Fakultät für Elektrotechnik und Informatik
Institut für Theoretische Informatik

Bachelorarbeit

Zero-Knowledge-Beweise in der digitalen Authentifikation

Niklas Schwabe

Hannover, 11. August 2018

Erstprüfer: Prof. Dr. Heribert Vollmer
Zweitprüfer: Dr. Arne Meier
Betreuer: Dr. Arne Meier

Eidesstattliche Erklärung

Hiermit versichere ich, die vorliegende Arbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die wörtlich oder inhaltlich aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Hannover, den 11. August 2018

_____

Niklas Schwabe

Inhaltsverzeichnis

1 Einleitung

1.1 Motivation

Die digitale Authentifikation einer Entität in einer elektronischen Kommunikation ist eine wichtige Voraussetzung, eine sichere Kommunikation mehrerer Teilnehmer zu etablieren und sicherzustellen. Durch den Einsatz von Authentifikationsverfahren können beispielsweise Man-in-the-Middle-Angriffe erschwert werden, die Angreifern das Mitlesen und die Manipulation von Nachrichten ermöglichen und somit die Integrität von Daten und die Vertraulichkeit von Kommunikationsverbindungen gefährden. Die Authentifikation von Teilnehmern auf einem Kommunikationskanal, eines Teilnehmers an elektronischen Wahlen oder eines Geräts im Netzwerk sind hierbei nur einige prominente Einsatzgebiete.

Eine populäre Methode zur Verschlüsselung eines Kommunikationskanals ist der Einsatz symmetrischer Kryptosysteme. Hierfür wird unter den Kommunikationsteilnehmern ein Schlüssel ausgetauscht, den im Folgenden alle Teilnehmer zur Ver- und Entschlüsselung der untereinander versendeten Nachrichten nutzen. Problematisch ist allerdings, dass der initiale Austausch des Schlüssels zunächst über einen unsicheren, unverschlüsselten Kanal erfolgt. Ein Diffie-Hellman-Schlüsselaustausch ermöglicht es zwei Kommunikationsteilnehmern, den gemeinsamen Schlüssel über diesen unsicheren Kanal zu vereinbaren, ohne dass dieser von Dritten ermittelt werden kann. Allerdings bietet der Diffie-Hellman-Schlüsselaustausch zunächst keine Mechanismen zur Authentifikation der Teilnehmer, durch einen Man-in-the-Middle-Angriff könnte also ein Angreifer Nachrichten ohne Wissen der Kommunizierenden modifizieren und sich als einer der Teilnehmer ausgeben, um an den gemeinsamen Schlüssel zu gelangen.

Um diesem Sicherheitsproblem vorzubeugen, werden häufig digitale Signaturenverfahren auf Basis einer Public-Key-Infrastruktur eingesetzt. Der Sendende signiert seine Nachricht mit seinem privaten Signierschlüssel, der Empfänger der Nachricht kann die Signatur mithilfe des zugehörigen öffentlichen Schlüssels des Signaturerstellers auf Echtheit überprüfen. Die Authentizität der öffentlichen Schlüssel wird dabei häufig in Kombination mit persönlichen Angaben zum Halter in einem Zertifikat zusammengefasst, das von einer zentralen, vertrauenswürdigen Drittpartei, beispielsweise einer Certificate Authority, verwaltet wird.

Soll in diesem Beispiel während der Authentifikation vom Einsatz einer zentralen, dritten Partei abgesehen werden und keine Signaturen verwendet wer-

den, kann der Einsatz eines Zero-Knowledge-Beweises erwogen werden. Ein Zero-Knowledge-Beweis ist ein probabilistisches Verfahren zur Authentifikation einer Entität. Ein Beweiser (*prover*) versucht hierfür, einen Verifizierer (*verifier*) davon zu überzeugen, ein bestimmtes Geheimnis zu kennen, ohne etwas über den Inhalt des Geheimnisses preiszugeben. Dies verringert das Risiko, sein Geheimnis während der Übertragung zum Empfänger offenzulegen. In der folgenden Arbeit sollen zunächst die Grundlagen eines Zero-Knowledge-Beweises dargelegt, beispielhaft einige Protokolle erläutert und nachfolgend verschiedene Einsatzgebiete von Zero-Knowledge-Beweissystemen in der digitalen Authentifikation präsentiert werden.

1.2 Einführendes Beispiel: Die magische Tür

Der grundlegende Ablauf eines Zero-Knowledge-Beweises kann anschaulich am folgenden Beispiel der „magischen Tür“ erläutert werden [BSW95]. Man stelle sich ein Gebäude mit dem in Abbildung 1.1 gezeigten Grundriss vor. Der Vorraum kann durch eine Tür betreten werden, außerdem zweigen, ebenfalls durch Türen getrennt, je ein Gang nach links sowie nach rechts ab. Die beiden Gänge enden in einer Sackgasse, sind allerdings an ihren Enden durch eine verschlossene Tür verbunden, die nur durch die Eingabe eines Zahlencodes entriegelt werden kann.

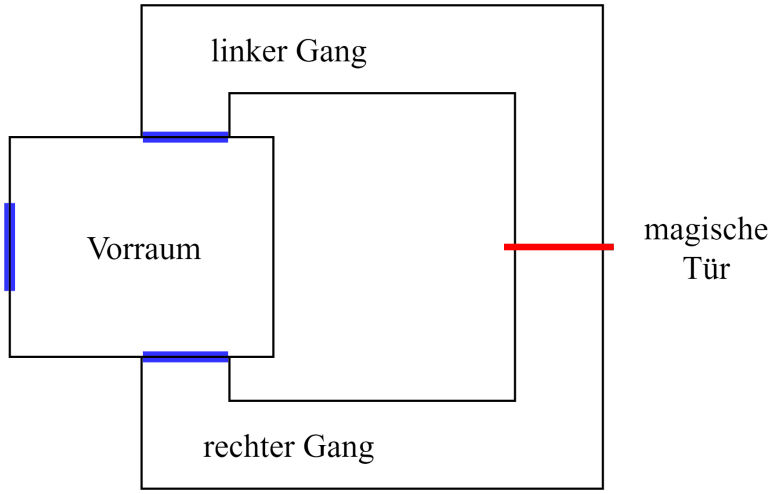

Abbildung 1.1: Die „magische“ Tür — blau kennzeichnet normale Türen, rot die magische Tür

Alice behauptet, diesen Zahlencode zu kennen. Ihr Ziel ist es, Bob von dieser Kenntnis zu überzeugen, ohne ihm den Code zu verraten. Mithilfe des folgenden Verfahrens ist dies möglich: Alice betritt allein den Vorraum, schließt die Tür hinter sich und wählt zufällig einen der Gänge aus. Sie geht hinein und schließt auch diese Tür. Somit ist gewährleistet, dass Bob keine Kenntnis darüber hat,

welchen Gang Alice gewählt hat. Nun darf Bob den Vorraum betreten. Er wirft eine Münze, um zu entscheiden, aus welchem Gang Alice herauskommen soll. Hat Alice zuvor denselben Gang gewählt, den Bob fordert, kann sie einfach wieder aus dem zuvor betretenen Gang heraustreten. Wählt Bob nicht denselben Gang, muss sie an der magischen Tür den Zahlencode eingeben, um in den anderen, von Bob geforderten Gang zu gelangen.

Kennt Alice den Zahlencode tatsächlich, kann sie in jedem Fall aus dem von Bob geforderten Gang heraustreten, gibt sie nur vor, den Code zu kennen, kann sie die magische Tür nicht öffnen und hat lediglich eine 50-prozentige Chance, Bob zu betrügen. Um die Sicherheit des Protokolls zu erhöhen, kann der gesamte Ablauf k-mal wiederholt werden, um die Wahrscheinlichkeit eines Betrugs beliebig weit zu reduzieren — auf 2^{-k}. Tritt Alice auch nur ein einziges Mal nicht aus der von Bob geforderten Tür heraus, ist sie als Betrügerin entlarvt und der Beweis kann abgebrochen werden.

Intuitiv scheint klar, dass Alice während des Verfahrens keinerlei Informationen über ihr Geheimnis, den Zahlencode zum Entriegeln der Tür, preisgibt. Um diese Eigenschaft des Protokolls formal zu beweisen, muss gezeigt werden, dass sämtliche Informationen, die Bob während des Protokolls erhält, auch simuliert werden könnten, wenn Alice das Kennwort gar nicht gekannt hätte.

Man nehme an, Bob filme den Ablauf mit einer Kamera. Mit der dabei entstehenden Aufnahme, dem Transkript, soll es Bob nicht möglich sein, Dritte davon zu überzeugen, dass Alice tatsächlich den Zahlencode kennt. Sei Malice eine Person, die den Code der Tür nicht kennt. Bob könnte dennoch ein Transkript anfertigen, in dem Malice k-mal aus der geforderten Tür heraustritt. Dazu müsste er lediglich jede Aufzeichnung der Kamera löschen, in der Malice nicht aus dem korrekten Gang hervorkommt. Insgesamt müssten also gegebenenfalls mehr als k Durchgänge gefilmt werden, um k Aufzeichnungen zu erhalten, in denen Malice den Gang durch die korrekte Tür verlässt. Das entstehende Transkript wäre nicht mehr unterscheidbar von einem ehrlich erzeugten Transkript. Somit hat der Beweis keine Aussagekraft für Dritte, Bob ist der Einzige, der durch die Interaktion mit Alice davon überzeugt werden kann, dass Alice den Zahlencode tatsächlich kennt.

2 Grundlagen

2.1 Turingmaschinen

Zunächst sollen zwei Varianten von Turingmaschinen definiert werden [GMR89, AB09]. Die grundlegende Funktionsweise einer Turingmaschine wird hierbei vorausgesetzt. Eine formale Definition einer deterministischen Turingmaschine sowie weitere Ausführungen finden sich beispielsweise in Michael Sipsers *„Introduction to the Theory of Computation“* [Sip97].

Definition 2.1. Eine *probabilistische Turingmaschine (PTM)* ist eine Turingmaschine mit zwei deterministischen Übergangsfunktionen δ_0 und δ_1. In jedem Berechnungsschritt wird eine der Transitionen per Zufall ausgewählt. Die Wahrscheinlichkeit ist dabei für beide Übergangsfunktionen δ_0 und δ_1 mit $\frac{1}{2}$ gleich groß. Die (zufällige) Wahlmöglichkeit zwischen δ_0 und δ_1 in jedem Schritt kann hierbei als fairer Münzwurf betrachtet werden.

Alternativ kann eine probabilistische Turingmaschine auch als deterministische Turingmaschine mit einem zusätzlichen, mit einer unendlichen Abfolge an zufälligen Bits beschriebenen Band aufgefasst werden. Das Band kann nur der Reihe nach gelesen werden, für jede Münze, die die Maschine wirft, wird das als nächstes folgende Bit gelesen.

Definition 2.2. Sei M eine probabilistische Turingmaschine und $x \in \{0,1\}^*$. Dann beschreibt $\Pr[M(x) = 1]$ die Wahrscheinlichkeit, dass die Maschine M bei Eingabe x akzeptiert.

Definition 2.3. Eine *interaktive Turingmaschine (ITM)* ist eine probabilistische Turingmaschine mit zwei zusätzlichen Bändern, je einem read-only- und einem write-only-Kommunikationsband.

2.2 Nicht-Unterscheidbarkeit von Wahrscheinlichkeitsverteilungen

Um die Eigenschaften eines Zero-Knowledge-Beweises präzise zu beschreiben, soll in diesem Abschnitt der Begriff der *Nicht-Unterscheidbarkeit (indistinguishability)* zweier Wahrscheinlichkeitsverteilungen formal eingeführt werden [GMR89, Gol01, BJY97]. Hierzu ist es zunächst nötig, vernachlässigbare Funktionen zu definieren.

Definition 2.4 (Vernachlässigbare Funktion)**.** Eine Funktion $\mu\colon \mathbb{N} \to \mathbb{R}$ heißt *vernachlässigbar*, wenn Folgendes gilt:

$$\forall c \in \mathbb{N}, \exists n_0 \in \mathbb{N}, \text{ so dass } \forall n \geq n_0 : \mu(n) < n^{-c}.$$

Mithilfe der voranstehenden Definition ist es nun möglich, Wahrscheinlichkeitsverteilungen anhand ihrer Unterschiede zu klassifizieren. Es lassen sich hierzu verschiedene Grade der Nicht-Unterscheidbarkeit definieren.

Definition 2.5 (Nicht-Unterscheidbarkeit)**.** Sei $L \subseteq \{0,1\}^*$ eine Sprache. Seien außerdem $U = \{U_x\}_{x \in L}$ und $V = \{V_x\}_{x \in L}$ Familien von Wahrscheinlichkeitsverteilungen. Dann definiere:

- Die Familien U und V sind *perfekt nicht-unterscheidbar*, wenn $U = V$ gilt.
- Die Familien U und V sind *statistisch nicht-unterscheidbar*, wenn eine vernachlässigbare Funktion $\mu(\cdot)$ existiert, so dass für alle Turingmaschinen M und alle $x \in L$ gilt:

  $$|\Pr[M(U_x) = 1] - \Pr[M(V_x) = 1]| \leq \mu(|x|).$$

- Die Familien U und V sind *berechenbar nicht-unterscheidbar*, wenn eine vernachlässigbare Funktion $\mu(\cdot)$ existiert, so dass für alle probabilistischen, polynomialzeit-beschränkten Turingmaschinen M und alle $x \in L$ gilt:

  $$|\Pr[M(U_x) = 1] - \Pr[M(V_x) = 1]| \leq \mu(|x|).$$

2.3 Interaktive Beweissysteme

Zur Definition von Zero-Knowledge-Beweisen ist es zunächst sinnvoll, interaktive Beweissysteme einzuführen. 1989 stellten Goldwasser, Micali und Rackoff ihre Idee vor, Interaktivität in klassische, statische Beweise zu integrieren [GMR89].

Hierzu tauschen ein Beweiser und ein Verifizierer sogenannte Challenges und Responses aus. Der Beweiser hat das Ziel, dem Verifizierer eine Aussage zu beweisen, der Verifizierer stellt Fragen (*challenges*), der Beweiser versucht, den Verifizierer durch seine Antworten (*responses*) von der Wahrheit seiner Behauptung zu überzeugen. Die vorgestellten interaktiven Beweissysteme sind dabei probabilistisch, das heißt der Verifizierer kann nicht mit 100-prozentiger Sicherheit von der Wahrheit der Aussage des Beweisers überzeugt werden, es ist lediglich möglich, die Wahrscheinlichkeit beliebig an 100 % anzunähern. Dagegen soll die Wahrscheinlichkeit, dass ein Beweiser, dessen Behauptung unwahr ist, den Verifizierer von der Wahrheit der Aussage überzeugen kann, beliebig weit an 0 angenähert werden können. Um die Wahrscheinlichkeit von Falschausgaben des Beweises zu verringern, kann das Protokoll mehrfach wiederholt werden.

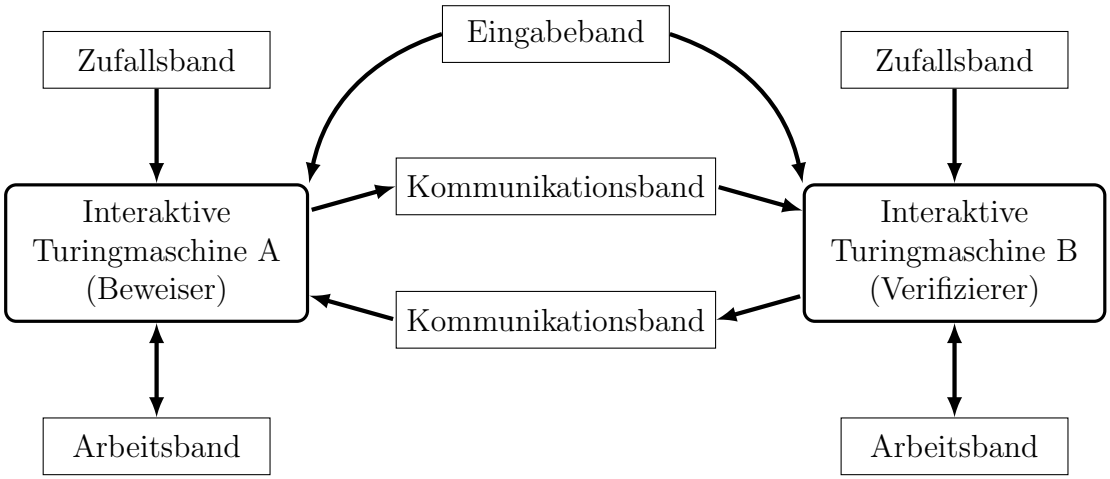

Abbildung 2.1: Interaktives Protokoll — eingehende Pfeile erlauben Lesezugriff, ausgehende Pfeile Schreibzugriff auf das zugehörige Band

Definition 2.6. Ein *interaktives Protokoll* ist ein geordnetes Paar von interaktiven Turingmaschinen (A, B). Das read-only-Kommunikationsband von Maschine A ist hierbei das write-only-Kommunikationsband von Maschine B, das write-only-Band von A ist dasselbe wie das read-only-Band von B. Außerdem teilen sich beide Maschinen ein Eingabeband.

Sei x die Eingabe auf dem Eingabeband. Dann soll A keinen komplexitätstheoretischen Einschränkungen unterliegen, Maschine B soll polynomiell-zeitbeschränkt in $|x|$ arbeiten.

Das Protokoll läuft dabei wie folgt ab: Es ist immer genau eine der Turingmaschinen aktiv, die andere ist untätig. Eine Phase besteht aus einer internen Berechnung unter Einbeziehung der zur Verfügung stehenden Bänder sowie dem abschließenden Schreiben einer Nachricht auf das zugehörige write-only-Kommunikationsband. Danach pausiert die Maschine und ihr Gegenstück wird aktiv.

Beginnend mit Maschine B arbeiten die interaktiven Turingmaschinen abwechselnd so lange, bis eine der Maschinen untätig wird, ohne eine Nachricht auf ihr Kommunikationsband geschrieben zu haben. Außerdem kann B das Protokoll durch eine „accept“- oder „reject“-Ausgabe terminieren.

Definition 2.7. Sei $L \subseteq \{0,1\}^*$ eine Sprache und (P, V) ein interaktives Protokoll. (P, V) ist ein *interaktives Beweissystem für* L, wenn es folgende Eigenschaften erfüllt:

- (Vollständigkeit). Ein interaktives Beweissystem (P, V) ist *vollständig*, wenn V für eine gegebene Eingabe $x \in L$ terminiert und mit einer Wahrscheinlichkeit von mindestens $1 - |x|^{-k}$ akzeptiert (für jedes beliebige $k > 0$).

- (Korrektheit). Ein interaktives Beweissystem (P^*, V) ist *korrekt*, wenn V für eine gegebene Eingabe $x \notin L$ und jeder beliebigen ITM P^* terminiert und mit einer Wahrscheinlichkeit von höchstens $|x|^{-k}$ akzeptiert (für jedes beliebige $k > 0$).

Führen also ein ehrlicher Beweiser, das heißt ein Beweiser, der eine wahre Aussage beweisen möchte, und ein ehrlicher Verifizierer das Protokoll durch, soll der Verifizierer mit hoher Wahrscheinlichkeit akzeptieren. Ist der Beweiser unehrlich, versucht also eine falsche Aussage zu beweisen, so soll der Verifizierer mit hoher Wahrscheinlichkeit ablehnen.

Erfüllt ein interaktiver Beweis diese beiden Eigenschaften, wird er häufig als *Proof of Knowledge* bezeichnet [Sim02].

2.4 Zero-Knowledge-Beweissysteme

Interaktive Zero-Knowledge-Protokolle sind besondere interaktive Beweissysteme, die zusätzlich noch eine weitere Eigenschaft erfüllen, die Zero-Knowledge-Eigenschaft. Während des Beweises sollen keine Informationen über das Geheimnis des Beweisers preisgegeben werden. Diese Informationen umfassen jegliches Wissen, mithilfe dessen der Verifizierer nach der Kommunikation mit dem Beweiser Berechnungen effizienter durchführen könnte als zuvor. Diese Eigenschaft lässt unter Zuhilfenahme eines Simulators formalisieren.

Definition 2.8. Sei (P, V) ein interaktives Beweissystem mit Eingabe x. Ein *Transkript* $view(x)$ bezeichnet die Gesamtheit an Daten, die während des Ablaufs des interaktiven Protokolls anfallen. Es beinhaltet ausschließlich die Eingabe x, die fairen Münzwürfe von Beweiser und Verifizierer sowie die zwischen P und V ausgetauschten Nachrichten.

Definition 2.9. Ein *Simulator* M ist eine probabilistische, in Polynomialzeit laufende Turingmaschine, die ohne den Beweiser P ein Transkript des Beweises anfertigt. Das entstehende Transkript ist nicht von einem authentischen, während eines Beweises mit P entstandenem Transkript zu unterscheiden.

Sowohl die echten, als auch die simulierten Transkripte werden von probabilistischen Turingmaschinen erzeugt und besitzen daher eine Wahrscheinlichkeitsverteilung. Diese sind nicht voneinander unterscheidbar. Da eine solche Verteilung ohne Wissen des Geheimnisses simuliert werden konnte, bedeutet dies im Umkehrschluss, dass in den echten Transkripten ebenfalls keinerlei Informationen über das Geheimnis enthalten sein können. Daher kann nun die Zero-Knowledge-Eigenschaft für interaktive Beweissysteme auch formal begründet werden.

Definition 2.10. Sei L eine Sprache und $x \in L$. Ein interaktives Beweissystem (P, V) für L hat die Zero-Knowledge-Eigenschaft, wenn für jeden beliebigen Verifizierer V ein Simulator M für den durchgeführten Beweis existiert, das

heißt die beiden Familien von Wahrscheinlichkeitsverteilungen $\{\text{view}(x)\}_{x\in L}$ und $\{M(x)\}_{x\in L}$ nicht-unterscheidbar sind.

Die Zero-Knowledge-Eigenschaft kann in drei unterschiedlich starke Kategorien eingeteilt werden.

- Ein interaktives Beweissystem liefert *perfect zero-knowledge*, wenn die Wahrscheinlichkeitsverteilungen von echten und simulierten Transkripten identisch sind, das heißt $\text{view}(x) = M(x)$.
- Ein interaktives Beweissystem liefert *statistical zero-knowledge*, wenn nach Definition 2.5 gilt, dass die Familien von Wahrscheinlichkeitsverteilungen $\{\text{view}(x)\}_{x\in L}$ und $\{M(x)\}_{x\in L}$ statistisch nicht-unterscheidbar sind.
- Ein interaktives Beweissystem liefert *computational zero-knowledge*, wenn nach Definition 2.5 gilt, dass die Familien von Wahrscheinlichkeitsverteilungen $\{\text{view}(x)\}_{x\in L}$ und $\{M(x)\}_{x\in L}$ berechenbar nicht-unterscheidbar sind.

Ablauf eines Zero-Knowledge-Beweises

Auf Basis der eingeführten Eigenschaften eines interaktiven Zero-Knowledge-Beweissystems soll nun ein allgemeiner Überblick über den grundlegenden Ablauf eines Zero-Knowledge-Protokolls gegeben werden. Am Protokoll nehmen zwei Parteien teil:

- Der *Beweiser* P hat das Ziel, einen Verifizierer von der Kenntnis einer Information zu überzeugen. Dabei darf er sein Geheimnis s allerdings nicht preisgeben.
- Der *Verifizierer* V hat die Aufgabe, zu überprüfen, ob P das vorgegebene Wissen nachweisen kann. Dafür stellt V eine Reihe von Aufgaben, die für P nur dann lösbar sind, wenn er das Geheimnis tatsächlich kennt. Kennt P es nicht, könnte er nur durch Zufall eine korrekte Antwort auf die gestellte Aufgabe geben.

Eine Beweisrunde besteht bei einem Zero-Knowledge-Beweis in der Regel aus drei Schritten. Zunächst legt sich P auf einen zufälligen Wert aus einer zuvor gewählten Menge fest. Dieser wird durch ein sogenanntes *Commitment-Verfahren* verschleiert und an V übermittelt. V kann nun eine *Challenge*, häufig ein Bit $c \in \{0, 1\}$, wählen und an P schicken. P kann unter Einsatz seines Geheimnisses s auf die Frage c antworten, mit einer sogenannten *Response* r. Ist die Antwort auf die Frage korrekt, muss V akzeptieren, ist sie falsch, wird das Protokoll abgebrochen und P ist als Betrüger entlarvt. Der Einsatz von zufälligen Werten auf Seiten des Beweisers und des Verifizierers sorgen dafür, dass ein betrügender P^* mit 50-prozentiger Wahrscheinlichkeit eine korrekte Lösung liefern kann, obwohl er das

Geheimnis s nicht kennt. Er muss lediglich erraten, welche Challenge c ihm von V gestellt wird. Trotzdem ist der Einsatz von Zufall unabdingbar, da er dafür sorgt, dass nicht alle Beweisrunden gleich ablaufen. Um die Wahrscheinlichkeit zu verringern, dass P^* zufällig eine korrekte Lösung rät, werden Zero-Knowledge-Beweise daher in mehreren Runden ausgeführt. Somit kann die Wahrscheinlichkeit für einen Betrug durch P^* durch einem Beweis mit k Runden auf 2^{-k} reduziert werden.

Merkmale eines Zero-Knowledge-Beweises

Zusammenfassend sollen an dieser Stelle nun die zuvor erarbeiteten Eigenschaften von Zero-Knowledge-Protokollen präsentiert werden [Sim02].

- Wird der Beweis hinreichend häufig wiederholt, kann ein betrügender Beweiser, der keine Kenntnis über das Geheimnis besitzt, den Verifizierer nur mit vernachlässigbar geringer Wahrscheinlichkeit davon überzeugen, das Geheimnis zu kennen.
- Der Verifizierer lernt während des Protokolls nicht über das Geheimnis des Beweisers. Selbst ein betrügender Verifizierer, der sich nicht an den Ablauf des Protokolls hält, erhält keine Informationen über das Geheimnis, außer, dass P den Beweis tatsächlich kennt.
- Der Verifizierer kann Dritte nicht davon überzeugen, dass P tatsächlich ein Geheimnis kennt, da sämtliche während des Protokolls angefallene Aufzeichnungen auch ohne P hätten simuliert werden können.

2.5 Komplexitätsklassen

Definition 2.11. Sei M eine deterministische Turingmaschine und p ein Polynom. Eine Sprache L liegt in *Nondeterminstic Polynomial Time* (**NP**), wenn für alle x gilt:

- $x \in L \iff \exists w$, so dass M bei Eingabe $\langle x, w \rangle$ akzeptiert.
- M arbeitet bei Eingabe $\langle x, w \rangle$ in Zeit $p(|x|)$.

Definition 2.12. Eine Sprache L liegt in *Interactive Polynomial Time* (**IP**), wenn für L ein interaktives Beweissystem existiert. Es gilt: **IP** = **PSPACE** [Sha90].

Definition 2.13. Eine Sprache L liegt in *Bounded-Error Polynomial Probabilistic Time* (**BPP**), wenn eine PTM M existiert, so dass:

- Es existiert ein Polynom f, so dass M nach höchstens $f(|x|)$ Schritten für alle Eingaben x hält,

- Wenn $x \in L$, dann $\Pr[M(x) = 1] \geq \frac{2}{3}$,
- Wenn $x \notin L$, dann $\Pr[M(x) = 1] < \frac{1}{3}$.

Goldreich, Micali und Wigderson stellten 1986 die Behauptung auf, dass für jedes Problem in **NP** ein zugehöriges Zero-Knowledge-Beweissystem existieren müsse [GMW86]. Unter der Voraussetzung, das sichere Verschlüsselungsfunktionen existieren, präsentierten sie dazu ein Zero-Knowledge-Beweissystem des **NP**-vollständigen Problems der *3-Färbbarkeit* von Graphen. Durch eine Polynomialzeitreduktion lässt sich jedes Problem in **NP** auf das Problem der 3-Färbbarkeit reduzieren.

Satz 2.14. Wenn theoretisch sichere Verschlüsselungsfunktionen existieren, dann existiert für jedes Problem in **NP** ein Zero-Knowledge-Beweissystem.

Auf die Vorstellung des von Goldreich, Micali und Wigderson geführten Beweises soll an dieser Stelle verzichtet werden. Anstatt ein Zero-Knowledge-Beweissystem für das Problem der 3-Färbbarkeit anzugeben, wird in Abschnitt 3.2 ein Zero-Knowledge-Beweissystem für das ebenfalls **NP**-vollständige Entscheidungsproblem **SAT** angegeben.

3 Zero-Knowledge-Protokolle

3.1 Fiat-Shamir-Protokoll

Zunächst soll das wohl bekannteste Zero-Knowledge-Verfahren vorgestellt werden, das von Amos Fiat und Adi Shamir entwickelte *Fiat-Shamir-Prokotoll*.

Definition 3.1. Für jede natürliche Zahl $n > 0$ umfasst die Menge $\mathbb{Z}_n^*$ alle zu n teilerfremden ganzen Zahlen von 1 bis $n-1$. Eine Zahl $z \in \mathbb{Z}_n^*$ ist ein *quadratischer Rest modulo n*, wenn ein $x \in \mathbb{Z}_n^*$ existiert, so dass $z \equiv x^2 \pmod{n}$ gilt. x wird auch als Quadratwurzel von $z \pmod{n}$ bezeichnet.

Protokoll

Das Fiat-Shamir-Protokoll lässt sich in zwei Phasen aufteilen: Die *Schlüsselgenerierungsphase* und die *Anwendungsphase*.

In der *Schlüsselgenerierungsphase* werden, wahlweise vom Beweiser P oder einer vertrauenswürdigen, dritten Partei zwei große, verschiedene Primzahlen p und q sowie deren Produkt $n = p \cdot q$ generiert. n wird öffentlich gemacht, die Primfaktoren dürfen jedoch nicht preisgegeben werden. Außerdem wählt P eine zufällige Zahl $s \in \mathbb{Z}_n^*$, hierbei handelt es sich um das Geheimnis, von dessen Kenntnis der Verifizierer V überzeugt werden soll. Zuletzt berechnet P den quadratischen Rest $z = s^2 \pmod{n}$ und übermittelt z dem Verifizierer.

V arbeitet polynomiell-zeitbeschränkt und kann daher durch z kein Wissen über das Geheimnis s des Beweisers erlangen. Es lässt sich zeigen, dass die Berechnung von Quadratwurzeln modulo n genauso schwierig wie das Faktorisieren von n ist, sofern n ein Produkt zweier verschiedener Primzahlen ist [BSW95, S. 122]. Es ist bekannt, dass das Problem der Primfaktorzerlegung in der Klasse **NP** liegt, ob ein effizienter, in Polynomialzeit arbeitender Algorithmus zur Lösung des Problems existiert, ist allerdings unklar.

Es sei anzumerken, dass es möglich ist, Quadratwurzeln modulo n in Polynomialzeit zu berechnen, wenn es sich beim Modulus um eine Primzahl handelt. Daher ist es notwendig, p und q geheimzuhalten.

In der *Anwendungsphase* versucht P, einen beliebigen Verifizierer V unter Geheimhaltung von s von der Kenntnis einer Quadratwurzel von z zu überzeugen.

Beweiser P		**Verifizierer** V
geg.: n, z, s		geg.: n, z
(1) Wähle zufällige Zahl $r \in \mathbb{Z}_n^*$, wähle zufällig $b \in \{1, -1\}$, berechne $x \equiv b \cdot r^2 \pmod{n}$.	$\xrightarrow{x}$	
		(2) Wähle zufälliges Bit $c \in \{0, 1\}$.
	$\xleftarrow{c}$	
(3) $y := \begin{cases} r, & \text{wenn } c = 0 \\ r \cdot s, & \text{wenn } c = 1. \end{cases}$		
	$\xrightarrow{y}$	
		(4) Prüfe, ob $y^2 \equiv x \cdot z^c \pmod{n}$ oder $y^2 \equiv -x \cdot z^c \pmod{n}$.

Abbildung 3.1: Fiat-Shamir-Protokoll

1. P wählt eine zufällige Zahl $r \in \mathbb{Z}_n^*$ und bildet daraus einen quadratischen Rest $x \equiv r^2 \pmod{n}$. Außerdem wird zufällig ein Vorzeichen $b \in \{1, -1\}$ für x gewählt. x wird an V übermittelt. Die Wahl eines zufälligen Vorzeichens b verhindert, dass ein Bit von $r \pmod{n}$ preisgegeben wird.
2. V sendet ein zufällig gewähltes Bit $c \in \{0, 1\}$ zum Beweiser P.
3. P muss nun eine Quadratwurzel offenlegen.
   - $b = 0$: P schickt r an V, da $r^2 \equiv x \pmod{n}$
   - $b = 1$: P schickt $r \cdot s$ an V, da $(r \cdot s)^2 \equiv x \cdot z \pmod{n}$
4. Sei y der empfangene Wert. V überprüft, ob entweder $y^2 \equiv x \cdot z^c \pmod{n}$ oder $y^2 \equiv -x \cdot z^c \pmod{n}$ durch das gesendete y erfüllt wird. Gilt eine der beiden Gleichungen, beginnt die nächste Beweisrunde. Gelten die Gleichungen nicht, stoppt der Verifizierer und bricht das Protokoll ab.

Der vorgestellte Ablauf beschreibt eine einzige Beweisrunde. Insgesamt werden die Schritte k-mal wiederholt. Ein betrügender Beweiser hat in jeder Runde eine 50-prozentige Chance, das Zufallsbit c des Verifizierers zu erraten und eine passende Lösung zu übermitteln, es ist ihm allerdings niemals möglich, beide Lösungen zugleich zu ermitteln.

Behauptung 3.2. Das präsentierte Protokoll arbeitet korrekt, vollständig und besitzt die Zero-Knowledge-Eigenschaft.

Beweis. Es wird gezeigt, dass das vorgestellte Protokoll die Eigenschaften *Vollständigkeit*, *Korrektheit* und die *Zero-Knowledge-Eigenschaft* aufweist.

- (*Vollständigkeit*). Kennt P das Geheimnis s, kann er V davon überzeugen, da in $\mathbb{Z}_n^*$ folgendes gilt:

$$y^2 \equiv (r \cdot s^c)^2 \equiv r^2 \cdot s^{2c} \equiv r^2 \cdot (s^2)^c \equiv x \cdot z^c \pmod{n}.$$

- (*Korrektheit*). Kennt P das Geheimnis s nicht, kann P die Gleichung $y^2 \equiv x \cdot z^c \pmod{n}$ beziehungsweise $y^2 \equiv -x \cdot z^c \pmod{n}$ nur für ein einziges c erfüllen. Vermutet P, dass $c = 0$, geht der Beweiser wie dargestellt vor und übermittelt die Quadratwurzel von r inklusive des Vorzeichens b. Es gilt

$$y^2 \equiv r^2 \equiv x \pmod{n} \text{ oder } y^2 \equiv r^2 \equiv -x \pmod{n}$$

  Vermutet er, dass $c = 1$, übermittelt er stattdessen $x' = x \cdot z^{-1}$ und das Vorzeichen b. Um $y^2 \equiv x' \cdot z = x \cdot z^{-1} \cdot z = x \pmod{n}$ in Schritt (4) zu erfüllen, muss in Schritt (3) nur noch $y := r$ übermittelt werden, das Geheimnis benötigt P nicht.
  Da das Protokoll bereits bei einem einzigen y abbricht, das keine der Gleichungen in Schritt (4) erfüllt, sinkt die Wahrscheinlichkeit für ein erfolgreiches Betrügen von P bei k Durchgängen auf 2^{-k} und kann durch mehrfache Wiederholung der Schritte beliebig reduziert werden.

- (*Zero-Knowledge-Eigenschaft*). Ein Simulator S lässt sich wie folgt definieren, um einen Dialog mit V zu simulieren.
  1. S wählt ein zufälliges $r \in \mathbb{Z}_n^*$ sowie ein zufälliges Bit $c' \in \{0, 1\}$. $x \equiv r^2 \cdot z^{-c'} \pmod{n}$ wird an V übermittelt.
  2. V wählt ein zufälliges Bit $c \in \{0, 1\}$ und sendet es an S.
  3. Ist $c = c'$, kann S mit $y := r$ eine korrekte Antwort liefern. Eine Überprüfung durch V wäre erfolgreich, da gilt:

$$x \cdot z^c \equiv r^2 \cdot z^{-c} \cdot z^c \equiv r^2 \equiv y^2 \pmod{n}$$

  4. Ist $c \neq c'$, lösche die Runde.

Das während der Kommunikation mit S entstehende Transkript ist von einem echten Transkript nicht zu unterscheiden, daher gilt die Zero-Knowledge-Eigenschaft. □

3.2 SAT

SAT ist eines der bekanntesten **NP**-vollständigen Probleme. Daher soll hier, ausgehend von der Arbeit von Gilles Brassard und Claude Crépeau, ein interaktives Zero-Knowledge-Protokoll für **SAT** vorgestellt werden [BC86].

Definition 3.3. Die Sprache **SAT** (*satisfiability*) wird definiert als

$$\mathbf{SAT} = \{\langle\varphi\rangle \mid \varphi \text{ ist eine aussagenlogische, erfüllbare Formel}\}.$$

Bevor ein Zero-Knowledge-Protokoll für **SAT** angegeben werden kann, sind vorab einige Voraussetzungen einzuführen.

Zu Beginn soll eine Methode präsentiert werden, mit der einzelne Bits verschlüsselt werden können.

Definition 3.4 (Verschlüsseln eines Bits)**.** Seien p und q verschiedene, große Primzahlen und $n = p \cdot q$ deren Produkt. Des Weiteren sei y ein zufällig gewählter quadratischer Rest modulo n. Die Quadratwurzel von y sei nicht bekannt.

Ein Beweiser P hat nun das Ziel, ein Bit b zu verschlüsseln. Dazu wählt er ein zufälliges, geheim zu haltendes $w \in \mathbb{Z}_n^*$ und berechnet unter dessen Einsatz $z \equiv w^2 \cdot y^b \pmod{n}$. Diese Verschlüsselung z enthält dann keine Information über das ursprüngliche Bit b.

Möchte P die Verschlüsselung für einen Verifizier V öffnen, ist dies möglich, indem er eine Quadratwurzel präsentiert. Ist $b = 0$, das heißt $z \equiv w^2 \pmod{n}$, ist w eine Quadratwurzel von z. Ist $b = 1$, das heißt $z \equiv w^2 \cdot y \pmod{n}$, ist w eine Quadratwurzel von $z \cdot y^{-1} \pmod{n}$.

Da das finalen Protokoll eine Methode erfordert, in der keine geheimen Bits aufgedeckt werden dürfen, wird nun ein Verfahren vorgestellt, das den Vergleich zweier bereits verschlüsselter Bits ermöglicht, ohne die ursprünglichen Bits aufzudecken.

Definition 3.5 (Vergleich zweier verschlüsselter Bits)**.** Seien b_1 und b_2 je ein Bit. Dann sind $z_1 \equiv w_1^2 \cdot y^{b_1} \pmod{n}$ und $z_2 \equiv w_2^2 \cdot y^{b_2} \pmod{n}$ deren Verschlüsselungen nach obigem Verfahren (vergleiche Definition 3.4), w_1 und w_2 die Zeugnisse von z_1 beziehungsweise z_2. Es sei außerdem $z = z_1 \cdot z_2 \pmod{n}$.

Ein Beweiser P hat nun das Ziel, einen Verifizier V von der Gleichheit beziehungsweise Ungleichheit der Bits b_1 und b_2 zu überzeugen, ohne die Bits selbst preiszugeben. Um dies zu zeigen, muss ein w wie folgt gewählt werden.

$$w := \begin{cases} w_1 \cdot w_2 \cdot y \pmod{n}, & \text{wenn } b_1 = 1, b_2 = 1 \\ w_1 \cdot w_2, & \text{sonst.} \end{cases}$$

Ist $b_1 = b_2$, dann gilt $z = w^2 \pmod{n}$, ist $b_1 \neq b_2$, dann gilt $z = w^2 \cdot y \pmod{n}$. Möchte P die Gleichheit von b_1 und b_2 darlegen, kann durch Preisgabe von w die

Kenntnis einer Quadratwurzel von z belegen, sind b_1 und b_2 verschieden, liefert w eine Quadratwurzel von $z \cdot y^{-1} \pmod{n}$. Die Bits b_1 und b_2 selbst werden dabei nicht preiszugeben.

Das Verfahren in Definition 3.5 kann selbstverständlich für beliebig lange Strings genutzt werden, hierfür wird das vorgestellte Verfahren bitweise auf den String angewendet. Ist das Ziel jedoch, die Ungleichheit zweier Strings zu beweisen, ohne dabei Informationen über die Zeichenketten zu verbreiten, ist ein bitweiser Vergleich der Strings nicht ohne Informationsfluss möglich. Es wäre bei Anwendung des Verfahrens aus Definition 3.5 für den Verifizierer ersichtlich, an welchen Indizes sich die Zeichenketten unterscheiden.

Protokoll: Ungleichheit von Strings

Daher soll nun ein Protokoll gezeigt werden, das es ermöglicht, die Ungleichheit zweier Zeichenketten $s = s_1 s_2 ... s_k$ und $s' = s'_1 s'_2 ... s'_k$ zu beweisen, ohne Informationen über die Strings, wie beispielsweise einen Index, an dem sich s und s' unterscheiden, veröffentlichen zu müssen. Dazu werden die Strings durch den Beweiser P bitweise zu $z = z_1 z_2 ... z_k$ und $z' = z'_1 z'_2 ... z'_k$ verschlüsselt, die dafür notwendigen Zeugnisse sind $w_1, w_2, ..., w_k$ sowie $w'_1, w'_2, ..., w'_k$. Es sei außerdem $v_i = z_i \cdot z'_i \pmod{n}$ für alle $i \in \{1, 2, ..., k\}$.

1. Der Beweiser P wählt eine zufällige Permutation σ aus $\{1, 2, ..., k\}$ und zufällige $x_i \in \mathbb{Z}_n^*$ für alle $i \in \{1, 2, ..., k\}$. Er berechnet $a_i = x_i^2 \cdot v_{\sigma(i)} \pmod{n}$ und übermittelt diese an den Verifizierer V.

2. Der Verifizierer V wählt ein zufälliges Bit $c \in \{0, 1\}$ und sendet es an P.

3. Je nach gewähltem c muss P nun eine der folgenden Aufgaben erfüllen:
   - $c = 0$: Der Beweiser P gibt ein bestimmtes i preis, so dass a_i eine Stelle verschlüsselt, an der sich s und s' unterscheiden, und öffnet die Verschlüsselung durch Übermittlung von $w = x_i \cdot w_{\sigma(i)} \cdot w'_{\sigma(i)} \pmod{n}$. Der Verifizierer V erfährt dabei nichts über die Position, an der sich s und s' unterscheiden, da die wahre Position durch σ permutiert wurde.
   - $c = 1$: Der Beweiser P gibt die Permutation σ preis und zeigt mithilfe des Verfahrens aus Definition 3.5, dass a_i und $v_{\sigma(i)}$ das gleiche Bit verschlüsseln für jedes $i \in \{1, 2, ..., k\}$.

4. Der Verifizierer V überprüft die Nachricht von P:
   a) $c = 0$: V prüft, ob $a_i = w^2 \cdot y \pmod{n}$ erfüllt ist.
   b) $c = 1$: V prüft mithilfe des Verfahrens aus Definition 3.5 für jedes $i \in \{1, 2, ..., k\}$, ob a_i und $v_{\sigma(i)}$ das gleiche Bit verschlüsseln.

Wird die jeweilige Gleichung nicht erfüllt, bricht V das Protokoll ab, hat P eine korrekte Antwort geliefert, beginnt die nächste Runde des Protokolls.

Das Protokoll besteht aus r Runden, so dass die Wahrscheinlichkeit für ein erfolgreiches Betrügen von P auf 2^{-r} gesenkt werden kann.

Behauptung 3.6. Das präsentierte Protokoll bildet ein interaktives Zero-Knowledge-Beweissystem.

Beweis. Es wird gezeigt, dass das vorgestellte Protokoll die Eigenschaften *Vollständigkeit*, *Korrektheit* und die *Zero-Knowledge-Eigenschaft* aufweist.

- (*Vollständigkeit*). Seien s und s' zwei verschiedene Zeichenketten. Dann kann P, wenn er s und s' kennt, V immer davon überzeugen, dass die Strings an mindestens einer Position unterscheiden, da Folgendes gilt:

$$a_i \equiv x_i^2 \cdot v_{\sigma(i)} \equiv x_i^2 \cdot w_{\sigma(i)}^2 \cdot w_{\sigma(i)}'^2 \cdot y \equiv w^2 \cdot y \pmod{n},$$

  mit $v_i \equiv w_i^2 \cdot w_i'^2 \cdot y \pmod{n}$, wenn $s_i \neq s_i'$. Das Zeugnis w ist also eine Quadratwurzel von $a_i \cdot y^{-1} \pmod{n}$.

- (*Korrektheit*). Sei s und s' zwei identische Zeichenketten, es gelte demnach $s_i = s_i'$ für alle $i \in \{1, 2, ..., k\}$. Dann kann P den Verifizierer eine Runde des Protokolls nur mit einer Wahrscheinlichkeit von $\frac{1}{2}$ erfolgreich abschließen, bei r Runden reduziert sich die Wahrscheinlichkeit auf 2^{-r}. Dazu müsste P in jeder Runde die korrekte Challenge c erraten. Wählt V das Bit $c = 0$, muss P ein a_i so wählen, dass a_i und $v_{\sigma(i)}$ unterschiedliche Bits verschlüsseln. Erwartet P, dass $c = 1$, wählt er eine korrekte Permutation σ, so dass a_i und $v_{\sigma(i)}$ für alle $i \in \{1, 2, ..., k\}$ dieselben Bits verschlüsseln.

- (*Zero-Knowledge-Eigenschaft*). Ein Simulator S lässt sich wie folgt definieren, um einen Dialog mit V zu simulieren.

  1. S wählt eine zufällige Permutation σ und zufällige $x_i \in \mathbb{Z}_n^*$ für alle $i \in \{1, 2, ..., k\}$ sowie ein zufälliges Bit $c' \in \{0, 1\}$. Er berechnet folgend $a_i = x_i^2 \cdot v_{\sigma(i)} \pmod{n}$ und übermittelt diese an den Verifizierer V.
  2. V wählt ein zufälliges Bit $c \in \{0, 1\}$ und sendet es an S.
  3. Ist $c = c'$, kann S eine korrekte Antwort liefern.
  4. Ist $c \neq c'$, lösche die Runde.

  Das während der Kommunikation mit S entstandene Transkript ist von einem echten Transkript nicht zu unterscheiden, daher gilt die Zero-Knowledge-Eigenschaft.

□

Protokoll: Boole'sche Funktionen in Zero-Knowledge

Definition 3.7. Eine *permutierte Wahrheitstabelle* für die boole'sche Funktion $B\colon \{0,1\}^t \to \{0,1\}$ ist ein binärer String, der sämtliche mögliche Belegungen von B sowie das Ergebnis jeder Belegung enthält. Der String wird hierzu in 2^t Blöcke unterteilt, die ersten t Bits definieren die Belegung, das letzte Bit eines Blocks das Ergebnis der jeweiligen Belegung.

Sei B eine erfüllbare boole'sche Funktion und $b_1, b_2, ...b_t$ eine erfüllende Belegung. Dann kann ein Verifizierer mit folgendem Protokoll ohne Veröffentlichung der b_i von der Erfüllbarkeit der Funktion überzeugt werden.

1. P übermittelt eine Verschlüsselung z_i für jedes b_i $(1 \leq i \leq k)$ an V. P wählt außerdem eine zufällige permutierte Wahrheitstabelle der booleschen Funktion B, verschlüsselt die Bits des Strings und sendet die verschlüsselten Bits ebenfalls an V.

2. V schickt ein zufällig gewähltes Bit $c \in \{0,1\}$ an P.

3. Ist $c = 0$, öffnet P die gesamte Verschlüsselung der permutierten Wahrheitstabelle und schickt diese an V. Ist $c = 1$, wählt P den (permutierten) Block aus, der die Belegung $b_1, b_2, ..., b_t$ und das Ergebnis b darstellt, und schickt ihn an V.

4. Ist $c = 0$, überprüft V, ob es sich tatsächlich um eine permutierte Wahrheitstabelle handelt. Ist $c = 1$, überprüft V, ob die Verschlüsselung $z_1, z_2, ...z_t$ der Belegung und die Verschlüsselung des Blocks der permutierten Wahrheitstabelle identisch sind.

Das Protokoll kann k mal wiederholt werden, um die Wahrscheinlichkeit für einen erfolgreichen Betrug von P auf 2^{-k} zu reduzieren.

Um eine erfüllende Belegung einer boole'schen Formel $\varphi\colon \{0,1\}^k \to \{0,1\}$ ohne Veröffentlichen der Belegung nachzuweisen, kann der Beweiser P mit dem Verifizierer V für jeden Operator in φ das zuvor präsentierte Protokoll durchführen und zuletzt zeigen, dass die Formel φ durch die Belegung erfüllt wird.

3.3 Graphenisomorphie

Ausgehend von einem von Gerardo I. Simari vorgestellten Protokoll soll ein Zero-Knowledge-Beweissystem der Sprache **GI** dargelegt werden [Sim02].

Definition 3.8. Zwei Graphen $G_0 = (V_0, E_0)$ und $G_1 = (V_1, E_1)$ sind zueinander *isomorph*, wenn eine bijektive Abbildung $\pi\colon V_0 \to V_1$ existiert, so dass

$$(u, v) \in E_0 \iff (\pi(u), \pi(v)) \in E_1$$

gilt. Eine solche Abbildung π wird als *Isomorphismus* zwischen den Graphen G_0 und G_1 bezeichnet.

Sind die beiden Knotenmengen V_0 und V_1 identisch, lässt sich π vereinfacht als Permutation der Knotenmenge V_0 auffassen. Man schreibe $G_1 = \pi(G_0)$.

Definition 3.9. Die Sprache **GI** (*graph isomorphism*) wird definiert als

$$\mathbf{GI} = \{\langle G_0, G_1\rangle \mid G_0 \text{ und } G_1 \text{ sind zueinander isomorphe Graphen}\}.$$

Protokoll

Seien $G_0 = (V_0, E_0)$ und $G_1 = (V_1, E_1)$ zwei Graphen und σ ein Isomorphismus zwischen G_0 und G_1 mit $V_0 = V_1$, das heißt $G_1 = \sigma(G_0)$. Sei dieser Isomorphismus das Geheimnis des Beweisers P. Folgendes Protokoll ermöglicht es P, einen beliebigen Verifizierer V unter Geheimhaltung von σ von der Kenntnis des Isomorphismus zu überzeugen.

1. P wählt eine zufällige Permutation π der Menge $\{1, 2, ..., |V_0|\}$ sowie ein Bit $b \in \{0, 1\}$ und generiert den Graphen $H = \pi(G_b)$. Der Graph H wird an V übermittelt.

2. V sendet ein zufällig gewähltes Bit $c \in \{0, 1\}$ zum Beweiser P.

3. P muss nun einen Isomorphismus zwischen G_c und H offenlegen.
   - $b = c$: P schickt π an V, da $H = \pi(G_b)$.
   - $b = 0$, $c = 1$: P schickt $\pi \circ \sigma^{-1}$ an V, da $H = \pi(G_0) = \pi(\sigma^{-1}(G_1))$.
   - $b = 1$, $c = 0$: P schickt $\pi \circ \sigma$ an V, da $H = \pi(G_1) = \pi(\sigma(G_0))$.

4. Sei ρ die empfangene Permutation. V überprüft, ob es sich bei ρ um einen Isomorphismus zwischen G_c und H handelt. Gilt $H = \rho(G_c)$, beginnt die nächste Beweisrunde, ist $H \neq \rho(G_c)$, stoppt der Verifizierer und bricht das Protokoll ab.

Beweiser P geg.: G_0,G_1,σ		**Verifizierer** V geg.: G_0,G_1
(1) Wähle zufällige Permutation π, wähle zufälliges Bit $b \in \{0,1\}$, berechne $H = \pi(G_b)$.		
	$\xrightarrow{H}$	
		(2) Wähle zufälliges Bit $c \in \{0,1\}$.
	$\xleftarrow{c}$	
(3) $\rho := \begin{cases} \pi, & \text{wenn } b = c \\ \pi \circ \sigma^{-1}, & \text{wenn } b = 0, c = 1 \\ \pi \circ \sigma, & \text{wenn } b = 1, c = 0. \end{cases}$		
	$\xrightarrow{\rho}$	
		(4) Prüfe, ob $H = \rho(G_c)$.

Abbildung 3.2: Zero-Knowledge-Protokoll des Graphenisomorphieproblems

Der vorgestellte Ablauf beschreibt eine einzige Beweisrunde. Insgesamt werden die Schritte k-mal wiederholt. Hat der Beweiser keine Kenntnis über die geheime Permutation σ, besteht in jeder Beweisrunde die Möglichkeit, dass P dennoch eine Permutation an V übermitteln kann, die der Überprüfung des Verifizierers standhält. Dies ist mit einer Wahrscheinlichkeit von $\frac{1}{2}$ möglich und geschieht immer dann, wenn das von P zufällig gewählte Bit dem von V gewählten Zufallsbit entspricht ($b = c$). In diesem Fall kann P die zufällig erzeugte Permutation π übermitteln und muss das ihm nicht bekannte Geheimnis σ nicht einsetzen. Da das Protokoll bereits bei einer einzigen Permutation abbricht, für die $H \neq \rho(G_c)$ gilt, sinkt die Wahrscheinlichkeit für ein erfolgreiches Betrügen von P bei k Durchgängen auf 2^{-k}.

Behauptung 3.10. Das präsentierte Protokoll bildet ein interaktives Zero-Knowledge-Beweissystem für die Sprache **GI**.

Beweis. Es wird gezeigt, dass das vorgestellte Protokoll die Eigenschaften *Vollständigkeit*, *Korrektheit* und die *Zero-Knowledge-Eigenschaft* aufweist.

- (*Vollständigkeit*). Sei $\langle G_0, G_1 \rangle \in \mathbf{GI}$. Dann ist H sowohl zu G_0 als auch zu G_1 isomorph. Der Verifizierer iteriert durch alle k Wiederholungen und akzeptiert mit einer Wahrscheinlichkeit von 1, da in jeder Runde des Protokolls gilt, dass $H = \pi(G_b) = \rho(G_c)$.

- (*Korrektheit*). Sei $\langle G_0, G_1 \rangle \notin \mathbf{GI}$. Dann ist der durch die zufällig gewählte Permutation π generierte Graph H entweder zu G_0 oder zu G_1 isomorph. $H = \pi(G_b) = \rho(G_c)$ gilt lediglich dann, wenn $b = c$. Dies ist nur mit einer Wahrscheinlichkeit von $\frac{1}{2}$ der Fall und kann durch mehrfache Wiederholung der Schritte beliebig reduziert werden.

- (*Zero-Knowledge-Eigenschaft*). Während einer Beweisrunde gibt der Beweiser P lediglich π oder $\pi \circ \sigma$ preis. Ein Simulator S lässt sich wie folgt definieren, um einen Dialog mit V zu simulieren.

  1. S wählt eine zufälliges Bit $b \in \{0, 1\}$ und erstellt mithilfe einer ebenfalls zufälligen Permutation π den Graphen $H = \pi(G_b)$. H wird an V übermittelt.
  2. V wählt ein zufälliges Bit $c \in \{0, 1\}$ und sendet es an S.
  3. Ist $c = b$, kann S mit $\rho := \pi$ einen gültigen Isomorphismus zwischen H und G_c liefern.
  4. Ist $c \neq b$, lösche die Runde.

  Das während der Kommunikation mit S entstandene Transkript ist von einem echten Transkript nicht zu unterscheiden, daher gilt die Zero-Knowledge-Eigenschaft.

□

Bemerkung 3.11. László Babai gelang es im Jahr 2017, einen *quasi-polynomiellen Algorithmus* für **GI** anzugeben [Bab16]. Die Zero-Knowledge-Eigenschaft des Protokolls bleibt dabei erhalten, da weiterhin keine Informationen über das Geheimnis des Beweisers an den Verifizierer übermittelt werden und ein Simulator die gesamte Interaktion simulieren könnte. Dennoch schwächt die Entdeckung des quasi-polynomiellen Algorithmus die Sicherheit des Protokolls in der praktischen Anwendung, da die Berechnung einer Isomorphie zwischen zwei Graphen nun einen geringeren Berechnungsaufwand erfordert. Potenzielle Angreifer könnten für die öffentlichen Graphen G_0 und G_1 die geheime Isomorphie des Beweisers P mit geringerem Berechnungsaufwand selbst berechnen und sich folgend als ebendieser ausgeben.

4 Nicht-interaktive Zero-Knowledge-Beweise

Die bisher vorgestellten Zero-Knowledge-Beweise waren ausnahmslos interaktiv. Die Kommunikation zwischen Beweiser und Verifizierer kann in der praktischen Anwendung jedoch problematisch sein, da eine stabile Verbindung während des gesamten Protokolls gehalten werden muss. Diese Interaktion benötigt während des Protokolls die meisten Ressourcen [BSMP91, S. 1086]. Daher stellt sich die Frage, ob es möglich ist, die Anwendbarkeit von Zero-Knowledge-Protokollen durch eine Abschwächung der bisherigen Bedingungen zu verbessern.

Interaktive Zero-Knowledge-Beweise haben folgende Eigenschaften:

- Der Beweiser und Verifizierer tauschen während des Protokolls Nachrichten aus.
- Der Verifizierer wirft in jeder Runde des Protokolls eine faire Münze, um die Challenge für den Beweiser festzulegen.
- Der Beweiser nutzt während des Beweises ein Problem, das der Verifizierer in Polynomialzeit nicht selbst lösen kann.

Blum, de Santis, Micali und Persiano fassten daher mehrere Ideen zusammen, die Interaktion zwischen dem Beweiser und dem Verifizierer zu beschränken, und formalisierten sie zu einer Definition von *nicht-interaktiven Zero-Knowledge-Beweisen* [BSMP91]. Darin soll die einzige Interaktion während eines Protokolls die Übermittlung des vom Beweiser erstellten Beweises stattfinden.

Goldreich und Oren [GO94] konnten zeigen, dass Zero-Knowledge-Beweise mit einem solch geringem Maß an Interaktion nur für *triviale* Sprachen, das heißt Sprachen in **BPP**, angegeben werden können. Diese Probleme eignen sich jedoch nicht zur Authentifikation eines Beweisers, da jeder Kommunikationsteilnehmer in der Lage ist, die Lösung des Problems mithilfe von probabilistischen Algorithmen selbst in Polynomialzeit zu ermitteln.

Blum et al. entwickelten daher ein nicht-interaktives Zero-Knowledge-Beweissystem, in dem der Beweiser und der Verifizierer zu Beginn des Protokolls eine zufällige Bitfolge, einen sogenannten *Common Random String*, von einer unabhängigen, dritten Partei erhalten. Durch diese lassen sich die ersten beiden Nachrichten eines interaktiven Zero-Knowledge-Beweissystems, das heißt das vom

Beweiser versendete Commitment und die darauf folgende, zufällige Challenge des Verifizierers, ersetzen. Der Beweiser kennt hierbei sämtliche Münzwürfe in Voraus, die Zero-Knowledge-Eigenschaft hängt lediglich vom Zufall des *Common Random Strings*, nicht aber von der Geheimhaltung und Unvorhersehbarkeit der Münzwürfe des Verifizieres ab. Ein solches Beweissystem lässt sich wie folgt formalisieren.

Definition 4.1. Sei $L \subseteq \{0,1\}^*$ eine Sprache und (P, V) ein Paar probabilistischer, polynomialzeit-beschränkter Turingmaschinen. (P, V) ist ein *nicht-interaktives Beweissystem für* L, wenn es folgende Eigenschaften erfüllt:

- (Vollständigkeit). Ein nicht-interaktives Beweissystem (P, V) ist *vollständig*, wenn V für eine gegebene Eingabe $x \in L$ terminiert und mit einer Wahrscheinlichkeit von mindestens $1-|x|^{-k}$ akzeptiert (für jedes beliebige $k > 0$).
- (Korrektheit). Ein nicht-interaktives Beweissystem (P^*, V) ist *korrekt*, wenn V für eine gegebene Eingabe $x \notin L$ und jeder beliebigen PTM M terminiert und mit einer Wahrscheinlichkeit von höchstens $|x|^{-k}$ akzeptiert (für jedes beliebige $k > 0$).

Der Beweiser erhält hierbei x sowie einen *Common Random String* σ als Eingabe und ermittelt daraus den *Beweis*. Diesen Beweis erhält der Verifizierer V zusätzlich zu x und σ und entscheidet auf Grund dessen entweder abzulehnen oder zu akzeptieren.

Analog zur Definition eines interaktiven Zero-Knowledge-Beweissystem kann auch für nicht-interaktive Beweise ein Simulator angegeben werden, durch den das Beweissystem die Zero-Knowledge-Eigenschaft erhält.

Definition 4.2. Sei $L \subseteq \{0,1\}^*$ eine Sprache und $x \in L$. Ein nicht-interaktives Beweissystem (P, V) hat die *Zero-Knowledge-Eigenschaft*, wenn für jeden beliebigen Verifizierer V ein Simulator M für den durchgeführten Beweis existiert, das heißt die beiden Familien von Wahrscheinlichkeitsverteilungen $\{\text{view}(x)\}_{x \in L}$ und $\{M(x)\}_{x \in L}$ ununterscheidbar sind.

5 Authentifikation durch Passwörter

Motivation

Ein stetig wachsender Anteil aller Seitenzugriffe im World Wide Web erfolgt über HTTPS-Verbindungen. Dabei kommt das Verschlüsselungsprotokoll TLS zum Einsatz, das für eine sichere Verbindung zwischen Server und Webbrowser sorgen soll. Zur Authentifikation der Kommunikationsteilnehmer werden von Certificate Authorities (CAs) ausgestellte Zertifikate eingesetzt, mit denen der Absender seine zu übermittelnden Nachrichten signiert.

Würden die Kommunikationsteilnehmer während der Interaktion nicht durch ein solches Zertifikat authentifiziert, wäre das Protokoll anfällig für Man-in-the-Middle-Angriffe. Ein Mithörender könnte während des Schlüsselaustauschs den öffentlichen Schlüssel des Servers abfangen und ihn durch den eigenen öffentlichen Schlüssel ersetzen. Der in der Antwort des Clients erhaltene öffentliche Schlüssel würde ebenfalls durch den eigenen öffentlichen Schlüssel ersetzt. Folgend könnte der Mithörer sowohl mit dem Client als auch mit dem Server einen gemeinsamen Schlüssel wählen. Die Kommunikation zwischen Client und Server erfolgt demzufolge immer über den Mithörenden, der alle versendeten Nachrichten mitlesen und modifizieren kann.

Das Sicherheit basiert hierbei auf der Verlässlichkeit der Certificate Authorities, nur vertrauenswürdige, verifizierte Domains sollten validiert werden. Wird eine Certificate Authority kompromittiert, ist keines der von dieser Certificate Authority ausgestelltes Zertifikat mehr verlässlich. Daher lässt sich die Überlegung anstellen, auf die Certificate Authorities aus vertrauenswürdige, dritte Partei zu verzichten und stattdessen ein dezentrales System zur Authentifikation einzusetzen.

Protokoll

Das im Folgenden vorgestellte Protokoll implementiert eine Möglichkeit, einen Client gegenüber einem Server durch die Eingabe eines Nutzernamens und eines Passworts zu authentifizieren, ohne dabei auf Zertifikate einer Certificate Authority zurückzugreifen. Das Protokoll hat darüber hinaus den Vorteil, dass das Pass-

wort den Webbrowser des Clients zu keinem Zeitpunkt verlässt. Das Versenden von Passwörtern über eine HTTPS-Verbindung mit TLS-Verschlüsselung bietet zwar ein Mindestmaß an Schutz vor dem Mitlesen des Passworts durch einen Angreifer und ist somit einer gänzlich unverschlüsselten Verbindung vorzuziehen, hat aber durchaus Schwachstellen. Eine fehlerhafte Implementierung des Protokolls kann schwerwiegende Sicherheitsrisiken zur Folge haben, das wohl bekannteste Sicherheitsleck ist der 2014 in der OpenSSL-Bibliothek entdeckte Fehler *Heartbleed* (CVE-2014-0160), der Angreifern das Abgreifen sensitiver Daten ermöglichte. Darüber hinaus suggeriert eine HTTPS-Verbindung einem alltäglichen Nutzer ein falsches Maß an Sicherheit. Kein Nutzer sollte aufgrund einer Übertragung via HTTPS sorglos sensible Daten versenden, ohne den Empfänger zweifelsfrei zu vertrauen.

Grzonkowski und Corcoran [GC14] entwickelten auf Grundlage des Zero-Knowledge-Beweissystems für **GI** in Abschnitt 3.3 ein Zero-Knowledge-Protokoll, das es ermöglicht, einen Client durch die Eingabe eines Passworts gegenüber einem Webserver zu authentifizieren, ohne dass jenes Passwort an den Server übermittelt werden muss.

Das Passwort wird hierbei im Browser des Clients in eine Permutation π umgewandelt. Dazu wird zunächst mittels einer Einwegfunktion der Hashwert des eingegebenen Passworts berechnet. Grzonkowski und Corcoran wählten dabei SHA-1 als Hashfunktion, das Passwort variabler Länge wird dementsprechend auf einen 160 Bit-Hexadezimalstring abgebildet. Die Kollisionsresistenz der gewählten Hashfunktion spielt dabei eine wichtige Rolle bei der Geheimhaltung des Passworts. Es sei an diesem Punkt anzumerken, dass es einem Team aus Sicherheitsforschern 2017 gelungen, für SHA-1 gezielt Kollisionen herbeizuführen [SBK+17]. Es sollte daher in Betracht gezogen werden, die SHA-1-Hashfunktion durch eine sichere, kollisionsresistente Alternative zu ersetzen, die weitere Struktur des Protokolls lässt dies problemlos zu.

Der daraus resultierende Bitstring lässt sich als Ganzzahl auffassen und soll im Folgenden als Seed für die Erstellung einer Permutation dienen. Hierzu wird der Hashwert in die Form $a_0 \cdot 0! + a_1 \cdot 1! + a_2 \cdot 2! + ... + a_k \cdot k!$ gebracht (für ein $k \in \mathbb{N}$).

Satz 5.1. Jede natürliche Zahl n kann für ein $k \in \mathbb{N}$ eindeutig durch

$$n = a_0 \cdot 0! + a_1 \cdot 1! + a_2 \cdot 2! + ... + a_k \cdot k!$$

dargestellt werden und wird auch als *fakultätsbasiertes Zahlensystem* bezeichnet. Jedes $n < k!$ lässt sich durch das Zahlensystem mit k Faktoren $a_1, a_2, ...a_k$ darstellen, darüber hinaus gilt immer, dass $a_0 = 0$.

Die dabei auftretenden Werte $a_0, a_1, a_2, ..., a_k$ lassen sich als Lehmer-Code interpretieren und in eine Permutation der Menge $\{0, 1, 2, ..., k\}$ decodieren. Dabei wird nach folgendem Algorithmus vorgegangen.

Algorithm 1 Decodieren eines Lehmer-Codes

```
1: function DECODE(a[ ])                          ▷ wobei a = [a_0, a_1, ..., a_k]
2:     for i ← 0 to k do
3:         s.insert(i)          ▷ Initialisierung der zu permutierenden Menge
4:     end for
5:     for i ← 0 to k do
6:         permutation[i] ← s.elementAt(a[i])
7:         s.remove(a[i])
8:     end for
9:     return permutation
10: end function
```

Die Funktion `elementAt`$(a[i])$ in Zeile 6 wählt dabei aus der Datenstruktur s eine Zahl aus, so dass exakt $a[i]$ Elemente in s existieren, die kleiner sind als die gewählte Zahl.

Beispiel 5.2. Es soll eine Permutation anhand der Eingabe 67 generiert werden. Die Zahl lässt sich im fakultätsbasierten Zahlensystem wie folgt darstellen.

$$67 = 2 \cdot 4! + 3 \cdot 3! + 0 \cdot 2! + 1 \cdot 1! + 0 \cdot 0!$$

Man wähle daher $a = [2, 3, 0, 1, 0]$. Der präsentierte Algorithmus berechnet aus a nun eine Permutation. Zunächst wird die Datenstruktur s mit den Werten 0-4 befüllt, das heißt $s = [0, 1, 2, 3, 4]$. Die Durchführung der zweiten Schleife in den Zeilen 5-8 soll nun detaillierter beleuchtet werden.

i	$a[i]$	Beschreibung	s	Permutation
-	-	Inhalt bei Beginn der Schleife	[0,1,2,3,4]	[ ]
0	2	Wähle die Zahl 2 aus s, da s exakt **zwei** Zahlen kleiner als 2 enthält.	[0,1,3,4]	[2]
1	3	Wähle die Zahl 4 aus s, da s exakt **drei** Zahlen kleiner als 4 enthält.	[0,1,3]	[2,4]
2	0	Wähle die Zahl 0 aus s, da s **keine** Zahl kleiner als 0 enthält.	[1,3]	[2,4,0]
3	1	Wähle die Zahl 3 aus s, da s exakt **eine** Zahl kleiner als 3 enthält.	[1]	[2,4,0,3]
4	0	Wähle die Zahl 1 aus s, da s **keine** Zahl kleiner als 1 enthält.	[ ]	[2,4,0,3,1]

Die durch die Eingabe der Dezimalzahl 67 und dem zugehörigen Lehmer-Code $(2, 3, 0, 1, 0)$ generierte Permutation ist somit $\pi = (2, 4, 0, 3, 1)$.

Die Umwandlung des eingegebenen Passworts in eine Permutation ermöglicht es dabei, ein klassisches, auf der Eingabe eines Passworts basierendes Authentifikationssystem und ein Zero-Knowledge-Beweissystem zu verbinden. Im Folgenden soll nun der Ablauf der erstmaligen *Registrierung* und der *Authentifikation* eines Clients dargelegt werden.

Meldet sich ein Nutzer erstmals bei einem Webserver an, muss er zunächst einmalig den Prozess der *Registrierung* durchlaufen. Hierzu wählt der Nutzer einen Benutzernamen sowie ein Passwort. Das Passwort wird durch den Webbrowser nach der im voranstehenden Absatz eingeführten Methode in eine Permutation π umgewandelt. Die Permutation dient dem Client als privater Schlüssel, wird nicht gespeichert und verlässt zu keinem Zeitpunkt den Browser des Clients. Darauf folgend generiert der Browser einen zufälligen Graphen G_0, dessen Knotenanzahl der Größe der Permutationsmenge entspricht. Dann wird mittels der Permutation π ein zu G_0 isomorpher Graph G_1 gebildet, das heißt $G_1 = \pi(G_0)$. Die beiden Graphen dienen als öffentlicher Schlüssel und werden gemeinsam mit dem gewählten Nutzernamen an den Server übermittelt und dort abgespeichert. Wird als Hashfunktion wie beschrieben SHA-1 gewählt, haben die erstellten Graphen eine Größe von bis zu 41 Knoten, da die größtmögliche 160-bit-Hexadezimalzahl fakultätsbasiert mit $k = 41$ darstellbar ist. Daraus resultiert eine Größe von $41^2 = 1681$ bits pro Graph, wenn der Graph in Form einer Adjazenzmatrix gespeichert wird.

Als Protokoll setzen Grzonkowski und Corcoran ein Zero-Knowledge-Beweissystem der Sprache **GI** ein. Ein ähnliches Protokoll wurde bereits in Abschnitt 3.3 vorgestellt. Um lange Wartezeiten während des Authentifikationsprozesses zu vermeiden, werden mehrere Instanzen des Beweissystems nebenläufig ausgeführt (*concurrent zero-knowledge* [DNS04]). Um die Zero-Knowledge-Eigenschaft der Beweise zu erhalten, müssen nach Ergebnissen von Dwork et al. zeitliche Einschränkungen eingeführt werden, um sowohl den Beweiser als auch den Verifizierer vor unehrlichen Konterparts zu schützen. Ohne zeitliche Einschränkung könnte beispielsweise ein betrügender Verifizierer die Ausführung des Protokolls so lange verzögern, bis aus einer anderen Protokollinstanz potenziell nutzbare Informationen geliefert werden können. Durch den Einsatz von *Timeouts* (α) und *Delays* (β) lässt sich die Zero-Knowledge-Eigenschaft des Protokolls erhalten.

Die Konstanten α und β werden vor Beginn des Beweises durch den Browser des Clients festgelegt. Der Client prüft während des Protokollsablaufs, ob die vom Server gesendeten Challenges innerhalb der vorgeschriebenen Zeit α zugestellt wurden. Ist dies nicht der Fall, wird der Authentifikationsprozess abgebrochen. Außerdem verzögert der Client seine Antwort um mindestens β nach Protokollbeginn, um sich vor einem womöglich betrügenden Verifizierer zu schützen.

Das Protokoll wurde in Java und JavaScript implementiert und in verschiedenen Browsern getestet, um den Performanceunterschieden der JavaScript-Integration-

en Rechnung zu tragen. Für den praktischen Einsatz des Protokolls muss zudem festgelegt werden, wie viele Challenges gestellt werden müssen, um die Gefahr auf ein akzeptables Maß zu reduzieren, dass sich ein betrügender, den ehrlichen Nutzer lediglich imitierenden Beweiser erfolgreich authentifizieren kann. Grzonkowski und Corcoran halten hier 40 Challenges für ausreichend. Zuletzt muss geprüft werden, ob die Implementierung des Protokolls den zeitlichen Anforderungen eines Authentifikationsprozesses entspricht. Die Authentifikation benötigt im Mittel je nach gewähltem Browser bis zu 1,5 Sekunden, ausführliche Daten zur Authentifikationsdauer in Abhängigkeit zur gewählten Anzahl der Challenges finden sich in der Arbeit von Grzonkowski und Corcoran [GC14]. Eine Authentifikation innerhalb von 1,5 Sekunden wird dabei einerseits den Arbeitsfluss des Nutzers nicht unterbrechen und andererseits beim Nutzer das Gefühl erzeugen, dass im Hintergrund tatsächlich ein Authentifikationsprozess stattfindet.

Sicherheit

Folgend soll die Sicherheit des präsentierten Protokolls evaluiert werden. Es sollte erneut angemerkt werden, dass für die Sprache **GI** ein quasi-polynomieller Algorithmus existiert, der die Sicherheit des Protokolls schwächt [Bab16]. Ein betrügender Beweiser, der in Besitz der Graphen G_0 und G_1 eines Nutzers ist und dessen Benutzernamen kennt, könnte sich durch Berechnung der zugehörigen Isomorphie als valider Nutzer authentifizieren. Auch der Einsatz der SHA-1-Hashfunktion schwächt die Sicherheit des Protokolls, es ist jedoch möglich, die Hashfunktion durch eine moderne, stärkere Alternative zu ersetzen.

Eine Kerneigenschaft von Zero-Knowledge-Beweisen liefert eine Angriffsmöglichkeit für betrügende Verifizierer. Dieser muss dazu jeden vom Beweiser geschickten Graphen, die zugehörige, gewählte Challenge und die jeweilige Antwort des Beweisers gespeichert werden. Wählt der Beweiser einen Graphen als Commitment, der in einem vergangenen Beweis bereits verwendet wurde, kann der Verifizierer durch Erfragen der gegenteiligen Challenge die geheime Permutation des Beweisers aus dessen Antworten ermitteln. Die dabei anfallende Menge an Daten ist jedoch zu groß für eine Speicherung, ein solcher Angriff gilt daher als nicht durchführbar.

Neben einem Angriff auf das Zero-Knowledge-Protokolls sollten auch Angriffe auf das Passwort in Betracht gezogen werden. Für die vorgestellte Authentifikationsmethode gelten dieselben Angriffsvektoren wie für vergleichbare Passwortsysteme. Angreifer könnten Brute-Force-Angriffe oder Wörterbuchangriffe auf das Passwort durchführen, um sich Zugriff zu verschaffen. Werden bei der Wahl des Passworts gängige Richtlinien befolgt, beispielsweise die Empfehlungen des *NIST*, dem *National Institute of Standards and Technology*, und dementsprechend Einschränkungen des Webservers getroffen, die schwache Passwörter verhindern, sind

auch diese Angriffe weitgehend wirkungslos. Andersartige Angriffe wie beispielsweise die Installation eines Keyloggers auf dem Gerät des Opfers zur Erlangung des Passworts oder durch Social Engineering versprechen höhere Erfolgswahrscheinlichkeiten. Diese Art der Angriffe schwächt jedoch nicht die Sicherheit des Protokolls an sich.

Zusammenfassend kann festgehalten werden, dass das vorgestellte Protokoll eine praxistaugliche Authentifikation im Web ermöglicht. Das Protokoll ist widerstandsfähig gegen Angriffe über das Netzwerk, da das eigentliche Passwort beziehungsweise die geheime Permutation nie über das Netzwerk versendet werden müssen. Das Protokoll bedarf daher nicht zwingend einer verschlüsselten Verbindung. Um die Verbindung zu verschlüsseln und gleichzeitig auch den Server gegenüber dem Client zu authentifizieren, wird von Grzonkowski und Corcoran dennoch ein schlichter *Encrypted Key Exchange* angedacht.

Durch die Nutzung von SHA-1 als Hashfunktion und den Einsatz eines Zero-Knowledge-Beweissystems für **GI** liefert das Protokoll keine optimale Sicherheit mehr und ist in dieser Form veraltet. Daher sollte von einem Einsatz in der Praxis ohne vorherige Anpassungen abgesehen werden.

6 Authentifikation in Peer-to-Peer-Netzwerken

Motivation

Peer-to-Peer-Infrastrukturen bilden ein Netzwerk aus untereinander verbundenen, gleichberechtigten Geräten, den sogenannten Peers. Jeder Peer kann dabei Dienste anderer Knoten in Anspruch nehmen und selbst Dienste im Netzwerk anbieten. Peer-to-Peer-Netzwerke arbeiten dezentralisiert, es ist also keine zentrale Organisation nötig, um das Netz zu verwalten. Dies unterscheidet ein Peer-to-Peer-Netzwerk von einer Client-Server-Infrastruktur, in der ein Client die Dienste des Servers in Anspruch nimmt, selbst aber keine Dienste anbietet.

Sucht ein Peer im Netzwerk nach einer bestimmten Datei, muss die Datei mithilfe von Suchalgorithmen im Netzwerk ausfindig gemacht werden. Hierbei ist zu beachten, dass jeder beliebige Knoten diese Datei anbieten kann, die erhaltene Ressource sollte daher auf Authentizität geprüft werden. Wird der Datenverkehr im Peer-to-Peer-Netzwerk vollständig anonymisiert, ist eine Einschätzung der Authentizität einer Datei nicht möglich. Eine Bestätigung der Echtheit und Authentizität der angebotenen Ressourcen und zugehörigen Peers liefert dagegen zum Beispiel die Einführung eines Rankings für jeden Peer. Diese kann durch eine zentralisierte, vertrauenswürdige Drittpartei, zum Beispiel eine Certificate Authority, bereitgestellt werden, welche die Authentifikation der Peers übernimmt. Dies setzt jedoch voraus, dass jeder Peer der Certificate Authority vollständig vertraut und zudem geheime Informationen mit der Certificate Authority teilt. Dies erhöht das Risiko, dass persönliche Daten von Angreifern erlangt werden können und erhöht darüber hinaus die Komplexität des Systems.

Wird auf eine Authentifikation durch Certificate Authorities verzichtet, werden Man-in-the-Middle-Attacken ermöglicht. Die Balance zwischen Vertrauen in einen Knoten und die Anonymität eines Peers spielt demzufolge in Peer-to-Peer-Infrastrukturen eine entscheidende Rolle. Ein von Lu et al. entwickeltes Protokoll namens *Pseudo Trust* soll die Authentifikation beider Kommunikationsteilnehmer bei gleichzeitiger Anonymität durch den Einsatz eines Zero-Knowledge-Protokolls und Pseudoidentitäten sichergestellt werden [LHL+08].

Protokoll

Um die Verknüpfung eines Peers A mit personenbezogenen Daten, zum Beispiel der IP-Adresse, zu vermeiden, soll zunächst ein Onion-Pfad die Verbindung zu einem Knoten im Peer-to-Peer-System ermöglichen. Die Verbindung zwischen dem Peer und dem Endknoten des Peers kann beispielsweise durch das *APFS-Protokoll* (*Anonymous Peer-to-Peer File Sharing* [SLS01]) implementiert werden. Der Endknoten leitet dabei für Peer A bestimmte Nachrichten an A weiter. Weder der Endknoten noch andere Peers im Netzwerk wissen dabei, dass sich A an der Endposition der Onion-Verbindung befindet.

Zunächst muss jedem Peer im System eine eindeutige *Pseudoidentität* (PI) und ein zugehöriges *Zertifikat* (PIC), welches die Pseudoidentität authentifiziert, zugeordnet werden. Im folgenden werden mehrere Hashfunktionen eingesetzt, alle Hashfunktionen h_i werden hierbei durch leicht modifizierte SHA-1-Hashfunktionen implementiert.

Als Erstes wählt der Peer A zwei verschiedene, große Primzahlen p und q und berechnet $n = p \cdot q$. Die Primfaktoren werden gemeinsam mit ID_A, der wahren Identität des Peers A, mithilfe einer Hashfunktion $h_1(\cdot)$ zu einem m-stelligen Seed gehasht. Es gilt demnach:

$$Seed_A = h_1(ID_A, p, q) \in \{0,1\}^m.$$

Dies einbeziehend wird die Pseudoidentität PI_A ermittelt. Es gilt:

$$PI_A = h_2(Seed_A, n) \in \{0,1\}^m.$$

Des Weiteren werden mittels der Hashfunktion $h_3(\cdot)$ die quadratischen Reste $v_{j_1}, ..., v_{j_k}$ erstellt. Es gilt:

$$v_{j_i} = h_3(Seed_A, j_i, n) \pmod n \in \mathbb{Z}_n^* \text{ für jedes } i \in \{1, 2, ..., k\},$$

wobei $j_1, j_2, ..., j_k$ verschiedene Ganzzahlen seien. Der Peer A berechnet außerdem die kleinstmögliche Quadratwurzel s_{j_i} jedes quadratischen Rests v_{j_i}, es gilt demnach $v_{j_i} \equiv s_{j_i}^2 \pmod n$. Dies ist angesichts der Kenntnis der Primfaktoren p und q effizient berechenbar, weitere Anmerkungen zu dessen Ursache finden sich in Abschnitt 3.1. Die ermittelten Werte werden zusammengefasst in

$$J = \{j_i, v_{j_i}\} \text{ für alle } i \in \{1, 2, ..., k\}.$$

Das zu PI_A gehörige Zertifikat PIC_A fasst die zuvor ermittelten Werte in einer Menge zusammen. Daher gelte:

$$PIC_A = \{PI_A, n, J, Seed_A\}.$$

Das Zertifikat kann öffentlich gespeichert und verbreitet werden, geheim gehalten werden müssen lediglich die Werte p und q, die echte Identität des Peers sowie die Quadratwurzeln s_{j_i}.

Mögliche Vertrauenswerte können nun mit der eindeutigen Pseudoidentität verknüpft werden, ohne dabei Informationen über die wahre Identität preiszugeben. Fordert der Peer A eine Ressource an, kann er aus der durch den Suchalgorithmus ermittelten Menge an Peers, die die Ressource zur Verfügung stellen, einen Peer mit gutem Ruf auswählen. Peer A kann nun an den gewählten Peer B eine Aufforderung zur Authentifikation schicken. Auch das Zertifikat PIC_A seiner Pseudoidentität wird dabei übermittelt. Möchte Peer B seine Identität beweisen, sendet er das Zertifikat PIC_B seiner Pseudoidentität PI_B an A. Dann authentifizieren die Peers A und B gegenseitig ihre Pseudoidentitäten, zunächst A seine Pseudoidentität gegenüber B, dann beweist B die Authentizität seine Pseudoidentität gegenüber A. Hierzu wird ein Zero-Knowledge-Protokoll verwendet, dass im Folgenden vorgestellt werden soll.

Das vorgestellte ähnelt dabei stark dem in Abschnitt 3.1 präsentierten Fiat-Shamir-Protokoll. Dabei wird wie folgt vorgegangen, wenn sich Peer A gegenüber Peer B authentifizieren möchte:

1. Peer A wählt eine zufällige Zahl $c \in \mathbb{Z}_n^*$ und berechnet $x \equiv c^2 \pmod{n}$. Der quadratische Rest x wird an Peer B übermittelt.

2. Peer B überprüft zunächst, ob $PI_A = h_2(Seed_A, n) \in \{0,1\}^m$ und $v_{j_i} = h_3(Seed_A, j_i, n) \pmod{n}$ für alle $i \in \{1, 2, ..., k\}$ gelten. Ist dies nicht der Fall, bricht B das Protokoll ab. Außerdem wählt B zufällig die Challenges $e_{j_i} \in \{0,1\}$ für jedes $i \in \{1, 2, ..., k\}$ und übermittelt diese an Peer A.

3. Peer A berechnet folgende Gleichung und übermittelt y an Peer B.

$$y = c \cdot \left(\prod_{i=1}^{k} s_{j_i}^{e_{j_i}} \pmod{n} \right)$$

4. Peer B überprüft, ob die folgende Gleichung gilt. Kennt A die geheimen Werte s und c, gelten $s_{j_i}^2 \equiv v_{j_i} \pmod{n}$ und $c^2 \equiv x \pmod{n}$ und die Gleichung gilt.

$$y^2 = x \cdot \left(\prod_{i=1}^{k} v_{j_i}^{e_{j_i}} \pmod{n} \right)$$

   Gilt die Gleichung nicht, wird der Authentifikationsprozess abgebrochen. Ist die Gleichung erfüllt, gilt Peer A gegenüber B als authentifiziert.

Während des Authentifikationsprozesses wird zudem ein Diffie-Hellman-Schlüsselaustausch durchgeführt. Durch den Einsatz des resultierenden Sitzungsschlüssels wird während der auf die Authentifikation folgenden Datenübertragung die Integrität der übermittelten Daten und die Vertraulichkeit der Verbindung gewährleistet.

Sicherheit

Die präsentierte Vorgehensweise adressiert sowohl den Wunsch der Peers nach Anonymität als auch die Verpflichtung zu einer Authentifikation, die ohne eine zentrale Certificate Authority auskommt. Die eingesetzte Hashfunktion soll dabei erstens dafür sorgen, dass aus der Pseudoidentität keine Rückschlüsse auf die wahre Identität gezogen werden können (*hiding*), zweitens soll es nicht effizient möglich sein, eine Kollision zweier Hashwerte herbeizuführen (*collision resistance*), so dass zwei verschiedene Identitäten dieselbe Pseudoidentität erhalten. Ließen sich Kollisionen gezielt herbeiführen, wäre es einem betrügerischen Peer A' unter Umständen möglich, einen Peer A zu imitieren, indem die Eingabe in die Hashfunktion so gewählt würde, dass sich die Pseudoidentitäten PI_A und $PI_{A'}$ glichen. Da die von Lu et al. gewählte Hashfunktion SHA-1 mittlerweile als unsicher gilt und Kollisionen gezielt herbeigeführt werden können [SBK+17], sollte ein Austausch durch eine moderne, sichere Hashfunktion erwogen werden.

Das eingesetzte Zero-Knowledge-Protokoll implementiert im Peer-to-Peer-Netzwerk ebenfalls einen Schutzmechanismus vor der Nachahmung eines Peers durch einen Angreifer. Hält ein Peer seine Identität ID, die Primzahlen p und q sowie die quadratischen Reste s_{j_i} geheim, ist es einem Betrüger nur mit einer Wahrscheinlichkeit von 2^{-k} möglich, die Kenntnis dieser Geheimnisse dennoch zu beweisen und sich somit zu authentifizieren. Lu et al. wählen die Länge k des Challenge-Vektors hierbei mit $k = 80$, um die Chance auf einer erfolgreichen Imitation auf ein vernachlässigbares Maß zu reduzieren. Die Länge des Modulus n sollte dabei auf mindestens 1024 bits festgelegt werden, somit wäre die Zerlegung des öffentlich einsehbaren n in Primfaktoren nicht effizient durchführbar. Ein Beweis der Zero-Knowledge-Eigenschaft des Protokolls wurde bereits in Abschnitt 3.1 präsentiert.

Durch die geschickte Wahl der Authentifikationsreihenfolge der Peers A und B werden zudem Denial-of-Service-Attacken während des Authentifikationsprozesses vermieden. Peer A initiiert den Authentifikationsprozess mit der Wahl eines geeigneten Peers B, der die gewünschte Ressource anbietet. Darum ist A zunächst wenig anfällig für DoS-Angriffe. Würde der anbietende Peer B mit Authentifikation beginnen, könnten Angreifer unter einer gefälschten Pseudoidentität PI viele Anfragen zur Authentifikation an B senden. Peer B müsste für jede Anfrage ein c zufällig wählen, daraus den quadratischen Rest x errechnen und an die

Angreifer übermitteln. Dann würde B auf die Challenges der Angreifer warten, die angesichts des Tatsache, dass es sich um einen Angriff und nicht tatsächlich um einen Authentifikationsversuch handelte, nie vom Angreifer geschickt würden. Dies könnte Peer B gegebenenfalls handlungsunfähig machen.

Wird dagegen, wie zuvor präsentiert, zunächst Peer A authentifiziert, wäre die Angreifer gezwungen, gültige Pseudoidentitäten zu verwenden und für die Anfrage außerdem einen quadratischen Rest x zu berechnen. Andernfalls würde Peer B den Authentifikationsprozess umgehend abbrechen. Die Berechnungskomplexität des Beweisers ist dabei im Vergleich zu der des Verifizierers hoch, und damit für den Angreifer schwerer durchführbar. Zuletzt würde eine Verwendung von echten Pseudoidentitäten während einer DoS-Attacke den Ruf der Identitäten zu verringern und damit zukünftige Angriffe noch weiter erschweren.

Abschließend lässt sich demnach festhalten, dass das vorgestellte Protokoll sowohl Anonymität als auch die Authentifikation jedes Peers gewährleisten kann. Es wird eine fälschungssichere, verifizierbare Pseudonymisierung der Peers im Netzwerk vorgenommen und eine Verbindung über Onion-Pfade aufgebaut, um die Anonymität der Peers im Netzwerk zu schützen. Darüber hinaus bietet das Protokoll im Vergleich zu Public-Key-Infrastrukturen den Vorteil, dass auf eine zentrale, vertrauenswürdige Institution, beispielsweise eine Certificate Authority, verzichtet werden kann, ohne dabei eine Gefährdung durch Man-in-the-Middle-Angriffe zuzulassen.

7 Zerocoin: Anonyme Authentifikation für Bitcoin-Transaktionen

Motivation

Der Bitcoin ist die erste und wohl bekannteste Kryptowährung. Beim 2008 erstmals vorgestellten Open-Source-Projekt handelt es sich um eine digitales Zahlungsmittel, dem ein gleichnamiges, dezentrales, das heißt ohne eine zentrale Bank oder sonstige Autorität auskommendes Peer-to-Peer-Netzwerk zugrunde liegt. Die Vertrauen in Bitcoin basiert dabei auf der Annahme, dass sich die Mehrheit der Peers im Netzwerk nicht betrügerisch verhält. Alle anfallenden Daten im Bitcoin-Netzwerk werden dabei an jeden Knoten im Netzwerk gesendet. Bei den gesendeten Daten handelt es sich dabei entweder um Transaktionen oder um Blöcke. Als Transaktionen gelten dabei jegliche Aufteilung, Zusammenführung oder Überweisung von Bitcoins, während in den Blöcken alle überprüften und als valide befundenen Transaktionen gespeichert werden. Um alle Transaktionen in Blöcken abspeichern zu können, müssen regelmäßig neue Blöcke generiert werden. In rechenintensiven Prozessen können Knoten versuchen, einen validen Block zu bilden. Hierzu muss eine *Nonce*, eine zufällige 32-bit-Zahl B gefunden werden, die folgende Gleichung erfüllt [MGGR13]:

$$\text{SHA256}(\text{SHA256}(B)) = (0^{\ell} \| \{0,1\}^{256-\ell}).$$

Der Wert ℓ wird hierbei regelmäßig durch das Netzwerk festgelegt, um der Rechenleistung des Netzwerks gerecht zu werden. Es soll etwa alle 10 Minuten ein neuer Block generiert werden, je größer ℓ gewählt wird, desto rechenintensiver wird die Berechnung einer Nonce B. Der beschriebene Prozess wird dabei als *Mining* bezeichnet. Hat ein Knoten eine Nonce B ermittelt, welche die Gleichung erfüllt, wird der Block mit ausstehenden Transaktionen befüllt und die Nonce B als *Proof of Work* an alle Knoten des Netzwerks gesendet. Die Knoten können die Validität der Nonce B leicht prüfen. Ist B validiert, wird der neu erstellte Block der *Blockchain*, einer Sammlung aller erstellten Blöcke und Transaktionen, hinzugefügt. Miner, die der Blockchain Blöcke hinzufügen, erhalten als Anreiz eine festgelegte Menge an Bitcoins pro hinzugefügtem Block und erhalten außerdem Gebühren für hinzugefügte Transaktionen.

Eine Transaktion besteht aus einem Input und einem Output. Der Output enthält dabei die Menge der zu transferierenden Bitcoins sowie den öffentlichen Schlüssel des Empfängers. Der Input umfasst eine oder mehrere Referenzen auf den Output vorheriger Transaktionen. Die Transaktion wird außerdem vom Ersteller unter Einsatz seines privaten Schlüssel signiert, um nachzuweisen, dass er selbst jeweils der Begünstigte der referenzierten Transaktionen ist und somit über die angegebenen Bitcoins verfügt. Wird die Transaktion im Netzwerk verbreitet und einem Block hinzugefügt, gilt die Transaktion als durchgeführt.

Weitere Ausführungen zu Funktionalität und Eigenschaften von Bitcoin finden sich zudem in der Arbeit von Barber et al. [BBSU12].

Alle Transaktionen in Bitcoin sind in der Blockchain gespeichert und demzufolge öffentlich einsehbar. Zahlungen in Bitcoin laufen daher lediglich pseudonym ab, die wahre Identität eines Akteurs wird nur durch seine öffentliche Bitcoinadresse verborgen. Hat ein Peer im Bitcoin-Netzwerk das Ziel, seine Identität zu verbergen und anonym Transaktionen zu tätigen, hat er die Möglichkeit, regelmäßig neue Bitcoinadressen zu generieren, um eine Nachverfolgung der Transaktionen zu erschweren. Dies liefert jedoch keine vollständige Anonymität. Eine weitere, aufwendigere Möglichkeit zur Verschleierung von Transaktionen ist der Einsatz von Mixing-Diensten (*laundry services*). Hierbei zahlen mehrere Nutzer dem Dienst ihre Bitcoins ein und geben neue Bitcoin-Adressen als Ziel an. Die Mixing-Dienste vermischen darauf die Geldmittel der verschiedenen Nutzer und transferieren diese auf die angegebenen, neuen Bitcoin-Adressen. Die eigenen Geldwerte gehen dabei an andere Nutzer des Dienstes, während man selbst Coins anderer Nutzer erhält. Somit wird die Nachverfolgung der Bitcoins schwierig. Mixing-Dienste fordern dabei verhältnismäßig hohe Gebühren, des weiteren muss in eine dritte Partei vertraut werden. Der Nutzer geht dabei das Risiko ein, seine eingezahlten Bitcoins nicht zurückzuerhalten, darüber hinaus muss er darauf vertrauen, dass die Mixing-Dienste den Verlauf des Mischens nicht protokollieren. Dies wäre nicht vereinbar mit dem Ziel, anonyme Zahlungen zu tätigen.

Miers et al. entwickelten daher eine Erweiterung des Bitcoin-Protokolls, den auf Zero-Knowledge-Beweisen basierenden *Zerocoin*, der vollständig anonyme Transaktionen im Bitcoin-Netzwerk ermöglicht [MGGR13]. Nutzer können ihre Bitcoins in anonyme Zerocoins umwandeln, mit ihnen handeln und sie bei Bedarf wieder in Bitcoins zurückverwandeln. Zerocoin basiert dabei auf einem ähnlichen Konzept wie die Mixing-Dienste, nimmt jedoch Anpassungen auf Protokollebene vor und beseitigt somit die Notwendigkeit einer vertrauenswürdigen Drittpartei.

Protokoll

Zunächst ist es notwendig, den Begriff des *Akkumulators* näher zu erläutern. Ein kryptografischer Akkumulator bildet eine Menge an Werten auf einen festen Wert

konstanter Größe ab. Mithilfe dieses festen Wertes lässt sich die Existenz eines bestimmten Elements in der zugrundeliegenden Menge effizient beweisen, ohne dabei den Wert des Elements preiszugeben. Dieser Beweis wird auch als *Proof of Membership* bezeichnet.

Der verwendete Akkumulator basiert dabei auf der *Strong-RSA-Annahme* von Baric und Pfitzmann [BP97] und wird von Miers et al. [MGGR13] wie folgt implementiert:

Setup des Akkumulators: Sei λ ein Sicherheitsparameter. Dann wähle zwei zufällige Primzahlen p und q (jeweils beschränkt durch ein Polynom von λ), und berechne $N = p \cdot q$. Wähle außerdem einen zufälligen quadratischen Rest $u \in \mathbb{Z}_N^*$. Gebe (N, u) aus. Das heißt:

$$(N, u) \leftarrow \texttt{SetupAccumulator}(\lambda).$$

Akkumulieren: Sei (N, u) der Rückgabewert des vorherigen Schrittes und außerdem $\mathbf{C} = \{c_1, c_2, ..., c_n\}$ eine Menge an Primzahlen. Dann berechne den Akkumulator $A = u^{c_1 c_2 ... c_n} \pmod N$. Das heißt:

$$A \leftarrow \texttt{Accumulate}((N, u), \mathbf{C}).$$

Generierung eines Zeugnisses: Seien (N, u) und $\mathbf{C}$ wie oben beschrieben. Sei darüber hinaus $v \in \mathbf{C}$. Dann ist das Zeugnis w der Akkumulator aller Werte in der Menge $\mathbf{C}$, mit Ausnahme des Wertes v, also $w = \texttt{Accumulate}((N, u), \mathbf{C} \setminus \{v\})$. Das heißt:

$$w \leftarrow \texttt{GenerateWitness}((N, u), v, \mathbf{C}).$$

Verifikation des Akkumulators: Seien (N, u), der Akkumulator A und w wie oben gewählt. Sei v ein Wert. Dann berechne $A' \equiv w^v \pmod N$. Ist $A' = A$ und v eine Primzahl, akzeptiere. Dann ist v ein Element in der zu A gehörigen Menge. Ist mindestens eine der Bedingungen nicht erfüllt, lehne ab.

$$\{0, 1\} \leftarrow \texttt{VerifyAccumulator}((N, u), A, v, w).$$

Ein solcher Akkumulator ist kollisionsresistent unter der Strong-RSA-Annahme [CL02]. Das bedeutet, dass es nicht durchführbar ist, für ein $v \notin \mathbf{C}$ ein zugehöriges w zu finden, so dass $\texttt{VerifyAccumulator}((N, u), A, v, w) = 1$ gilt. Der Nachweis, dass ein bestimmter Wert in der zugrundeliegenden Menge enthalten ist, ist dabei möglich, ohne den konkreten Wert preisgeben zu müssen [CL02]. Man bezeichnet diesen Nachweis auch als *zero-knowledge proof of set membership*.

Durch den Einsatz eines solchen Akkumulators lässt sich nun die Funktionsweise von Zerocoin erläutern.

Setup von Zerocoins: Sei λ ein Sicherheitsparameter und (N, u) das Ergebnis des Akkumulatorsetups $\texttt{SetupAccumulator}(\lambda)$. Dann wähle zwei Primzahlen p und q, so dass $p = 2^w \cdot q + 1$ für alle $w \geq 1$ gilt. Wähle zwei zufällige Generatoren g und h, so dass $\mathbb{G} = \langle g \rangle = \langle h \rangle$. Sei $\mathbb{G}$ außerdem eine Untergruppe von $\mathbb{Z}_q^*$. Gebe (N, u, p, q, g, h) aus. Das heißt:

$$(N, u, p, q, g, h) \leftarrow \texttt{SetupZerocoin}(1^\lambda).$$

Generierung von Zerocoins: Zunächst werden eine zufällige Zahl $r \in \mathbb{Z}_q^*$ und eine zufällige Seriennummer $S \in \mathbb{Z}_q^*$ gewählt. Als Commitment für S wird der Zerocoin $c \equiv g^S \cdot h^r \pmod{p}$ berechnet, so dass c eine Primzahl bildet. Gebe (c, skc) mit $skc = (S, r)$ aus. Das heißt:

$$(c, skc) \leftarrow \texttt{MintZerocoin}(N, u, p, q, g, h).$$

Möchte ein Peer P demnach einen Zerocoin c eines bestimmten Werts generieren, muss P die Funktion `MintZerocoin` ausführen und die Ausgabe skc geheim halten. Das Commitment c wird darauf folgend als Output einer Bitcoin-Transaktion angegeben. Wurde die Transaktion validiert und einer Blockchain hinzugefügt, wird c zusätzlich dem globalen Akkumulator A hinzugefügt. Der Akkumulator A enthält die Commitments aller Zerocoins und verwaltet demzufolge Bitcoins im Wert aller Zerocoins, die sich in Umlauf befinden.

Aufwenden von Zerocoins: Sollen Zerocoins zurück in Bitcoins umgewandelt werden, muss zunächst bewiesen werden, dass das Commitment c in der Menge $\mathbf{C}$, die alle generierten Zerocoins umfasst, enthalten ist und c zudem ein korrektes Commitment der Werte S und r ist. Dazu berechnet der Besitzer des Zerocoins den Akkumulator $A = \texttt{Accumulator}((N, u), \mathbf{C})$ aller Zerocoins sowie das Zeugnis $w = \texttt{GenerateWitness}((N, u), c, \mathbf{C})$, das die Mitgliedschaft von c in $\mathbf{C}$ nachweist. Ausgegeben wird (π, S), wobei π ein nicht-interaktiver Zero-Knowledge-Beweis, genauer eine *Signature of Knowledge* sei. Der Beweis soll dabei unter Geheimhaltung von c, w und r einerseits zeigen, dass c ein korrektes Commitment der Werte S und r darstellt und andererseits nachweisen, dass c in A akkumuliert wird. Nur so kann gewährleistet werden, dass der Zerocoin tatsächlich ausgegeben werden kann. So lässt sich eine Verbindung zwischen dem Generieren und dem Aufwenden eines Zerocoins erfolgreich verbergen. Eine Verbindung zwischen dem Coin c und der Seriennummer S kann nur durch die Zufallszahl r hergestellt werden, welche der Besitzer des Coins während des Beweises geheim hält, oder durch die Kenntnis darüber, welchen Zerocoin c der Besitzer in $\mathbf{C}$ nachgewiesen hat. Auch dies kann durch den von Miers et al. verwendeten Zero-Knowledge-Beweise verborgen werden. Der Zero-Knowledge-Beweis wird hierbei auf Basis eines Protokolls von Claus-Peter Schnorr geführt [Sch91], welches auf der Schwierigkeit basiert, diskrete Logarithmen zu bestimmen. Um nicht-interaktive Zero-Knowledge-Beweise zu erhalten, wird außerdem die Fiat-Shamir-Heuristik angewendet [FS86].

Sei $R \in \{0,1\}^*$ dabei ein Transaktionsstring, in dem beispielsweise die Zieladresse der Transaktion gespeichert werden kann. Das heißt:

$$(\pi, S) \leftarrow \texttt{SpendZerocoin}((N, u, p, q, g, h), c, skc, R, \mathbf{C}).$$

Der Besitzer des Zerocoins hängt den Beweis und die Seriennummer in Form einer Transaktion an die Bitcoin-Blockchain an, nachdem (π, S) von ausreichend Peers validiert worden ist.

Verifizierung des Zerocoins: Berechne zunächst $A = \texttt{Accumulate}((N, u), \mathbf{C})$. Ist der Zero-Knowledge-Beweis π korrekt, akzeptiere, sonst lehne ab.

$$\{0,1\} \leftarrow \texttt{VerifyZerocoin}((N, u, p, q, g, h), \pi, S, R, \mathbf{C}).$$

Es muss also lediglich überprüft werden, ob es sich bei π um einen validen Beweis handelt und außerdem zu prüfen, ob die Seriennummer S erstmals eingesetzt wurde, ist dies nicht der Fall, wird die Transaktion unterbunden, um ein mehrfaches Ausgeben desselben Zerocoins zu verhindern.

Sicherheit

Das verwendeten Zero-Knowledge-Beweise basieren auf der Annahme, dass diskrete Logarithmen schwer zu lösen sind. Außerdem wird der eingesetzte Akkumulator unter der Strong-RSA-Annahme konstruiert. Der Besitzer eines Zerocoins kann sich bei einer Transaktion über das vorgestellte Verfahren als rechtmäßiger Besitzer des Coins authentifizieren, ohne den ursprünglich generierten Zerocoin offenlegen zu müssen. Daraus resultiert ein Zugewinn an Anonymität für den Besitzer. Dennoch ist das mehrfache Aufwenden eines Zerocoins durch die Überprüfung der Seriennummer S nicht möglich, nur die erste Transaktion eines Zerocoins mit Seriennummer S wird erfolgreich validiert.

Da die Anzahl der sich in Umlauf befindlichen Zerocoins öffentlich ermittelbar und zudem der Wert jedes Zerocoins einsehbar ist, sollte eine möglichst große Nutzerbasis sichergestellt werden oder die Menge möglicher Zerocoin-Beträge verkleinert werden. Sind nur wenige Zerocoins gleichen Wertes im Umlauf, kann die Anonymität des Halters unter Umständen nicht sichergestellt werden, wenn die Zerocoins zurück in Bitcoin umgewandelt werden, da sich Generierung und Aufwenden des Zerocoins durch den Wert des Zerocoins miteinander verknüpfen lassen. Die Berechnungsdauer einer Transaktion eines Zerocoins liegt bei der Implementierung von Miers et al. deutlich höher als eine vergleichbare Transaktion eines Bitcoins durchzuführen.

Zusammenfassend kann festgehalten werden, dass Zerocoin eine anonyme Erweiterung für das Handeln mit Bitcoin implementiert. Die Nutzung von Zerocoin

anstelle von Bitcoin liefert einen Zugewinn an Anonymität, hat im Gegenzug jedoch eine erhöhte Verifikationsdauer und einen höheren Speicherbedarf zu Folge. Die Verifikationszeiten liegen signifikant höher als die Dauer einer Transaktionsverifikation eines Bitcoins, auch die Größe der Beweise erhöht sich durch die Nutzung von Zerocoins. Daher wird angeraten, die gelieferten Beweise nicht in der Blockchain, sondern separat zu speichern.

8 Fazit

Mit der „magischen“ Tür wurde zunächst ein anschauliches Beispiel eines Zero-Knowledge-Beweises präsentiert. Darauf folgend wurde das Konzept von Zero-Knowledge-Beweissystemen nach den Definitionen von Goldwasser, Micali und Rackoff [GMR89] formalisiert und komplexitätstheoretisch eingeordnet. Mit dem Fiat-Shamir-Protokoll und ein Beweissystem der Sprache **GI** wurden außerdem zwei konkrete Zero-Knowledge-Protokolle vorgestellt, die in den im Verlauf der Arbeit präsentierten Anwendungsfällen tatsächlich eingesetzt werden. Darüber hinaus wurde auch ein Protokoll des **NP**-vollständigen Problems **SAT** herausgearbeitet. Für alle Protokolle wurden jeweils die Eigenschaften eines Zero-Knowledge-Beweissystem hervorgehoben und bewiesen. Zuletzt wurde noch in einem Exkurs in das Themengebiet der nicht-interaktiven Zero-Knowledge-Beweise eingeführt.

Für die zuvor dargelegten, theoretischen Konzepte wurden im folgenden Verlauf der Arbeit mehrere praktische Implementierungen demonstriert. Die exemplarisch ausgewählten Anwendungsgebiete decken dabei verschiedene Gebiete der Authentifikation ab.

Zunächst wurde eine Alternative für ein klassisches Passwortsystem aufgezeigt, in dem sich ein Nutzer im Web per Passwort bei einem Server authentifiziert. Durch das Umwandeln des Passworts in eine Graphenisomorphie und dem Einsatz des Zero-Knowledge-Beweissystems für **GI** konnte eine Methode entwickelt werden, mit der das eingegebene Passwort nicht mehr über das Netzwerk versendet werden muss und demensprechend besser gegen Angriffe über das Netzwerk geschützt ist [GC14].

Als Zweites wurde eine Implementierung des Fiat-Shamir-Protokolls vorgestellt, um zwei Peers in einem Peer-to-Peer-Netzwerk vor dem Austausch von Daten gegenseitig zu authentifizieren. Das präsentierte Vorgehen liefert dabei sowohl Anonymität für beide Peers als auch eine dezentrale Struktur, die ohne vertrauenswürdige, zentrale Drittparteien auskommt [LHL+08].

Als letztes Beispiel wurde Zerocoin, eine Erweiterung der Kryptowährung Bitcoin, vorgestellt. Da in der Bitcoin-Blockchain jegliche jemals getätigte Transaktionen öffentlich einsehbar sind, sollte mit Zerocoin eine Erweiterung vorgestellt werden, die Anonymität beim Zahlen mit Bitcoin anbietet. Bei der Implementierung des Zerocoins werden dabei während der Transaktion keine Informationen über den Besitzer des Zerocoins veröffentlicht, die Transaktionen bleiben

vollständig anonym. Ein Besitzer von Zerocoins kann sich dabei als rechtmäßiger Besitzer des Zerocoin authentifizieren, ohne Informationen über sich selbst oder den Empfänger preiszugeben. Auch Zerocoin arbeitet dezentralisiert und benötigt keine zentrale, vertrauenswürdige Institution [MGGR13].

In einer digitalen Welt, in der Anonymität eine immer entscheidendere Rolle bei der Wahl einer Technologie spielt, können Zero-Knowledge-Beweise tatsächlich einen Zugewinn an Anonymität erreichen und gleichzeitig Kommunikationsteilnehmer untereinander authentifizieren. Um sowohl Anonymität als auch Authentizität zu erreichen, werden Zero-Knowledge-Beweise häufig mit weiteren kryptografischen Mechanismen kombiniert. Die vorgestellten Implementierungen haben dabei jedoch häufig eine schlechtere Laufzeit als vergleichbare Implementierungen, die ohne Zero-Knowledge-Beweise auskommen.

Es sollte darüber hinaus angemerkt werden, dass es sich bei Zero-Knowledge-Beweisen um probabilistische Verfahren handelt. Das bedeutet, dass immer eine geringe Restwahrscheinlichkeit bleibt, mit der sich betrügerische Beweiser als jemand authentifizieren können, der sie nicht tatsächlich sind. Je weiter diese Wahrscheinlichkeit gesenkt werden soll, desto mehr Beweisrunden müssen geführt werden und desto rechenintensiver wird dementsprechend das Protokoll. Dagegen ermöglichen es Zero-Knowledge-Beweise, geheime Daten zu schützen, indem sie während eines Beweises nur lokal zur Authentifikation eingesetzt werden müssen und die Übertragung über das Netzwerk entfällt.

Der Einsatz von Zero-Knowledge-Beweisen kann daher vor allem dann in Betracht gezogen werden, wenn verteilte Dienste zum Einsatz kommen und auf zentrale, vertrauenswürdige Drittparteien verzichtet werden soll. Außerdem können Zero-Knowledge-Beweise als Subprotokoll in Kombination mit weiteren kryptografischen Methoden Authentizität bei gleichzeitiger Anonymität sicherstellen.

Literaturverzeichnis

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, 2009.

[Bab16] László Babai. Graph isomorphism in quasipolynomial time [extended abstract]. In *Proceedings of the 48th Annual ACM SIGACT Symposium on Theory of Computing, STOC 2016, Cambridge, MA, USA, June 18-21, 2016*, pages 684–697, 2016.

[BBSU12] Simon Barber, Xavier Boyen, Elaine Shi, and Ersin Uzun. Bitter to better - how to make Bitcoin a better currency. In *Financial Cryptography and Data Security - 16th International Conference, FC 2012, Kralendijk, Bonaire, Februray 27-March 2, 2012, Revised Selected Papers*, pages 399–414, 2012.

[BC86] Gilles Brassard and Claude Crépeau. Non-transitive transfer of confidence: A perfect zero-knowledge interactive protocol for SAT and beyond. In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 188–195, 1986.

[BJY97] Mihir Bellare, Markus Jakobsson, and Moti Yung. Round-optimal zero-knowledge arguments based on any one-way function. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 280–305, 1997.

[BP97] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *Advances in Cryptology - EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, May 11-15, 1997, Proceeding*, pages 480–494, 1997.

[BSMP91] Manuel Blum, Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Noninteractive zero-knowledge. *SIAM J. Comput.*, 20(6):1084–1118, 1991.

[BSW95] Albrecht Beutelspacher, Jörg Schwenk, and Klaus-Dieter Wolfenstetter. *Moderne Verfahren der Kryptographie*, volume 6. Springer, 1995.

[CL02] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, pages 61–76, 2002.

[DNS04] Cynthia Dwork, Moni Naor, and Amit Sahai. Concurrent zero-knowledge. *J. ACM*, 51(6):851–898, 2004.

[FS86] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 186–194, 1986.

[GC14] Sławomir Grzonkowski and Peter Corcoran. A practical zero-knowledge proof protocol for web applications. *Journal of Information Assurance & Security*, 9(7):329–343, 2014.

[GMR89] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to prove all NP-statements in zero-knowledge, and a methodology of cryptographic protocol design. In *Advances in Cryptology - CRYPTO '86, Santa Barbara, California, USA, 1986, Proceedings*, pages 171–185, 1986.

[GO94] Oded Goldreich and Yair Oren. Definitions and properties of zero-knowledge proof systems. *J. Cryptology*, 7(1):1–32, 1994.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[LHL+08] Li Lu, Jinsong Han, Yunhao Liu, Lei Hu, Jinpeng Huai, Lionel M. Ni, and Jian Ma. Pseudo Trust: Zero-knowledge authentication in anonymous p2ps. *IEEE Trans. Parallel Distrib. Syst.*, 19(10):1325–1337, 2008.

[MGGR13] Ian Miers, Christina Garman, Matthew Green, and Aviel D. Rubin. Zerocoin: Anonymous distributed e-cash from Bitcoin. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 397–411, 2013.

[SBK+17] Marc Stevens, Elie Bursztein, Pierre Karpman, Ange Albertini, and Yarik Markov. The first collision for full SHA-1. In *Advances in*

Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I, pages 570–596, 2017.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sha90] Adi Shamir. IP=PSPACE. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume I*, pages 11–15, 1990.

[Sim02] Gerardo I Simari. A primer on zero knowledge protocols. Technical report, Universidad Nacional del Sur, Bahia Blanca, Buenos Aires, Argentina, 2002.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation*. PWS Publishing Company, 1997.

[SLS01] V. Scarlata, Brian Neil Levine, and Clay Shields. Responder anonymity and anonymous peer-to-peer file sharing. In *9th International Conference on Network Protocols (ICNP 2001), 11-14 November 2001, Riverside, CA, USA*, pages 272–280, 2001.