung ist uns das für einige wenige nicht gelungen. Die Listen enthalten dennoch der Vollständigkeit halber die Duplikate.

A.1. Variablen aus dem Kapitel *Oblivious Bingo Voting*

a_l	Mitglied der Wahlleitung
$\mathcal{A}$	Angreifer
$B_{i,j}$	Chiffrat: Wurde die Füllstimme verwendet? $\texttt{ENC}(0)$, sonst $\texttt{ENC}(1)$
$b_{i,j}$	Bit: Wurde die Füllstimme verwendet? 0, sonst 1
β_j	Wählerentscheidung
c, $c_{i,j}$	Commitment
C, C'	blindes Commitment: Tripel (c,u,v)
D, D'	blindes Commitment: Tripel (c,u,v)
e_j	Wähler / Elector
g	Erzeuger
h	Erzeuger
i	Laufvariable Kandidat (siehe p_i)
j	Laufvariable Wähler (siehe e_j)
k	Schwellenwert für Threshold-Verschlüsselung
l	Laufvariable für Mitglieder der Wahlleitung (siehe a_l)
$N_{i,j}$	Zufallszahl / Füllstimme (siehe ebenfalls R)
n_A	Anzahl der Mitglieder der Wahlleitung
$n_{\mathcal{A}}$	Zahl der vom Angreifer $\mathcal{A}$ korrumpierten Stimmen
n_β	Zahl der abgegebenen Stimmzettel
n_E	Anzahl der Wähler
n_i	Anzahl benutzter Füllstimmen
n_P	Anzahl der Kandidaten
p	Primzahl
p_i	Kandidat / Partei
$\mathrm{pk}_{\mathcal{T}}$	öffentlicher Schlüssel der Wahlleitung (Tally)
q	Primzahl
Q_j	Liste der Commitments für den Bingo-Voting-Beleg
r	Zufallszahl / ohne bestimmten Verwendungszweck
R, R_j	Zufallszahl / frischer Zufall statt Füllstimme N
$\mathcal{S}$	Simulator
$\mathrm{sk}_{\mathcal{T},l}$	geheimer Schlüssel eines Mitglieds der Wahlleitung l
$\mathrm{s}_l^{(sig)}$	Signaturschlüssel von a_l
$\mathrm{s}_l^{(ver)}$	Verifikationsschlüssel für Signatur von a_l
σ	Signatur
U	Liste der Commitments für Auszählung (unverwendete)
$u_{i,j}$	Chiffrat eines blinden Commitments: enthält den Commitment-Wert
$v_{i,j}$	Chiffrat eines blinden Commitments: enthält den Commitment-Zufall
$\mathbf{X}$	Auszählungsergebnis
$\zeta^{(a_l)}$	Commitment auf blinde Commitments

A.2. Variablen aus dem Kapitel *Coercion Resistant Revoting*

a	temporäre Variable
a_l	Mitglied der Wahlleitung
A_i	Listen (Schritt in der Auszählung)
$\mathcal{A},\ \mathcal{A}'$	Angreifer
b	temporäre Variable
b_i	Stimmzettel
B_i	Listen (Schritt in der Auszählung)
β_j	Wählerentscheidung
c	temporäre Variable
C	Liste der Kandidaten
$d_{j,j'}$	verschlüsselter, maskierter Quotient zweier öffentlicher Schlüssel pk_j und $\mathrm{pk}_{j'}$: ($\mathtt{ENC}(1)$, wenn $\mathrm{pk}_j = \mathrm{pk}_{j'}$)
$\hat{d}_{j,j'}$	konvertierters $d_{j,j'}$: ($\mathtt{ENC}(1)$, wenn $\mathrm{pk}_j \neq \mathrm{pk}_{j'}$)
$d,\ \bar{d}$	gleichverteilt zufällig gewähltes Bit
e_j	Wähler / Elector
E_i	Listen (Schritt in der Auszählung)
$g,\ h$	Erzeuger
$i,\ j$	Laufvariablen: Kandidat (siehe p_i), Wähler (siehe e_j)
k_i	Sicherheitsparameter
K_i	Listen mit verschlüsselten öffentlichen Schlüsseln der Wähler
l	Länge des Bulletin Boards (Zahl der abgegebenen Stimmen) oder Laufvariable für Mitglieder der Wahlleitung (siehe a_l)
L_{Voter}	Liste der Wahlberechtigten
$\mathrm{L}_0,\ \mathrm{L}_{trash}, \ldots$	Listen während der Auszählung
$L_i[x]$	x-tes Element aus der Liste L_i
m	Laufvariable für Revotes oder Klartext
n_A	Anzahl der Mitglieder der Wahlleitung
$n_{\mathcal{A}}$	Zahl der vom Angreifer $\mathcal{A}$ korrumpierten Stimmen
n_β	Zahl der abgegebenen Stimmzettel
n_E	Anzahl der Wähler
n_P	Anzahl der Kandidaten
o_j	Flag, ob Stimme überschrieben: $o_j = \mathtt{ENC}(a)$ Stimme überschrieben: $a \neq 1$; sonst: $a = 1$
p	Primzahl
pk_i	öffentlicher Schlüssel i
$\mathrm{pk}_{\mathcal{T}}$	öffentlicher Schlüssel der Wahlleitung (Tally)
p_i	Kandidat / Partei
q	Primzahl
r	Zufallszahl (meist ohne bestimmten Verwendungszweck)
$\mathcal{S}$	Simulator
$\widetilde{sk}$	ungültiger geheimer Schlüssel
sk_i	geheimer Schlüssel i
$\mathrm{sk}_{\mathcal{T},l}$	geheimer Schlüssel eines Mitglieds der Wahlleitung l
σ	Signatur
ts_j	Zeitstempel
$\mathbf{X}$	Auszählungsergebnis

A.3. Funktionen

Verschlüsselung von m mit...	
$\texttt{ENC}(m)$	...implizit verwendetem Zufall und Schlüssel
$\texttt{ENC}(m, r)$	...explizit benanntem Zufall r
$\texttt{ENC}_{\text{pk}}(m)$	...explizit benanntem Schüssel pk
$\texttt{ENC}_{\text{pk}}(m, r)$	...explizit benanntem Zufall r und Schlüssel pk

Entschlüsselung von c mit...	
$\texttt{DEC}(c)$	...implizit verwendetem Zufall
$\texttt{DEC}_{\text{sk}}(c)$	...explizit benanntem geheimen Schlüssel

Signaturen	
$\texttt{SIG}_{sk}(m)$	Signieren der Nachricht m mit Schlüssel sk
$\texttt{VERIF}_{pk}(m,\ \sigma)$	Verifizieren der Signatur σ mit m und pk

sonstiges	
$\texttt{COM}(m)$	Commitment auf m
$\texttt{COM}(m, r)$	Commitment auf m mit explizit angegebener Zufallszahl r
$\texttt{bCOM}(m)$	blindes Commitment (verwendete Zufallszahlen implizit)
$\texttt{re-rand}(C)$	Rerandomisierung von C (Zufallszahlen implizit)
`KeyGen`	Schlüsselerzeugung (für verschiedene Bausteine, mit gegebenem Sicherheitsparameter)
`Shuffle`	Mix-Net, ausgeführt von Mitgliedern einer Gruppe (erhält als Eingabe eine Liste von Chiffraten, Commitments oder aus Chiffraten oder Commitments zusammengesetzten Objekten)
$\texttt{H}(m)$	Hash der Nachricht m
$\texttt{ggT}(a, b)$	größter gemeinsamer Teiler zwischen a und b

Weitere Bausteine	
$\texttt{PET}(a, b)$	Plaintext-Equality-Test zwischen a und b
$\texttt{EPET}(a, b)$	encrypted Plaintext-Equality-Test zwischen a und b
pOT	Physical Oblivious Transfer
RNG	Zufallszahlengenerator

B. Blurry-Box Hackers-Contest

Für die Hackers-Contest von Blurry-Box wurde ein Quiz-Spiel entwickelt, das mit einer Deutschlandreise verknüpft ist. Die hier angegebenen Skizzen und Pseudocode-Stücke sollen die Idee, die sich hinter dem Schutz verbirgt, verdeutlichen. *Es handelt sich dabei nicht um einsetzbaren Code.*

Auch wenn das Spiel speziell für den Contest entwickelt wurde, wurde das Spiel nicht um die Schutzmethoden herum entwickelt. Das hier angegebene Konzept der Umsetzung der Blurry-Box-Methoden nutzt die Konzepte des Spiels. Das Spiel wurde von der Firma Wibu-Systems programmiert und die Schutzfunktionen manuell implementiert. Die Umsetzung des Schutzes basierte auf diesen Skizzen, folgt ihnen jedoch nicht in allen Punkten exakt.

B.1. Spielidee

Der Spieler erhält ein Startguthaben und kann damit von Stadt zu Stadt reisen. Dafür wählt er ein Ziel und ein Verkehrsmittel. Von seinem aktuellen Standpunkt, dem Ziel und dem Verkehrsmittel hängen die Kosten für die Reise ab. Verfügt der Spieler nicht über ein ausreichend hohes Guthaben, kann er die Wahl (begrenzt oft) ändern. In der neuen Stadt angekommen, werden ihm Fragen gestellt, für die er einen Einsatz wählt. Wählt der Spieler aus den Antwortvorschlägen die richtige Antwort aus, gewinnt er Geld; wählt er eine falsche Antwort, reduziert sich sein Guthaben. Bei bestimmten Abfolgen (Städtereihenfolge, Folge von richtigen Antworten, ...) werden sogenannte *Easter Eggs* freigeschaltet. Dabei kann es sich um ein zusätzliches Verkehrsmittel handeln, ein Zusatzguthaben oder ähnliches.

Sowohl die Kosten- als auch die Gewinnfunktion ist in mehrere Varianten unterteilt. Welche Ziele von den Städten aus erreicht werden können, stellt ebenfalls Varianten dar. Zusätzlich ist jede Frage eine Variante. Welche Frage ausgewählt wird, ist abhängig vom Spielstand.

Die Auswahl, welche Variante als nächstes entschlüsselt werden muss, wird vom Dongle anhand der aktuellen Spielsituation bestimmt. So gibt es eine festgelegte Folge zwischen *Zielstadt wählen*, *Verkehrsmittel wählen*, *Einsatz wählen*, *Frage*, *Gewinn berechnen* und gegebenenfalls einem *Easter Egg*, bevor der Spieler erneut eine Zielstadt wählen kann. Nur nach der Wahl des Verkehrsmittels kann der Wähler, wenn er die Kosten angezeigt bekommt, zurück zur Stadtauswahl springen. Versucht der Spieler durch Manipulation der Daten einen anderen Ablauf zu wählen, wird die Lizenz gelöscht.

Die Hacker erhielten zusätzlich zum geschützten Spiel, dem Dongle mit Lizenz und der Schutzbeschreibung ein Beispiel-Script, das das Spiel *fernsteuert*. Das sollte es den Hackern erleichtern, automatisierte Spieldurchläufe zu generieren. Eine automatisierte Auswahl der *richtigen* Antwort ist jedoch nicht möglich, da diese Information nur vom Dongle entschlüsselbar vorliegt und darauf basierend die Gewinnfunktion auswählt; im Falle einer falsch gewählten Antwort ist die Gewinnfunktion negativ. Die Anwendung der Funktion kann das Dongle zwar nicht prüfen. Jedoch ist es prinzipiell möglich zu prüfen, ob sich der Kontostand im richtigen Maße ändert. Ein höherer Kontostand nach einer falsch beantworteten Frage ist ebenso unmöglich wie ein Gewinn, der höher ist als der Einsatz.

B.2. Methoden

Im Folgenden geben wir nochmal einen kurzen Überblick über die Methoden, die im Hackers-Contest angewendet wurden, um die Software zu schützen. Die hier gewählte Reihenfolge dient dem besseren Verständnis. Die Anwendung der Methoden findet jedoch nicht in der hier aufgeführten Reihenfolge statt.

1. **Erstellung von Varianten:** Der Programmcode wird in einzelne Blöcke eingeteilt. In unserem Spiel für den Hackers-Contest sind das die folgenden: Städte, Verkehrsmittel, Reisekosten, Fragen und Gewinn.
2. **Modifikation von Varianten:** Die einzelnen Blöcke werden durch Spezialisierung zu Varianten:

   Städte: Die von einer bestimmten Stadt aus zu bereisenden Städte werden für jede Stadt separat als Variante gespeichert. Die Liste der möglichen Ziele von jeder einzelnen Stadt aus ist eine eigene Variante.

   Verkehrsmittel: Die Verkehrsmittel, die zwischen zwei Städten gewählt werden können, werden für jedes Tupel an Städten separat gespeichert. Die Liste der möglichen Verkehrsmittel für jede mögliche Städtekombination ist eine Variante.

   Reisekosten: Die Reisekostenfunktion wird für jedes Tupel (Stadt $\times$ Stadt $\times$ Verkehrsmittel) nochmals anhand der Uhrzeit in Intervalle aufgeteilt. Jedes Intervall ist eine Variante.

   Fragen: Jede Frage ist eine eigene Variante.

   Gewinn: Die Gewinn-/Verlustfunktion wird anhand der Antwortzeit und dem Einsatz in einzelne Intervalle eingeteilt. Jedes Intervall ist eine Variante.
3. **Verschlüsselung der Varianten:** Alle Varianten werden einzeln mit einem symmetrischen Verschlüsselungsverfahren und einem für jede Variante zufällig gezogenen Variantenschlüssel verschlüsselt. Der Variantenschlüssel wird mit dem Schlüssel des Dongles verschlüsselt. (Siehe dazu Abbildung B.1.)

4. **Einfügen von Fallen:** Es werden zusätzliche verschlüsselte Blöcke eingefügt, auf die aus einem Programmdurchlauf nie verwiesen wird. Als Variantenschlüssel wird ein Schlüssel gewählt, der dem Dongle als *falscher Schlüssel* bekannt ist. Dieser wird – ebenso wie bei den gültigen Varianten – mit dem Schlüssel des Dongles verschlüsselt an die Variante angehängt. Entschlüsselt das Dongle einen solchen *falschen* Variantenschlüssel, sperrt es die Lizenz.

5. **Variantenauswahl im Dongle:** Die Berechnung, welche Variante als nächstes ausgeführt werden muss, um das Spiel in seinem vorgesehenen Ablauf fortzusetzen, findet im Dongle statt. Dieser Code wird aus dem ursprünglichen Programmcode extrahiert und aus den Varianten entfernt. Siehe dazu Abschnitt B.3.5.

6. **Dongle als Zustandsspeicher:** Das Dongle kann den Zustand, in dem sich das Spiel befindet, speichern, um festzustellen, ob die Ausführungsreihenfolge eingehalten wird. Der Speicher, den wir dazu im Dongle benötigen, muss nicht groß sein, da der eigentliche Zustand ausgelagert werden kann. Um Manipulationen am ausgelagerten Zustand zu vermeiden, wird der ausgelagerte Zustand signiert. Um Replayangriffe zu verhindern, zieht das Dongle bei jeder Aktualisierung eine frische Zufallszahl, die Teil der signierten Nachricht wird, und speichert diese im internen Speicher. Acht Bit scheinen uns dafür ausreichend zu sein. Jedes weitere Bit reduziert die Wahrscheinlichkeit, dass der Angreifer einen erfolgreichen Replayangriff durchführt. Beim Laden des Zustandes wird die Signatur zusammen mit dem im internen Speicher befindlichen Zufall geprüft.

B.3. Variantenauswahl und Zustandsspeicher im Dongle

In diesem Abschnitt wird die Umsetzung der Methode 5 (Variantenauswahl im Dongle) und 6 (Zustandsspeicher im Dongle) mit Hilfe von Pseudocode skizziert. Abschnitt B.3.1 gibt einen Überblick über den Aufbau der Varianten. In Abschnitt B.3.2, B.3.3 und B.3.4 werden Hilfsklassen und -methoden beschrieben, um die Beschreibung des zentralen Codeblocks in Abschnitt B.3.5 kurz und lesbar zu halten: In Abschnitt B.3.2 werden einige Datenstrukturen definiert; der Abschnitt B.3.3 beschreibt Methoden, die als gegeben angenommen werden; Abschnitt B.3.4 enthält Funktionen, die das Dongle bereitstellen muss.

B.3.1. Überblick

Jede Variante besteht aus vier Teilen:

1. einer nicht verschlüsselten Variantenidentifikationsnummer V-ID,

2. einem verschlüsselten Header, der im Dongle entschlüsselt wird. Er enthält hauptsächlich den Schlüssel, mit dem der nächste Block verschlüsselt ist,

3. einem verschlüsselten Codeblock, der die eigentliche Variante enthält. Das ist der Programmteil, auf den es letztendlich ankommt. Der entschlüsselte Code wird auf dem PC ausgeführt und

4. einem verschlüsselten Codeblock, der den allgemeinen Sprungcode ergänzt (siehe Abschnitt B.3.5). Der Code wird im Dongle ausgeführt.

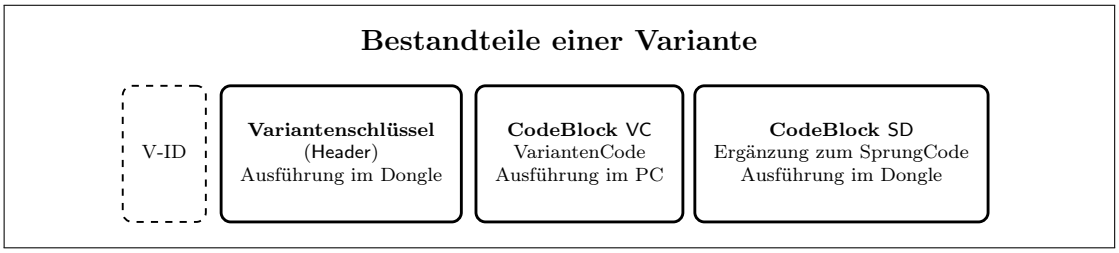

Abbildung B.1.: Aufbau einer Variante. Eine Variante besteht aus einer (unverschlüsselten) Variantenidentifikationsnummer und drei verschlüsselten Blöcken: Der Header enthält Code zur Ablaufkontrolle und den Schlüssel für den CodeBlock VC. Der CodeBlock VC enthält lediglich den Code, der auf dem PC ausgeführt wird: die eigentliche Variante. Der CodeBlock SD enthält nun wieder den Code, der im Dongle ausgeführt wird und die ID der nächsten Variante berechnet.

B.3.2. Ablaufkontrolle und State

Um zu verhindern, dass ein Angreifer (Spieler) die Varianten effizient durchprobieren kann, soll vom Ablauf des Spiels nicht abgewichen werden können. Dazu wird sichergestellt, dass die Parameter, von denen der Ablauf des Spiels abhängt, nicht frei gesetzt werden können.

Datentyp für den Status des Programmablaufs

Der *State* enthält die wesentlichen Parameter, die definieren, in welchem Zustand sich der Spieler befindet. Das gilt neben der Position im Zyklus (Ziel, Verkehrsmittel, Einsatz, Antwort) auch für den Fortschritt im Spiel. Einige Parameter (Tageszeit und Guthaben) werden im Lauf des Spiels neu gesetzt, ohne dass das Dongle die Werte genau kennt. Lediglich eine Prüfung der Plausibilität der Änderung kann hier durch das Dongle übernommen werden, da es grob abschätzen kann, in welchem Rahmen sich die Werte ändern können. Andere Werte können nur durch das Dongle geändert werden. Dazu gehören unter anderem der Rundenzähler oder welcher Schritt im Zyklus als nächstes ausgeführt werden muss. Diese Werte werden vom Dongle signiert und an den Angreifer/PC zurückgegeben. Um Replay-Attacken zu verhindern, muss ein im Dongle gespeicherter Wert, der sich nach der Prüfung und vor der Erstellung der nächsten Signatur ändert, Teil der Signatur sein.

```
1   class State{
2     int[] nextVid;           // naechste Variante
3     int nextType;            // Typ der naechsten Variante
4     int lapCounter;          // Rundenzaehler
5     int otherTarget;         // wie oft das Ziel korrigiert wurde
6     int currentCity;         // aktueller Aufenthaltsort
7     int target;              // Reiseziel
8     int vehicle;             // Verkehrsmittel (CurrentCity nach Target)
9     int stake;               // Einsatz
10    int durationStart;       // Dongle-Zeit, zu der die Frage gestellt wurde
11    cypher EasterEgg;        // Bit-Array; Laenge = Zahl der EasterEggs; 1=aktiv, 0=inaktiv
12    // Liste erweiterbar
13
14    /** Serialisiert alle oben stehenden Felder: nextVid, nextType, lapCounter,
15      * otherTarget, currentCity, target, vehicle, stake und durationStart. Der
16      * Rueckgabewert wird fuer die Signatur (sign() siehe Abschnitt B.3.3) genutzt
17      */
18    function byte[] serialize();
19
20    /* sym. Signatur */
21    Hmac hmac;
22
23
24    /* Element, die nicht in der sym. Sig. enthalten sind
25       da sie ausserhalb des Dongles geaendert werden */
26    int timeOfDay;           // Uhrzeit/Tageszeit
27    int asstes;              // Guthaben, durch Gewinn erspielt
28
29  }
```

```
1   enum TypeOfCode{
2     ziele,
3     verkehrsmittel,
4     transportkosten,
5     frage,
6     gewinn
7   }
```

Datentypen für Eingaben an das Dongle

Dem Dongle müssen Parameter übergeben werden können, anhand deren es entscheidet, welche Variante als nächstes ausgeführt werden muss. Diese Daten liegen einerseits im State, den das Dongle jedes Mal lädt, und andererseits werden Daten direkt „vom Benutzer“ eingegeben. Welche Daten das sind, hängt von der Verzweigung (der Position im Spiel) ab. Dazu werden die folgenden Klassen verwendet:

```
1   class InputParameter{
2
3   }
```

```
1   class InputParameterZiele : InputParameter{
2
3   }
```

```
1   class InputParameterVerkehrsmittel : InputParameter{
2     int target;
3   }
```

```
1   class InputParameterTransportkosten : InputParameter{
2     int verkehrsmittel;
3   }
```

```
1   class InputParameterFrage : InputParameter{
2     boolean newTarget;
3     int stake;
4   }
```

```
1   class InputParameterGewinn : InputParameter{
2     int answer;
3     int duration;
4   }
```

Datentypen für variantenspezifische Daten

Statt jeder Variante den kompletten Sprungcode anzuhängen, wurde dieser zu einem für alle gültigen Block zusammengefasst. Hin und wieder kann es jedoch sein, dass es spezifische Daten speziell zu den einzelnen Varianten gibt. Ein Beispiel sind die Fragen: Jede Frage ist eine eigene Variante. Welches die korrekte Antwort zu einer Frage ist, wird in diesem Datenblock codiert.

```
1   class SpecializedData{
2     TypeOfCode type;
3     int Vid;
4     function T specializedDataFunction();
5   }
```

```
1   class SpecializedDataZiele : SpecializedData{
2     int VidGetQuestionCurrentCity;            // Varianten-ID von "getFrage()" fuer
        entsprechende Stadt
3   }
```

```
1   class SpecializedDataVerkehrsmittel : SpecializedData{
2
3   }
```

```
1   class SpecializedDataTransportkosten : SpecializedData{
2
3   }
```

```
1   class SpecializedDataFrage : SpecializedData{
2     int VidGetTargetsCurrentCity;             // Varianten-ID von "getZiele()" fuer
        entsprechende Stadt
3   }
```

```
1   class SpecializedDataGewinn : SpecializedData{
2     int correctAnswer;
3     function boolean specializedDataFunction(inOut String answer, out int easterEgg);
4   }
```

B.3.3. Funktionen

Dieser Abschnitt enthält die Schnittstellenbeschreibungen einiger Funktionen, die im Programmcode (siehe Kapitel B.3.5) genutzt werden. Die genaue Implementierung hängt von der Umsetzung der Varianten oder anderen Funktionen im Dongle ab.

8 Bit Zufall ziehen und speichern

```
1   /** Funktion zieht einen zufaelligen Wert (8Bit?), speichert diesen
2     * im Dongle und gibt ihn zurueck.
3     */
4   function byte NewRandByte();
```

8 Bit Zufall laden

```
1   /** Funktion gibt den im Dongle gespeicherten zufaelligen (8Bit-)Wert
2     * zurueck.
3     */
4   function byte GetRandByte();
```

HMAC berechnen

```
1   /** Funktion berechnet HMAC
2     * Quelle: Wikipedia
3     */
4   function Hmac getHmac(byte[] data, byte randomByte)
5   {
6     byte key = randomByte || getPrivateHmacKey();
7     // Quelle: Wikipedia
8     if (length(key) > blocksize)
9     {
10      key = hash(key) // keys longer than blocksize are shortened
11    }
12    if (length(key) < blocksize)
13    {
14      // keys shorter than blocksize are zero-padded (where || is concatenation)
15      key = key << [0x00 * (blocksize - length(key))] // Where * is repetition.
16    }
17
18    o_key_pad = [0x5c * blocksize] XOR key // blocksize of the underlying hash function
19    i_key_pad = [0x36 * blocksize] XOR key
20
21    return hash(o_key_pad || hash(i_key_pad || message)) // Where || is concatenation
22
23  }
```

Symmetrische Signatur erstellen

```
1   /** Funktion signiert data symmetrisch. Dazu wird ein HMAC der Daten
2     * -- gemeinsam mit geheimen Daten des Dongles und nur bis zur naechsten
3     * Pruefung gueltiger, im Dongle gespeicherter Daten (gegen Replay) --
4     * erstellt.
5     */
6   function Hmac sign(byte[] data)
7   {
8     return getHmac(data, newRandByte());
9   }
```

Symmetrische Signatur verifizieren

```
1   /** Funktion verifiziert HMAC.
2     * hash(data, geheimeDaten, Seriennummer oder zufaelligerWert)
3     * verif verwendet sign, um auf Gleichheit zu testen.
4     */
5   function bool verif(byte[] data, Hmac hmac)
6   {
7     return (hmac == return getHmac(data, getRandByte()));
8   }
```

Zeitangabe runden

```
1   /** Funktion "rundet" Daten entsprechend der Variantenaufteilung Gewinn
2     * anhand der Antwortzeit
3     */
4   function int roundTime(byte[] data);
```

Einsatz runden

```
1   /** Funktion "rundet" Daten entsprechend der Variantenaufteilung Gewinn
2     * anhand des Einsatzes
3     */
4   function int roundStake(byte[] data);
```

Lizenz löschen / sperren

```
1   /** Funktion loescht die zur Ausfuehrung des Programms notwendige Lizenz
2     * im Dongle. Der Rueckgabewert ist -- wie sonst auch -- ein State.
3     */
4   function State lizenzLoeschen();
```

Verschlüsseln

```
1   /** Funktion verschluesselt ByteArray mit internem Schluessel des Dongles
2     */
3   function cypher enc(Byte[] Data);
```

Entschlüsseln

```
1   /** Funktion entschluesselt ByteArray mit internem Schluessel des Dongles
2     */
3   function plaintext dec(Byte[] Data);
```

B.3.4. System Calls

Zeit Ein Aufruf der Methode getCurrentDongleTime() gibt einen Wert zurück, der es zulässt, zuverlässige Aussagen über Zeitdifferenzen zu machen. Das Dongle kann Zeitdifferenzen sekundengenau angeben, solange es mit Strom versorgt wird.

Hash Das Dongle kann einen kryptographischen Hash berechnen (zum Beispiel SHA 512): hash()

Zufall erzeugen Das Dongle muss in der Lage sein, gute Zufallszahlen für kryptographische Funktionen zu erzeugen, ohne dass diese für den Angreifer in Erfahrung zu bringen sind.

zufälliges Byte speichern Das Dongle kann 8 Bit (Zufall erzeugen und) im Dongle speichern, ohne dass diese für den Angreifer in Erfahrung zu bringen sind.

zufälliges Byte lesen Das Dongle kann die oben gespeicherten 8 Bit wieder lesen und für weitere Berechnungen im Dongle zur Verfügung stellen, ohne dass diese für den Angreifer in Erfahrung zu bringen sind.

lösche Lizenz Das Dongle kann die Lizenz sperren/löschen, wenn illegale Aktionen festgestellt werden: lizenzLoeschen()

verschlüsseln Das Dongle kann Nachrichten symmetrisch verschlüsseln (zum Beispiel mit dem AES): enc()
Das Dongle verfügt dazu über einen geheimen Schlüssel, den der Angreifer nicht in Erfahrung bringen kann.

entschlüsseln Das Dongle kann diese Chiffrate wieder entschlüsseln, ohne dass der Klartext oder der Schlüssel für den Angreifer in Erfahrung zu bringen ist: dec()

symmetrischer Schlüssel für HMAC Das Dongle verfügt über einen geheimen Schlüssel für die Berechnung *symmetrischer Signaturen* (HMAC). Es stellt diesen Schlüssel für die internen Berechnungen zur Verfügung: getPrivateHmacKey(). Der Angreifer kann diese nicht in Erfahrung bringen.

B.3.5. Variantenauswahl

Sprungcode

Der folgende Codeblock wird einmal verschlüsselt auf der Festplatte gespeichert und zur Ausführung in das Dongle geladen. Der Code enthält – in diesem Fall – alle (größeren / relevanten) Sprunganweisungen. Welcher Teil davon ausgeführt wird, hängt von den übergebenen Parametern ab: Nach einem Kontrollabschnitt, der zu Beginn ausgeführt wird, entscheidet die Position im Spiel in dem Switch-Case-Statement, welcher Teil ausgeführt werden muss. Zum Schluss werden wieder einige generische Befehle ausgeführt, die zum Beispiel den State aktualisieren und für die Rückgabe an den PC signieren.

```
1   State getNextVid(in InputParameter input, in State state, in encryptedData
      VidSpecializedData)
2   {
3     int[] nextVid;
4     SpecializedData decVidSpecialicedData;   // Entschluesselte V-ID-speziallisierten Daten
5     State currentState;                      // Entschluesselter State
6     byte[] jumpRelevantData;
7     decVidSpecializedData = (SpecializedData)dec(VidSpecializedData);
8
9     /// Kontrolle der Ausführungsreihenfolge: Methode 6 ///
10    /* pruefen, ob der State unveraendert ist */
11    if (verif(state.serialize(), state.hmac))
12      currentState = state;
13    else
14      return lizenzLoeschen();
15
16    /* pruefen, ob der Type of Code stimmt */
17    if (currentState.nextType != decVidSpecializedData)
18      return lizenzLoeschen();
19
20    /* Ausfuehrungserlaubnis pruefen */
21    if (currentState.nextVid[0] != decVidSpecilizedData.Vid)
22      return lizenzLoeschen();
23    else
24      currentState.nextVid.removeFirst();
25
26
27    /* Typspezifischen Code ausfuehren */
28    switch (state.nextType)
29    {
30    /// Vorbereitung der Variantenauswahl „Ziele“ ///
31      case (ziele)
32        SpecializedDataZiele specData = (SpecializedDataZiele)decVidSpecialicedData;
33        if (currentState.otherTarget < 3){
34          jumpRelevantData.append(currentState.currentCity);
35        }
36        else{
37          nextVid.append(specData.VidGetQuestionCurrentCity);
38        }
39        currentState.nextType = TypeOfCode.Verkehrsmittel;
40        break;
41
42    /// Vorbereitung der Variantenauswahl „Verkehrsmittel“ ///
43      case (verkehrsmittel)
44        SpecializedDataVerkehrsmittel specData
45                          = (SpecializedDataVerkehrsmittel)decVidSpecialicedData;
46        InputParameterVerkehrsmittel specInput = (InputParameterVerkehrsmittel)input;
47
48        currentState.target = specInput.target;
49
```

```
50          jumpRelevantData.append(currentState.currentCity);
51          jumpRelevantData.append(specInput.target);
52          currentState.nextType = TypeOfCode.Transportkosten;
53          break;
54
55        /// Vorbereitung der Variantenauswahl ,,Transportkosten'' ///
56          case (transportkosten)
57            SpecializedDataTransportkosten specData
58                                = (SpecializedDataTransportkosten)decVidSpecialicedData;
59            InputParameterTransportkosten specInput = (InputParameterTransportkosten)input;
60
61            currentState.vehicle = specInput.vehicle;
62
63            jumpRelevantData.append(currentState.currentCity);
64            jumpRelevantData.append(currentState.target);
65            jumpRelevantData.append(specInput.vehicle);
66            jumpRelevantData.append(currentState.timeOfDay);
67            currentState.nextType = TypeOfCode.Fragen;
68            break;
69
70        /// Vorbereitung der Variantenauswahl ,,Frage'' ///
71          case (frage)
72            SpecializedDataFrage specData = (SpecializedDataFrage)decVidSpecialicedData;
73            InputParameterFragen specInput = (InputParameterFrage)input;
74
75            /* Ziel korrigieren? */
76            if (input.newTarget) {
77              currentState.otherTarget += 1;
78            }
79            else {
80              /* neue Runde; State entsprechend setzen */
81              currentState.currentCity = currentState.target;
82              currentState.target = 0;
83              currentState.vehicle = 0;
84              currentState.otherTarget = 0;
85              currentState.stake = specInput.stake;
86
87              /* Werte fuer die Berechnung der neuen Variante */
88              jumpRelevantData.append(currentState.target);
89              jumpRelevantData.append(specInput.stake);
90              jumpRelevantData.append(round(currentState.lapCounter));
91            }
92            currentState.nextType = TypeOfCode.Gewinn;
93            break;
94
95        /// Vorbereitung der Variantenauswahl ,,Gewinn'' ///
96          case (gewinn)
97            SpecializedDataGewinn specData = (SpecializedDataGewinn)decVidSpecialicedData;
98            InputParameterFragen specInput = (InputParameterFragen)input;
99
100           /* Ueberpruefe die Antwort auf ihre Richtigkeit */
101           int vidEasterEgg = null;
102           boolean answerIsCorrect
103                       = specData.specializedDataFunction(specInput.answer, vidEasterEgg))
104
105           /* EasterEgg-Behandlung */
106           if (vidEasterEgg != null) // EasterEgg oder Bild geklickt
107           {
108             nextVid.append(vidEasterEgg);
109
110             if (specInput.answer = null) // answer auf null, wenn keine Antwort enthalten
111               break;
112
113           }
114
115           /* Pruefe, ob die angegebene Antwortzeit plausibel ist */
116           antwortdauer = (currentState.durationStart - getCurrentDongleTime());
117           abweichung = abs(specInput.duration - antwortdauer);
118           if (abweichung  > 10)
119             ThrowDataManipuatlionException(Duration);
120
121           /* State: Eine weitere Runde gespielt */
122           currentState.lapCounter += 1;
123
124           /* Wert fuer die Berechnung der Gewinnvariante */
125           jumpRelevantData.append(answerIsCorrect);
126           jumpRelevantData.append(roundTime(specInput.duration));
127           jumpRelevantData.append(roundStake(currentState.stake));
128           currentState.nextType = TypeOfCode.Ziele;
129           break;
130
131         default
```

```
132           return lizenzLoeschen();
133       }
134
135       /// Berechnung der Varianten-ID: Methode 5 ///
136       /* Wenn diese naechste Varianten-ID noch nicht gesetzt ist, muss sie berechnet werden */
137       if (nextVid.length = 0)
138         nextVid.append(hash(jumpRelevantData, Dongle-Geheimnis));
139
140       currentState.nextVid.append(nextVid);
141       currentState.durationStart = getCurrentDongleTime();
142       currentState.hmac = sign(currentState.serialize());
143
144       return currentState;
145     }
```