Kryptografische Grundlagen von Bitcoin

Bachelorarbeit von
Lasse Bohlen und Jan Niklas Hasse

im Fachbereich Mathematik
der Universität Bremen

Erstgutachter: Prof. Dr. Michael Hortmann
Zweitgutachter: Prof. Dr. Rolf Drechsler

Abgabe: 06.09.2013

Inhaltsverzeichnis

Teil I

Einleitung

Bitcoin ist eine virtuelle Währung, die seit 2009 existiert. Zu ihren Besonderheiten zählt, dass sie im Unterschied zu anderen Online-Bezahlsystemen, wie Kreditkarten oder PayPal, dezentral verwaltet wird und eine eigene Währung darstellt. Außerdem unterliegt sie gewissen konstruktionsbedingten Gesetzmäßigkeiten, die nicht durch eine privilegierte Gruppierung beeinflussbar sind. Damit wird ein lang existierendes Problem der Geldwirtschaft behoben: Es muss beim Transferieren oder Sparen von Geld nicht auf das Wohlwollen einer dritten Partei vertraut werden. Es gibt somit keine Institution, der es vergleichbar zu einer privaten Bank, einer Zentralbank oder Regierung möglich wäre, Kontostände einzusehen oder die Menge eines Zahlungsmittels zu beeinflussen. Stattdessen können Veränderungen der Bitcoin-Struktur nur dann durchgesetzt werden, wenn sie in der Allgemeinheit des Bitcoin-Netzwerks akzeptiert werden. So ist sichergestellt, dass kein Staat, privates Unternehmen oder einzelne Personen Beeinflussungen zu ihrem Vorteil vornehmen können.

In der folgenden Ausarbeitung soll zunächst in den Aufbau und die Funktionsweise des Bitcoin-Netzwerks eingeführt werden. Anschließend sollen zwei Mechanismen dargestellt werden, die für die Funktionalität und Sicherheit wichtig sind. Zuerst wird dabei auf den Algorithmus zur asymmetrischen Verschlüsselung eingegangen, der für die Anonymität und die digitalen Signaturen genutzt wird. Er dient einerseits als Nachweis dafür, dass eine getätigte Überweisung durch einen dazu berechtigten Teilnehmer des Netzwerks vorgenommen wurde, andererseits sorgt er für Pseudonymbildung und damit für Anonymität. Letztere ist damit allerdings noch nicht vollständig gewährleistet. Daher möchten wir in Teil IV unserer Arbeit eine Bitcoin-Erweiterung vorstellen, die ein Konzept zur vollständigen Anonymisierung verwirklicht.

Der einleitende Teil „Bitcoin“, der die Funktionsweise von Bitcoin vorstellt, wurde von beiden Verfassern in direkter Zusammenarbeit erstellt. Die Bearbeitung des zweiten Abschnitts „Digitale Signaturen“ übernahm primär Lasse Bohlen und der darauf folgenden Teil „Anonymität“ wurde vorrangig von Jan Niklas Hasse verfasst. Wir bedanken uns außerdem besonders bei unserem Betreuer Prof. Dr. Michael Hortmann für den Vorschlag des interessanten Themas, seine Unterstützung, wertvolle Tipps und anreichende Diskussionen während dieser Arbeit.

Teil II

Bitcoin

1 Warum Bitcoin?

In heutigen Kontosystemen verwaltet eine Instanz wie z. B. die Bank die Konten der Teilnehmer. Das heißt, sie weiß über sämtliche Kontostände und Transaktionen Bescheid. Alle Teilnehmer sind auf sie angewiesen. Möchte Alice Geld an Bob überweisen, muss sie der Bank zunächst mitteilen, wie viel Geld von welchem Konto an wen transferiert werden soll. Nun kann die Bank feststellen, ob Alice berechtigt ist, diese Überweisung auszuführen. Dazu muss geprüft werden, ob Alice auf das Konto zugreifen darf und ob der Überweisungbetrag vom Kontostand abgedeckt wird. Um all dies nachvollziehen zu können, muss die Bank vollen Zugriff auf das Konto und auf vertrauliche Daten von Alice haben. Diese Macht könnte von einzelnen Personen in der Bank, der Bank selbst oder der Bank übergeordneten Behörden missbraucht werden.

Außerdem unterliegen aktuelle Währungen dem direkten Einfluss von Zentralbanken, Geschäftsbanken und Regierungen. Seit dem Ende des Goldstandards kann die Geldmenge somit theoretisch beliebig erhöht werden.

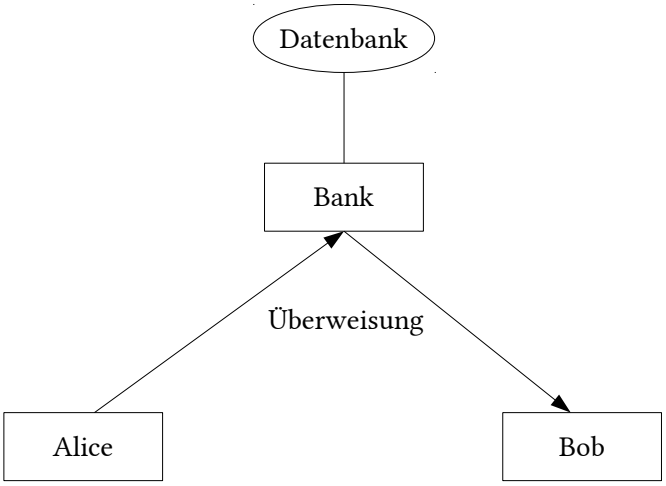

Abbildung 1: Klassisches Kontosystem

Mit Bitcoin will man laut [Nak] die benannten Umstände ändern. Keine Instanz soll die Datenbank kontrollieren, in der Transaktionen bzw. Kontostände gespeichert werden. Zudem soll es nicht wie bei traditionellem Geld möglich sein, dass die Geldmenge beliebig erweiterbar ist, sondern dass sie sich stattdessen für alle Marktteilnehmer vorhersehbar entwickelt.

2 Idee des Netzwerks

Bitcoin setzt ein Peer-2-Peer-System ein, in dem alle Teilnehmer gleichberechtigt sind. Das System agiert über das Internet und man benötigt zur Teilnahme eine Software, die *Bitcoin-Client* genannt wird. Jeder Teilnehmer speichert seine eigene Kopie der Datenbank, in der alle Transaktionen, die jemals getätigt wurden, gespeichert sind. Führt jemand eine Überweisung durch, muss dies allen anderen Teilnehmern im Netzwerk mitgeteilt werden, damit jeder seine Datenbank aktuell halten kann. Mit diesen Daten haben die Teilnehmer einer Transaktion die Möglichkeit, eine Transaktion zu verifizieren, weil sie in der Lage sind, die Deckung des Absenderkontos zu überprüfen.

Da in so einem System nun jeder Teilnehmer Zugriff auf sämtliche Kontostände hat, werden die Koten nicht durch die Namen ihrer Besitzer, sondern durch Pseudonyme repräsentiert. Dies gewährleistet allerdings noch keine vollständige Anonymität, was wir in Teil IV thematisieren.

In einem solchen Netzwerk muss die Integrität der Datenbank gewährleistet werden. Außerdem müssen Teilnehmer sich im Netzwerk authentifizieren, damit niemand in ihrem Namen Überweisungen durchführen kann. Im Folgenden werden wir diese Aspekte genauer erläutern.

[Nak]

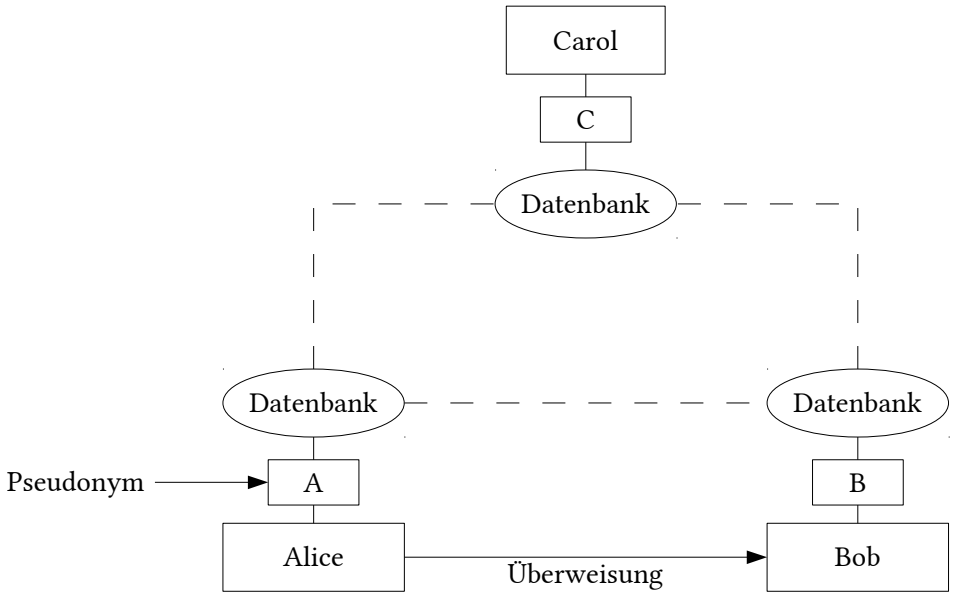

Abbildung 2: Dezentrales Kontosystem

3 Hashfunktion

Für die weiteren Ausführungen werden wir sichere kryptografische Hashfunktionen benötigen. Dieses sind Abbildungen, die folgende Kriterien erfüllen:

$$\mathcal{H} : X \to H$$
$$x \mapsto \mathcal{H}(x)$$

1. $\mathcal{H}(x)$ lässt keine Rückschlüsse auf x zu. (Einwegfunktion)
2. $x_1 \neq x_2$, für die gilt $\mathcal{H}(x_1) = \mathcal{H}(x_2)$, sind sehr schwer zu finden. (Kollisionsfreiheit)
3. $\mathcal{H}(x)$ ändern sich pseudo-zufällig, auch wenn nur geringe Änderungen an x vorgenommen werden. (Lawineneffekt)

Diese Art von Funktionen findet an mehreren Stellen Anwendung. In der Bitcoin-Implementierung wird hauptsächlich SHA-256 benutzt, welche zum SHA-2-Standard gehört. Sie besitzt eine Ausgabelänge von 256 Bits [Sha13]. Um ihre Komplexität zu erhöhen, wird sie an vielen Stellen zweimal hintereinander angewendet.

Außerdem verwendet Bitcoin RIPEMD-160, welche eine Ausgabe von 160 Bit besitzt [Bos].

4 Aufbau einer Transaktion

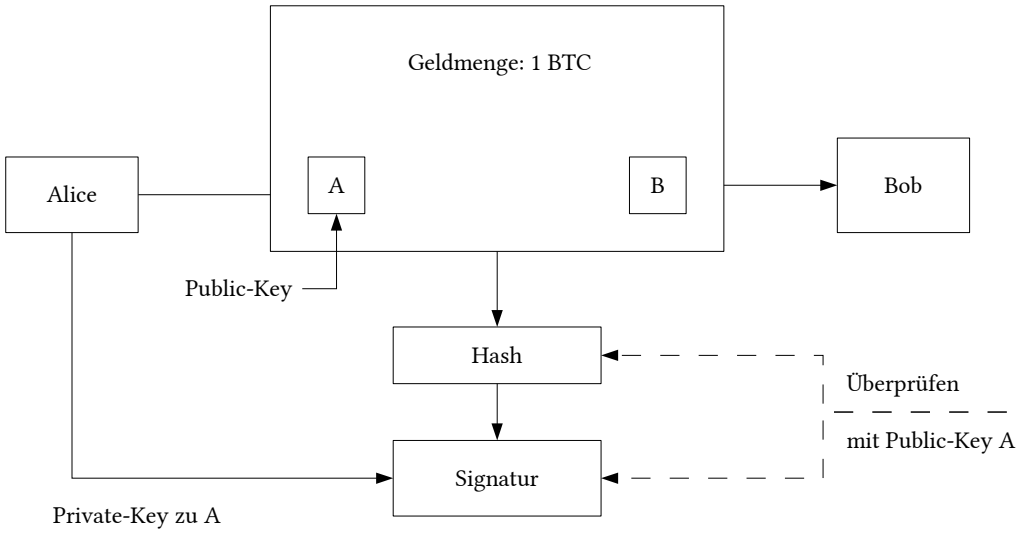

Abbildung 3: Vereinfachte Darstellung einer Transaktion

In einer Transaktion werden alle für den Transfer von Bitcoins wichtigen Informationen gespeichert. Dies sind Daten, aus denen die Herkunft des Geldes und der Empfänger bestimmt werden können. Außerdem enthält die Transaktion die umzubuchende Geldmenge und eine Signatur. Letztere stellt sicher, dass die Transaktion wirklich von dem Absender erstellt bzw. genehmigt wurde. Hierzu werden die Daten der Transaktion gehasht, dieser Hash wird mit dem Private-Key des Absenders verschlüsselt und so die Signatur erstellt. Andere Teilnehmer des Netzwerks können nun mittels des Public-Keys des Absenders den verschlüsselten Hash entschlüsseln und mit einem selber erstellten Hash der Transaktionsdaten vergleichen (siehe Abb. 3). Man nutzt für das Erstellen der Signaturen die gesamten

Transaktionsdaten, da nur so sichergestellt ist, dass sie im Nachhinein nicht geändert werden können. Auf das verwendete Public-Key-System gehen wir in Teil III genauer ein.

Es ist außerdem möglich, dass es bei einer Transaktion mehrere Empfänger oder Absender gibt. Für jeden Absender muss natürlich eine eigene Signatur hinzugefügt werden. Im Gegensatz zu einem üblichem Kontosystem gibt man als Absender kein Konto an, sondern verweist auf eine ältere Transaktion, bei der man selbst der Begünstigte war. Dies können auch mehrere Transaktionen sein, sie werden in ihrer Gesamtheit als *Eingabe-Transaktionen* bezeichnet. [Nak]

Sämtliche Daten werden dabei hintereinander binär gespeichert, um Platz zu sparen. Der genaue Aufbau sieht nach [Tra13] wie folgt aus:

Bezeichnung	Anmerkung
Versionsnummer	momentan 1
Anzahl Eingabe-Transaktionen	>0
Eingabe-Transaktionen	siehe Tabelle unten
Anzahl Ausgabe-Transaktionen	>0
Ausgabe-Transaktionen	siehe Tabelle unten
Sperrzeit	normalerweise nicht benutzt

4.1 Eingabe-Transaktion

Die einzelnen Eingabe-Transaktionen, die eine Transaktion beinhaltet, sind jeweils wie folgt aufgebaut:

Bezeichnung	Anmerkung
Hash der verwendeten Transaktion	SHA-256
Nummer der Ausgabe-Transaktion	Da die verwendete Transaktion mehrere Ausgabe-Transaktionen haben kann, muss hier eine Index-Nummer angegeben werden.
Signatur	Hash verschlüsselt mit Private-Key
Public-Key	entschlüsselt die Signatur

Da zum Erstellen der Signatur die kompletten Daten der Transaktion notwendig sind, werden zur Berechnung dieser die Signaturfelder der Eingabetransaktion vorerst mit Nullen gefüllt. Dann wird der Hash der Transaktionsdaten berechnet, dieser jeweils mit den zugehörigen Public-Keys der Eingabetransaktionen verschlüsselt und das Ergebnis in die entsprechenden Signaturfelder eingesetzt. [Tra13]

4.2 Ausgabe-Transaktion

Eine einzelne Ausgabe-Transaktion beinhaltet zwei Werte:

Name	Anmerkung
Wert in der Einheit Satoshi	10^8 Satoshi = 1 Bitcoin
Hash des Public-Keys des Empfängers	siehe 8

Hieraus folgt, dass die kleinst möglichste zu überweisende Geldmenge 10^{-8} Bitcoins beträgt. Dies entspricht momentan ungefähr 0,0001 Eurocent[1].

[Tra13]

4.3 Transaktionsgebühr

Die Summe der Bitcoins der Ausgabe-Transaktionen muss durch die Eingabe-Transaktionen gedeckt sein. Es ist jedoch möglich, dass der Wert der Eingabe-Transaktionen den der Ausgabe-Transaktionen überschreitet. Der Überschuss stellt die freiwillige Transaktionsgebühr dar. Diese wird der *Coinbase* hinzugefügt, einer speziellen Transaktion, die wir im nächsten Abschnitt erklären werden.

[Fee13]

4.4 Wechselgeld

Bei einer Überweisung wird für die Quelle des Geldes nicht etwa ein Konto angegeben, sondern eine vergangene Transaktion, bei der der Zahlende selbst der Begünstigte war. Dies wollen wir an einem Beispiel näher erläutern: Alice überweist Bob 20 Bitcoins, wobei in der Ausgabe-Transaktion der Hash seines Public-Keys verwendet wird. Möchte Bob nun Carol 5 Bitcoins überweisen, erstellt er eine neue Transaktion, in dessen Eingabe-Transaktion er den Hash der alten Transaktion angibt, sowie seine Signatur zum verwendeten Public-Key. Für die Ausgabe-Transaktion benutzt er Carols Public-Key und gibt einen Wert von 5 Bitcoins an. Dies würde nun dazu führen, dass 15 Bitcoins als Transaktionsgebühr entfallen. Da Bob dieses Geld lieber behalten möchte, fügt er der Transaktion eine weitere Ausgabe-Transaktion hinzu, die ihn selbst mit 15 Bitcoins begünstigt. Diese neue Ausgabe-Transaktion kann er in Zukunft nutzen und hat nur, wie geplant, 5 Bitcoins weniger.

Eine Ausgabe-Transaktion, die den Zahlenden selber wieder begünstigt, ist in der Praxis üblich und wird, da es dem Bezahlen mit echtem Geld ähnelt, als Wechselgeld bezeichnet [Chn13].

5 Datenbank

Die bei Bitcoin verwendete Datenbank wird, wie bereits erwähnt, nicht zentral verwaltet. Stattdessen besitzt jeder seine eigene Kopie dieser Datenbank. Alle Änderungen müssen im gesamten Bitcoin-Netzwerk mitgeteilt werden, damit jeder mit derselben Version ausgestattet ist. Nachfolgend soll erklärt werden, wie entschieden wird, welche die aktuell gültige Datenbank ist und wie ihre Integrität sichergestellt wird.

Die Datenbank enthält alle bisher getätigten Transaktionen. So kann jeder, der sie kennt, kontrollieren, ob eine neue Transaktion gültig ist, also die verwendeten Eingabe-Transaktionen nicht zuvor in einer anderen Transaktion benutzt wurden. Für die Datenbank werden die Daten in einzelne Blöcke unterteilt.

Ein einzelner Block besteht aus dem Hashwert des vorherigen Blocks und mehreren Transaktionen. Dies können beliebig viele sein, momentan wird die Blockgröße allerdings vom Bitcoin-Client beschränkt. Der im Block enthaltene Hashwert aus den Daten seines vorhergehenden Blocks gewährleistet,

[1] Kurs vom 01.09.2013 von `www.bitcoin.de`

dass nachträglich keine Änderungen mehr an einem einzelnen Block durchgeführt werden können.

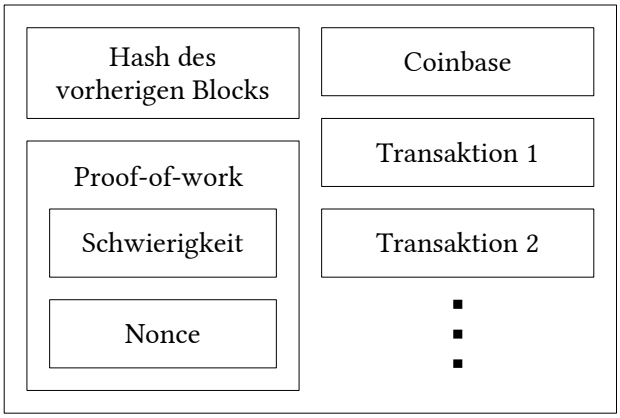

Abbildung 4: Aufbau eines Blocks

Eine beliebige nachträgliche Änderung der Daten in einem Block würde seinen Hashwert verändern. Da der nachfolgende Block den Hashwert beinhaltet, ändern sich auch dessen Daten und damit sein Hashwert. Dies gilt nun auch für den darauf folgenden Block. Es würden sich also alle nachfolgenden Blöcke und deren Hashwerte ändern. Nun könnte jeder Teilnehmer des Bitcoin-Netzwerks, der aus dem Netzwerk einen neuen Teil der Datenbank (ein oder mehrere Blöcke) abruft, sofort kontrollieren, ob dieser zu seiner bisherigen Version passt, indem er kontolliert, ob der erste neue Block, den er erhält, den Hash seines letzten Blocks verwendet. Überprüft er nun, ob die Blöcke des neuen Teils auch jeweils den passenden Hashwert enthalten, kann er sicher sein, dass er einen neuen gültigen Abschnitt der Datenbank erhalten hat.

Zunächst scheint es sehr einfach, die Datenbank zu fälschen, da das Berechnen von mehreren Hashwerten in kürzester Zeit möglich ist. Doch dies wird durch das *Proof-of-work*-System verhindert, welches das schnelle Berechnen der Hashwerte praktisch unmöglich macht. Hierbei wird der sogenannte *Nonce* verwendet. Darauf werden wir im nächsten Abschnitt genauer eingehen.

Die gesamte Datenbank, also die Verkettung der Blöcke, nennt man die *Blockchain*. Der erste Block, welcher als einziger keinen Hash eines Vorgängers enthält, wird der *Genesis-Block* genannt.

[Nak, Prt13]

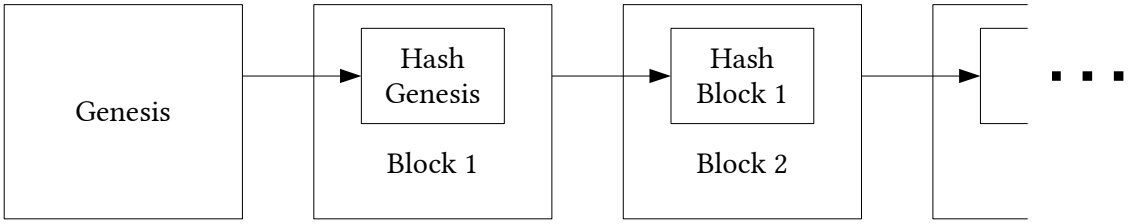

Abbildung 5: Blockchain

Wie im nächsten Abschnitt deutlich werden wird, benötigt man zum Erstellen eines Blocks eine gewisse Rechenzeit. Damit ein Anreiz geschaffen wird, trotzdem Blöcke zu erzeugen und neue Trans-

aktionen so in die Datenbank aufzunehmen, gibt es eine spezielle Transaktion in einem Block, welche keinen Absender besitzt. Sie ist die einzige Möglichkeit, im Bitcoin-Netzwerk Geld zu erzeugen, und wird *Coinbase* genannt. Der Ersteller des Blocks erhält sie mit den ihr zugefügten Transaktionsgebühren als Belohnung. Es ist jedem Teilnehmer des Bitcoin-Netzwerks möglich, einen Block zu erstellen.

6 Proof-of-work

Jeder Block in der Blockchain muss die Lösung eines sogenannten Proof-of-works enthalten. Damit wird gewährleistet, dass es nur noch mit einer verschwindend geringen Wahrscheinlichkeit möglich ist, innerhalb von kürzester Zeit sehr viele Blöcke zu erstellen. Dazu stellt man Anforderungen an den Hashwert, den die Daten eines neuen Blocks ergeben. Diese Anforderung an den Hash nennt sich *Challenge* und kann nur durch viele Versuche gelöst werden. Die genauen Mechanismen des Verfahrens sollen nachfolgend in Anlehnung an [Nak] erklärt werden.

Sei $\mathcal{H}$ nun eine kryptografische k-Bit-Hashfunktion (siehe 3):

$$\begin{aligned} \mathcal{H} : \{0,1\}^* &\longrightarrow \{0,1\}^k \\ x &\longmapsto y \end{aligned}$$

Dann sei $y\,[i]$ das i-te Bit von rechts und $a||b$ das Hintereinanderlegen zweier Bitfolgen. Die Aufgabe besteht nun darin, ein beliebiges $n \in \{0,1\}^*$ für feste Daten d zu finden, für das gilt:

$$\mathcal{H}\,(d||n) = y \qquad \text{mit} \qquad y\,[1] = y\,[2] = \cdots = y\,[s] = 0$$

$$s \in \mathbb{N},\ s \leq k$$

Dabei stellt s die sogenannte *Schwierigkeit* des Proof-of-works dar, weil so die Anzahl der festgeschriebenen Stellen des Hashs bestimmt wird und es, wie oben beschrieben schwierig ist, einen gewissen Hashwert zu erhalten. Aufgrund der Eigenschaften einer Hashfunktion lässt sich n (genannt *Nonce*) nur durch Durchprobieren finden. Da ein Computer zum Berechnen eines Hashs Zeit benötigt, ist eine gewisse durchschnittliche Rechenzeit notwendig, um n zu finden. Gelingt dies einem Teilnehmer, lässt es sich anderen leicht beweisen, indem das n offenbart wird. Zum Überprüfen muss nun lediglich ein einziger Hash berechnet werden. Der Finder des Nonce hat damit das Vollbringen einer Arbeit bewiesen, daher der Name Proof-of-work. Als Belohnung erhält er die Coinbase. Dass er n offenbart, stellt für ihn kein Risiko dar, weil andere Teilnehmer es nicht verwenden können. Würde ein anderer Teilnehmer dasselbe n in seinem Block verwenden, hätte er trotzdem einen komplett anderen Hashwert, da er eine andere Coinbase verwenden müsste, um einen Vorteil zu erlangen, und er so ein anderes d hätte.

Da beim Erstellen der Blöcke Bitcoins erschaffen werden, nennt man diesen Vorgang *Mining* und die Teilnehmer daran *Miner*.

6.1 Anpassung der Schwierigkeit

Da die Rechenleistung im Bitcoin-Netzwerk aufgrund wachsender Teilnehmerzahl und technischen Fortschritts mit der Zeit zunimmt, muss die Schwierigkeit s angepasst werden. Ziel ist es, dass nur

ungefähr alle 10 Minuten ein Proof-of-work gelöst wird und damit ein neuer Block erzeugt werden kann. Dazu wird die Schwierigkeit alle 2016 Blöcke, was ungefähr zwei Wochen entspricht, neu berechnet. Dabei orientiert man sich an der Rechenleistung des Netzwerks in der Vergangenheit, so dass in Zukunft dies zu einer durchschnittlichen Zeit, bis jemand den Proof-of-work löst, von 10 Minuten führt. Das Geldwachstum wird hiermit eingeschränkt und ist sehr gut vorhersehbar.

Außerdem wird die Anzahl der Bitcoins in der Coinbase ca. alle 2 Jahre (alle 210000 Blöcke) halbiert. Bitcoin startete im Jahr 2009 mit einer Belohnung von 50 Bitcoins pro Block. Mittlerweile ist man bei 25 Bitcoins pro Block. Es lässt sich auch die maximale Anzahl an Bitcoins berechnen:

$$\sum_{k=0}^{\infty} 210.000 \cdot \frac{50}{2^k} = 10.500.000 \cdot \underbrace{\sum_{k=0}^{\infty} \frac{1}{2^k}}_{=2} = 21.000.000$$

Außer Acht gelassen haben wir hierbei, dass $\frac{50}{2^k}$ auf 8 Stellen nach dem Komma gerundet werden muss. Der Fehler ist aber gering, sodass man sagen kann, dass es maximal etwas weniger als 21 Millionen Bitcoins geben kann.

Da irgendwann durch die Coinbase keine Bitcoins bzw. nur noch sehr wenige gewonnen werden, entfällt ein Anreiz zum Erstellen der Blöcke. Man hofft, dass die Transaktionsgebühren, die Teilnehmer des Netzwerks freiwillig bei einer Transaktion abgeben können, ausreichen, um genug Leute zum Mining zu motivieren. Miner bevorzugen bereits jetzt Transaktionen mit höherer Gebühr, wodurch diese schneller in die Blockchain aufgenommen werden.

6.2 Mehrheit der Rechenleistung

Um nun zu entscheiden, welche Version der Blockchain gültig ist, speichern Clients immer die längste ihnen bekannte Version der Blockchain. In dieser steckt im Durchschnitt die größte Rechenzeit, da für ihr Erstellen am meisten Proof-of-works nötig waren. Möchte ein Angreifer die Blockchain zu seinen Gunsten manipulieren, benötigt er also mehr Rechenleistung als die Hälfte der Rechenleistung des Bitcoin-Netzwerks. Dies ist momentan bereits fast unmöglich, da die geschätzte Rechenleistung des Bitcoin-Netzwerks zur Zeit weit über der Summe der Rechenleistung der 500 besten Supercomputer liegt [Bru13].

7 Netzwerk

Die Arbeitsschritte eines Clients, der am Mining beteiligt ist, sind nach [Nak] wie folgt:

1. Neu eingegebene Transaktionen werden an alle Teilnehmer gesendet.
2. Transaktionen, die noch nicht in der Blockchain enthalten sind, werden zu einem Block zusammengefügt.
3. Es wird versucht, den Proof-of-work für diesen Block zu finden.
4. Wenn man den Proof-of-work findet, wird der fertige Block an alle anderen Teilnehmer gesendet.

5. Der Block wird nur akzeptiert, wenn alle Transaktionen gültig sind und alle Eingabe-Transaktionen nicht bereits in anderen Transaktionen verwendet wurden.

6. Wurde der Block akzeptiert, arbeiten die anderen Teilnehmer nun an einem Block, der den akzeptierten als Vorgänger hat und dessen Hashwert beinhaltet.

Kommt es zu dem Fall, dass zwei Teilnehmer zeitnah einen Block fertigstellen und im Netzwerk verteilen, arbeiten die anderen jeweils mit dem Block, den sie zuerst erhalten haben. Sobald für eine der beiden Versionen ein weiterer Block gefunden wurde, verwerfen die Teilnehmer, die mit dem anderen Block arbeiten, ihre Version der Blockchain, da sie kürzer ist. Vereinfacht gesagt wird immer die Blockchain akzeptiert, in der am meisten Arbeit steckt.

Möchte man wirklich sicher sein, dass eine Transaktion, die man erhalten hat, endgültig ist, sollte man also mindestens zwei Blöcke abwarten. Dies dauert im Schnitt 20 Minuten und ist damit immer noch schneller als die meisten traditionellen Banküberweisungen. Hierbei ist allerdings zu beachten, dass nicht automatisch alle Transaktionen, die in dem verworfenen Block lagen, nicht mehr gültig sind. Denn diese können genauso gut in der neuen Version der Blockchain vorkommen.

8 Bitcoin-Adresse

Möchte man jemandem Bitcoins überweisen, benötigt man den in der Ausgabe-Transaktion einzufügenden Hashwert, der den Empfänger identifiziert. Dieser Hashwert ist Teil einer Bitcoin-Adresse. Um eine eigene Bitcoin-Adresse zu berechnen, benötigt man ein Public-Private-Schlüsselpaar. Woraus dieses besteht, wird in Teil III genauer erläutert. Zunächst wird der öffentliche Schlüssel P mittels SHA-256 gehasht. Dieser Hash wird nochmal mit dem Hashstandard RIPEMD-160 gehasht, um eine etwas kürzere Adresse zu erhalten.

$$h = \text{RIPEMD-160}\left(\text{SHA-256}\left(P\right)\right)$$

Dieser Hashwert wird auch in einer Ausgabe-Transaktion verwendet, um den Besitzer zu identifizieren (siehe 4.2).

Es handelt sich aber noch nicht um die endgültige Bitcoin-Adresse. Damit es möglich ist, dass es auch noch andere Typen von Adressen gibt (z. B. Gemeinschaftskonten), wird vor diesen Hash ein Versionsbyte v gehängt. Bei normalen Adressen ist dies momentan immer 0.

Um schnell überprüfen zu können, ob eine Bitcoin-Adresse gültig ist, werden noch vier Prüfbytes angehängt. Dazu wird die bisherige Adresse $v||h$ noch zweimal mit SHA-256 gehasht und die ersten 4 Bytes davon ans Ende gesetzt.

$$\begin{aligned} b &= \text{SHA-256}\left(\text{SHA-256}\left(v||h\right)\right) \\ a &= v||h||b[1\ldots4] \end{aligned}$$

So kann jeder leicht nachrechnen, ob eine von ihm erhaltene Bitcoin-Adresse a gültig ist.

Um die Adresse so kurz wie möglich aber trotzdem für Anwender lesbar zu halten, wird a nicht als Binärdaten oder in Hexadezimalschreibweise angegeben. Stattdessen wird eine Kodierung zur Basis 58

verwendet (genannt *Base-58-Kodierung*). Hierzu werden alle Zahlen sowie Groß- und Kleinbuchstaben verwendet. Die Repräsentanten 0, O, l und I werden weggelassen, da man sie oft nicht auseinander halten kann. Folgende Tabelle gibt einen Überblick über die Reihenfolge der Kodierung:

Wert	Repräsentant	Wert	Repräsentant	Wert	Repräsentant	Wert	Repräsentant
0	1	15	G	30	X	45	n
1	2	16	H	31	Y	46	o
2	3	17	J	32	Z	47	p
3	4	18	K	33	a	48	q
4	5	19	L	34	b	49	r
5	6	20	M	35	c	50	s
6	7	21	N	36	d	51	t
7	8	22	P	37	e	52	u
8	9	23	Q	38	f	53	v
9	A	24	R	39	g	54	w
10	B	25	S	40	h	55	x
11	C	26	T	41	i	56	y
12	D	27	U	42	j	57	z
13	E	28	V	43	k		
14	F	29	W	44	m		

[Adr13, Bas13]

8.1 Beispiel

An einem kleinen Beispiel nach [Adr13] soll das Erstellen einer Bitcoin-Adresse verdeutlicht werden, wobei alle Zahlen in Hexadezimalschreibweise angegeben werden:

$$\begin{aligned} P = {} & 0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B235 \\ & 22CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6 \end{aligned}$$

$$\begin{aligned} h &= \text{RIPEMD-160}\left(\text{SHA-256}\left(D\right)\right) \\ &= 010966776006953D5567439E5E39F86A0D273BEE \end{aligned}$$

$$v = 00$$

$$\begin{aligned} b &= \text{SHA-256}\left(\text{SHA-256}\left(00010966776006953D5567439E5E39F86A0D273BEE\right)\right) \\ &= D61967F63C7DD183914A4AE452C9F6AD5D462CE3D277798075B107615C1A8A30 \end{aligned}$$

$$a = 00010966776006953D5567439E5E39F86A0D273BEE\underbrace{D61967F6}_{b[1\ldots4]}$$

Umgewandelt zur Base-58-Kodierung:

$$a = 16\text{UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM}$$

9 Sonstiges

9.1 Wallet

Wie bereits deutlich wurde, kann man alle Bitcoins ausgeben, solange man im Besitz der Private-Keys ist, die durch die jeweiligen Transaktionen begünstigt wurden. Da es möglich ist, nahezu beliebig viele neue Schlüsselpaare zu erstellen (siehe 11.1) und ein einzelner Netzwerkteilnehmer sehr viele solcher Paare besitzen kann, fassen heutige Bitcoin-Clients diese im sogenannten *Wallet* zusammen. Dabei handelt es sich meist um eine Datei, welche oft durch ein Passwort geschützt wird.

Gehen ein oder mehrere Private-Keys verloren, ist es nicht mehr möglich, die entsprechenden Bitcoins auszugeben. Somit sind sie dem Geldkreislauf entzogen und praktisch nicht mehr vorhanden. Theoretisch könnte dies in Zukunft ein Problem von Bitcoin darstellen, da es aufgrund der geringeren Geldmenge zu einer Deflation kommen würde.

[Wal13]

9.2 Bitcoin-Exchange

Möchte man Bitcoins in andere Währungen tauschen, kann man dies an einem sogennanten *Bitcoin-Exchange* durchführen. Es gibt verschiedene private Unternehmen, die diesen Service anbieten. Meist verlangen diese Unternehmen für ihren Dienst eine Tauschgebühr, ähnlich wie beim Devisenhandel. Der populärste Bitcoin-Exchange ist zur Zeit Mt.Gox [Gox13].

10 Sicherheit

Angriffe auf das Bitcoin-Netzwerk sind prinzipiell möglich. Allerdings, wie im Folgenden gezeigt wird, können diese - nach jetzigem Kenntnisstand - nur durchgeführt werden, wenn man Zugang zu enorm hoher Rechenleistung hat. Da man diese Rechenleistung im Bitcoin-Netzwerk auch dazu nutzen kann, mittels Mining Geld zu verdienen, erscheinen die Anreize für einen solchen Angriff minimal. Ein Angreifer müsste sich also entscheiden, ob er die Sicherheit des Netzwerk unterstützt und dafür belohnt wird oder die Sicherheit des Netzwerks verringert, wodurch er vielleicht mehr Bitcoins erhalten könnte. Hierbei sollte er allerdings bedenken, dass Bitcoins beträchtlich an Wert verlieren, wenn bekannt wird, dass ihm ein solcher Angriff gelungen ist.

10.1 Angriff auf die Blockchain

Es sei:

p = Wahrscheinlichkeit, dass irgendein ehrlicher Teilnehmer den nächsten Block erstellt.

q = Wahrscheinlichkeit, dass der Angreifer den nächsten Block erstellt.

w_z = Wahrscheinlichkeit, dass der Angreifer einen Rückstand von z Blöcken jemals aufholt.

Nehmen wir zunächst an $p > q$. Die Wahrscheinlichkeit, dass der Angreifer den Rückstand von einem Block gegenüber der Menge der ehrlichen Teilnehmer aufholen kann, entspricht:

$$\begin{aligned} w_1 &= \sum_{i=1}^{\infty} a_i q^i p^{i-1} \\ &= \sum_{i=1}^{\infty} a_i q^i \left(1-q\right)^{i-1} \\ &= \sum_{i=1}^{\infty} a_i q^i \left(\left(-1\right)^{i-1} \left(q-1\right)^{i-1}\right) \end{aligned}$$

Für die a_i gelten dabei die folgenden Rekursionsformeln

$$\begin{aligned} a_1 &= 1 \\ a_{i+1} &= \binom{2i+1}{i} - \sum_{j=1}^{i} a_j \binom{2i-2j}{i-j} \end{aligned}$$

die man aus folgender einfacher Überlegung erhalten kann: Zunächst definieren wir die Ereignisse, dass ein Angreifer den nächsten Block erstellt, als Q und das Gegenereignis als P. Dann führen uns folgende Ereignisketten zum gewünschten Ereignis:

$$\begin{array}{ll} a_1 = 1 & Q \\ a_2 = 1 & PQQ \\ a_3 = 2 & PPQQQ, PQPQQ \\ a_4 = 5 & PPPQQQQ, PPQPQQQ, PPQQPQQ, PQPPQQQ, PQPQPQQ \\ & \vdots \end{array}$$

Diese Ereignisketten erfüllen dabei mehrere Bedingungen:

1. Q kommt einmal häufiger vor als P.
2. In der i-ten Zeile kommt Q genau i-mal vor.
3. Keine der Ereignisketten aus der i-ten Zeile kommen als Anfang in der $(i+1)$-ten Zeile vor.

Mit dem binomischen Lehrsatz

$$\left(x+y\right)^n = \sum_{k=0}^{n} \binom{n}{k} x^{n-k} y^k$$

gilt:

$$w_1 = \sum_{i=1}^{\infty} a_i q^i \sum_{k=0}^{i-1} \binom{i-1}{k} \left(-q\right)^k$$

$$= \sum_{i=1}^{\infty} a_i q^i \sum_{k=0}^{i-1} \binom{i-1}{k} (-1)^k q^k$$
$$= \sum_{i=1}^{\infty} a_i \sum_{k=0}^{i-1} \binom{i-1}{k} (-1)^k q^{k+i}$$

Schreiben wir diese Summe aus, so haben die Summanden, nach der Laufvariable i gelistet, folgende Gestalt:

$$\begin{array}{ll}
i=1 & a_1q^1 \\
i=2 & a_2q^2 - a_2q^3 \\
i=3 & a_3q^3 - 2a_3q^4 + a_3q^5 \\
i=4 & a_4q^4 - 3a_4q^5 + 3a_4q^6 - a_4q^7 \\
i=5 & a_5q^5 - 4a_5q^6 + 6a_5q^7 - 4a_5q^8 + a_5q^9 \\
i=6 & a_6q^6 - 5a_6q^7 + 10a_6q^8 - 10a_6q^9 + 5a_6q^{10} - a_6q^{11} \\
\vdots & \vdots
\end{array}$$

Wir fassen auf den Diagonalen alle selben q-Potenzen zusammen, womit für die Summe folgt:

$$\sum_{i=1}^{\infty} a_i \sum_{k=0}^{i-1} \binom{i-1}{k} (-1)^k q^{k+i} = \sum_{i=1}^{\infty} \left(\sum_{k=0}^{\left[\frac{i-1}{2}\right]} \binom{i-1-k}{k} (-1)^k a_{i-j} \right) q^i$$

Mittels Induktion kann man nun zeigen:

$$\sum_{k=0}^{\left[\frac{i-1}{2}\right]} \binom{i-1-k}{k} (-1)^k a_{i-j} = 1$$

Es folgt insgesamt:

$$w_1 = \sum_{i=1}^{\infty} a_i q^i p^{i-1} = \sum_{i=1}^{\infty} q^i = q \sum_{i=0}^{\infty} q^i = \frac{q}{1-q} = \frac{q}{p}$$

Damit haben wir eine leichte Formel gefunden, welche die Wahrscheinlichkeit beschreibt, dass ein Angreifer den Rückstand von einem Block aufholt. Die Wahrscheinlichkeit für das Aufholen von k Blöcken entspricht dann $w_z = w_1^z = \left(\frac{p}{q}\right)^z$.

Im Fall $q \geq p$ gilt offensichtlich $w_1 = 1$ und $w_z = 1 \quad \forall z \in \mathbb{N}$. Zusammengefasst können wir also schreiben:

$$w_z = \begin{cases} 1 & q \geq p \\ \left(\frac{q}{p}\right)^z & \text{sonst} \end{cases}$$

Wenn wir den Fall $q < p$ annehmen, der nach [Bru13] weitaus realistischer ist, geht diese Funktion aufgrund des exponentiellen Zusammenhangs sehr schnell gegen Null, falls mehrere Blöcke aufgeholt werden müssen. Für genauere Details siehe [Nak, §11].

10.2 Double-Spend

Ein allgemeines Problem von digitalem Geld ist es, dass beim Geldtransfer nur ein Austausch von Daten stattfindet. Diese Daten sind beliebig kopierbar. Der Zahlende verliert durch das Senden der Daten also nichts, wie er es tun würde, wenn er mit Bargeld bezahlt. Deswegen muss von Bitcoin sichergestellt werden, dass ein Begünstigter sicher sein kann, dass der Zahlende die entsprechenden Bitcoins wirklich besitzt. Es ist also ein wichtiges Ziel von Bitcoin, dass es keinem Angreifer gelingt, Bitcoins, die er eigentlich nicht mehr besitzt, trotzdem noch auszugeben. Diesen Angriff nennt man *Double-Spend*. Ein Double-Spend ist theoretisch im Bitcoin-Netzwerk nicht möglich, da alle Clients ihre Blockchain überprüfen können und so ungedeckte Transaktionen dem Geschädigtem sofort auffallen. Problematisch wird es nur, wenn es einen Fehler im Quellcode des Bitcoin-Client oder im Protokoll gibt. Dabei kann es passieren, dass ein Teil des Netzwerks mit einer anderen Version der Blockchain arbeitet, die ein anderer Teil nicht als gültig ansieht. Das Netzwerk spaltet sich also ab einer gewissen Stelle auf und ist sich nicht mehr darüber einig, mit welcher Datenbank gearbeitet werden muss. Hierbei spricht man von einer *Blockchain-Fork*.

Genau dies geschah im März 2013. Der am weitesten verbreitete Bitcoin-Client *bitcoind* stieg in Version 0.8 auf eine neue Datenbank-Implementierung um. Diese erlaubte nun mehr als 5000 Transaktionen pro Block, während Version 0.7 und noch ältere Clients diese ablehnten. Als ein Miner mit der Version 0.8 genau solch einen Block erstellte, baute nur ein Teil der anderen Miner diesen in ihre Version der Blockchain ein. Der andere Teil des Netzwerks lag einen Block zurück und konnte den Vorsprung nicht mehr einholen, da die Mehrheit der Rechenleistung bereits bei Clients mit Version 0.8 lag. Wären diese Clients noch in der Minderheit gewesen, hätte nach einiger Zeit die 0.7-Variante der Blockchain die andere überholt und wäre somit wieder die längste und damit gültige Blockchain geworden. Somit wäre das Problem von allein gelöst worden. Stattdessen existierten für mehrere Stunden zwei Versionen der Blockchain und ein manuelles Eingreifen war notwendig.

Die Entwickler von *bitcoind* mussten sich entscheiden, ob man die 0.7-Variante oder die 0.8-Variante weiter benutzen wollte. In letzterem Fall hätten alle Clients dazu aufgefordert werden müssen, auf die neue Bitcoin-Version zu aktualisieren. Da man dies in kurzer Zeit nicht für möglich hielt, benachrichtigte man stattdessen die wichtigsten Miner. Sie sollten wieder Version 0.7 installieren und somit dieser Variante der Blockchain helfen, länger zu werden. Nach 6 Stunden war es soweit, was nun auch dazu führte, dass Teilnehmer des Netzwerks mit Version 0.8 die andere Variante akzeptierten und ihre verwarfen.

Innerhalb dieses kurzen Zeitraums gelang es einem Mitglied, seine Bitcoins zweimal auszugeben, indem er eine Transaktion auf der 0.7-Variante und eine andere (mit anderem Empfänger) auf der 0.8-Variante platzieren konnte. Dies hätte ihm nur dann einen Vorteil verschafft, wenn er bereits eine Leistung für die Bitcoins erhalten hätte, sich z. B. damit etwas gekauft hätte. Allerdings wurde seine Transaktion in der 0.8-Blockchain wieder verworfen, als das Problem nach 6 Stunden gelöst wurde.

[But13, Dbl13]

Teil III

Digitale Signaturen

In diesem Kapitel möchten wir vorstellen, wie das digitale Signieren innerhalb des Bitcoin-Netzwerks umgesetzt ist. Dazu gehen wir zuerst allgemein auf die Public-Key-Kryptografie ein. Dieser Teil beinhaltet die theoretischen Grundlagen sowie das konkrete Beispiel der RSA-Verschlüsselung. Anschließend soll das in Bitcoin verwendete Public-Key-System, welches auf Gruppen über Elliptischen Kurven (EK) basiert, vorgestellt werden. Hierzu wird zunächst in die zugrundeliegende Mathematik der EK eingeführt und die Gruppenstruktur, die auf eben diesen definiert werden kann, aufgezeigt. Der auch in Bitcoin verwendete ECDSA-Standard wird darauf folgend beschrieben. Dabei sollen verschiedene Möglichkeiten von Angriffen auf diese Verschlüsslung diskutiert werden und so verdeutlichen welche Elliptischen Kurven sich für digitalen Signaturen eignen. Weiterführend wird dargelegt werden welche spezielle EK gewählt wurde und aus welchen Gründen. Dazu wird auf die Effizienz von verschiedenen EK eingegangen und EK-Kryptografie mit RSA-Kryptografie verglichen. Zuletzt wird geklärt werden, warum sich gerade diese Art von Verschlüsselung für Bitcoin eignet.

11 Public-Key-Kryptografie

11.1 Verschlüsselung

Bis in die 70er Jahre gab es nur Symmetrische Verschlüsselungsverfahren. D. h. der Schlüssel zum Verschlüsseln einer Nachricht wurde auch dazu verwendet die Chiffre wieder in Klartext umzuwandeln. Als nun das erste Public-Key-System eingeführt wurde, löste man das Problem des Schlüsselaustauschs. Außerdem wurde es möglich Nachrichten zu signieren. Diese Public-Key-Systeme sind dabei im Allgemeinen formal wie folgt definiert:

1. Es ist möglich mehrere Schlüsselpaare (S, P) zu erzeugen.

2. Für die Abbildungen $S(t)$ und $P(c)$ gilt:

$$\begin{aligned} S : T &\to C \\ t &\mapsto S(t) = c \end{aligned}$$

$$\begin{aligned} P : C &\to T \\ c &\mapsto P(c) = t \end{aligned}$$

   Dabei wird S als Secret-Key (auch Private-Key), P als Public-Key, T als Textraum und C als Chiffreraum bezeichnet.

3. Kennt man P ist es praktisch unmöglich die zugehörige Umkehrabbildung S zu bestimmen.

4. Für zwei Schlüsselpaare (S_i, P_i) und (S_j, P_j) gilt immer:

$$P_i(S_j(t)) \neq t \qquad \forall t \in T \text{ und } i \neq j$$

[Pub13, Wer02]

11.2 Digitale Signaturen

Wie bereits erwähnt, ist es mit solchen Public-Key-Systemen möglich Nachrichten $m \in T$ zu signieren. Dazu wird ein Schlüsselpaar (S, P), welches der Signierende erzeugt hat, benötigt. Nun wird wie folgt vorgegangen:

1. Zunächst wird der Public-Key P im Netzwerk der Empfänger veröffentlicht.
2. Man einigt sich im Netzwerk auf eine Hashfunktion $\mathcal{H}$.
3. Man berechnet $S(\mathcal{H}(m))$.
4. m und $S(\mathcal{H}(m))$ werden im Netzwerk veröffentlicht.
5. Nun kann jeder Empfänger von m und $S(\mathcal{H}(m))$ kontrollieren ob $P(S(\mathcal{H}(m))) = \mathcal{H}(m)$

Da nur der Absender Kenntnis von S hat, ist sichergestellt, dass er die Nachricht im Netzwerk verbreitet hat.

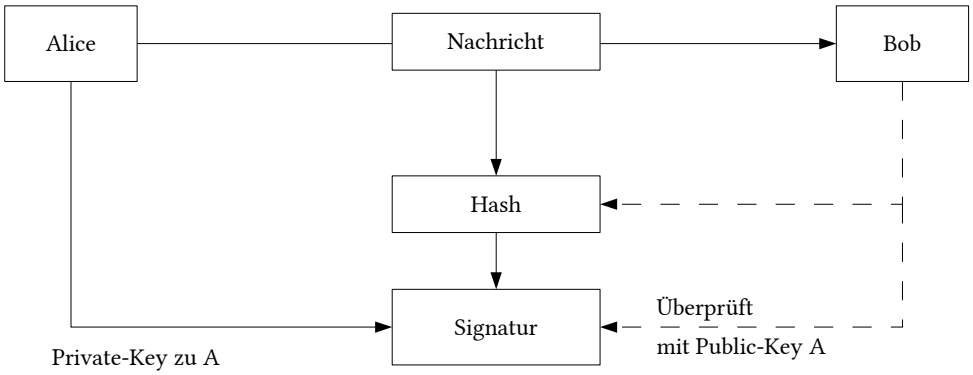

Abbildung 6: Schema digitaler Signaturen

[Sig13]

11.3 Diskreter Logarithmus

Viele Public-Key-Systeme und auch der ECDSA-Algorithmus, der in Bitcoin verwendet wird, basieren auf einer geeigneten Gruppe G mit einer Verknüpfung $+$ und dem neutralen Element 0. Man betrachte dabei eine geeignete Untergruppe $\langle P \rangle$ wobei P die Ordnung $n \in \mathbb{N}$ besitzt. Berechnet man nun in solch

einer Gruppe, die das diskrete Logarithmus-Problem besitzt:

$$\underbrace{P + P + ... + P}_{k-mal} = kP = Q$$

dann ist für das entstehende Paar (P, Q) sehr viel Rechenaufwand nötig, um das passende k zu bestimmen. Der Name diskretes Logarithmus-Problem (DLP) leitet sich daher ab, dass man bei vielen Public-Key-Systemen mit multiplikativen Untergruppen bereits bekannter Körper arbeitet. Schreibt man die Verknüpfung mit $\cdot$ anstelle von $+$, so stellt sich das Problem wie folgt dar:

$$\underbrace{P \cdot P \cdot ... \cdot P}_{k-mal} = P^k = Q$$

Hierbei entspräche das Suchen des passenden ks dem Lösen einer Exponentialfunktion. Schreibt man dieses Problem in der obigen Form, wäre der Begriff diskretes Produkt-Problem intuitiver. Wir haben uns dennoch für die erste Form entschieden, da es so bei der Gruppe über EK üblich ist.

[Wer02]

11.4 RSA

Als nächstes möchten wir eine Einführung in das wohl am weitesten verbreitete Public-Key-System geben. Es hat den Namen RSA, was als Kürzel für die Namen der Designer Ron Rivest, Adi Shamir und Leonard Adleman steht.

In diesem System sind privater und öffentlicher Schlüssel wie folgt aufgebaut: Als Grundlage wählt sich ein Nutzer zwei große und verschiedene Primzahlen p und q. Aus diesen bestimmt er dann die natürlich Zahl $n := pq$, genannt RSA-Modul und berechnet anschließend den Wert der Eulerschen φ-Funktion von n. Dieser Wert entspricht der Anzahl der zu n teilerfremden Zahlen, die kleiner als n sind. Da man die Primfaktorzerlegung von n kennt, ist dies einfach:

$$|\{m \in \mathbb{N} : m < n,\ \mathrm{ggT}(m, n) = 1\}| = \varphi(n) = (p-1) \cdot (q-1)$$

Als nächstes wählt der Nutzer noch eine Zahl e zwischen 1 und $\varphi(n)$. Das Tupel (n, e) bildet nun den öffentlichen Schlüssel des Nutzers. Um den privaten Schlüssel, der auch aus einer natürlichen Zahl d besteht, zu erzeugen, bestimmt er d mit

$$ed \equiv 1 \mod \varphi(n) \qquad \text{und } d < \varphi(n)$$

Dies ist augenscheinlich nur möglich wenn man $\varphi(n)$ kennt.

Nun wollen wir noch zeigen, wie der Nutzer mit diesem Schlüsselpaar eine Nachricht signieren kann. Dazu gehen wir im Folgenden davon aus, dass die Nutzerin, genannt Alice (A), die das Schlüsselpaar erzeugt hat, einem beliebigen anderen Nutzer Bob (B) des Netzwerks beweisen möchte, dass eine Nachricht m tatsächlich von ihr stammt. Hierzu benötigt sie noch eine kryptografische Hashfunktion $\mathcal{H} : M \to \mathbb{Z}_n$ und nutzt folgendes Protokoll:

1. A erzeugt $h = \mathcal{H}(m)$

2. Sie nutzt die Verschlüsselungsabbildung E und bestimmt $E(h)$:

$$\begin{aligned} E : \mathbb{Z}_n &\to \mathbb{Z}_n \\ h &\mapsto E(h) := h^d \mod n \end{aligned}$$

3. Nun kann A das Tupel $(m, E(h))$ an B senden.

Um zu überprüfen, ob die Nachricht wirklich von A kommt, benötigt B den öffentlichen Schlüssel von A. Dieser ist im Netzwerk verfügbar.

4. B überprüft dann auf Richtigkeit:

$$E(h)^e = h^{de} \equiv h \mod n$$

5. Ist diese Gleichheit erfüllt, so akzeptiert er die Nachricht bzw. das Tupel $(m, E(h))$.

Wenn nach dem Protokoll vorgegangen wird, kann B sicher sein, dass A die Absenderin der Nachricht m bzw. des Tupels $(m, E(h))$ ist. Diese Sicherheit soll nun mathematisch begründet werden.

Ein möglicher Angreifer soll aus der Signatur $E(h)$ und der Nachricht m nicht den privaten Schlüssel d berechnen kann. Zunächst gilt, kennt man $E(h)$ und h, welche die Gleichheit $E(h) = h^d \mod n$ erfüllen, so ist es schwer d zu berechnen. Dieses Problem entspricht einem, wie im vorherigem Kapitel beschriebenem, diskretem Logarithmus-Problem auf der Gruppe $\mathbb{Z}_n^*$. Da nun p und q verschiedene Primzahlen sind und $h < n = pq$ gilt:

$$\mathrm{ggT}(h, p) \in \{1, p\} \quad \vee \quad \mathrm{ggT}(h, q) \in \{1, q\}$$

1. Fall: $\mathrm{ggT}(h, p) = p$
$\Rightarrow h = p^r$, so folgt offensichtlich $h^{de} = p^{rde} \equiv h = p^r \mod p$

2. Fall: $ggT(h, p) = 1$
so folgt mit dem kleinen Satz von Fermant:

$$h^{de} = h^{1+a\varphi(n)} = h \cdot h^{a\varphi(p)\varphi(q)} = h \cdot \underbrace{\left(h^{a\varphi(q)}\right)^{\varphi(p)}}_{\equiv 1 \mod p} \equiv h \mod p$$

3. und 4. Fall: $ggT(h, p) = 1 \quad \vee \quad ggT(h, p) = p$
kann äquivalent zu den anderen beiden Fällen behandelt werden. So kann gezeigt werden, dass:

$$h^{de} = h \mod q$$

Mittels des Chinesischem-Restsatzes lässt sich beweisen, dass $h^{de} \equiv h \mod n$ gilt. Es ist nur mit sehr hohem Aufwand für einen Fälscher möglich d zu berechnen, da man dazu $\varphi(n)$ bzw. die Primfaktoren p und q kennen muss.

Für die Praxis ist es daher wichtig p und q so groß zu wählen, dass alle bekannten Verfahren von Primfaktorzerlegung sehr aufwändig sind. Ferner ist nicht bewiesen, dass das Finden der Primfaktorzerlegung ein schwieriges Problem ist. D. h. es ist nicht sicher, ob nicht in naher Zukunft eine effiziente Methode gefunden wird. Die Verwendung der Hashfunktion ist notwendig, da so eine Nachricht beliebiger Länge mit dem selben Schlüssel verschlüsselt werden kann. Allerdings kann eine Hashfunktion auch zu Problemen führen, da z. B. eine Kollision, von einem Fälscher dazu genutzt werden kann, eine Unterschrift unter eine nicht signierte Nachricht zu setzten.

[Wer02]

11.5 Andere geeignete Gruppen

Es ist leicht vorstellbar, dass noch andere Gruppenklassen, als die im RSA-System verwendeten multiplikativen Untergruppen von Restklassen dazu geeignet sind Nachrichten zu signieren bzw. zu verschlüsseln. Eine allgemeine Theorie, wie diese Verschlüsselungen und Signaturen funktionieren und aufgebaut sein können, wurde von T. ElGamel aufgezeigt [EG85]. In seinem Werk stellt er vor, wie nach dem genannten RSA-Beispiel zu vermuten war, dass für eine Verschlüsselung nach seinem Schema eine abelsche Gruppe von Nöten ist, auf denen eine diskretes Logarithmus Problem existiert. Dieser Logarithmus darf mit jedem bekannten Verfahren nur mit extrem hohen Rechenaufwand lösbar sein. Eine zusätzliche Forderung, die aus der Praxis der Computer gestützten Kryptografie resultiert, besagt, dass eine geeignet Gruppe endlich sein muss, da sie sonst numerisch nicht komplett erfasst werden kann. Jedoch sollte die Gruppenordnung nicht allzu gering gewählt werden, weil sonst mittels der Brute-Force-Methode oder ähnlichen naiven Verfahren dass DLP gelöst werden kann.

Eine sehr gut geeignete Gruppe lässt sich auf einer sogenannten Elliptischen Kurve über endlichen Körpern definieren. Diese liefert eine starkes DLP, für welches vergleichsweise wenige und eher ineffiziente Angriffs-Algorithmen existieren. Da sich dieses Public-Key-System als sehr effizient bewiesen hat, kommt es in der Praxis oft zur Anwendung. So setzt Bitcoin beim nötigen digitalen Signieren einer Transaktion auf EK, zumal dadurch viel Speicherplatz in der Blockchain eingespart werden kann12.7.

[Wer02]

12 Elliptische Kurven

Im Folgenden soll in die Mathematik von Elliptischen Kurven (EK) grundlegend eingeführt werden. Dabei soll zunächst deutlich werden, was genau eine EK ist und wie auf ihr eine Gruppe mit einem diskreten Logarithmus Problem definiert werden kann. Zunächst werden wir jedoch eine kurz gehaltene intuitive Einführung geben, was man sich unter einen solchen Kurve bzw. Gruppe vorstellen kann. Dies soll helfen bei den späteren mathematischen Ausführungen den Überblick zu behalten.

12.1 Intuitive Einführung

12.1.1 Elliptische Kurven

EK sind die Nullstellenmengen N_f von einem Polynom $f(x, y)$ mit zwei Variablen. Diese sind über einen Körper $\mathbb{K}$ definiert. Die Elemente einer EK sind also alle Paare $(x, y) \in \mathbb{K}^2$ für die gilt $f(x, y) = 0$.

Diese Polynome sind nicht beliebig gewählt, wie in den nächsten Abschnitten deutlich werden wird. Wählt man als Körper beispielsweise $\mathbb{R}$ und als Polynom $f(x, y) = x^3 - 3x - y^2 + 7$ so ergibt sich grafisch folgendes Bild:

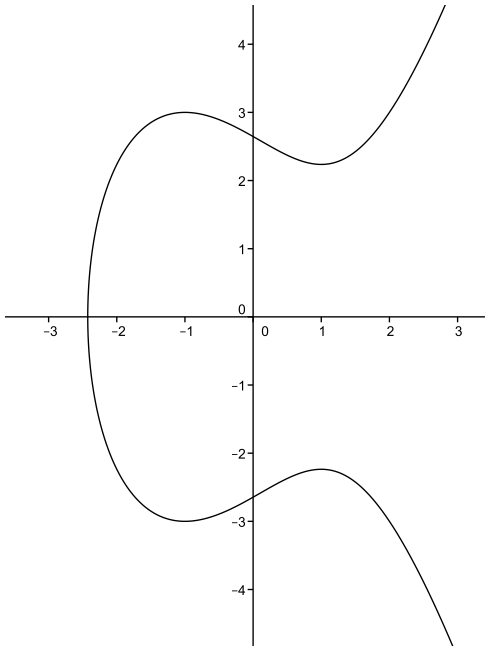

Abbildung 7: Beispiel $y^2 = x^3 - 3x + 7$

In den später vorgestellten kryptografischen Anwendungen wird $\mathbb{K}$ immer endlich sein. Daher ergibt sich keine Kurve im eigentlichem Sinne sondern nur eine Punktmenge in $\mathbb{K}^2$.

12.1.2 Gruppe auf Elliptischen Kurven

Um eine Gruppe auf so einer Menge N_f zu definieren bedarf es einer Verknüpfung $+ : N_f^2 \to N_f$ die die Gruppenaxiome erfüllt. Um dies zu erreichen wird $+$ folgendermaßen definiert: Wollen wir zwei Punkte $A, B \in N_f$ verknüpfen, erzeugen wir eine Gerade g die durch die beiden Punkte verläuft. Nun ist es bei EK so, dass für jede solche Gerade noch ein dritter Punkt $C' \in N_f$ existiert, der ebenfalls auf der so bestimmten Gerade g liegt. Diesen Punkt spiegelt man anschließend noch an der x-Achse und erhält so den Punkt C mit $C = A + B$ (siehe Abb. 8)

Um nun die Verknüpfung zu vervollständigen, muss man noch definieren, wie man ein Gruppenelement mit sich selber verknüpft, also wie man $A + A$ behandelt. In Diesem Fall wird keine diskrete Gerade erzeugt. Daher wählen wir die eindeutige Tangente in A und behandeln diese genauso wie im Fall $A + B$.

Glaubt man, dass immer der dritte bzw. zweite Punkt C' existiert und eindeutig ist, kann man leicht einsehen, dass diese Verknüpfung in N_f abgeschlossen ist. Als neutrales Element wählt man sich einen zusätzlichen Punkt in der unendlichen Ferne, sodass die erzeugte Gerade parallel zu y-Achse verläuft. Nun kann man sich auch die Existenz des inversen Elements überlegen. Spiegelt man einen Punkt an der x-Achse und verknüpft ihn mit dem ursprünglichen Punkt, ist der dritte erhaltene Punkt

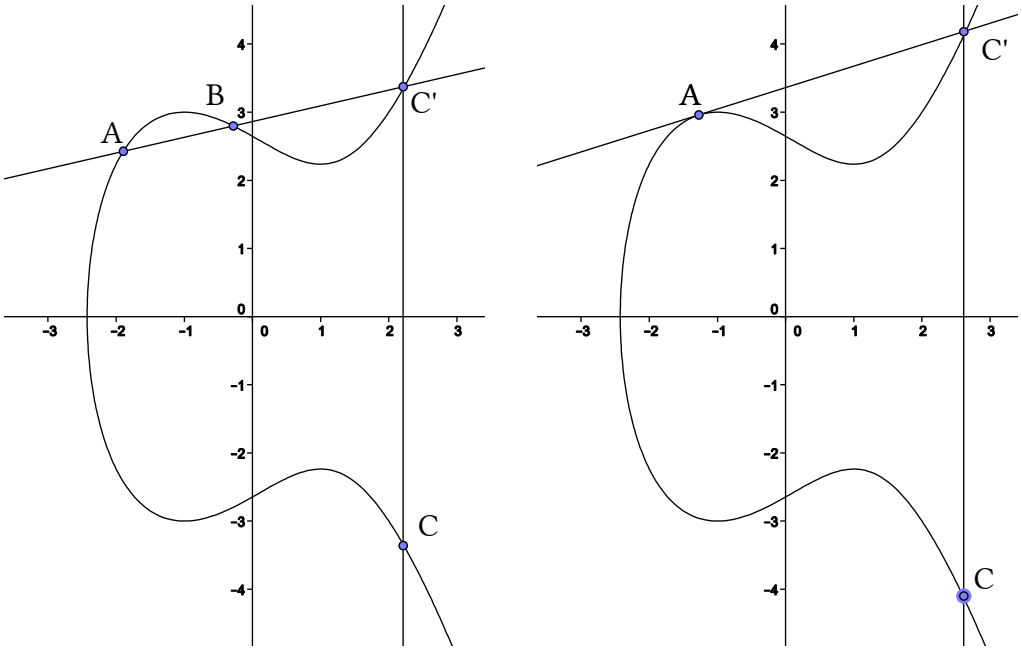

Abbildung 8: Addition und Dopplung von Punkten auf der Kurve $y^2 = x^3 - 3x + 7$

der in der unendliche Ferne. Zusätzlich erhalten wir aus dieser grafischen Vorstellung unmittelbar die Einsicht, warum die Gruppenverknüpfung + abelsch ist. Denn für die erzeugte Gerade spielt es keine Rolle, ob wir $A + B$ oder $B + A$ schreiben. Die Gültigkeit des Assoziativ-Gesetzes besitzt leider keine solch einfache grafische Veranschaulichung.

12.2 Definition

Als Nächstes sollen nun EK definiert und genauer untersucht werden. Diese mathematische Einleitung orientiert sich dabei in ihrer Struktur an dem Buch „Elliptische Kurven in der Kryptografie“ von Werner A. [Wer02]. Dabei wurden viele Definitionen und Beweisansätze übernommen, weiter ausgeführt und/oder vereinfacht. Wir empfehlen daher bei gesteigertem Interesse oder unbeantworteten Fragen dort nachzuschlagen. Die hier gegebene Einführung wurde so angepasst, dass sie all das mathematische Grundwissen zu EK vermittelt, das wir für spätere Ausführungen benötigen. Zuerst wird eine formale Definition von EK gegeben und anschließend die enthaltenen Begrifflichkeiten erklärt.

Elliptische Kurven: Eine EK ist eine nicht-singuläre projektive ebene Kurve eines homogenen Polynoms vom Grad drei über einem Körper $\mathbb{K}$, welches folgende Gestalt hat:

$$f(X, Y, Z) = Y^2Z + aXYZ + bYZ^2 - X^3 - cX^2Z - dXZ^2 - eZ^3$$

dabei sind $a, b, c, d, e \in \mathbb{K}$.

Diese Begrifflichkeiten, werden nun im Folgenden erklärt.

12.2.1 Affine Kurve

Um den Begriff rojektive ebene Kurve genauer zu erläutern, führen wir zuerst die affinen Kurven ein:

Definition: Sei h ein beliebiges von Null verschiedenes Polynom mit zwei Variablen und Koeffizienten in einem Körper $\mathbb{K}$.

$$h(x,y) = \sum_{i,j>0} a_{ij} x^i y^j \qquad a_{ij} \in \mathbb{K} \qquad \exists a_{ij} \neq 0$$

man nennt die Menge der Nullstellen N_h von h also die Menge der Paare $(x,y) \in \mathbb{K}^2$ mit $h(x,y) = 0$ affine ebene Kurve. Diese Definition deckt sich mit der Darstellung im Beispiel des intuitiven Ansatzes, genügt jedoch nicht um alle mathematischen Beweise und Untersuchungen ausführlich zu begründen. Daher wollen wir nun mittels einer Abbildung zu einer, wie in der Definition von EK bereits benannten, projektiven Kurve übergehen. Dazu müssen wir jedoch zuerst die Begriffe des projektiven Raums und der projektiven Kurve einführen.

12.2.2 Projektiver Raum

In einem projektiven Raum werden die später beschriebenen projektiven Kurven und somit auch EK definiert. Der projektive Raum über einem Körper $\mathbb{K}$ entspricht einem normalen $\mathbb{K}$-Vektorraum in dem jedoch alle Vielfachen von Vektoren zu einem identischem Element zusammengefasst werden. Die in $\mathbb{K}^3$ verschiedenen Punkte (X,Y,Z) und (X',Y',Z') sind im projektiven Raum also äquivalent, falls es ein $\lambda \in \mathbb{K}$ gibt, sodass gilt:

$$X = \lambda X' \qquad Y = \lambda Y' \qquad Z = \lambda Z'$$

Diese so definierte Äquivalenzrelation bezeichnen wir mit $\approx$ und definieren den projektiven Raum $P(\mathbb{K})$ über $\mathbb{K}$ wie folgt:

$$P(\mathbb{K}) = \left\{\mathbb{K}^3 \setminus (0,0,0)\right\} / \approx$$

Man kann sich diesen Raum auch als die Menge aller Ursprungsgeraden in $\mathbb{K}^3$ vorstellen, dabei entspricht jede Gerade genau einem Element.

12.2.3 Projektive Kurven

Eine projektive Kurve ist nun die Nullstellenmenge eines Polynoms f über dem Projektiven Raum $P(\mathbb{K})$. Wir bezeichnen diese Menge mit NP_f. Hierbei ist zu beachten, dass eine solche Nullstellenmenge nur sinnvoll definiert ist, wenn für das Polynom im projektiven Raum gilt:

$$f(X,Y,Z) = 0 \quad \Rightarrow \quad f(\lambda X, \lambda Y, \lambda Z) = 0 \qquad \forall \lambda \in \mathbb{K}$$

Eine Klasse von Polynomen, die diese Eigenschaft besitzt, sind die homogenen Polynome.

12.2.4 Homogenes Polynom

Sei nun f eine homogenes Polynom mit drei Variablen:

$$f(X,Y,Z) = \sum_{i,j,k>0} a_{ijk} X^i Y^j Z^k \qquad a_{ijk} \in \mathbb{K} \qquad \exists a_{ijk} \neq 0$$

Man nennt f nun homogen, wenn für alle 3-Tupel der Exponenten (i,j,k) für die $a_{ijk} \neq 0$ ist, ein festes $n \in \mathbb{N}$ existiert sodass gilt: $i+j+k=n$. Es gilt nun:

Beh.: Sei f homogenes Polynom, dann gilt:

$$f(X,Y,Z) = 0 \quad \Rightarrow \quad f(\lambda X, \lambda Y, \lambda Z) = 0$$

Bew.:

$$\begin{aligned} f(\lambda X, \lambda Y, \lambda Z) &= \sum_{i,j,k>0} a_{ijk} (\lambda X)^i (\lambda Y)^j (\lambda Z)^k \\ &= \sum_{i,j,k>0} a_{ijk} \lambda^{i+j+k} X^i Y^j X^k \\ &= \lambda^n \sum_{i,j,k>0} a_{ijk} X^i Y^j X^k \\ &= \lambda^n f(X,Y,Z) \end{aligned}$$

□

Daher definieren wir sinnvoll.

12.2.5 Projektive Kurven

Sei f ein homogenes Polynom vom Grad drei über $\mathbb{K}^3$. Dann bezeichnen wir die Nullstellenmenge:

$$NP_f = \{(X,Y,Z) \in P(\mathbb{K}) : f(X,Y,Z) = 0\}$$

als projektive ebene Kurve.

Projektion: Wir wollen nun unsere affine Kurve N_h in den projektiven Raum abbilden und somit zu einer projektiven Kurve machen. Diese Abbildung werden wir als Projektion bezeichnen. Für diese Projektion ist es nötig ein Polynom f über $\mathbb{K}^3$ zu finden, das unter der Abb. Φ gilt:

$$\begin{aligned} \Phi: \quad & N_h \to NP_f \\ & (x,y) \mapsto (X,Y,Z) \end{aligned}$$

$$h(x,y) = 0 \quad \Rightarrow \quad f(\Phi(x,y)) = f(X,Y,Z) = 0$$

In Worten: Alle Nullstellen der affinen Kurve werden auf Nullstellen der projektiven Kurve abgebildet. Anschließend soll diese neue Nullstellenmenge NP_f untersucht werden.

Es ist klar, dass wir für f ein homogenes Polynom wählen müssen. Um dieses zu erzeugen gehen wir ausgehend von h wie folgt vor:

- Aus der Summendarstellung von $h = \sum_{i,j>0} a_{ij} x^i y^j$ bestimmen wir

$$d := \max_{ij:a_{ij}\neq 0} [(i+j)]$$

- Wir definieren:

$$f(X,Y,Z) = \sum_{i,j>0} a_{ij} X^i Y^j Z^{d-(i+j)}$$

Dieses neue Polynom ist eindeutig homogen und daher ist seine Nullstellenmenge sinnvoll definierbar in $P(\mathbb{K})$. Wählen wir dann noch Φ auf folgenden Weise:

$$\begin{aligned} \Phi: \quad & N_h \to NP_f \\ & (x,y) \mapsto \Phi(x,y) = (x,y,1) \end{aligned}$$

haben wir unsere Projektion gefunden. Denn es gilt:

Beh.:

$$(x,y) \in N_h \quad \Rightarrow \quad \Phi(x,y) = (x,y,1) \in NP_f$$

Bew.:

$$\begin{aligned} f(x,y,1) &= \sum_{i,j>0} a_{ij} x^i y^j \underbrace{1^{d-(i+j)}}_{=1} \\ &= \sum_{i,j>0} a_{ij} x^i y^j \\ &= h(x,y) \\ &= 0 \end{aligned}$$

□

Damit haben wir gezeigt, dass es möglich ist mittels einer Projektion von einer affinen zu einer projektiven Kurve zu wechseln. Aber wozu haben wir nun die komplexer erscheinen projektiven Kurven eingeführt? Ein Grund ist, dass auf projektiven Kurven, welche die Form einer EK haben, ein für die später beschriebene Gruppenstruktur wichtiger Punkt existiert. Dieser kann nur dort mathematisch korrekt beschrieben werden. Es handelt sich dabei um den Punkt $(0,1,0)$, welcher nicht durch die Projektion aus der affinen Kurve entsteht.

Beh.: **i)** $(0,1,0) \notin \Phi(N_h)$ **ii)** $(0,1,0) \in NP_f$

Bew.:

i) angenommen es gibt $(x_1, y_1) \in N_h$ mit $\Phi(x_1, y_1) = (0, 1, 0)$
$\Rightarrow \Phi(x_1, y_1) = (x_1, y_1, 1) \approx (0, 1, 0)$
$\Rightarrow \exists \lambda \in \mathbb{K}$ mit $\lambda y_1 = 1$ und $\lambda 1 = 0$

↯

ii) Da Y nie alleine in einem Monom von einer EK auftaucht, folgt sofort: $f(0, 1, 0) = 0$

□

Dieser Punkt $(0, 1, 0)$ wird, die im Beispiel schon erwähnte Rolle des unendlich fernen Punktes einnehmen, also das neutrale Element bilden. Deswegen wird er im Folgenden mit 0 bezeichnet. Außerdem lässt sich leicht zeigen, dass der Punkt 0 für keine projektive ebene Kurve, die die Form einer EK besitzt, singulär ist.

12.2.6 Singularität

In der Definition von EK war außerdem gegeben, dass eine solche projektive Kurve nicht singulär sein darf. Was dies bedeutet soll die nächste Definition klar machen.

Definition singulär: Eine projektive Kurve, welche vom Polynom f erzeugt wird, heißt singulär in dem Punkt (x_1, y_1, z_1), falls gilt:

$$f(x_1, y_1, z_1) = 0 \qquad \frac{d}{dX} f(x_1, y_1, z_1) = 0 \qquad \frac{d}{dY} f(x_1, y_1, z_1) = 0 \qquad \frac{d}{dZ} f(x_1, y_1, z_1) = 0$$

Dabei darf der Punkt (x_1, y_1, z_1) nicht nur aus $\mathbb{K}^3$ selbst gewählt werden, sondern aus dem Vektorraum über dem algebraischen Abschluss von $\mathbb{K}$, also aus $\overline{\mathbb{K}}^3$.

Definition nicht singulär: Eine projektive Kurve welche vom Polynom f erzeugt wird, heißt nichtsingulär wenn sie in keinem Punkt singulär ist.

Hier soll noch gezeigt werden, dass es genügt die zugrunde liegende affine Kurve zu untersuchen, um eine EK auf Singularität zu prüfen. Es gilt nämlich:

Beh.:

$$\frac{d}{dx} h(x_1, y_1) = \frac{d}{dy} h(x_1, y_1) = 0 \quad \Leftrightarrow \quad \frac{d}{dX} f(x_1, y_1, 1) = \frac{d}{dY} f(x_1, y_1, 1) = \frac{d}{dZ} f(x_1, y_1, 1) = 0$$

Bew.:

$$\begin{aligned} \frac{d}{dX} f(x_1, y_1, 1) &= \sum_{i,j>0} a_{ij} i x_1^{i-1} y_1^j \underbrace{1^{d-(i+j)}}_{=1} \\ &= \sum_{i,j>0} a_{ij} i x_1^{i-1} y_1^j \end{aligned}$$

$$= \frac{d}{dx} h\left(x_1, y_1\right)$$
$$= 0$$

Äquivalent dazu zeigt man $\frac{d}{dY} f\left(x_1, y_1, 1\right) = \frac{d}{dy} h\left(x_1, y_1\right) = 0$. Es bleibt zu zeigen, dass

$$\frac{d}{dx} h\left(x_1, y_1\right) = \frac{d}{dy} h\left(x_1, y_1\right) = 0 \quad \Leftrightarrow \quad \frac{d}{dZ} f\left(x_1, y_1, 1\right) = 0$$

$$\begin{aligned}
\frac{d}{dZ} f\left(x_1, y_1, 1\right) &= \sum_{i,j>0} a_{ij} x_1^i y_1^j \left(d - i - j\right) \underbrace{1^{d-i-j)}}_{=1} \\
&= d \sum_{i,j>0} a_{ij} x_1^i y_1^j - \sum_{i,j>0} a_{ij} i x_1^i y_1^j - \sum_{i,j>0} a_{ij} j x_1^i y_1^j \\
&= d \sum_{i,j>0} a_{ij} x_1^i y_1^j - x_1 \cdot \sum_{i,j>0} a_{ij} i x_1^{i-1} y_1^j - y_1 \cdot \sum_{i,j>0} a_{ij} j x_1^i y_1^{j-1} \\
&= \underbrace{df\left(x_1, y_1\right)}_{d \cdot 0 = 0} - x_1 \underbrace{\frac{d}{dx} h\left(x_1, y_1\right)}_{=0} - y_1 \underbrace{\frac{d}{dy} h\left(x_1, y_1\right)}_{=0} \\
&= 0
\end{aligned}$$

□

12.2.7 Bijektion

Die hier zuletzt eingeführte Begrifflichkeit der Singularität ist von essenzieller Bedeutung für die Gruppenstruktur. Doch bevor wir dazu kommen, wie diese Bedingung an projektiven ebenen Kurven leicht überprüft werden kann, wollen wir zuerst vorstellen, wie wir die Gleichung einer EK vereinfachen können. Dazu stellen wir eine Bijektion zwischen verschiedenen EK vor.

Satz: Sei NP_{f_1} eine EK über einem Körper $\mathbb{K}$ und somit von der Form:

$$f_1(X, Y, Z) = Y^2 Z + a_1 XYZ + b_1 YZ^2 - X^3 - c_1 X^2 Z - d_1 XZ^2 - e_1 Z^3$$

weiterhin sei $\mathrm{char}\left(\mathbb{K}\right) > 3$. Sodass die Zahlen 36, 3, 12, 216 und 108 nur die Primfaktoren 2 und 3 besitzen. Damit ist die Abbildung

$$\begin{aligned}
\Omega: \qquad & P\left(\mathbb{K}\right) \to P\left(\mathbb{K}\right) \\
& \left(x_1, y_1, z_1\right) \mapsto \left(36x_1 + 3a_1^2 + 12c_1 z_1, 216y_1 + 108a_1 x_1 + 108b_1 z_1 . z_1\right)
\end{aligned}$$

bijektiv und es gilt:

$$\Omega\left(NP_{f_1}\right) = NP_{f_2}$$

sodass NP_{f_2} ebenfalls eine EK ist und folgende Form besitzt:

$$f_2(X,Y,Z) = Y^2Z - X^3 - d_2XZ^2 - e_2Z^3$$

Wobei die Koeffizienten definiert sind als:

$$\begin{aligned} d_2 &= 27a_1^4 + 216a_1c_1 + 432c_1^2 - 1296d_1 - 648a_1b_1 \\ e_2 &= -54a_1^6 - 648a_1^4c_1 - 2592a_1^2c_1^2 + 3888a_1^2d_1 - 11664b_1^2 \\ &\quad - 3456c_1^3 + 15552c_1d_1 - 46656e_1 + 1944a_1e_1 \end{aligned}$$

Bew.: Zunächst zeigen wir leicht, dass die Abbildung Ω tatsächlich bijektiv ist, denn:

$$\begin{aligned} \Omega^{-1}: \quad & P(\mathbb{K}) \to P(\mathbb{K}) \\ & (x_1,y_1,z_1) \mapsto \left(\frac{1}{36}x_1 - \frac{3a_1^2+12c_1}{36}z_1, \frac{1}{216}\left(y_1 - \frac{a_1}{2}\left(\frac{1}{36}x_1 - \frac{3a_1^2+12c_1}{36}z_1\right) - \frac{b_1}{2}z_1\right), z_1\right) \end{aligned}$$

ist offenbar inverse zu Ω. Weiter lässt sich leicht nachrechnen:

$$\begin{aligned} & f_2(X,Y,Z) = 216f\left(\Omega^{-1}(X,Y,Z)\right) \\ \Rightarrow \quad & f_2(X,Y,Z) = 0 \quad \Leftrightarrow \quad 216f_1\left(\Omega^{-1}(X,Y,Z)\right) = 0 \\ \Rightarrow \quad & \Omega(NP_{f_1}) = NP_{f_2} \end{aligned}$$

Es ist schon deutlich geworden, das f_2 von seiner die Richtige Form besitzt um eine EK zu bilden. Nun müssen wir nur noch zeigen, das NP_{f_2} für keinen enthaltenen Punkt singulär ist. Mit der Kettenregel bestimmen wir zunächst die lokalen Ableitungen von f_2 für einen Punkt (x_1,y_1,z_1):

$$\begin{aligned} \frac{d}{dX}f_2(x_1,y_1,z_1) &= 216\left[\frac{1}{36}\frac{d}{dX}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right) - \frac{a_1}{15552}\frac{d}{dY}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right)\right] \\ \frac{d}{dY}f_2(x_1,y_1,z_1) &= \frac{d}{dY}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right) \\ \frac{d}{dZ}f_2(x_1,y_1,z_1) &= 216\left[\frac{3a_1^2+12c_1}{36}\frac{d}{dX}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right)\right. \\ &\qquad + \frac{3a_1^2-12c_1-36b_1}{15552}\frac{d}{dY}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right) \\ &\qquad \left. + \frac{d}{dZ}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right) + \frac{d}{dZ}f_1\left(\Omega^{-1}(x_1,y_1,z_1)\right)\right] \end{aligned}$$

Fordert man nun:

$$\frac{d}{dX}f_2(x_1,y_1,z_1) = \frac{d}{dY}f_2(x_1,y_1,z_1) = \frac{d}{dZ}f_2(x_1,y_1,z_1) = 0$$

So folgt aus den lokalen Ableitungen von f_2 sofort:

$$\frac{d}{dX} f_1\left(\Omega^{-1}(x_1, y_1, z_1)\right) = \frac{d}{dY} f_1\left(\Omega^{-1}(x_1, y_1, z_1)\right) = \frac{d}{dZ} f_1\left(\Omega^{-1}(x_1, y_1, z_1)\right) = 0$$

da allerdings NP_{f_1} nicht singulär ist und für alle $(x_1, y_1, z_1) \in NP_{f_2}$ gilt $\Omega^{-1}(x_1, y_1, z_1) \in NP_{f_1}$ ist dies ein Widerspruch.

□

Im Nachfolgenden werden wir uns auf EK von der Form von NP_{f_2} beschränken. Da auch im Bitcoin-Netzwerk eine solche EK verwendet wird und wir gezeigt haben, dass wir im Fallechar $(\mathbb{K}) > 3$ bijektiv zu einer solchen übergehen können, soll uns die Betrachtung dieser Art von EK genügen.

12.2.8 Diskriminante

Wie schon angekündigt, wollen wir eine einfache Variante präsentieren, wie man eine potenzielle EK auf Singularität prüfen kann. Wir beschränken uns nun auf EK, deren zugrundelegendes Polynom die Form $f(X, Y, Z) = Y^2Z - X^3 - dXZ^2 - eZ^3$ besitzt.

Wie wir in 12.2.6 schon gezeigt haben, reicht es hier die affine Kurve zu untersuchen. Möchten wir nun also nachweisen, dass NP_f nicht singulär ist, müssen wir zeigen:

$$\begin{aligned}
&\nexists (x_1, y_1) \in \overline{\mathbb{K}}^2 \text{ mit:} \\
&\qquad h(x_1, y_1) = y_1^2 - x_1^3 - cx_1 - e = 0 \\
&\frac{d}{dx} h(x_1, y_1) = -3x_1^2 - c = 0 \\
&\frac{d}{dy} h(x_1, y_1) = 2y = 0 \quad \Rightarrow y = 0
\end{aligned}$$

Dies ist äquivalent zu:

$$\begin{aligned}
&\nexists x_1 \in \overline{\mathbb{K}} \text{ mit:} \\
&\qquad \mu(x_1) = x_1^3 + cx_1 - e = 0 \\
&\frac{d}{dx} \mu(x_1) = 3x_1^2 + c = 0
\end{aligned}$$

Da wir uns in $\overline{\mathbb{K}}$ befinden, zerfällt μ in seine lineare Faktoren:

$$\mu(x_1) = (x_1 - \lambda_1)(x_1 - \lambda_2)(x_1 - \lambda_3)$$

und hat somit die möglichen 3 Nullstellen λ_1, λ_2 und λ_3. Leitet man μ aus dieser Form ab, erhält man folgende Darstellung für $\frac{d}{dx}\mu$:

$$\frac{d}{dx} \mu(x_1) = (x_1 - \lambda_1)(x_1 - \lambda_2) + (x_1 - \lambda_1)(x_1 - \lambda_3) + (x_1 - \lambda_2)(x_1 - \lambda_3)$$

man erkennt hier leicht, dass wenn alle λ_i verschieden sind, $\frac{d}{dx}\mu$ nie für ein $x_1 = \lambda_i$ Null werden kann.

Ist jedoch ohne Beschränkung der Allgemeinheit z. B. $\lambda_1 = \lambda_2$, so setzen wir $x_1 = \lambda_1$ und erhalten so:

$$\frac{d}{dx}\mu(x_1) = \underbrace{(\lambda_1 - \lambda_1)(\lambda_1 - \lambda_1)}_{=0} + \underbrace{(\lambda_1 - \lambda_1)}_{=0}(\lambda_1 - \lambda_3) + \underbrace{(\lambda_1 - \lambda_1)}_{=0}(\lambda_1 - \lambda_3) = 0$$

Wir sehen, dass wir x_1 wie gewünscht wählen können, genau dann wenn μ eine doppelte Nullstelle besitzt, also zwei λ_i identisch sind. Um nun das Polynom μ schnell darauf zu untersuchen, ob es eine solche Nullstelle hat, können wir die Diskriminante verwenden. Sie ist definiert als:

$$D(\mu) = (\lambda_1 - \lambda_2)^2 (\lambda_1 - \lambda_3)^2 (\lambda_2 - \lambda_3)^2$$

Wie sich zeigen lässt, kann man die Diskriminante direkt aus den Koeffizienten des Polynoms berechnen [Dis13]. Damit folgt:

$$D(\mu) = 4c^3 + 27e^2$$

Um nun festzustellen, ob eine projektive ebene Kurve NP_f von der Form einer EK nicht singulär ist, genügt es also zu überprüfen, ob die Ungleichung $4c^3 + 27e^2 \neq 0$ erfüllt ist. Daher nennt man $D = 4c^3 + 27e^2$ auch Diskriminante von NP_f.

12.3 Nachweis der Gruppenstruktur

Dieses Kapitel basiert ebenfalls auf der Literatur [Wer02]. Wie erwähnt, lässt sich auf einer EK über einem endlichem Körper $\mathbb{K}$ eine Gruppenstruktur definieren. Zu ihrer Beschreibung benötigen wir 2 Hilfsmittel.

12.3.1 Projektive Gerade

Im naiven Ansatz wurde bereits geschildert, dass man die Gruppenverknüpfung mittels einer Geraden, die mehrere Punkte einer EK verbindet, beschreiben kann. Da wir uns aber im projektiven Raum befinden, müssen wir erst definieren, wie dort eine Gerade aussieht. Eine solche Gerade ist ebenfalls die Nullstellenmenge eines Polynoms und ist folgendermaßen definiert:

Definition:

Sei $g(X, Y, Z)$ von der Form:

$$g(X, Y, Z) = rX + sY + tZ \qquad \text{mit } (0, 0, 0) \neq (r, s, t) \in \mathbb{K}^3$$

so nennen wir die Nullstellen Menge $NP_g = \{(x_1, y_1, z_1) \in P(\mathbb{K}) : g(X, Y, Z) = 0\}$ von g projektive Gerade.

Es sei erwähnt, dass diese Nullstellenmenge wiederum sinnvoll definiert werden kann, da g die Form eines homogenen Polynoms vom Grad 1 besitzt. Da in jedem Monom genau eine Variable vorkommt,

gilt bei den EK außerdem:

$$g_1(X,Y,Z) = g_2(X,Y,Z) \quad \Leftrightarrow \quad \exists \lambda \in \mathbb{K} \text{ sodass } (r_1,s_1,t_1) = \lambda(r_2,s_2,t_2)$$

Wir zeigen noch, dass zwei Punkte des projektiven Raums und somit auch zwei Punkte einer EK genau eine projektive Gerade definieren.

Beh.:

Seien $P_1, P_2 \in P(\mathbb{K})$. Dann gilt:

$$P_1 \neq P_2 \quad \Rightarrow \quad \exists! NP_g \text{ sodass } P_1, P_2 \in NP_g$$

Bew.:

$$\begin{aligned} & P_1, P_2 \in \{(x_1,y_1,z_1) \in P(\mathbb{K}) : g(X,Y,Z) = 0\} \\ \Leftrightarrow \quad & rx_1 + sy_1 + tz_1 = 0 = rx_2 + sy_2 + tz_2 \\ \Rightarrow \quad & A = \begin{pmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \end{pmatrix} \end{aligned}$$

Da die Matrix A den Rang zwei hat, folgt, dass die Lösungsvektoren (r,s,t) existieren und alle Lösungsvektoren Vielfache voneinander sind. Da diese Vielfachen, wie oben bereits erwähnt, immer dieselbe projektive Gerade erzeugen, ist der Beweis erbracht.

□

Eine weitere wichtige projektive Gerade ist die Tangente eines Punkts auf einer EK. Diese wird wie die Tangente im naiven Ansatz dazu benötigt, einen Punkt mit sich selbst zu verknüpfen. Wir definieren sie folgendermaßen:

Projektive Tangente: Sei NP_f eine EK und $P_1 = (x_1,y_1,z_1) \in NP_f$, nennen wir die projektive Gerade NP_g mit den Koeffizienten

$$r = \frac{d}{dX} f(x_1,y_1,z_1) \qquad s = \frac{d}{dY} f(x_1,y_1,z_1) \qquad t = \frac{d}{dZ} f(x_1,y_1,z_1)$$

projektive Tangente in P_1.

Der Punkt P_1liegt, wie leicht nachvollziehbar ist, auf NP_g:

Beh.:

$$P_1 = (x_1,y_1,z_1) \in NP_g$$

Bew.:

$$
\begin{aligned}
f(X,Y,Z) &= Y^2Z - X^3 - dXZ^2 - eZ^3 \\
r = \frac{d}{dX} f(x_1,y_1,z_1) &= -3x_1^2 - 2dz_1^2 \\
s = \frac{d}{dY} f(x_1,y_1,z_1) &= 2y_1z_1 \\
t = \frac{d}{dZ} f(x_1,y_1,z_1) &= y_1^2 - cdx - 3ez_1^2
\end{aligned}
$$

setzt man nun diese Koeffizienten sowie die Koordinaten des Punktes P_1 in g ein, erhält man:

$$
\begin{aligned}
g(x_1,y_1,z_1) &= \left(-3x_1^2 - 2dz_1^2\right) x_1 + (2y_1z_1)\, y_1 + \left(y_1^2 - dxz_1 - 3ez_1^2\right) z_1 \\
&= -3x_1^3 - 2dx_1z_1^2 + 2y_1^2z_1 + z_1y_1^2 - dx_1z_1^2 - 3ez_1^3 \\
&= 3\underbrace{\left(y_1^2z_1 - x_1^3 - dx_1z_1^2 - ez_1^3\right)}_{f(x_1,y_1,z_1)=0} \\
&= 0
\end{aligned}
$$

□

12.3.2 Schnittverhalten

Als zweites Hilfsmittel wird die Schnitthäufigkeit eingeführt. Diese wird Schnittpunkte einer projektiven Gerade und EK mittels einer natürlichen Zahl bewerten. Sie gibt an, wie häufig dieser Schnittpunkt gezählt wird. Diese Häufigkeit ist vergleichbar mit der Häufigkeit einer Nullstelle eines Polynoms.

Definition Schnitthäufigkeit:

Die Schnitthäufigkeit ist definiert für ein 3-Tupel (P_1, NP_f, NP_g) bestehend aus einem Punkt $P_1 = (x_1, y_1, z_1)$, einer EK NP_f, und einer projektiven Gerade. Sei nun $P_2 = (x_2, y_2, z_2)$ ein weiterer, von P_1 verschiedener Punkt auf NP_g. Wir erzeugen zunächst ein Polynom:

$$
\begin{aligned}
s(t) &= f(x_1 + \delta x_2, y_1 + \delta y_2, z_1 + \delta z_2) \\
&= \sum_{i=0}^{3} a_i \delta^i
\end{aligned}
$$

Dann ist die Schnitthäufigkeit S definiert als:

$$
S(P_1, NP_f, NP_g) = \begin{cases} S = 0 & P_1 \notin NP_f \\ S = 0 & P_1 \notin NP_g \\ S = i : min_i\{a_i \neq 0\} & \text{sonst} \end{cases}
$$

Es sei auf folgende Umstände aufmerksam gemacht:

Bemerkungen:

1.

$$P \in NP_f \cap NP_g \Rightarrow S\left(P, NP_f, NP_g\right) \geq 1$$

da gilt:

$$s(0) = f\left(x_1, y_1, z_1\right) = 0 \Rightarrow a_0 = 0$$

2.

$P \in NP_f \cap NP_g$ und NP_g ist die Tangente in $P \Rightarrow S\left(P, NP_f, NP_g\right) \geq 2$

da gilt:

$$s'(0) = \frac{d}{dX} f\left(x_1, y_1, z_1\right) x_2 + \frac{d}{dY} f\left(x_1, y_1, z_1\right) y_2 + \frac{d}{dZ} f\left(x_1, y_1, z_1\right) z_2 = 0 \Rightarrow a_1 = 0$$

Nun können wir über die Schnitthäufigkeit ggf. die Existenz des aus dem naiven Ansatz postulierten zusätzlichen Schnittpunktes zeigen. Dazu beweisen wir zunächst folgenden Satz:

Satz:

$$\sum_{P \in \mathbb{P}(\mathbb{K})} S\left(P, NP_f, NP_g\right) \in \{0, 1, 3\}$$

Bew.: Für den Beweis unterscheiden wir verschiedene Formen der Gerade NP_g:

1. Fall: $g\left(X, Y, Z\right) = 0 \cdot X + 0 \cdot Y + tZ$ und $t \neq 0$

$$\begin{aligned} &\Rightarrow Z = 0 \\ &\Rightarrow f\left(Y, X, 0\right) = -X^3 = 0 \\ &\Rightarrow X = 0 \end{aligned}$$

Hier ist nun sofort ersichtlich, dass der einzige Punkt auf $NP_g \cap NP_f$ der Punkt $P_1 = 0 = (0, 1, 0)$ ist. Als Hilfspunkt wählen wir uns $P_2 = (1, 0, 0)$, damit folgt:

$$s\left(\delta\right) = f(0 + \delta, 1, 0) = -\delta^3 \quad \Rightarrow \quad S\left(P, NP_f, NP_g\right) = \min_i \{a_i \neq 0\} = 3$$

2. Fall: $g\left(X, Y, Z\right) = rX + 0 \cdot Y + tZ$ und $r \neq 0$

$$\begin{aligned} &\Rightarrow \quad X = -\frac{t}{r} Z \\ &\Rightarrow \quad \text{wenn } P \in NP_g, \text{ dann } P = (0, 1, 0) \text{ oder } P = \left(-\frac{t}{r}, y, 1\right) \text{ für gewisse } y_i \in \mathbb{K} \end{aligned}$$

$(0,1,0)$ liegt nun immer auf der EK und es gilt mit dem Hilfspunkt $(-t,0,r)$:

$$\begin{aligned} s(\delta) &= f(-\delta t, 1, \delta r) \\ &= \delta r + \delta^3 t^3 + d\delta^3 t r^2 - e\delta^3 r^3 \\ &= \delta^3 \underbrace{\left(t^3 + dtr^2 - er^3\right)}_{=a_3} + \delta \underbrace{r}_{=a_1} \\ \Rightarrow & S((0,1,0), NP_f, NP_g) = 1 \end{aligned}$$

Liegt nun keiner der Punkte$\left(-\frac{t}{r}, y_i, 1\right)$ auf der Kurve, ist der Beweis erbracht. Falls doch, muss noch gezeigt werden, dass $\sum S\left(\left(-\frac{t}{r}, y_i, 1\right), NP_f, NP_g\right) = 2$. Es gilt:

$$\begin{aligned} f\left(-\frac{t}{r}, y, 1\right) &= y^2 + \frac{t^3}{r^3} - d\frac{t}{r} - e \\ y^2 &= -\frac{t^3}{r^3} + d\frac{t}{r} + e \end{aligned}$$

Aus dieser quadratischen Gleichung kann man erkennen, dass es zwei Lösungen für y gibt, sodass $\left(-\frac{t}{r}, y, 1\right)$ auf NP_f liegt. Für beide Punkte bestimmen wir die Schnitthäufigkeit. Dazu wählen wir uns den Hilfspunkt $P_2 = (0,1,0)$ und bestimmen $s(\delta)$:

$$\begin{aligned} s(\delta) &= f\left(-\frac{t}{r}, y+\delta, 1\right) \\ &= (y+\delta)^2 + \frac{t^3}{r^3} - d\frac{t}{r} - e \\ &= \delta^2 + \delta 2y + y^2 + \frac{t^3}{r^3} - d\frac{t}{r} - e \\ &= \delta^2 + \delta \underbrace{2y}_{=a_1} \\ \Rightarrow & \sum S\left(\left(-\frac{t}{r}, y_i, 1\right), NP_f, NP_g\right) = 2 \quad \forall y_i \neq 0 \end{aligned}$$

Falls y_i Null ist, gibt es keine zwei Lösung für y. Allerdings ist die Schnitthäufigkeit von $\left(-\frac{t}{r}, 0, 1\right)$, wie aus obiger Gleichung erkennbar ist 2. Damit ist dieser Fall abgedeckt.

3. Fall: $g(X,Y,Z) = rX + sY + tZ$ und $s \neq 0$
Der Beweis für diesen Fall findet sich in [Wer02, 38f].

□

12.3.3 Gruppenstruktur

Nun können wir die Gruppenstruktur beschreiben. Da wir wissen $\sum_{P \in \mathbb{P}(\mathbb{K})} S(P, NP_f, NP_g) \in \{0,1,3\}$ und $P \in NP_f \cap NP_g \Rightarrow S(P, NP_f, NP_g) \geq 1$ können wir definieren:

$P_1 \neq 0 \in NP_f \qquad \exists! NP_g : P_1, 0 \in NP_g$ ist.

$$-P_1 = P_1 \text{ falls } S(P_1, NP_f, NP_g) = 2 \quad \vee \quad -P_1 = P_2 : S(P_2, NP_f, NP_g) = 1 \text{ mit } 0 \neq P_2 \neq P_1$$

$$P_1 \neq P_2 \qquad P_1, P_2 \in NP_f \Rightarrow \exists! NP_g : P_1, P_2 \in NP_g$$

$$P_1 + P_2 = \begin{cases} P_1 & \text{falls } S(P_1, NP_f, NP_g) = 2 \\ P_2 & \text{falls } S(P_2, NP_f, NP_g) = 2 \qquad \text{wobei } S(P_3, NP_f, NP_g) = 1 \\ -P_3 & \text{sonst} \end{cases}$$

$$P_1 \in NP_f \qquad \exists! NP_g \text{ sodass } NP_g \text{ die Tangente in } P_1 \text{ ist.}$$

$$2P_1 = \begin{cases} P_1 & \text{falls } S(P_1, NP_f, NP_g) = 3 \qquad \text{wobei } S(P_2, NP_f, NP_g) = 1 \\ -P_2 & \text{sonst} \end{cases}$$

Es ist leicht nachvollziehbar, dass diese Definition mit den Bemerkungen und dem Satz aus dem vorigen Kapitel immer eindeutig und sinnvoll ist. Auch die Abgeschlossenheit und „Abelschheit“ der Gruppenverknüpfung wird mit ihnen direkt aus dieser Definition deutlich. In der Praxis ist diese Definition allerdings wenig hilfreich, da dort mit diskreten Punkten gerechnet werden muss. Dabei hilft uns die Sicherheit, dass der jeweilig gesuchte Punkt existiert, wenig. Stattdessen braucht man möglichst schnelle Algorithmen, um für gegebene Punkte P_1, P_2 auf einer EK Aufgaben wie$-P_1$,$2P_1$ oder $P_1 + P_2$eindeutig zu berechnen. Wir werden nun solche Algorithmen vorstellen. Dabei werden wir mit projektiven Kurven arbeiten, und dabei ausnutzen, dass diese mit beliebigen Skalaren multipliziert werden dürfen und somit z. B. die letzte Koordinate mit 1 gleichsetzen können. Außerdem sei erwähnt, dass vereinfacht angenommen werden kann, dass das Polynom unser EK die affine Form $h(x, y) = y^2 - x^3 - dx - e$ hat.

12.3.4 Neutrales Element

Betrachtet man den einfachen Fall, was bei der Addition mit 0 geschieht, erhält man den Beweis, dass ein neutrales Element existiert:

Beh.:

$$P_1 + 0 = P_1 \qquad \forall P_1 \in N_h$$

Bew.: In diesem Beweis müssen wir wieder zu unser projektiven Darstellung wechseln, da 0 nicht auf der Affinen Gerade liegt. Zuerst stellen wir die Geradengleichung auf, sodass $0, P_1 \in NP_g$.

$$\begin{aligned} & A = \begin{pmatrix} 0 & 1 & 0 \\ x_1 & y_1 & z_1 \end{pmatrix} \\ \Rightarrow \quad & s = 0 \\ \Rightarrow \quad & rx_1 = -tz_1 \\ & \text{oBdA: } z_1 = 1 \end{aligned}$$

$$\Rightarrow \quad -x_1 r = t$$

dann suchen wir nach weiteren Punkten P_2, welche auf der Gerade liegen können. Für diese muss gelten:

$$\begin{aligned}
&\Rightarrow g\left(x_2, y_2, z_2\right) = r x_2 - r x_1 z_2 = 0 \\
&\quad \text{oBdA: } z_2 = 1 \\
&\Rightarrow x_1 = x_2
\end{aligned}$$

Da wir nur Punkte von Interesse sind, die auf der Gerade und der EK liegen, folgt:

$$\begin{aligned}
f(x_1, y_2, 1) &= y_2^2 - x_1^3 - d x_1 - e = 0 \\
\Rightarrow y_2^2 &= x_1^3 + d x_1 + e = y_1^2
\end{aligned}$$

Ist nun $y_1 = 0$, ist der Beweis erbracht, denn daraus folgt $y_2 = 0 = y_1 \Rightarrow P_1 = P_2$. Ferner gilt für $S\left(0, NP_f, NP_g\right)$ mit dem Hilfspunkt $\left(-\frac{t}{r}, 0, 1\right)$:

$$\begin{aligned}
s\left(\delta\right) &= f(-\frac{t}{r}\delta, 1, \delta) \\
&= r^2\delta^2 - \frac{t^3}{r^3}\delta^3 - \underbrace{\frac{t}{r}}_{\neq 0}\delta + e\delta^3 \\
&\Rightarrow S\left(0, NP_f, NP_g\right) = 1 \\
&\Rightarrow S\left(P_1, NP_f, NP_g\right) = 2 \\
&\Rightarrow P_1 + 0 = P_1
\end{aligned}$$

Ist hingegen $y_1 \neq 0$, so gibt es für y_2 abgesehen von y_1 noch eine zweite Lösung und damit automatisch einen Punkt $P_2 \neq P_1$, der als Ergebnis von $P_1 + 0$ in Frage kommt. Für diesen folgt unmittelbar aus der Definition der Gruppenstruktur $P_2 = -P_1$ und damit: $P_1 + 0 = -(-P_2) = P_1$.

□

12.3.5 Inverses Element

Hier soll gezeigt werden, dass für jedes Element ein inverses Element existiert und sich mit geringem Aufwand bestimmen lässt:

Beh.:

$$P_1 = (x, y), P_2 = (x, -y) \quad \Rightarrow P_2 = -P_1$$

Bew.:

1. Fall: $y \neq 0$

Zunächst bestimmen wir die Gerade, die durch P_1 und P_2 verläuft.

$$\begin{aligned} A &= \begin{pmatrix} x_1 & y_1 & 1 \\ x_1 & -y_1 & 1 \end{pmatrix} \\ A &= \begin{pmatrix} x_1 & y_1 & 1 \\ 0 & -2y_1 & 0 \end{pmatrix} \\ \Rightarrow \quad & s = 0 \\ \Rightarrow \quad & 0 \in NP_g \cap NP_f \end{aligned}$$

Da nun für die Schnitthäufigkeiten von P_1 und P_2 gilt:

$$\begin{aligned} s(\delta) &= f(x, y-\delta, 1) \\ &= \underbrace{-2y}_{\neq 0}\delta + \delta^2 \\ \Rightarrow & S(P_1, NP_f, NP_g) = S(P_2, NP_f, NP_g) = 1 \end{aligned}$$

Daher folgt unmittelbar:

$$-P_1 = P_2 = (x, -y)$$

2. Fall: $y = 0 \quad \Rightarrow P_1 = P_2$

Zunächst bestimmen wir die Tangente, die durch P_1 verläuft.

$$r = \frac{d}{dX} f(x_1, 0, 1) = -3x_1 - d \qquad s = \frac{d}{dY} f(x_1, 0, 1) = 0 \qquad t = \frac{d}{dZ} f(x_1, 0, 1) = 2dx_1 - 3e$$

$$\begin{aligned} \Rightarrow \quad & s = 0 \\ \Rightarrow \quad & 0 \in NP_g \cap NP_f \end{aligned}$$

Da für die Schnitthäufigkeiten von P_1 gilt:

$$\begin{aligned} s(\delta) &= f(x, y-\delta, 1) \\ &= \underbrace{-2y}_{=0}\delta + \delta^2 \\ \Rightarrow \quad & S(P_1, NP_f, NP_g) = 2 \end{aligned}$$

Daraus folgt unsere Behauptung.

□

12.3.6 Addition zweier Elemente:

Etwas aufwendiger ist es, zwei allgemeine Elemente P_1, P_2 miteinander zu verknüpfen. Hier unterscheiden wir die beiden Fälle $P_1 \neq P_2$ und $P_1 = P_2$.

1. Fall: $P_1 \neq P_2$

Beh.:

$$P_1 + P_2 = -P_3 = (x_3, -y_3) \quad \text{wobei: } x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2 \text{ und } -y_3 = \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1$$

Bew.: Zunächst bestimmen wir die Gerade durch P_1 und P_2:

$$\begin{aligned}
& A = \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{pmatrix} \\
\Leftrightarrow \quad & \begin{pmatrix} x_1 & y_1 & 1 \\ x_2 - x_1 & y_2 - y_2 & 0 \end{pmatrix} \\
\Rightarrow \quad & r(y_2 - y_1) + s(x_2 - x_1) = 0 \\
\Rightarrow \quad & r = -s\left(\frac{y_2 - y_1}{x_2 - x_1}\right) \\
& \text{wir dürfen definieren: } s = -1 \\
\Rightarrow \quad & r = \frac{y_2 - y_1}{x_2 - x_1} \\
\Rightarrow \quad & \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x_1 - y_1 + t = 0 \\
\Rightarrow \quad & t = y_1 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x_1 \\
& = \frac{x_2 y_1 - x_1 y_2}{x_2 - x_1}
\end{aligned}$$

Alle weiteren Punkte in $NP_g \cap NP_f$ erfüllen nun folgende Gleichungen:

$$\begin{aligned}
& y = rx + t \\
& f(x, y, 1) = y^2 - x^3 - dx - e = 0 \\
\Rightarrow \quad & (rx + t)^2 - x^3 - dx - e = 0 \\
\Rightarrow \quad & -x^3 + r^2 x^2 + (2rt - d)x + t^2 - e = 0
\end{aligned}$$

In $\overline{\mathbb{K}}$ zerfällt dieses Polynom vom Grad drei in seine Linearfaktoren:

$$\begin{aligned}
\Rightarrow \quad & -x^3 + r^2 x^2 + (2rt - d)x + t^2 - e = k(x - x_1)(x - x_2)(x - x_3) \\
\Rightarrow \quad & -x^3 + r^2 x^2 + (2rt - d)x + t^2 - e = kx^3 - k(x_1 + x_2 + x_3)x^2 - \ldots \\
\Rightarrow \quad & k = -1 \qquad r^2 = x_1 + x_2 + x_3 \\
\Rightarrow \quad & x_3 = r^2 - x_1 + x_2 \\
\Rightarrow \quad & x_3 = \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 + x_2
\end{aligned}$$

Dann gilt mit den Gleichungen für y und t sofort:

$$\begin{aligned} y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x_3 + y_1 - \left(\frac{y_2 - y_1}{x_2 - x_1}\right) x_1 \\ \Rightarrow \quad y_3 &= \left(\frac{y_2 - y_1}{x_2 - x_1}\right) (x_3 - x_1) + y_1 \end{aligned}$$

Der Punkt P_3 liegt nun ebenfalls auf der Gerade und es muss nur noch $-P_3$ berechnet werden:

$$\begin{aligned} P_1 + P_2 = -P_3 &= (x_3, -y_3) \\ \Rightarrow \quad -y_3 &= \frac{y_2 - y_1}{x_2 - x_1} (x_1 - x_3) - y \end{aligned}$$

Dieser Beweis genügt ebenfalls den Möglichkeiten $P_1 + P_2 = -P_1$ bzw. $P_1 + P_2 = -P_2$, da dann aus $x_3 = x_1$ bzw. x_2 mittels der Darstellung als Linearfaktoren, folgt, dass P_1 bzw. P_2 die Schnitthäufigkeit zwei besitzt.

2. Fall:

$$P_1 = P_2 \quad \Rightarrow \quad P_1 + P_2 = 2P_1 = -P_3 = (x_3, -y_3)$$

Beh.:

$$2P_1 = P_3 \quad \text{wobei: } x_3 = \left(\frac{3x_1^2 - d}{2y_1}\right)^2 - 2x_1 \text{ und } -y_3 = \frac{3x_1^2 - d}{2y_1} (x_1 - x_3) - y_1$$

Bew.: Wir bestimmen die Tangente die durch P_1 verläuft:

$$r = \frac{d}{dX} f(x_1, y_1, 1) = -3x_1^2 - d \quad s = \frac{d}{dY} f(x_1, y_1, 1) = 2y_1 \quad t = \frac{d}{dZ} f(x_1, y_1, 1) = y_1^2 - 2dx_1 - 3e$$

$$-3x_1^2 x + 2y_1 y + y_1^2 - 2dx_1 - 3e = 0$$

$$\Rightarrow \quad y = \underbrace{\frac{3x_1^2 + d}{2y_1}}_{r'} x + \underbrace{\frac{y_1^2 - 2dx_1 - 3e}{2y_1}}_{t'} = r'x + t' \Rightarrow \text{ insbesondere gilt hier auch: } t' = y_1 - r'x_1$$

Mit Hilfe der EK-Gleichung suchen wir jetzt den zweiten Punkt auf der Geraden:

$$\begin{aligned} & f(x, y, 1) = y^2 - x^3 - dx - e = 0 \\ \Rightarrow & (r'x + t')^2 - x^3 - dx - e = 0 \\ \Rightarrow & -x^3 + r'^2 x^2 + (2r't' - d) x + t^2 - e = 0 \\ \Rightarrow & -x^3 + r'^2 x^2 + (2r't' - d) x + t^2 - e = k' (x - x_1)^2 (x - x_3) \\ \Rightarrow & -x^3 + r'^2 x^2 + (2r't' - d) x + t^2 - e = k'x^3 - (2x_1 + x_3) x^2 + \ldots \\ \Rightarrow & k' = -1 \\ & r'^2 = 2x_1 + x_3 \\ \Rightarrow & x_3 = r'^2 - 2x_1 \end{aligned}$$

$$= \left(\frac{3x_1^2 + d}{2y_1}\right)^2 - 2x_1$$

Mit der Formel für y und t' folgt:

$$\begin{aligned} y_3 &= \frac{3x_1^2 + d}{2y_1} x_3 + y_1 - \underbrace{\frac{3x_1^2 + d}{2y_1} x_1}_{=t'} \\ &= \frac{3x_1^2 + d}{2y_1} (x_3 - x_1) + y_1 \end{aligned}$$

Analog zum letzten Beweis wird gezeigt:

$$\begin{aligned} 2P_1 &= -P_3 = (x_3, -y_3) \\ -y_3 &= \frac{3x_1^2 + d}{2y_1} (x_1 - x_3) - y_1 \end{aligned}$$

□

Wir haben nun also Algorithmen um Berechnungen auf einer EK bzw. in der auf ihr befindlichen Gruppe durchzuführen. Mit diesen kann nun auch leicht der fehlende Beweis für die Gültigkeit der Assoziativ-Gesetze erbracht werden. Die Berechnungen lassen sich je nach zugrundeliegendem Körper und seinen additiven und multiplikativen Verknüpfungen direkt in informatischen Code übertragen. Da wir uns für diese informatische Umsetzung auf endliche Körper beschränken müssen, wählen wir für $\mathbb{K}$ einen Primkörper, der sich also ausdrücken lässt als $\mathbb{Z}_p$ wobei p eine Primzahl ist. In diesen lässt sich auch auch das inverse Element mit dem Euklidischen Algorithmus effizient bestimmen. So kann auch die benötigte Division einfach realisiert werden.

12.3.7 Diskretes Logarithmus-Problem auf Elliptischen Kurven

Um die EK für unsere kryptografischen Zwecke nutzen zu können, muss auf ihrer Gruppe ein diskretes Logarithmus-Problem bestehen. Dass diese Art von Problem grundsätzlich schwierig lösbar ist, lässt sich im Allgemeinen nicht zeigen. Es ist jedoch leicht vorstellbar, dass es die komplexe Verknüpfung auf EK schwierig macht für ein gegebenes Paar $(P, Q) \in NP_f$, für welches $kP = Q$ gilt, k tatsächlich zu bestimmen.

Es lässt sich wie in Abbildung 9 veranschaulicht auch geometrisch nicht erkennen wie oft ein Punkt mit sich selber addiert wurde, um zu einem gewissen anderem Punkt zu gelangen.

12.4 ECDSA

Nachdem wir nun in die Mathematik der EK ausführlich eingeführt haben, wollen wir beschreiben wie sie in der Praxis für digitale Signaturen genutzt wird. Dazu stellen wir hier den *ECDSA-Standard* vor, welcher auch von Bitcoin genutzt wird. Die Abkürzung ECDSA steht dabei für *Elliptic Curve Digital Signature Algorithm*. In dem Protokoll dieses Standards wird genau beschrieben, wie man etwas digital signiert und wie diese Signatur nachgeprüft werden kann. Die Verfahren basieren dabei auf der bereits erwähnten ElGammel-Verschlüsselung.

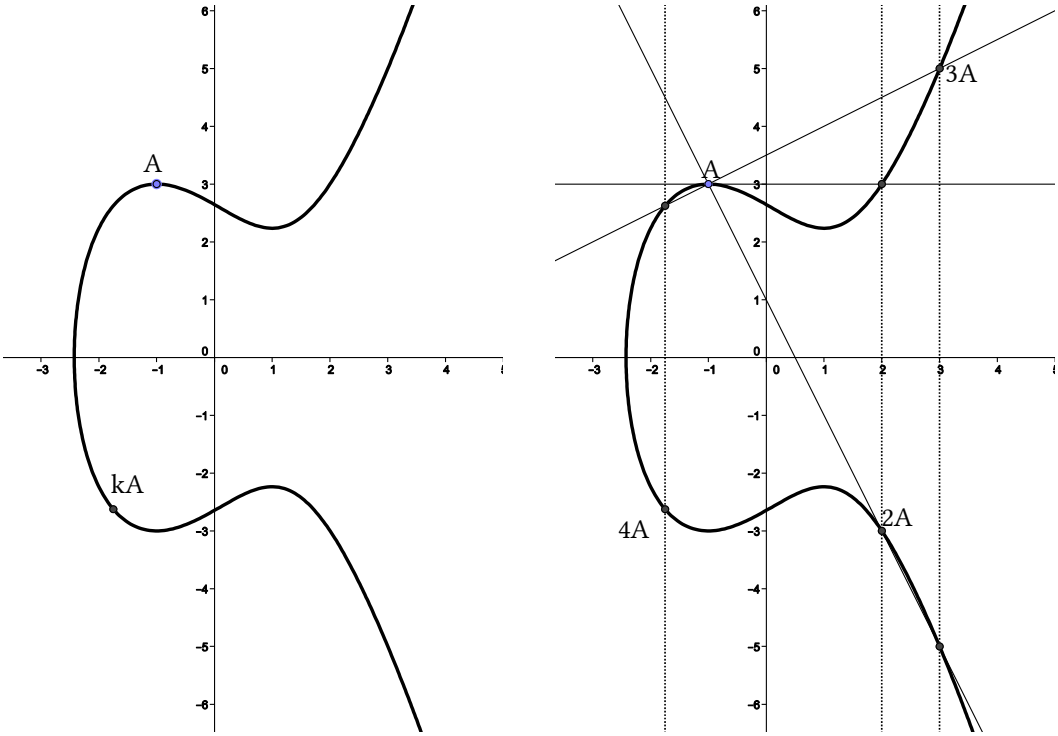

Abbildung 9: Grafisches DLP mit Auflösung

12.4.1 Protokoll

Zunächst müssen sich der Absender (A, Alice) und der Empfänger (B, Bob) einer Nachricht auf einige Parameter einigen. Dazu wird ein Tupel zwischen ihnen ausgetauscht. Dieses beinhaltet folgende Parameter:

1. p
   Die Ordnung des zugrundeliegenden Körpers $\mathbb{Z}_p$, muss eine möglichst große Primzahl sein. (Hier können auch Primzahlpotenzen genutzt werden, was aber in Bitcoin nicht getan wird).

2. e
   Der erste Koeffizient unseres Polynoms $f(x, y) = y^2 - x^3 - dx - e$, das unsere EK definiert. e ist ein Element von $\mathbb{K}$.

3. d
   Der zweite Koeffizient von f.

4. x
   Die erste Koordinate eines Punktes E auf der EK. Dieser erzeugt eine zyklische Untergruppe $\langle E \rangle$

5. y
   Die zweite Koordinate des Punktes E.

6. n
   Die Ordnung der Gruppe $\langle E \rangle$.

7. k
   Der Quotient aus der Anzahl der Punkte der EK selbst sowie der Untergruppe $\langle E \rangle$.

8. $\mathcal{H}$
   Eine kryptografische Hashfunktion.

Dieser Austausch kann über jeden Kanal erfolgen. In Netzwerken wie dem von Bitcoin werden diese Parameter oft vorher festgelegt und gelten für alle Teilnehmer. An diese Parameter werden weitere Anforderungen gestellt, um verschiedene Angriffsmöglichkeiten auszuschließen. Auf diese werden wir im nächsten Abschnitt eingehen. Nachdem diese Einigung stattgefunden hat, kann mit dem eigentlichen Signieren begonnen werden. Nehmen wir nun an das Alice die Nachricht m signieren möchte und somit Bob nachweisen, dass sie es war, die m erstellt hat. Alice führt dann folgende Schritte aus:

A erstellt ein Signatur für m

1. A wählt sich eine zufällige Zahl $d \in \{1, 2, \ldots, n-1\}$ und berechnet $P = dE$. d bildet dann den Privaten Schlüssel und P den öffentlichen. Dieser öffentliche Schlüssel stellt nun auch die Berechnungsgrundlage für die Bitcoin-Adresse dar und entspricht somit dem Pseudonym vom A im Netzwerk.

2. A wählt erneut eine zufällige Zahl $k \in \{1, 2, \ldots, n-1\}$ und berechnet $Q = kE$.

3. A bestimmt k^{-1} mittels des Euklidischen Algorithmus.

4. $Q = (x_Q, y_Q)$ ist nun ein Punkt in $\mathbb{Z}_p^2$ und A berechnet $r = x \mod n$. Falls $r = 0$ beginnt A erneut bei Schritt 2.

5. Dann muss noch $\mathcal{H}(m) = e \in \mathbb{N}$ und

6. $s = k^{-1}(e + rd) \mod n$ bestimmt werden. Falls $s = 0$ geht A wieder zu Schritt 2.

7. Als letztes schickt A das Tupel (m, r, s) an B.

B überprüft eine Signatur

1. B überprüft ob $r, s \in \{1, 2, \ldots, n-1\}$.

2. $\mathcal{H}(m) = e$ wird ebenfalls von B bestimmt.

3. Bob berechnet s^{-1} mittels des Euklidischen Algorithmus.

4. Er ermittelt den Punkt $R = s^{-1}(eE + rP) = (x_R, y_R)$.

5. Falls $x_R = r \mod n$ so ist die Unterschrift gültig und Bob weiß, dass die Nachricht m von A kommt.

Mathematischer Hintergrund

Die Sicherheit, dass wirklich A der Ersteller der Nachricht m und der Unterschrift war, beruht auf folgenden Aussagen:
Nur A war in der Lage $s = k^{-1}(e + rd) \mod n$ korrekt zu berechnen, da dafür d benötigt wird. Die Zufälligkeit von der k bzw. k^{-1} stellt auch sicher, dass wir aus s nicht d berechnen können. Außerdem gilt:

$$s^{-1}s = s^{-1}k^{-1}(e + rd) \equiv 1 \mod n \Rightarrow s^{-1}(e + rd) \equiv k \mod n$$

Daher gilt nun auch:

$$R = s^{-1}(eE + rP) = s^{-1}(eE + rdE) = s^{-1}(e + rd)E = kE$$

Damit ist auch klar das nun $x_Q = x_R \equiv r \mod n$. Diese Kongruenz weißt genauer betrachtet keine Gleichheit nach, jedoch gibt es, da n sehr groß gewählt wird, nur sehr wenige von Q verschiedene Punkte, die diese Gleichung ebenfalls erfüllen.

[Wer02, NSA10, ECC13]

12.4.2 Angriffe

Es gibt verschiedene Angriffe um die Sicherheit von ECDSA anzugreifen. Bei einem solchen Angriff versucht ein Fälscher d zu finden, um damit beliebige Nachrichten zu signieren. Dabei können zwei grundlegend verschiedene Arten differenziert werden. Eine Art von Angriffen sind allgemeine Verfahren um diskrete Logarithmus-Probleme anzugreifen. Diese werden wir zuerst diskutieren. Die anderen Angriffe versuchen Eigenschaften von EK auszunutzen. Wir werden nun einige solcher Angriffe aufzählen und die sich daraus ergebenen Konsequenzen diskutieren.

Brute-Force: Die einfachste und primitivste Methode ist Brute-Force. Dabei werden alle Zahlen $t_i \in \{1, 2, \ldots, n-1\}$ durchprobiert und immer t_iE berechnet bis das Ergebnis P entspricht. Daraus folgt dann: $d = t_i$. Hieraus ergibt sich, dass n möglichst groß sein sollte, da sonst mit viel Rechenleistung das Signier-System angreifbar wäre.

Babystep-Giantstep: Diese Methode basiert auf einer möglichen Zerlegung von d. Dazu bilden wir zunächst $m = \lceil \sqrt{n} \rceil$ und machen uns bewusst, dass es möglich ist d in $d = tm + r$ zu zerlegen. Dabei kann das Paar (t, r) so gewählt werden, dass $r < t \leq m$ ist. Aufgrund der geltenden Assoziativität gilt für ein passendes Paar (t, r) auch

$$P = dE = tmE + rE \Rightarrow P - rE = tmE$$

Die Idee ist es eine Liste von allen Möglichkeiten für tmE und $P - rE$ zu bestimmen und zu speichern. Anschließend vergleicht man diese Listen und sucht nach Übereinstimmungen. Sobald man eine gefunden hat, kennt man d und kann gefälschte Unterschriften leisten. Hierbei ist zu beachten, dass man mindestens $\sqrt{n}$ Punkte speichern muss und maximal $2\sqrt{n}$ Punkte berechnen muss. Allerdings

ist es sehr häufig notwendig auch Listen dieser Größe zu durchsuchen. Daher wird dieser Algorithmus ebenfalls für große n unbrauchbar.

Weitere allgemeine Verfahren: Zwei weitere allgemeine Verfahren sind die Pollard-p-Methode und das Pohling-Hellman-Verfahren. Auf ihnen beruhende Angriffe sind ebenfalls mit einer großen Prim-Untergruppenordnung n zu umgehen.

Kommen wir zu zwei speziell für EK entwickelte Verfahren:

MOV-Algorithmus: Dieses Verfahren reduziert das DLP auf einer EK über einen Körper $\mathbb{K}_p$ auf das DLP von der Gruppe $\mathbb{K}_{p^l}^*$. Um speziell dieses l und noch weitere nötige Parameter zu bestimmen, gibt es kein schnelles Verfahren. Für manche EK können diese Parameter jedoch effizient bestimmt werden. Eine wichtige Art solcher EK sind die sogenannten supersingulären Kurven, d. h. Kurven für welche gilt:

$$\text{char}\,(\mathbb{K}_p)\,|p+1-\text{ord}\,(NP_f)$$

Diese Kurven werden daher in ECDSA ausgeschlossen.

SSSA-Algorithmus Auch in diesem Algorithmus reduziert man das DLP für spezielle EK auf DLP in anderen Gruppen, welche mit den allgemeinen Verfahren schnell gelöst werden können. Die Klasse für die dies mit dem SSSA-Algorithmus möglich ist, bilden die anormalen Kurven. Dies sind Kurven für die die Gleichheit $\text{ord}\,(NP_f) = \text{ord}\,(\mathbb{K}_p)$ erfüllt ist. Da dieser Algorithmus auch auf Untergruppen mit der Ordnung n, für die gilt $n^k = p$, angewendet wird, werden neben den anormalen Kurven auch diese Kurven im ECDSA-Protokoll ausgeschlossen.

12.4.3 Geeignete Elliptische Kurven

Zusammenfassend möchten wir hier noch einmal alle Anforderungen an unsere EK auflisten. Dabei nehmen wir an, dass der zugrundeliegende Körper die Prim-Ordnung p besitzt und schon, die im ECDSA Protokoll angebenen Vorgaben, gelten:

1. wegen dem MOV-Algorithmus:
   $n = \text{ord}\,\langle E\rangle \nmid p^l - 1$ für ausreichend viele l, d. h. bis das DLP von $\mathbb{K}_{p^l}^*$ nicht mehr schneller zu lösen ist, als das eigentlich DLP.

2. Um Angriffe mit dem SSSA-Alogrithmus auszuschließen muss gelten:
   $n \neq p$

3. n muss je nach zur Verfügung stehender Rechenleistung sehr groß sein.

[Wer02]

12.5 Umsetzung in Bitcoin

Bitcoin verwendet auch den schon beschriebenen ECDSA-Standard. Dabei wurden die Parameter im Netzwerk festgelegt und sind somit für alle Nutzer die selben. Diese speziellen Parameter ermöglichen eine Implementierung deren Effizienz erheblich gesteigert ist. Wodurch diese genau entsteht, werden wir im nächsten Kapitel ausführlich behandeln. Zunächst sollen diese Parameter genannt werden, wobei sie im Hexadezimalsystem dargestellt werden.

$$p = \text{FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFF}$$
$$\text{FFFF FFFF FFFF FFFF FFFF FFFE FFFF FC2F}$$

$$e = 7$$

$$d = 0$$

$$x = \text{79BE 667E F9DC BBAC 55A0 6295 CE87 0B07}$$
$$\text{029B FCDB 2DCE 28D9 59F2 815B 16F8 1798}$$

$$y = \text{483A DA77 26A3 C465 5DA4 FBFC 0E11 08A8}$$
$$\text{FD17 B448 A685 5419 9C47 D08F FB10 D4B8}$$

$$n = \text{FFFF FFFF FFFF FFFF FFFF FFFF FFFF FFFE}$$
$$\text{BAAE DCE6 AF48 A03B BFD2 5E8C D036 4141}$$

$$k = 1$$

$$\mathcal{H} = \text{SHA-256}$$

Diese Parameter entsprechen einem Sicherheitsniveau von 256-bit. Wie sich numerisch bestimmen lässt, sind sowohl p als auch n Primzahlen. Weiter lässt sich nachrechnen, dass alle Kriterien, die für die Sicherheit notwendig sind, erfüllt sind. Diese Parameter wurden von der *Efficient Cryptography Group* (SECG) erstellt und in einem Dokument der Öffentlichkeit zugänglich gemacht. Dabei stehen in diesem Dokument mehrere Parameter-Gruppen für verschieden Sicherheitsstufen zur Verfügung. Für jede Sicherheitsstufe werden zwei verschiedene Parameter-Gruppen empfohlen. Die einen wurden nachweislich zufällig erzeugt, die anderen, welche auch hier gewählt wurden, ermöglichen die bereits erwähnte effiziente Implementierung.

[Sec12, Res00]

12.6 Effizienz

Die mögliche Implementierung mit gesteigerter Effizienz basiert auf verschiedenen Eigenschaften der Parameter. Hier soll nun eine dieser Parametereigenschaften sowie eine andere Möglichkeit zur Effizienzsteigerung vorgestellt werden. Im ersten Abschnitt wird ein einfaches Schema beschrieben, welches es erlaubt Vielfache von Punkten mit weniger Operationen zu berechnen. Anschließend soll dann noch eine Möglichkeit beschrieben werden, welche die Körperoperationsanzahl für solch eine Skalarmultiplikation, mittels eines schnell zu berechnenden Endomorphismus, weiter senkt. Wie deutlich werden wird, basiert diese schnelle Berechenbarkeit des Endomorphismus ebenfalls auf Parametereigenschaften.

12.6.1 Verdopplung von Punkten

Beim Vorstellen des ECDSA-Standards ist deutlich geworden, dass man sowohl zum Erstellen einer Signatur als auch zum Überprüfen einer Signatur mindestens eine Multiplikation eines Punkts durchführen muss. Besonders das Überprüfen einer Signatur muss äußert häufig vorgenommen werden. Da alle Nutzer, die versuchen einen Block zu erstellen, wie in Teil II beschrieben, für jede Transaktion, die sie in ihren Block aufnehmen wollen, die Signatur prüfen müssen. Auch Teilnehmer des Netzwerks, die ihre Blockchain aktualisieren möchten, müssen jede Signatur in den neu erhaltenen Blöcken auf ihre Zulässigkeit prüfen. Die einfachste Art ein solches k-Vielfaches eines Punkts zu berechnen, ist es, rekursiv diesen Punkt zu addieren. Dafür sind jedoch $k-1$ Punkt-Additionen nötig. Da jede dieser Punktadditionen 6 Additionen, 2 Multiplikationen sowie die Bestimmung eines inversen Elements im Körper verlangen, benötigt es $k-1$ mal 8 Körperoperationen plus $k-1$ mal die Anzahl an Körperoperationen, die im Durchschnitt für die Bestimmung des inversen Elements nötig sind. Wie können wir nun Operationen einsparen?

Algorithmus zur Bestimmung von kQ: Folgender C++-Code geht davon aus, dass eine Klasse `Point` definiert ist, die die Operatoren für die Addition und die Multiplikation mit der 2 definiert hat. Die Klasse `Integer` stellt eine Zahl in $\mathbb{Z}_n$ dar, auf deren Binäreinträge wir mittels dem []-Operator zugreifen können.

```
Point multiply(Point Q, const Integer k) {
        Integer iMax = 255;
        while (k[iMax] == 0 && iMax > 0) {
                --iMax;
        }
        Point N = 0;
        Point P = Q;
        for (int i = 0; i <= iMax; i++) {
                Q = 2 * Q;
                P += Q;
                if (k[i] == 0) {
                        N += Q;
                }
```

```
        }
        return P - N;
}
```

Dieser Algorithmus benötigt nur $\log_2 k$ Durchläufe, wobei jeder eine Punktdopplung und maximal 2 Punktadditionen benötigt. Daher benötigt er nur $log_2(k) \cdot (2\,(8) + 12)$ Körperoperationen plus $\log_2(k)$ mal die Anzahl der Operatoren, die für das Invertieren eines Elements notwendig sind. Damit liegt der Aufwand dieses Algorithmus nur in der Aufwandsklasse $\mathcal{O}\,(\log_2 k)$ im Gegensatz zum linearen Aufwand des naiven Weges.

12.6.2 Verwendung eines Endomorphismus

Wenn wir annehmen, dass wir einen Endomorphismus Φ kennen, der in der Untergruppe $\langle E\rangle$ multiplikativ wirkt, d. h. folgende Eigenschaften erfüllt:

1. $\forall P \in \langle E\rangle$ gilt: $\Phi\,(P) \in \langle E\rangle$
2. $\forall P_1, P_2 \in \langle E\rangle$ gilt: $\Phi\,(P_1 + P_2) = \Phi\,(P_1) + \Phi\,(P_2)$
3. $\exists! \lambda \in \mathbb{N}$ sodass $\forall P \in \langle E\rangle$ gilt: $\Phi\,(P) = \lambda P$
4. $\forall P \in \langle E\rangle$ ist $\Phi\,(P)$ ist sehr viel schneller zu berechnen als λP

dann können wir diesen nutzen, um die Berechnung von kP für alle $k \in \{1, \ldots, n-1\}$ und alle $P \in \langle E\rangle$ zu beschleunigen. Dazu gehen wir wie folgt vor: Wir zerlegen k in $r_1$und r_2, sodass gilt:

$$\begin{aligned} & k = r_1 + r_2\lambda \\ \Rightarrow \quad & kP = r_1P + r_2\lambda P \\ \Rightarrow \quad & kP = r_1P + r_2\Phi\,(P) \end{aligned}$$

Wenn wir r_1 und r_2, abhängig von λ, optimal wählen können, sind beide nur von der Größenordnung $\sqrt{n}$ und nicht wie k von der Größenordnung n. Nun müssen wir also anstelle von maximal n mal P nur etwa $2\sqrt{n}$ mal P berechnen. Daher können wir den Aufwand zur Berechnung von kP in Kombination mit dem im vorherigem Abschnitt vorgestellten Algorithmus der Klasse $\mathcal{O}\left(\log_2\left(\sqrt{k}\right)\right)$ zuordnen. Dabei wird natürlich vorausgesetzt, dass sich die Bestimmung von $\Phi(P)$ und der Zerlegung von k in derselben oder in einer niedrigeren Aufwandsklasse befindet. Es ist also interessant eine solche Abbildung mit einem passenden Algorithmus für die Zerlegung k zu finden.

Wir zeigen nun, dass ein beliebiger Endomorphismus neben den definitionsgemäßen Eigenschaften 1 und 2 in einer zyklischen Gruppe auch immer die Eigenschaft 3 erfüllt.

Beh.: Sei Φ ein Endomorphismus auf einer zyklischen Untergruppe wie z. B. $\langle E\rangle$

$$\Rightarrow \quad \exists! \lambda \in \mathbb{K} = \mathbb{Z}_p \text{ sodass } \forall P \in G \text{ gilt: } \Phi\,(P) = \lambda P$$

Bew.:

$$
\begin{aligned}
& \Phi(E) = P_1 \in \langle E \rangle \\
& \text{da } P_1 \in \langle E \rangle \\
\Rightarrow \quad & \Phi(E) = P_1 = \lambda E \text{ für genau ein } \lambda \in \mathbb{N} \\
\Rightarrow \quad & \Phi(P) = \Phi(\mu E) \quad \mu \in \mathbb{N} \\
& \qquad = \underbrace{\Phi(E) + \ldots + \Phi(E)}_{\mu \text{ mal}} \\
& \qquad = \mu \Phi(E) \\
& \qquad = \mu \lambda E = \lambda \mu E = \lambda P
\end{aligned}
$$

□

Nun brauchen wir also nur noch einen Endomorphismus, der die Eigenschaft 4 erfüllt. Um diesen zu finden, bedienen wir uns wieder bekannter Eigenschaften der Parameter aus 12.5. Dazu sei hier erwähnt, dass $p \equiv 1 \mod 3$. Hieraus folgt unmittelbar, dass die Ordnung von $\operatorname{ord}\left(\mathbb{Z}_p^*\right) = p - 1 \equiv 0 \mod 3$ ist. Daher gibt es mindestens ein Element in $\mathbb{Z}_p^*$, das die Ordnung 3 besitzt und ungleich 1 ist. Betrachten wir nun das affine Polynom h, dessen Nullstellen unserer EK entsprechen, finden wir ein Endomorphismus auf N_h wenn wir wie folgt vorgehen:

$$
\begin{aligned}
f(x, y) = y^2 - x^3 - 7 = 0 \\
y^2 = x^3 + 7
\end{aligned}
$$

Sei nun $\beta \in \mathbb{Z}_p^*$ und von der Ordnung 3 und der Punkt $E{=}(x_E, y_E)$ eine Lösung dieser Gleichung, so löst auch der Punkt $(\beta x_E, y_E)$ selbige. Denn es lässt sich rechnen:

$$
\begin{aligned}
& y_E^2 = (\beta x)_E^3 + 7 \\
\Leftrightarrow \quad & y_E^2 = \beta^3 x_E^3 + 7 \\
\Leftrightarrow \quad & y_E^2 = x_E^3 + 7
\end{aligned}
$$

Weisen wir nun nach, dass Φ eine derartige Abbildung ein Endomorphismus ist und machen uns bewusst, dass wir ihn mit nur einer Körperoperation bestimmen können, haben wir einen Endomorphismus, der die ersten 4 Eigenschaften besitzt.

Beh.:

$$
\begin{aligned}
\Phi : (x, y) \mapsto (\beta x, y) \quad \Rightarrow \quad \Phi(P_1 + P_2) = \Phi(P_1) + \Phi(P_2) \quad \forall P_1, P_2 \in \langle E \rangle \\
0 \mapsto 0
\end{aligned}
$$

Bew.:

1. Fall: $P_1 = P_2$

$$\begin{aligned}
\Phi(x,y) + \Phi(x,y) &= (\beta x, y) + (\beta x, y) \\
&= \left(\underbrace{\left(\frac{3\beta^2 x^2}{2y}\right)^2 - 2\beta x}_{x_3'}, \frac{3\beta^2 x^2 - d}{2y}(\beta x - x_3') - y \right) \\
&= \left(\underbrace{\beta^4}_{\beta} \left(\frac{3x^2}{2y_1}\right)^2 - 2\beta x, \beta^2 \frac{3x^2 - d}{2y_1}(\beta x - x_3') - y \right) \\
&= \left(\beta \underbrace{\left(\left(\frac{3x^2}{2y}\right)^2 - 2x\right)}_{x_3}, \underbrace{\beta^3}_{1} \frac{3x^2 - d}{2y}(x - x_3) - y \right) \\
&= \left(\beta \left(\left(\frac{3x^2}{2y}\right)^2 - 2x\right), \frac{3x^2 - d}{2y}(x - x_3) - y \right) \\
&= \Phi\left(\left(\frac{3x^2}{2y}\right)^2 - 2x, \frac{3x^2 - d}{2y}(x - x_3) - y \right) = \Phi((x,y) + (x,y))
\end{aligned}$$

2. Fall: $P_1 \neq P_2$

$$\begin{aligned}
\Phi(x_1,y_1) + \Phi(x_2,y_2) &= (\beta x_1, y_1) + (\beta x_2, y_2) \\
&= \left(\underbrace{\left(\frac{y_2 - y_1}{\beta x_2 - \beta x_1}\right)^2 - \beta x_1 - \beta x_2}_{x_3'}, \frac{y_2 - y_1}{\beta x_2 - \beta x_1}(\beta x_1 - \beta x_3') - y_1 \right) \\
&= \left(\left(\frac{1}{\beta}\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - \beta(x_1 - x_2), \frac{1}{\beta}\frac{y_2 - y_1}{x_2 - x_1}(\beta x_1 - x_3') - y_1 \right) \\
&= \left(\underbrace{(\beta^{-1})^2}_{\beta} \left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - \beta(x_1 - x_2), \frac{1}{\beta}\frac{y_2 - y_1}{x_2 - x_1}(\beta x_1 - x_3') - y_1 \right) \\
&= \left(\beta \underbrace{\left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2\right)}_{x_3}, \frac{1}{\beta}\frac{y_2 - y_1}{x_2 - x_1}(\beta x_1 - \beta x_3) - y_1 \right) \\
&= \left(\beta \left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2\right), \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1 \right) \\
&= \Phi\left(\left(\frac{y_2 - y_1}{x_2 - x_1}\right)^2 - x_1 - x_2, \frac{y_2 - y_1}{x_2 - x_1}(x_1 - x_3) - y_1 \right) \\
&= \Phi((x_1,y_1) + (x_2,y_2))
\end{aligned}$$

□

Damit haben wir einen guten Endomorphismus gefunden. Wie schon gezeigt, wirkt dieser auf $\langle E \rangle$ auch als multiplikative Abbildung. Allerdings kennen wir λ noch nicht und wissen somit nicht, auf welches ihrer Vielfachen Φ die Elemente abbildet. Die Kenntnis von λ ist aber von essenzieller Bedeutung, da wir ohne sie auch nicht die Zerlegung von k bestimmen können. Da Φ ein Endomorphismus ist, besitzt er auch ein charakteristisches Polynom. Es kann nachgewiesen werden, dass Endomorphismen $\mathrm{End}\,(NP_f)$ über den EK einen Ring bilden. Dabei sind die beiden Verknüpfungen definiert als:

$$\begin{aligned} " + " : \mathrm{End}\,(NP_f)^2 &\to \mathrm{End}\,(NP_f) \\ (\Lambda + \Omega)\,(P) &\mapsto \Lambda\,(P) + \Omega\,(P) \end{aligned}$$

$$\begin{aligned} " \circ " : \mathrm{End}\,(NP_f)^2 &\to \mathrm{End}\,(NP_f) \\ (\Lambda \circ \Omega)\,(P) &\mapsto \Lambda\,(\Omega\,(P)) \end{aligned}$$

Außerdem gilt, id ist das neutrale Element von $\mathrm{End}\,(NP_f)$ und da $\beta^3 = 1$ gilt:

$$\begin{aligned} & \Phi^3 = \mathrm{id} \\ \Rightarrow \quad & \left(\Phi^3 - \mathrm{id}\right) = 0 \\ \Rightarrow \quad & (\Phi - \mathrm{id})\left(\Phi^2 + \Phi + \mathrm{id}\right) = 0 \end{aligned}$$

Da nun wie aus seiner Konstruktion direkt geschlossen werden kann $\Phi \neq \mathrm{id}$, kann nun gefolgert werden:

$$\begin{aligned} \Rightarrow \quad & \underbrace{(\Phi - \mathrm{id})}_{\neq 0}\underbrace{\left(\Phi^2 + \Phi + \mathrm{id}\right)}_{=0} = 0 \\ \Rightarrow \quad & \Phi^2\,(P) + \Phi\,(P) + \mathrm{id}\,(P) = 0 \\ \Rightarrow \quad & \lambda^2 P + \lambda P + P = 0 \\ \Rightarrow \quad & \lambda^2 + \lambda + 1 = 0 \mod n \end{aligned}$$

Somit können wir also λ numerisch bestimmen. Nun brauchen wir nur noch einen effektiven Weg, der zu gegebenen k, λ und n passende r_1 und r_2 bestimmt. Dafür gibt es bereits Möglichkeiten wie z. B. den LLL-Algorithmus. Da dieser sehr aufwendig ist, können wir ihn nicht immer neu anwenden, wenn wir passende r_1 und r_2 bestimmen wollen. Eine andere Möglichkeit, die sich durch besondere Schnelligkeit auszeichnet, möchten wir nun beschreiben. Bevor wir damit anfangen sei jedoch erneut darauf aufmerksam gemacht, dass im Netzwerk die Parameter festgelegt wurden. Da wir auch β festlegen, bleibt der Endomorphismus Φ und damit λ immer gleich. Um nun die tatsächliche Zerlegung zu bestimmen, suchen wir zuerst Vektoren v_1, v_2 mit folgenden Eigenschaften:

1. $v_1, v_2 \in \mathbb{Z}_n^2$
2. $v_1 \neq \lambda v_2 \forall \lambda \in \mathbb{Q}$

3. $\|v_{1,2}\| = \sqrt{x_{1,2}^2 + y_{1,2}^2} \approx \sqrt{n}$

4. $f(v_{1,2}) = x_{1,2} + y_{1,2}\lambda = 0 \mod n$

Vektoren mit diesen Eigenschaften können wir mit dem erweiterten Euklidischen Algorithmus finden. Dieser Algorithmus liefert Tupel (s_i, t_i, r_i), sodass gilt:

$$\begin{aligned} & s_i n + t_i \lambda = r_i \\ \Rightarrow \quad & r_i + (-t_i)\,\lambda = s_i n \\ \Rightarrow \quad & r_i + (-t_i)\,\lambda = 0 \mod n \end{aligned}$$

Hieraus ist sofort ersichtlich, dass für jeden Vektor $v_i = (r_i, -t_i)$ die Bedingung $f(v_i) = 0$ erfüllt ist. Betrachtet man außerdem den Algorithmus selbst, so sieht man leicht:

$$\begin{aligned} |r_i| &> |r_{i+1}| \\ |t_i| &< |t_{i+1}| \end{aligned}$$

Sei nun l der letzte Index, sodass $r_l \geq \sqrt{n}$, dann sehen wir mit einer Analyse des erweiterten Euklidischen Algorithmus, dass die Vektoren $v_1 = (r_{l+1}, t_{l+1})$ und $v_2 = (r_{l+2}, t_{l+2})$ bzw. $v_2 = (r_l, t_l)$ linear unabhängig sind. Weiter gilt außerdem:

1. $\|v_1\|$ hat die Größenordnung $\sqrt{n}$.

2. $\|v_2\|$ ist nach einer heuristischen Analyse ebenfalls klein.

Wir haben also ein Paar (v_1, v_2) gefunden, welches völlig unabhängig von k ist und die Bedingungen 1. bis 4. erfüllt. Nun können wir eine Gleichung für einen beliebigen Vektor der Gestalt $(k, 0)$ erstellen:

$$\begin{pmatrix} k \\ 0 \end{pmatrix} = c_1 \underbrace{\begin{pmatrix} x_1 \\ y_1 \end{pmatrix}}_{v_1} + c_2 \underbrace{\begin{pmatrix} x_2 \\ y_2 \end{pmatrix}}_{v_2} \qquad c_1, c_2 \in \mathbb{Q}$$

Diese c_1 und c_2 können mit einfacher linearer Algebra berechnet werden. Anschließend können wir definieren:

$$\begin{aligned} \overline{c_1} &= \lceil c_1 \rfloor \in \mathbb{Z} \\ \overline{c_2} &= \lceil c_2 \rfloor \in \mathbb{Z} \\ v &= \overline{c_1} v_1 + \overline{c_2} v_2 \end{aligned}$$

v ist nun ein Vektor sehr nahe an $(k, 0)$. Somit gilt:

$$r = \begin{pmatrix} k \\ 0 \end{pmatrix} - v$$

r hat sehr kleine Euklidische Norm und es folgt aus der Konstruktion von v:

$$\begin{aligned} f(r) = r_1 + r_2\lambda = f\left(\begin{pmatrix} k \\ 0 \end{pmatrix} - v\right) \\ &= f\begin{pmatrix} k \\ 0 \end{pmatrix} - f(v) \\ &= f\begin{pmatrix} k \\ 0 \end{pmatrix} \\ &= k + 0\lambda \\ \Rightarrow \quad k &= r_1 + r_2\lambda \mod n \end{aligned}$$

Damit haben wir unsere Zerlegung gefunden. Um die Größenordnung von r_1 und r_2 abzuschätzen, können wir nun folgende Rechnung tätigen.

$$\begin{aligned} \|r\| = \sqrt{r_1^2 + r_2^2} &= \left\| \begin{pmatrix} k \\ 0 \end{pmatrix} - (\overline{c_1}v_1 + \overline{c_2}v_2) \right\| \\ &= \left\| \begin{pmatrix} k \\ 0 \end{pmatrix} - (c_1 + \tilde{c_1})\, v_1 - (c_2 + \tilde{c_2})\, v_2 \right\| \\ &= \left\| \begin{pmatrix} k \\ 0 \end{pmatrix} - \underbrace{(c_1v_1 + c_2v_2)}_{(k,0)} - (\tilde{c_1}v_1 + \tilde{c_2}v_2) \right\| \\ &= \|\tilde{c_1}v_1 + \tilde{c_2}v_2\| \\ &\leq \|\tilde{c_1}v_1\| + \|\tilde{c_2}v_2\| \\ &= |\tilde{c_1}|\,\|v_1\| + |\tilde{c_2}|\,\|v_2\| \\ &\approx |\tilde{c_1}|\,\sqrt{n} + |\tilde{c_2}|\,\sqrt{n} \\ &= \underbrace{(|\tilde{c_1}| + |\tilde{c_2}|)}_{\leq 1}\,\sqrt{n} \\ \Rightarrow \quad \sqrt{r_1^2 + r_2^2} &\lesssim \sqrt{n} \end{aligned}$$

Damit ist es ersichtlich, dass r_1 und r_2 in der Größenordnung von $\sqrt{n}$ liegen. Also haben wir auf effiziente Weise eine optimale Zerlegung gefunden. In der Anwendung hat sich gezeigt, dass durch Anwenden dieser Technik mit vorher bestimmten v_1 und v_2 etwa 30 % der Zeit, die zum Überprüfen einer Signatur benötigt wird, eingespart werden kann.

Mit dieser Technik und der zuvor vorgestellten Möglichkeit zur schnelleren Punkt-Addition haben wir unsere Effizienz zur Überprüfung einer Signatur stark gesteigert.

[HMV03, GLV99]

12.7 Vergleich zu RSA

Abschließend werden wir noch einen Vergleich zwischen den beiden vorgestellten Public-Key-Verfahren anstellen. Für einen Angriff auf das RSA-System stehen weit mehr Angriffsalgorithmen zu Verfügung als für das EK-System. Manche Verfahren, die RSA angreifen haben dabei subexponentielle Laufzeit [Wer02, 98ff]. Für EK hingegen existiert kein bekannter Angriff, für den erwartet werden kann, dass er in subexponentieller Laufzeit das DLP löst. So resultieren schon aus vergleichsweise geringen Schlüssellängen sehr hohe Sicherheitsniveaus. Diese erwarteten Werte wurden in folgender Tabelle für die in 12.5 bereits erwähnten empfohlenen Kurven zusammengefasst:

Kurven Kürzel	ord $(\mathbb{K})$ in Bit	RSA-Modulus in bits
⋮	⋮	⋮
secp192k1	192	1536
secp192r1	192	1536
secp224k1	224	2048
secp224r1	224	2048
secp256k1	***256***	***3072***
secp256r1	256	3072
⋮	⋮	⋮

Tabelle 1: Größenordnung einer EK und eines vergleichbar sicheren RSA-Moduls aus [Res00]

Wie hier zu sehen, müssen RSA-Module und damit auch die in der Datenbank zu speichernden Public-Keys aller Netzwerknutzer wesentlich größer sein, um das selbe Sicherheitsniveau, wie es relativ kurze Public-Keys des EK-Systems innehaben, zu erreichen. Damit eignen sich EK insbesondere für Anwendungen wie Bitcoin, bei denen große Datenmengen von vielen Leuten gespeichert werden müssen. [Res00, Wer02]

Teil IV

Anonymität

In diesem Teil wollen wir zuerst die Probleme, die Bitcoin bzgl. der Anonymität hat, vorstellen und besprechen, welche Lösungsmöglichkeiten dafür existieren. Eine vorgeschlagene Erweiterung zu Bitcoin ist *Zerocoin*, welche eine komplette Anonymisierung ermöglicht, ohne dabei wieder den Nachteil einzuführen, dass eine vertrauenswürdigen Instanz vonnöten ist. Um die Funktionsweise von Zerocoin zu verstehen, sind insbesondere zwei *Signatures-of-knowledge* notwendig. Hierfür erklären wir die Grundlagen von *Zero-Knowledge-Protokollen* und wie man deren Gültigkeit beweisen kann. Im letzten Abschnitt führen wir noch den *Akkumulator* ein, welcher eine abgewandelte Hashfunktion darstellt und auch von Zerocoin verwendet wird.

13 Probleme von Bitcoin

Obwohl in der Blockchain keine Namen gespeichert werden, hat Bitcoin dennoch ein Problem. Denn sobald ein Teilnehmer, z. B. bei einem Online-Shop, etwas kauft, ist dem Shop der echte Name, der sich hinter der Bitcoin-Adresse verbirgt, bekannt. Alle Teilnehmer des Bitcoin-Netzwerks besitzen dieselbe Version der Blockchain mit allen Transaktionen, also auch der Online-Shop. Es ist diesem nun möglich, die sonstigen Aktionen des Käufers nachzuvollziehen, z. B. woher die Bitcoins, mit denen der Käufer bezahlt hat, stammen und eventuell sogar, welche Einkäufe dieser sonst noch getätigt hat oder tätigen wird. Letzteres wird dem Shop ermöglicht, da er die Bitcoin-Adresse kennt, an die das Wechselgeld (siehe 4.4) gezahlt wird. In der Regel gehört diese wieder dem Käufer.

Des Weiteren kennt auch der Bitcoin-Exchange, sofern man damit Bitcoins erlangt, die Identität. Das bedeutet, dass man nicht nur dort, wo man Bitcoins ausgibt, seine Identität preisgibt, sondern auch meistens dort, wo man sie erhält.

Abbildung 10 zeigt beispielhaft einen Transaktionsverlauf. Die Pfeile stellen Transaktionen dar, wobei sich Alice beim Kauf in Shop 1 und 2 das Wechselgeld wieder an sich selbst überweist. Hierbei können sowohl der Bitcoin-Exchange, als auch Shop 1 und Shop 2 auf die anderen Aktivitäten von Alice schließen.

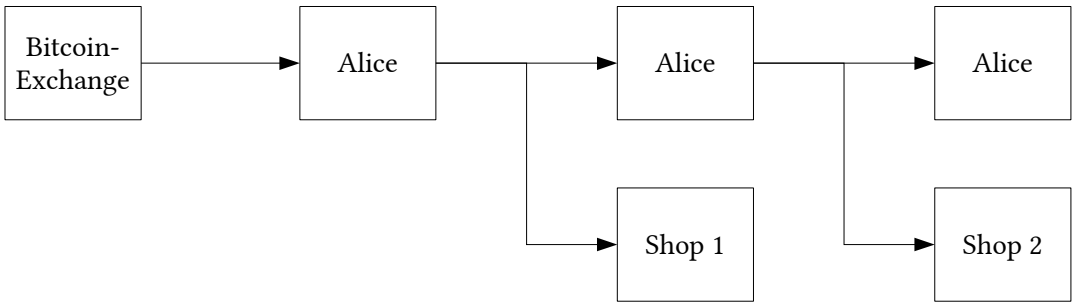

Abbildung 10: Beispiel zur Anonymität

Dasselbe Problem existiert auch bei anderen Überweisungen. Wenn Alice an Bob etwas bezahlt, kann nun Bob den Public-Key, von dem er die Bitcoins erhalten hat, Alice zuordnen und weiß damit Bescheid über ihre Identität. Selbst wenn Alice verschiedene Bitcoin-Adressen für jede Überweisung verwendet, ist es oft relativ schnell klar, wobei es sich um die eigentliche Transaktion handelt und wobei um Wechselgeld.

14 Laundry

Eine Lösung des Problems besteht darin, dass mehrere Personen ihre Bitcoins an eine sogenannte *Laundry* überweisen. Dabei schickt jeder dieser zwei Bitcoin-Adressen, z. B. über eine Internetseite. Bei der ersten Adresse handelt es sich um die, von der die Bitcoins überwiesen werden, bei der zweiten um die, auf die die Bitcoins überwiesen werden sollen. Nachdem die Laundry von der ersten Adresse Bitcoins erhalten hat, überweist sie diese nun wieder zurück, allerdings an die zweite Bitcoin-Adresse. Außenstehenden ist es nun nicht möglich herauszufinden, welche neue Bitcoin-Adresse zu welcher alten gehört. Ein Nachteil hierbei ist allerdings, dass die Laundry vertrauenswürdig sein muss. Denn sie weiß über die Identitäten Bescheid und könnte diese speichern und später offenbaren. Außerdem könnte eine böswillige Laundry einigen Leuten oder sogar allen ihre Bitcoins nicht zurück überweisen und diese stattdessen an eine eigene Adresse schicken.

Bei diesem Prinzip wird also wieder ein zentrales System geschaffen, wie man es von Banken kennt.

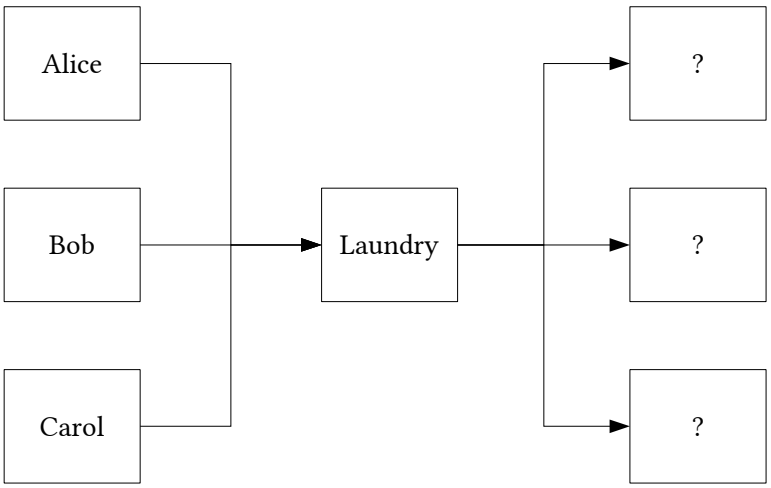

Abbildung 11: Beispiel einer Laundry mit 3 Teilnehmern

Außerdem ist es bei einer Laundry notwendig, dass alle Teilnehmer ihr den selben Betrag überweisen, da sonst natürlich sofort eine Zuordnung von Ein- und Auzahlung der Laundry möglich ist.

Ein weiterer Nachteil ist, dass viele Laundrys für ihren Dienst Gebühren verlangen. Zum Beispiel verlangt Bitcoin Fog eine Gebühr von 1 bis 3 % vom Transaktionswert [Fog13].

Es besteht also der Bedarf nach einem besseren System zur Anonymisierung.

15 Zerocoin

Mit Zerocoin wurde eine Erweiterung des Bitcoin-Netwerks geschaffen, die ähnlich wie eine Laundry funktioniert. Diese wird hierbei aber nicht von einer Instanz vertreten, sondern arbeitet dezentral. Dazu ist eine neue Transaktion notwendig, welche keinen Empfänger beinhaltet. Stattdessen wird eine Zerocoin geprägt, was dem Einzahlen in eine Laundry entspricht. Später kann der Präger diese Zerocoin wieder in eine Bitcoin umwandeln. Hierbei ist es nicht möglich, die Prägung und das Zurückverwandeln miteinander zu verbinden.

[MGGR13]

15.1 Beispiel

Im Folgenden wollen wir die Funktionsweise an einem Beispiel verdeutlichen: Alice hat eine Krankheit und möchte diesen Umstand gerne geheim halten. Sie möchte sich bei einer Online-Apotheke ein Medikament kaufen und verhindern, dass sich dies jemals zurück verfolgen lässt. Während das Paket, in dem das Medikament an sie geschickt wird, nur von der Post verfolgt werden kann, kann ihre Transaktion zum Bezahlen an die Apotheke theoretisch ewig (solange Bitcoin existiert) von allen Teilnehmern des Bitcoin-Netzwerks gesehen werden.

Alice prägt nun eine Zerocoin, indem sie sich dazu eine Zufallszahl r erstellt, eine Seriennummer S und eine dazu gehörige Zahl $\mathbf{c}$, genannt *Commitment*. Letztere veröffentlicht sie auf der Blockchain. Genauso wie sie verfahren mehrere andere Teilnehmer des Bitcoin-Netzwerks für ihre eigenen Anliegen.

Anschließend wandelt Alice die Zerocoin wieder in Bitcoins um. Dazu veröffentlicht sie die Seriennummer S zusammen mit einer Signatur. Diese Signatur beweist, ohne $\mathbf{c}$ und r zu verraten, dass sie ein r für eine Zerocoin kennt, die in der Vergangenheit auf der Blockchain geprägt wurde. Andere Teilnehmer akzeptieren diese Transaktion nur, wenn die Signatur gültig ist und S nicht bereits in einer vorherigen Transaktion verwendet wurde. Als Empfänger der Transaktion gibt Alice die Bitcoin-Adresse der Online-Apotheke an.

Da man Prägung und Ausgeben der Zerocoin nicht verbinden kann, sofern mehrere Leute Zerocoins geprägt haben, ist die Überweisung von Alice an die Apotheke nun nicht mehr zurückverfolgbar. Dieses System funktioniert also umso besser, je mehr Leute es benutzen, da die Verbindung sonst zu leicht erraten werden kann.

15.2 Commitment

Wie im Beispiel erklärt, muss Alice eine Seriennummer S erstellen und ein Commitment, welches verhindert, dass sie eine andere Nummer statt S beim Ausgeben der Zerocoin benutzt. Da andere Teilnehmer des Netzwerks die Seriennummer nicht aus dem Commitment errechnen können sollen, eignet sich ein exponentieller Zusammenhang mit $S = \log \mathbf{c}$ (siehe 11.3). Dies reicht noch nicht aus, da man umgekehrt aus einer veröffentlichten Seriennummer so das Commitment ausrechnen könnte und damit die Anonymität nicht mehr gewahrt wäre. Daher generiert Alice zusätzlich eine Zufallszahl r, die sie nie veröffentlicht.

Seien p, q Primzahlen für die gilt $p = 2^w q + 1$, $w \in \mathbb{N}$. Man wähle $\mathbf{g}, \mathbf{h} \in \mathbb{Z}_q^*$ zufällig[2] so, dass

[2]Das von Zerocoin dazu verwendete Verfahren wird in [GFD09, Appendix A.2.3] erklärt.

$\langle \mathbf{g} \rangle = \langle \mathbf{h} \rangle = \mathbb{Z}_q^*$. Dann wird das Commitment für S folgendermaßen berechnet:

$$\mathbf{c} = \mathbf{g}^S \mathbf{h}^r \mod p$$

Ist $\mathbf{c}$ keine Primzahl, muss eine neue Zufallszahl r benutzt werden. Den Grund für diese Anforderung erklären wir in 15.5.

Wenn eine Zerocoin ausgegeben wird, veröffentlicht man die Seriennummer S, damit eine Zerocoin nicht mehrmals ausgeben werden kann. Man beweist außerdem mit zwei Signaturen, dass das eigene Commitment $\mathbf{c}$ in der Blockchain veröffentlicht wurde, und dass man ein r kennt, sodass $\mathbf{c} = \mathbf{g}^S \mathbf{h}^r$. Im Folgenden wollen wir die Grundlagen, die für diese beiden Signaturen wichtig sind, erläutern.

15.3 Zero-Knowledge-Protokolle

Bei einem Zero-Knowledge-Protokoll kommunizieren Beweiser und Verifizierer. Ersterer möchte beweisen, dass eine bestimmte Aussage wahr ist. Er könnte nun den Beweis dazu einfach dem Verifizierer zeigen, sodass auch dieser diesen selbst durchführen kann. Sein Ziel ist es aber, den Verifizierer von der Wahrheit der Aussage zu überzeugen, ohne ihm zu verraten, wie oder warum er dies weiß. Der Verifizierer kann nach Durchführung des Zero-Knowledge-Protokolls nicht dieses selber anwenden, um z. B. eine dritte Person von der Wahrheit der Aussage zu überzeugen.

Zero-Knowledge-Protokolle werden oft auch Zero-Knowledge-Beweise genannt. Da sie aber keinen Beweis im mathematischen Sinne darstellen, sondern kryptografischen Protokollen entsprechen, werden wir sie im Folgenden Protokolle nennen.

15.3.1 Höhlen-Beispiel

Wir wollen kurz ein einfaches Zero-Knowledge-Protokoll nach [ZKB13] vorstellen. Peggy P (die Beweiserin) möchte Victor V (dem Verifizierer) zeigen, dass sie das magische Wort x kennt, um in einer Höhle eine Tür zu öffnen. Die Höhle ist dabei zirkulär aufgebaut mit einer Tür im Inneren. Peggy möchte Victor aber nicht das Wort verraten, sondern ihn nur überzeugen, dass sie es kennt.

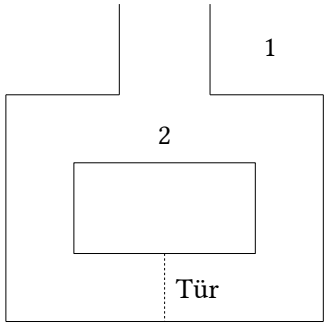

Abbildung 12: Grundriss der Höhle

Victor und Peggy starten an Position 1 (siehe Abb. 12). Peggy betritt nun die Höhle und wählt per Zufall einen Weg links oder rechts bis vor die Tür, wobei Victor nicht sieht, welchen Weg sie nimmt,

nur dass sie die Höhle betritt. Nun begibt er sich zu Position 2 und ruft in die Höhle, aus welchem Ausgang Peggy diese verlassen soll. Mit einer Wahrscheinlichkeit von $\frac{1}{2}$ muss sie dabei durch die Tür, wofür sie das magische Wort x benutzen muss. Die beiden wiederholen dieses Protokoll nun so oft, bis Victor überzeugt ist, dass Peggy x kennt. Verlässt sie die Höhle durch den falschen Ausgang, bricht Victor ab und glaubt Peggy nicht, dass sie das geheime Wort für die Tür kennt.

Mit einer Wahrscheinlichkeit von $\frac{1}{2}$ ist es möglich, dass Peggy Glück hat und Victor genau möchte, dass sie aus dem Ausgang kommt, in den sie gegangen ist. Hierbei braucht sie das geheime Wort nicht zu kennen. Die Wahrscheinlichkeit, dass Victor nach k Versuchen glaubt, sie kenne das Wort, obwohl sie dies nicht tut, beträgt also:

$$\mathbb{P}\left(V \text{ akzeptiert } \tilde{P}\right) = 2^{-k}$$

$\tilde{P}$ bezeichnet einen unehrlichen Beweiser, der sich also nicht ans Protokoll hält oder x nicht kennt. Mit hohem k wird die Wahrscheinlichkeit vernachlässigbar klein, weswegen man dieses Protokoll *korrekt* nennt. Die Wahrscheinlichkeit, dass V glaubt, dass P x kennt und P dies auch wirklich tut, lautet entsprechend:

$$\mathbb{P}\left(P \text{ kennt } x \mid V \text{ akzeptiert } P\right) = 1 - 2^{-k}$$

Da diese Wahrscheinlichkeit mit hohem k, bis auf eine von k abhängige vernachlässigbare Funktion, gleich 1 ist, nennt man das Protokoll *vollständig*. Um zu zeigen, dass das Protokoll auch *zero-knowledge* ist, zeigt man, dass es einen Simulator S gibt, welcher P ersetzt. Wir verdeutlichen dies, indem wir uns zuerst den richtigen Ablauf des Protokolls ansehen:

- Peggy betritt die Höhle durch einen Eingang (rechts oder links), welchen nur sie kennt.
- Victor ruft Peggy zu, durch welchen Ausgang sie die Höhle verlassen soll.
- Peggy verlässt die Höhle durch den Ausgang, den Victor gerufen hat.

Der Simulator simuliert nun diesen Ablauf ohne Kenntnis von x, kann aber die Zufallswahl von V bestimmen. Er steckt quasi mit Victor unter einer Decke und die beiden haben sich abgesprochen.

- Der Simulator betritt die Höhle durch einen zufälligen Eingang r.
- Da er über Victors Zufallswahl bestimmen kann, kann er festlegen was Victor ruft. Dieser ruft dem Simulator nun zu, er solle die Höhle durch Ausgang r verlassen.
- Er verlässt die Höhle durch den Ausgang r, den Victor gerufen hat, wobei er nie durch die Tür gehen muss.

Für einen Außenstehenden ist es nun nicht möglich, die beiden Abläufe zu unterscheiden. Während des Ablaufs befindet er sich immer an Victors Position. Es sei nun P^* entweder Peggy oder der Simulator. Dann sieht der Ablauf beider Protokolle für den Außenstehenden wie folgt aus:

- P^* betritt die Höhle.
- Victor ruft in die Höhle, durch welchen Ausgang P^* die Höhle verlassen soll. Welchen Ausgang er ruft, erfolgt in jedem Durchlauf des Protokolls nicht nach einem Muster, sondern zufällig.

- P^* verlässt durch genau diesen Ausgang die Höhle.

Hiermit ist bewiesen, dass es für die Sicht eines Außenstehenden auf den Ablauf des Protokolls keinen Einfluss hat, ob P^* Peggy oder der Simulator ist. Der Außenstehende kann also auch nichts erfahren, außer dass eine Zufallszahl in die Höhle gerufen wird. Da der Außenstehende dieselbe Sicht wie Victor besitzt, heißt dies, dass auch dieser nichts über x erfahren kann. Das Protokoll ist also *zero-knowledge*. Unsere Vorgehensweise, dies mit einem Simulator zu zeigen, ist bei Zero-Knowledge-Protokollen üblich.

15.3.2 Das Schnorr-Protokoll

Wir wollen nun ein etwas mathematischeres Beispiel anführen. Das folgende Protokoll wurde von Claus-Peter Schnorr in [Sch91] beschrieben und wird deswegen Schnorr-Protokoll genannt. Hierbei kennt Peggy x für

$$a = g^x \mod p$$

Es handelt sich um das diskrete Logarithmus-Problem auf $\mathbb{Z}_p^*$ (siehe 11.3). Victor kennt a, g und p. Peggy möchte Victor nun beweisen, dass sie x kennt, ohne es zu verraten.

- Sie wählt eine Zufallszahl r, berechnet $t := g^r \mod p$ und schickt Victor t.
- Victor wählt eine Zufallszahl c und schickt diese an Peggy.
- Peggy berechnet $k := r + cx \mod p$ und schickt k an Victor.
- Victor akzeptiert, falls $g^k \stackrel{?}{=} t \cdot a^c$.

Verglichen mit dem Höhlen-Beispiel stellt r den Eingang dar, den Peggy am Anfang betritt. Victor sah dies von der äußeren Position, begab sich dann zum Eingang der Höhle, sodass Peggy ihre Wahl nicht mehr ändern konnte. Im Schnorr-Protokoll entspricht dieser Vorgang dem Wert t, den Peggy an Victor schickt. Ihr gewähltes r kann sie nicht mehr ändern, außer sie könnte einen diskreten Logarithmus berechnen. Die Zufallszahl c, die Victor wählt, entspricht dem Befehl von Victor, den er in die Höhle ruft. Peggys Antwort k beweist, dass sie x kennt, genauso wie das Verlassen der Höhle durch den richtigen Ausgang im Höhlen-Beispiel die Kenntnis des Worts x bewiesen hat (nach ausreichend vielen Durchläufen).

Hält sich Peggy an das Protokoll und kennt x, dann gilt:

$$g^k = g^{r+cx} = g^r g^{cx} = t \cdot (g^x)^c = t \cdot a^c$$

Damit ist das Protokoll vollständig.

Beh.: Kennt Peggy x nicht, kann sie nicht mit korrektem k Victor antworten. Das Protokoll ist also korrekt.

Bew.: Folgender Beweis stammt aus [Mau09, §2.1]. Wir lassen Peggy in zwei Durchläufen zweimal dasselbe t wählen und erhalten von Victor c und c'. Wir gehen davon aus, sie könnte mit k und k'

antworten, die von Victor akzeptiert werden. Es gilt also $g^k = t \cdot a^c$ und $g^{k'} = t \cdot a^{c'}$ und damit:

$$g^{k-k'} = \frac{g^k}{g^{k'}} = \frac{t \cdot a^c}{t \cdot a^{c'}} = a^{c-c'} = g^{x(c-c')}$$

Und damit

$$\begin{aligned} & k - k' \equiv x\,(c - c') \mod p \\ \Rightarrow \qquad & x \equiv \frac{k - k'}{c - c'} \mod p \end{aligned}$$

Das Inverse von $c - c'$ existiert, da p prim, und ist leicht zu berechnen. Peggy muss nun also, um geantwortet zu haben, x gekannt haben, sonst widerspräche dies dem diskreten Logarithmus-Problem. □

Knowledge-Extractor: Wir haben bei diesem Beweis eine spezielle Situation eingenommen, indem wir über Peggys Zufallswahl t bestimmt haben. Dies ist die Rolle des sogenannten *Knowledge-Extractors*, der auch bei anderen Protokollen verwendet wird. Dieser versucht x aus zwei korrekten Antworten zu extrahieren, was dann auf einen Widerspruch zu bekannten Annahmen wie dem diskreten Logarithmus-Problem zurückgeführt wird. Da wir davon ausgehen, dass z. B. das DLP nicht effizient lösbar ist, widerspräche unser Knowledge-Extractor diesem und wir müssten daher davon ausgehen, dass x bereits Peggy bekannt war. In anderen Worten: Da der Knowledge-Extractor ein effizientes Programm darstellt, solch eines aber nicht existieren darf für die Berechnung eines diskreten Logarithmus, muss das korrekte Antworten von Peggy auf Victors c ohne die Kenntnis von x in Exponentialzeit (also ineffizient) geschehen. Dieses Antworten würde also dem Lösen eines DLP entsprechen.

Zero-knowledge: Wir müssen nun nur noch zeigen, dass das Protokoll auch zero-knowledge ist. Der Simulator erzeugt dazu, ohne Kenntnis von x, folgende Übertragung (vgl. [Hoh07]):

- Der Simulator wählt Zufallszahlen $k, c \in \mathbb{Z}_p$ und berechnet $t := \frac{g^k}{a^c}$ und schickt t an Victor.
- Er lässt Victor mit c antworten, da er Kontrolle über dessen Zufallswahl hat.
- Der Simulator antwortet mit k.
- Victor akzeptiert, da $g^k = g^k \frac{a^c}{a^c} = \frac{g^k}{a^c} \cdot a^c = t \cdot a^c$.

Der Simulator kann $(a^c)^{-1}$ berechnen, da p prim. Für einen Außenstehenden sehen beide Übertragungen, bestehend aus (t, c, k) mit $g^k = t \cdot a^c$, identisch aus und sind gleichmäßig zufällig verteilt. Das vom Simulator erstellte t ist gleichmäßig verteilt, da g Erzeuger von $\mathbb{Z}_p$.

Wir haben nun ein weiteres Beispiel für ein Zero-Knowledge-Protokoll gesehen. Wieder wählt der Beweiser eine Zufallszahl, anschließend schickt (bzw. ruft) der Verifizierer eine zufällige Zahl, worauf der Beweiser mit einer Zahl (bzw. Aktion) reagiert, die beweist, dass er x kennt. Die vom Verifizierer erzeugte Zufallszahl nennt man *Challenge*[3] und wir bezeichnen sie deswegen mit c.

[3] Nicht zu verwechseln mit der Challenge beim Proof-of-work in 6.

15.3.3 Signature-of-knowledge

Es handelte sich bei den beiden bisherigen Beispielen um interaktive Protokolle, da Beweiser und Verifizierer miteinander kommunizieren müssen. Bei Zerocoin möchten wir nun anderen Teilnehmern des Netzwerks, die damit die Rolle des Verifizierers einnehmen, beweisen, dass wir beim Ausgeben einer Zerocoin auch eine geprägt haben. Dies soll in Zero-Knowledge geschehen, da wir nicht verraten wollen, um welche Zerocoin-Prägung es sich handelt. Die vorgestellten Zero-Knowledge-Protokolle sind für Zerocoin allerdings ungeeignet, da es zu aufwendig wäre, beim Ausgeben einer Zerocoin mit jedem Teilnehmer interaktiv zu kommunizieren. Zudem sind nicht immer alle Bitcoin-Teilnehmer online und müssten nachträglich mit allen Teilnehmern, die Zerocoins ausgegeben haben, Kontakt aufnehmen, um das interaktive Protokoll durchzuführen.

Es ist möglich ein interaktives Zero-Knowledge-Protokoll umzuwandeln in ein nicht-interaktives *Signatur-Schema*, das aber dasselbe leistet. Bei diesem muss der Verifizierer nur noch den letzten Schritt durchführen: das Überprüfen und Akzeptieren des Beweises bzw. der Signatur. Dies kann auch wesentlich später als das Erstellen der Signatur erfolgen.

Die Rolle des Verifizierers im Protokollablauf besteht darin, die Challenge c zu erzeugen. Dabei handelt es sich um eine Zufallszahl, die verhindern soll, dass der Beweiser schummeln kann. Dieser muss sich nämlich vorher festlegen. Im Höhlen-Beispiel war dies der Eingang, den er wählt, beim Schnorr-Protokoll die Zahl $t = g^r$. Wichtig ist, dass die Challenge zufällig ist, damit der Beweiser diese nicht im Voraus weiß. Wir erreichen dies, indem wir c durch einen Hashwert ersetzen, den der Beweiser selber erstellt. Dabei sollte er alle öffentlichen Werte der Signatur hashen, damit er keine mehr nachträglich austauschen kann. Aufgrund der Eigenschaften der Hashfunktion (siehe 3) ist damit c auch pseudo-zufällig. Da für ein bestimmtes $c = \mathcal{H}(d)$ ein d schwer zu finden ist, kann der Beweiser c nicht so bestimmen, dass er die Signatur fälschen kann.

Ein Problem hierbei ist, dass einige Werte, die zum Berechnen des Hashwertes notwendig sind, selber von c abhängen. Dies wird gelöst, indem ein Wert s vom Beweiser erstellt wird, der nur mittels des geheimen Wertes x erzeugt werden kann. Beim Verifizieren können von c abhängige Werte einfach mit dem vorläufigem Hashwert des Beweisers errechnet werden. Wie dies konkret realisiert wird, zeigen wir in den folgenden Kapiteln.

Das endgültige Tupel, bestehend aus s und dem Hashwert c, nennt man eine *Signature-of-knowledge* des Wertes x für eine Aussage, die dieser erfüllen soll.

[FS87, Cam98]

15.3.4 Eigenschaften des Zero-Knowledge-Protokolls

Wir wollen nun die Eigenschaften definieren, die ein Zero-Knowledge-Protokoll bzw. eine Signature-of-knowledge, wie wir sie brauchen, erfüllen muss (vgl. [Mau09]):

1. Vollständigkeit: V akzeptiert den Beweis nur, wenn P wirklich x kennt.

2. Korrektheit: Mittels des *Knowledge-Extractors*, welcher über die Zufallswahl von P bestimmen kann, können wir aus zwei Signaturen, die V akzeptiert, ein gültiges x extrahieren. Dies bedeutet, wenn V eine Signatur von P akzeptiert, kennt P auch x.

3. Zero-Knowledge: Es existiert ein Simulator S, welcher ein „Gespräch“ zwischen P und V nachstellen kann, welches von einem Außenstehenden nicht von einem echten Gespräch unterscheidbar ist.

Ist es nur möglich einen Simulator für Verifizierer, die sich ehrlich verhalten, zu finden, so nennt man dies *honest-verifier zero-knowledge*. Da bei der Signature-of-knowledge der Verifizierer nicht mehr die Challenge erstellt, sondern nur noch überprüft, reicht uns diese Art der Zero-Knowledge aus. Nur beim Erstellen der Challenge wäre es einem Verifizierer nämlich möglich zu schummeln und keine wirklichen Zufallszahlen zu benutzen. Dies wird jedoch durch Benutzen einer Hashfunktion vermieden.

15.3.5 Doppelt diskreter Logarithmus

Wir wollen nun ein Beispiel aus [Cam98, §5.3] für eine Signature-of-knowledge zeigen, welche beweist, dass man den doppelt diskreten Logarithmus kennt. Sei $G = \langle g \rangle$ eine zyklische Gruppe mit Ordnung $n = pq$ und $a \in \mathbb{Z}_n^*$. Dann sei der doppelt diskrete Logarithmus von $y \in G$ zu den Basen g und a eine Zahl x für die gilt:

$$y = g^{a^x}$$

Es seien λ, ϵ Sicherheitskonstanten (siehe [Cam98, S. 92]), $\mathcal{H} : \{0,1\}^* \to \{0,1\}^l$ eine Hashfunktion und $k \in \mathbb{N}$, sodass $k \leq l$. Der Beweiser wählt zufällige $r_i \in \left\{0, \ldots, 2^{\epsilon\lambda} - 1\right\}$ mit $i = 1, \ldots, k$ und berechnet

$$t_i := g^{(a^{r_i})}$$

sowie den Hashwert

$$c := \mathcal{H}\left(y||g||a||t_1|| \ldots ||t_k\right)$$

und davon ausgehend

$$s_i := \begin{cases} r_i & \text{falls } c\,[i] = 0 \\ r_i - x & \text{sonst} \end{cases}$$

Dann ist das Tupel $(c, s_1, \ldots, s_k)$ eine Signature-of-knowledge des doppelt diskreten Logarithmus x von y zu den Basen g und a. Um zu überprüfen, ob die Signatur gültig ist, berechnet der Verifizierer

$$\bar{t}_i = \begin{cases} g^{(a^{s_i})} & \text{falls } c\,[i] = 0 \\ y^{(a^{s_i})} & \text{sonst} \end{cases}$$

sowie

$$\bar{c} = \mathcal{H}\left(y||g||a||\bar{t}_1|| \ldots ||\bar{t}_l\right)$$

und überprüft, ob $c \stackrel{?}{=} \bar{c}$ gilt.

Vollständigkeit: Falls $c\,[i] = 0$ und damit $s_i = r_i$, gilt:

$$\bar{t}_i = g^{(a^{s_i})} = g^{(a^{r_i})} = t_i$$

Falls $c[i] = 1$ und damit $s_i = r_i - x$, gilt:

$$\begin{aligned}\bar{t}_i &= y^{(a^{s_i})} \\ &= y^{\left(a^{r_i - x}\right)} \\ &= y^{a^{r_i} a^{-x}} \\ &= g^{a^x a^{r_i} a^{-x}} \\ &= g^{a^{r_i}} = t_i\end{aligned}$$

Da somit $\bar{t}_i = t_i$ für alle $i = 1, \ldots, k$ gilt $\bar{c} = c$. □

Korrektheit: Der Knowledge-Extractor wählt die Zufallszahlen für den Beweiser und lässt ihn solange antworten, bis er zwei Signaturen (c, s) und (c', s') mit $c \neq c'$ erhalten hat. Aufgrund der Kollisionsfreiheit der verwendeten Hashfunktion geschieht dies sofort, außer mit vernachlässigbarer Wahrscheinlichkeit. Ohne Beschränkung der Allgemeinheit sei nun $c[j] = 0$ und $c'[j] = 1$ für ein $j \in \{1, \ldots, k\}$. Dann ist nach [Cam98, S. 94]:

$$t_j = g^{(a^{s_j})} = y^{\left(a^{s'_j}\right)} = g^{a^x \left(a^{s'_j}\right)}$$

$$\begin{aligned}\Rightarrow \quad & a^{s_j} \equiv a^x a^{s'_j} \mod n \\ \Rightarrow \quad & a^x \equiv a^{s_j - s'_j} \mod n \\ \Rightarrow \quad & x = s_j - s'_j\end{aligned}$$

Da wir auch davon ausgehen, dass das Berechnen eines doppelt diskreten Logarithmus nicht effizient möglich ist, beweist dies die Korrektheit. □

Honest-verifier zero-knowledge: Wir konstruieren einen Simulator nach [Cam98, S. 94f]. Dieser wählt c zufällig aus dem Wertebereich von $\mathcal{H}$ und alle $r_i = s_i$ aus $\left\{0, \ldots, 2^{\epsilon\lambda} - 1\right\}$. Mit diesen berechnet er:

$$t_i := \begin{cases} g^{(a^{r_i})} & \text{falls } c[i] = 0 \\ y^{(a^{r_i})} & \text{sonst} \end{cases}$$

für $i = 1, \ldots, k$. Um zu beweisen, dass diese Werte von einem außenstehendem Beobachter statistisch ununterscheidbar von denen des richtigen Protokollablaufs sind, reicht es die beiden Wahrscheinlichkeitsverteilungen $\mathbb{P}(R_1 = r_1, \ldots, R_k = r_k)$ und $\mathbb{P}(S_1 = s_1, \ldots, S_k = s_k)$ zu betrachten. Erstere steht für die Verteilung der r_i wie sie der Simulator wählt und letztere für die s_i des ehrlichen Beweisers. Es gilt:

$$\mathbb{P}(R_1 = r_1, \ldots, R_k = r_k) = \prod_{i=1}^{k} \mathbb{P}(R_i = r_i)$$

$$= \prod_{i=1}^{k} 2^{-\epsilon\lambda}$$
$$= 2^{-k\epsilon\lambda}$$

Wenn der Beweiser die r_i gleichverteilt zufällig in $\left\{0, \ldots, 2^{\lambda\epsilon} - 1\right\}$ wählt und x beliebig aus $\left\{0, \ldots, 2^{\lambda} - 1\right\}$, dann müssen wir den Fall bedenken, dass $c\,[i] = 1$, da dann

$$s_i = r_i - x$$

Betrachten wir zunächst den Fall, dass $x > r_i$, also $-\left(2^{\lambda} - 1\right) \leq s < 0$. Es gilt

$$\mathbb{P}\left(-\left(2^{\lambda} - 1\right) \leq S_i < 0\right) \leq \frac{2^{\lambda} - 1}{2^{\lambda\epsilon}} < \frac{2^{\lambda}}{2^{\lambda\epsilon}}$$

da wir $2^{\lambda} - 1$ Möglichkeiten für r_i aus $2^{\lambda\epsilon}$ Möglichkeiten insgesamt wählen. Entsprechend für genau ein s_i gilt:

$$\mathbb{P}\left(-\left(2^{\lambda} - 1\right) \leq S_i = s_i < 0\right) < 2^{-\lambda\epsilon}$$

Zusammengefasst für alle Fälle gilt:

$$\mathbb{P}\left(S_i = s_i\right) \begin{cases} = 0 & \text{falls } s_i < -\left(2^{\lambda} - 1\right) \\ < 2^{-\epsilon\lambda} & \text{falls } -\left(2^{\lambda} - 1\right) \leq s_i < 0 \\ = 2^{-\epsilon\lambda} & \text{falls } 0 \leq s_i \leq 2^{\epsilon\lambda} - 2^{\lambda} \\ \leq 2^{-\epsilon\lambda} & \text{falls } 2^{\epsilon\lambda} - 2^{\lambda} < s_i \leq \left(2^{\epsilon\lambda} - 1\right) \\ = 0 & \text{falls } \left(2^{\epsilon\lambda} - 1\right) < s_i \end{cases}$$

Dies gilt für alle Verteilungen von $c\,[i]$ über $\{0, 1\}$. Gleicherweise ist die Wahrscheinlichkeit

$$\mathbb{P}\left(S_1 = s_1, \ldots, S_k = s_k\right) \begin{cases} = 0 & \text{falls } s_i < -\left(2^{\lambda} - 1\right) \vee s_i > 2^{\epsilon\lambda} - 1 \text{ für ein } i \\ = 2^{-k\epsilon\lambda} & \text{falls } 0 \leq s_i \leq 2^{\lambda\epsilon} - 2^{\lambda} \quad \forall i \\ \leq 2^{-k\epsilon\lambda} & \text{sonst} \end{cases}$$

Betrachten wir nun:

$$P := \sum_{\alpha \in \mathbb{Z}^k} \left| \underbrace{\mathbb{P}\left(S_1 = \alpha_1, \ldots S_k = \alpha_k\right)}_{P_S} - \underbrace{\mathbb{P}\left(R_1 = \alpha_1, \ldots R_k = \alpha_k\right)}_{P_R} \right|$$

Hierbei brauchen wir nur α zu betrachten, für $-\left(2^{\lambda} - 1\right) \leq \alpha_i < 0$ oder $2^{\lambda\epsilon} - 2^{\lambda} < \alpha_i \leq 2^{\epsilon\lambda} - 1$ für ein i. Insgesamt gibt es $\left(2^{\lambda\epsilon} + 2^{\lambda} - 1\right)^k$ Möglichkeiten für α bei denen $P_S \neq 0 \ \wedge \ P_R \neq 0$. Da gelten muss

$$\sum_{\alpha \in \mathbb{Z}^k} \mathbb{P}\left(S_1 = \alpha_1, \ldots S_k = \alpha_k\right) = 1$$

können wir diese Anzahl weiter begrenzen auf $2^{k\lambda\epsilon}$. Für α mit $0 \leq \alpha_i < 2^{\lambda\epsilon} - 2^{\lambda} + 1 \; \forall i$ gilt $P_S = P_R = 2^{-k\epsilon\lambda}$. Das heißt, für $2^{k\lambda\epsilon} - \left(2^{\lambda\epsilon} - 2^{\lambda} + 1\right)^k$ Möglichkeiten gilt $|P_S - P_R| \leq 2^{-\lambda\epsilon}$. Daraus folgt:

$$\begin{aligned} P &\leq \frac{2^{k\lambda\epsilon} - \left(2^{\lambda\epsilon} - 2^{\lambda} + 1\right)^k}{2^{k\lambda\epsilon}} \\ &= \frac{2^{k\lambda\epsilon} - 2^{k\lambda\epsilon}\left(1 + \frac{1-2^{\lambda}}{2^{\lambda\epsilon}}\right)^k}{2^{k\lambda\epsilon}} \\ &\leq 1 - \left(1 + k\frac{1-2^{\lambda}}{2^{\lambda\epsilon}}\right) \\ &= k\frac{2^{\lambda}-1}{2^{\lambda\epsilon}} \\ &< \frac{k}{\left(2^{\lambda}\right)^{\epsilon-1}} \end{aligned}$$

Das heißt, für ausreichend große Sicherheitsparameter λ und ϵ sind die beiden Wahrscheinlichkeitsverteilungen praktisch nicht unterscheidbar. Die Signatur ist damit honest-verifier zero-knowledge. □

15.4 Signature-of-knowledge der Seriennummer

Nachdem wir die Grundlagen erläutert haben, wollen wir im Folgenden nun die beiden von Zerocoin tatsächlich verwendeten Signatures-of-knowledge vorstellen. Zuerst sei hier die der Seriennummer genannt. Dabei soll bewiesen werden, dass man ein r zu dem offengelegten S kennt, sodass gilt $\mathbf{c} = \mathbf{g}^S\mathbf{h}^r$. Dazu werden $g, h \in \mathbb{Z}_{\hat{n}}^*$ neu gewählt mit $\langle g \rangle = \langle h \rangle$ und $\operatorname{ord}(g) = p$. Dabei ist $\hat{n}$ ein neues RSA-Modul und p das aus 15.2. Dann sei

$$y_1 := g^{\mathbf{g}^x\mathbf{h}^r}h^w$$

ein neues Commitment mit Zufallszahl w, wobei y_1 nicht prim sein muss. Der Beweiser generiert die Zufallszahlen $r_1, \ldots, r_k$ und $v_1, \ldots, v_k$ mit $k \in \mathbb{N}$ und berechnet

$$t_i = g^{\mathbf{g}^x\mathbf{h}^{r_i}}h^{v_i}, \qquad i \in \{1, \ldots, k\}$$

sowie

$$\begin{aligned} c &= \mathcal{H}\left(y_1||\mathbf{g}||\mathbf{h}||g||h||x||t_1||\ldots||t_k\right) \\ s_i &= \begin{cases} r_i & \text{falls } c[i] = 0 \\ r_i - r & \text{sonst} \end{cases} \\ \hat{s}_i &= \begin{cases} v_i & \text{falls } c[i] = 0 \\ v_i - w\mathbf{h}^{r_i - r} & \text{sonst} \end{cases} \end{aligned}$$

mit $i \in \{1, \ldots, k\}$. Das Tupel $(c, s_1, \ldots, s_k, s'_1, \ldots, s'_k)$ stellt die Signature-of-knowledge dar. Zum Überprüfen berechnen die Miner

$$\overline{c} = \mathcal{H}\left(y_1 || \mathbf{g} || \mathbf{h} || g || h || x || \overline{t_1} || \ldots || \overline{t_k}\right)$$

mit

$$\overline{t_i} = \begin{cases} g^{\mathbf{g}^x \mathbf{h}^{s_i}} h^{\hat{s}_i} & \text{falls } c[i] = 0 \\ y_1^{\mathbf{h}^{s_i}} h^{\hat{s}_i} & \text{sonst} \end{cases}$$

und überprüfen ob $c \stackrel{?}{=} \overline{c}$.

[Lib13, MGGR13]

Vollständigkeit: Wenn $c[i] = 0$:

$$\begin{aligned} \overline{t_i} &= g^{\mathbf{g}^x \mathbf{h}^{s_i}} h^{\hat{s}_i} \\ &= g^{\mathbf{g}^x \mathbf{h}^{r_i}} h^{v_i} \end{aligned}$$

Wenn $c[i] = 1$:

$$\begin{aligned} \overline{t_i} &= y_1^{\mathbf{h}^{s_i}} h^{\hat{s}_i} \\ &= y_1^{\mathbf{h}^{r_i - r}} h^{v_i - w\mathbf{h}^{r_i - r}} \\ &= \left(g^{\mathbf{g}^x \mathbf{h}^r} h^w\right)^{\mathbf{h}^{r_i - r}} h^{v_i - w\mathbf{h}^{r_i - r}} \\ &= g^{\mathbf{g}^x \mathbf{h}^z \mathbf{h}^{r_i - r}} h^{w\mathbf{h}^{r_i - r}} h^{v_i - w\mathbf{h}^{r_i - r}} \\ &= g^{\mathbf{g}^x \mathbf{h}^{r_i}} h^{w\mathbf{h}^{r_i - r} + v_i - w\mathbf{h}^{r_i - r}} \\ &= g^{\mathbf{g}^x \mathbf{h}^{r_i}} h^{v_i} \end{aligned}$$

□

Korrektheit: Der Knowledge-Extractor erhält wie in 15.3.5 zwei Signaturen $(c, s, \hat{s})$ und $(c', s', \hat{s}')$. Ohne Beschränkung der Allgemeinheit sei $c[j] = 0$ und $c'[j] = 1$ für ein $j \in \{1, \ldots, k\}$. Dann ist

$$\begin{aligned} t_j &= g^{\mathbf{g}^x \mathbf{h}^{s_j}} h^{\hat{s}_j} && (\text{da } c[j] = 0) \\ &= y_1^{\mathbf{h}^{s'_j}} h^{\hat{s}'_i} && (\text{da } c'[j] = 1) \\ &= g^{\mathbf{g}^x \mathbf{h}^r \mathbf{h}^{s'_j}} h^{\mathbf{h}^{s'_j} w} h^{\hat{s}'_j} \\ &= g^{\mathbf{g}^x \mathbf{h}^{r + s'_j}} h^{\mathbf{h}^{s'_j} w + \hat{s}'_j} \end{aligned}$$

$$\begin{aligned} \Rightarrow \quad & r + s'_j \equiv s_j && \mod q \\ \Rightarrow \quad & r \equiv s_j - s'_j && \mod q \end{aligned}$$

sowie

$$\begin{aligned} \Rightarrow \qquad & \hat{s}_j \equiv \mathbf{h}^{s'_j} w + \hat{s}'_j \mod p \\ \Rightarrow \qquad & w = \frac{\hat{s}_j - \hat{s}'_j}{\mathbf{h}^{s'_j}} \mod p \end{aligned}$$

Das Inverse von $\mathbf{h}^{s'_j}$ exisitert, da p prim. □

Honest-verifier zero-knowledge: Der Simulator wählt c zufällig und alle r_i und v_i wie im Protokoll. Dann setzt er $s_i = r_i$ und $\hat{s}_i = v_i$. Nun berechnet er:

$$t_i := \begin{cases} g^{\mathbf{g}^x \mathbf{h}^{s_i}} h^{\hat{s}_i} & \text{wenn } c[i] = 0 \\ y_1^{\mathbf{h}^{s_i}} h^{\hat{s}_i} & \text{sonst} \end{cases}$$

Der Beweis, dass die Wahrscheinlichkeitsverteilung der Variablen des Simulators der eines echten Beweisers gleicht, ist analog zu dem in 15.3.5.

15.5 Akkumulator

Um mit Zero-Knowledge zu beweisen, dass man eine von den bisherigen Coins kennt, muss man in seinem Beweis die Commitments aller Coins verwenden. Würde man wirklich mit den Daten aller Coins rechnen, käme es schnell zu einem Performanceproblem. Daher benutzt man stattdessen eine Art Hashwert der Coins. Statt einer Hashfunktion wird allerdings ein *Akkumulator* $\mathcal{A}$ verwendet. Es sei:

$$\begin{aligned} \mathcal{A} : \{0,1\}^* \times \{0,1\}^* &\to \{0,1\}^k \\ (x, y) &\mapsto \mathcal{A}(x, y) \end{aligned}$$

Diese Funktion muss folgende Bedingungen erfüllen (ähnlich zur Hashfunktion, siehe 3):

1. $\mathcal{A}(x, y)$ lässt keine Rückschlüsse auf x und y zu. (Einwegfunktion)

2. $(x_1, y_1) \neq (x_2, y_2)$, für die gilt $\mathcal{A}(x_1, y_1) = \mathcal{A}(x_2, y_2)$, sind sehr schwer zu finden. (Kollisionsfreiheit)

3. $\mathcal{A}(x, y)$ ändert sich pseudo-zufällig, auch wenn nur geringe Änderungen an x oder y vorgenommen werden. (Lawineneffekt)

Zusätzlich soll gelten:

4. $\mathcal{A}(\mathcal{A}(x, y_1), y_2) = \mathcal{A}(\mathcal{A}(x, y_2), y_1) \qquad \forall x, y_1, y_2 \qquad$ (quasi-kommutativ)

Dies schließt das Verwenden einer Hashfunktion aus.

[BM94]

15.5.1 Witness

Es sei $v \in \{0,1\}^k$ und $x \in \{0,1\}^*$. Ein Wert $w \in \{0,1\}^k$ wird *Witness* für x genannt, wenn gilt:

$$\mathcal{A}(w, x) = v$$

[CL02]

15.5.2 Konstruktion des Akkumulators

Im Folgenden zeigen wir, wie der von Zerocoin verwendete Akkumulator aufgebaut ist. Er orientiert sich an der Konstruktion in [CL02, §3.2]. Es sei $n = pq$ ein RSA-Modul (siehe 11.4) $n = pq$ mit $p = 2p' + 1$ und $q = q' + 1$, wobei p, p', q, q' prim. Nun wird ein u bestimmt, für das aus Sicherheitsgründen gilt:

$$u \equiv x^2 \mod n \quad \text{(für ein beliebiges } x \in \mathbb{Z}_n^*) \qquad \wedge \qquad u \neq 1$$

Die Menge der Zahlen, die sich darstellen lassen als $x^2 \mod n$ für ein beliebiges $x \in \mathbb{Z}_n^*$, nennen wir im Folgenden QR_n. Der Wert u bildet den *Startwert* für unseren Akkumulator. Das RSA-Modul n wird am Anfang festgelegt und bei jeder Berechnung verwendet. Nun können wir den Akkumulator definieren als:

$$\begin{aligned} \mathcal{A} : \mathbb{Z}_n^* \times \mathbf{C} &\to \mathbb{Z}_n^* \\ (u, \mathbf{c}) &\mapsto u^{\mathbf{c}} \mod n \end{aligned}$$

Dabei ist $\mathbf{C}$ die Menge aller Primzahlen aus $[A, B]$. A, B können frei gewählt werden, solange $2 < A \wedge B < A^2$. Es ist leicht zu sehen, dass der Akkumulator quasi-kommutativ ist:

$$\mathcal{A}(\mathcal{A}(u, \mathbf{c}_1), \mathbf{c}_2) = (u^{\mathbf{c}_1})^{\mathbf{c}_2} = u^{\mathbf{c_1} \cdot \mathbf{c_2}} = (u^{\mathbf{c}_2})^{\mathbf{c}_1} = \mathcal{A}(\mathcal{A}(u, \mathbf{c}_2), \mathbf{c}_1)$$

Die Eigenschaften Einwegfunktion und Lawineneffekt gehen aus der Konstruktion hervor (siehe [CL02, Theorem 3]). Um die Kollionsfreiheit zu zeigen, benötigen wir vorher folgende Annahme:

Starke RSA-Annahme: Für jeden probabilistischen polynomiellen Algorithmus A ist die Wahrscheinlichkeit, dass er bei Eingabe von n und $u \in \mathbb{Z}_n^*$ zwei Zahlen x und e mit

$$x^e \equiv u \mod n$$

berechnen kann, vernachlässigbar. [BGZ08]

Beh.: Gilt die starke RSA-Annahme, ist der Akkumulator aus 15.5.2 kollisionsfrei.

Bew.: Dieser Beweis stammt aus [BP97, Theorem 5]. Wir nehmen an, ein Angreifer findet u' und y' prim, sodass

$$u'^{y'} = u^y \mod n$$

Dann kann er auch die starke RSA-Annahme knacken: Er berechnet $a, b \in \mathbb{Z}$ mit $ay + by' = 1$ mit dem erweiterten euklidischen Algorithmus (dies ist möglich, da y, y' prim und damit $\text{ggT}(y, y') = 1$). Es sei $x := u'^a u^b$. Dann gilt:

$$\begin{aligned} x^{y'} &\equiv u'^{ay'} u^{by'} \\ &\equiv \left(u'^{y'}\right)^a u^{by'} \\ &\equiv u^{ay+by'} \\ &\equiv u \mod n \end{aligned}$$

Und damit wäre die starke RSA-Annahme für $x^{y'} = u \mod n$ gelöst. ↯

15.5.3 Verwendung in Zerocoin

Da alle Zerocoin-Prägungen ein Commitment $\mathbf{c}$ beinhalten und es sich dabei um eine Primzahl handelt, kann man nun beim Ausgeben einer Zerocoin alle Commitments akkumulieren. Diesen Akkumulatorwert v veröffentlicht man in der speziellen Transaktion zum Ausgeben einer Zerocoin. Da der Akkumulator quasi-kommutativ ist, können andere Teilnehmer des Bitcoin-Netzwerks diesen Wert nachrechnen, ohne dieselbe Reihenfolge zu benutzen. Um die Rechenzeit zum Erstellen von v zu verringern, ist es daher außerdem möglich, dass Miner bereits in ihrem Block Akkumulator-Zwischenwerte veröffentlichen, die alle bisherigen Zerocoin-Commitments beinhalten.

Nun möchten wir beweisen, dass ein Commitment akkumuliert wurde in v, ohne es aber zu verraten. Die Signature-of-knowledge, die dies ermöglicht, möchten wir im nächsten Kapitel vorstellen.

15.6 Signature-of-knowledge des Akkumulators

Zunächst erzeugen wir ein neues Commitment für das alte Commitment $\mathbf{c}$:

$$\mathfrak{C} = \mathfrak{g}^{\mathbf{c}} \mathfrak{h}^{\varphi}$$

Hierbei sind $\mathfrak{g}, \mathfrak{h}$ neue Erzeuger mit $\langle \mathfrak{g} \rangle = \langle \mathfrak{h} \rangle \subseteq \mathbb{Z}_n^*$ und $\varphi \in \mathbb{Z}_q$ eine Zufallszahl. Dabei gilt für $n = pq$, dass p, p', q, q' Primzahlen mit $p = 2p' + 1$ und $q = 2q' + q$, sowie $B2^{k'+k''+2} < A^2 - 1 < \frac{q}{2}$. Dabei ist k' die Bitlänge der in der folgenden Signatur verwendeten Hashfunktion und k'' ein Sicherheitsparameter (vgl. [CL02, §3.3]). Zusätzlich werden die Erzeuger $g, h \in QR_n$ für diese Signatur nach demselben Prinzip neu gewählt und entsprechen nicht denen aus 15.4.

Die Signature-of-knowledge beweist, ähnlich wie in [CL02, §3.3], dass man ein Witness u kennt, sodass:

$$u^{\mathbf{c}} = v \mod n$$

Hierzu berechnet (vgl. [Lib13, AccumulatorProofOfKnowledge.cpp]) der Beweiser die Zufallszahlen η, ε, ζ aus $\mathbb{Z}_{\lfloor \frac{n}{4} \rfloor}$, sowie:

$$C_u = uh^{\varepsilon}$$

Zum Erstellen der Signature-of-knowledge wählt der Beweiser folgende weitere Zufallszahlen:

$$r_\alpha \in \left(-B2^{k'+k''}, \ldots, B2^{k'+k''}\right)$$

$$r_\gamma, r_\varphi, r_\psi, r_\sigma, r_\xi \in \mathbb{Z}_q$$

$$\begin{aligned} r_\varepsilon, r_\eta, r_\zeta &\in \left(-\left\lfloor\frac{n}{4}\right\rfloor 2^{k'+k''}, \ldots, \left\lfloor\frac{n}{4}\right\rfloor 2^{k'+k''}\right) \\ r_\beta, r_\delta &\in \left(-\left\lfloor\frac{n}{4}\right\rfloor q2^{k'+k''}, \ldots, \left\lfloor\frac{n}{4}\right\rfloor q2^{k'+k''}\right) \end{aligned}$$

Nun möchte er folgende Tatsachen in Zero-Knowledge beweisen:

$$\mathfrak{C} = \mathfrak{g}^{\mathbf{c}}\mathfrak{h}^{\varphi} \tag{1}$$

$$\mathfrak{g} = \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{r_\gamma} \mathfrak{h}^{r_\psi} \tag{2}$$

$$\mathfrak{g} = (\mathfrak{g}\mathfrak{C})^{r_\sigma}\, \mathfrak{h}^{r_\xi} \tag{3}$$

$$C_{\mathbf{c}} = h^{\mathbf{c}} g^{\eta} \tag{4}$$

$$C_r = h^{\varepsilon} g^{\zeta} \tag{5}$$

$$v = C_u^{\mathbf{c}} \left(\frac{1}{h}\right)^{r_\beta} \tag{6}$$

$$1 = C_r^{\mathbf{c}} \left(\frac{1}{h}\right)^{r_\delta} \left(\frac{1}{g}\right)^{r_\beta} \tag{7}$$

$$\mathbf{c} \in \left[-B2^{k'+k''+2}, B2^{k'+k''+2}\right]$$

Dazu berechnet er:

$$\begin{aligned} \mathfrak{t}_1 &:= \mathfrak{g}^{r_\alpha}\mathfrak{h}^{r_\varphi} \\ \mathfrak{t}_2 &:= \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{r_\gamma} \mathfrak{h}^{r_\psi} \\ \mathfrak{t}_3 &:= (\mathfrak{g}\mathfrak{C})^{r_\sigma}\, \mathfrak{h}^{r_\xi} \\ t_1 &:= h^{r_\varepsilon} g^{r_\zeta} \\ t_2 &:= h^{r_\alpha} g^{r_\eta} \\ t_3 &:= C_u^{r_\alpha} \left(\frac{1}{h}\right)^{r_\beta} \\ t_4 &:= C_r^{r_\alpha} \left(\frac{1}{h}\right)^{r_\delta} \left(\frac{1}{g}\right)^{r_\beta} \end{aligned}$$

$$c = \mathcal{H}\left(\mathfrak{g}||\mathfrak{h}||g||h||\mathfrak{C}||C_{\mathbf{c}}||C_u||C_r||\mathfrak{t}_1||\mathfrak{t}_2||\mathfrak{t}_3||t_1||t_2||t_3||t_4\right)$$

$$\begin{aligned} s_\alpha &:= r_\alpha - c\mathbf{c} \\ s_\eta &:= r_\eta - c\eta \end{aligned}$$

$$
\begin{aligned}
s_\varphi &:= r_\varphi - c\varphi \mod q \\
s_\beta &:= r_\beta - c\varepsilon\mathbf{c} \\
s_\varepsilon &:= r_\varepsilon - c\varepsilon \\
s_\gamma &:= r_\gamma - c\left(\mathbf{c}-1\right)^{-1} \mod q \\
s_\zeta &:= r_\zeta - c\zeta \\
s_\delta &:= r_\delta - c\zeta\mathbf{c} \\
s_\psi &:= r_\psi + c\varphi\left(\mathbf{c}-1\right)^{-1} \mod q \\
s_\sigma &:= r_\sigma - c\left(\mathbf{c}+1\right)^{-1} \mod q \\
s_\xi &:= r_\xi + c\varphi\left(\mathbf{c}+1\right)^{-1} \mod q
\end{aligned}
$$

Das Tupel $(C_\mathbf{c}, C_u, C_r, \mathfrak{t}_1, \mathfrak{t}_2, \mathfrak{t}_3, t_1, t_2, t_3, t_4, c, s_\alpha, s_\eta, s_\varphi, s_\beta, s_\varepsilon, s_\gamma, s_\zeta, s_\delta, s_\psi, s_\sigma, s_\xi)$ bildet die Signature-of-knowledge. Zum Überprüfen der Signatur müssen folgende Werte verglichen werden:

$$
\begin{aligned}
\mathfrak{t}_1 &\stackrel{?}{=} \mathfrak{C}^c\mathfrak{g}^{s_\alpha}\mathfrak{h}^{s_\varphi} \\
\mathfrak{t}_2 &\stackrel{?}{=} \mathfrak{g}^c\left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{s_\gamma}\mathfrak{h}^{s_\psi} \\
\mathfrak{t}_2 &\stackrel{?}{=} \mathfrak{g}^c\left(\mathfrak{g}\mathfrak{C}\right)^{s_\sigma}\mathfrak{h}^{s_\xi} \\
t_1 &\stackrel{?}{=} C_r^c h^{s_\varepsilon} g^{s_\zeta} \\
t_2 &\stackrel{?}{=} C_\mathbf{c}^c h^{s_\alpha} g^{s_\eta} \\
t_3 &\stackrel{?}{=} v^c C_u^{s_\alpha}\left(\frac{1}{h}\right)^{s_\beta} \\
t_4 &\stackrel{?}{=} C_r^{s_\alpha}\left(\frac{1}{h}\right)^{s_\delta}\left(\frac{1}{g}\right)^{s_\beta} \\
s_\alpha &\stackrel{?}{\in} \left[-B2^{k'+k''+1}, B2^{k'+k''+1}\right]
\end{aligned}
$$

Sowie:

$$
c \stackrel{?}{=} \mathcal{H}\left(\mathfrak{g}||\mathfrak{h}||g||h||\mathfrak{C}||C_\mathbf{c}||C_u||C_r||\hat{t}_1||\hat{t}_2||\hat{t}_3||t_1||t_2||t_3||t_4\right)
$$

Vollständigkeit: Wir setzen dazu in die zu überprüfenden Gleichungen ein:

$$
\begin{aligned}
\mathfrak{C}^c\mathfrak{g}^{s_\alpha}\mathfrak{h}^{s_\varphi} &= \mathfrak{g}^{c\mathbf{c}}\mathfrak{h}^{c\varphi}\mathfrak{g}^{r_\alpha - c\mathbf{c}}\mathfrak{h}^{r_\varphi - c\varphi} \\
&= \mathfrak{g}^{c\mathbf{c}-c\mathbf{c}}\mathfrak{h}^{c\varphi-c\varphi}\mathfrak{g}^{r_\alpha}\mathfrak{h}^{r_\varphi} \\
&= \mathfrak{g}^{r_\alpha}\mathfrak{h}^{r_\varphi} = \mathfrak{t}_1 \\
\mathfrak{g}^c\left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{s_\gamma}\mathfrak{h}^{s_\psi} &= \mathfrak{g}^c\left(\frac{\mathfrak{g}^\mathbf{c}\mathfrak{h}^\varphi}{\mathfrak{g}}\right)^{r_\gamma - \frac{c}{\mathbf{c}-1}}\mathfrak{h}^{r_\psi + \frac{c\varphi}{\mathbf{c}-1}} \\
&= \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{r_\gamma}\mathfrak{g}^c\mathfrak{g}^{(\mathbf{c}-1)\frac{-c}{\mathbf{c}-1}}\mathfrak{h}^{\varphi\frac{-c}{\mathbf{c}-1}}\mathfrak{h}^{r_\psi+\frac{c\varphi}{\mathbf{c}-1}} \\
&= \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{r_\gamma}\mathfrak{h}^{r_\psi} = \mathfrak{t}_2
\end{aligned}
$$

$$
\begin{aligned}
\mathfrak{g}^c\left(\mathfrak{g}\mathfrak{C}\right)^{s_\sigma}\mathfrak{h}^{s_\xi} &= \mathfrak{g}^c\left(\mathfrak{g}\mathfrak{g}^{\mathbf{c}}\mathfrak{h}^{\varphi}\right)^{r_\sigma-\frac{c}{\mathbf{c}+1}}\mathfrak{h}^{r_\xi+\frac{c\varphi}{\mathbf{c}+1}}\\
&= \left(\mathfrak{g}\mathfrak{C}\right)^{r_\sigma}\mathfrak{g}^c\mathfrak{g}^{(\mathbf{c}+1)\frac{c}{\mathbf{c}+1}}\mathfrak{h}^{\varphi\frac{-c}{\mathbf{c}+1}}\mathfrak{h}^{\frac{c\varphi}{\mathbf{c}+1}}\mathfrak{h}^{r_\xi}\\
&= \left(\mathfrak{g}\mathfrak{C}\right)^{r_\sigma}\mathfrak{h}^{r_\xi} = \mathfrak{t}_3\\
C_r^c h^{s_\varepsilon} g^{s_\zeta} &= h^{\varepsilon c} g^{\zeta c} h^{r_\varepsilon - c\varepsilon} g^{r_\zeta - c\zeta}\\
&= h^{r_\varepsilon} g^{r_\zeta} = t_1\\
C_{\mathbf{c}}^c h^{s_\alpha} g^{s_\eta} &= h^{\mathbf{c}c} g^{\eta c} h^{r_\alpha - c\mathbf{c}} g^{r_\eta - c\eta}\\
&= h^{r_\alpha} g^{r_\eta} = t_2\\
v^c C_u^{s_\alpha}\left(\frac{1}{h}\right)^{s_\beta} &= v^c C_u^{r_\alpha - c\mathbf{c}}\left(\frac{1}{h}\right)^{r_\beta - c\varepsilon\mathbf{c}}\\
&= C_u^{r_\alpha}\left(\frac{1}{h}\right)^{r_\beta} u^{\mathbf{c}c} u^{-c\mathbf{c}} h^{-\varepsilon c\mathbf{c}} h^{c\varepsilon\mathbf{c}}\\
&= C_u^{r_\alpha}\left(\frac{1}{h}\right)^{r_\beta} = t_3\\
C_r^{s_\alpha}\left(\frac{1}{h}\right)^{s_\delta}\left(\frac{1}{g}\right)^{s_\beta} &= C_r^{r_\alpha - c\mathbf{c}}\left(\frac{1}{h}\right)^{r_\delta - c\zeta\mathbf{c}}\left(\frac{1}{g}\right)^{r_\beta - c\varepsilon\mathbf{c}}\\
&= C_r^{r_\alpha}\left(\frac{1}{h}\right)^{r_\delta}\left(\frac{1}{g}\right)^{r_\beta} h^{-\varepsilon c\mathbf{c}} g^{-\zeta c\mathbf{c}} h^{c\zeta\mathbf{c}} g^{c\varepsilon\mathbf{c}}\\
&= C_r^{r_\alpha}\left(\frac{1}{h}\right)^{r_\delta}\left(\frac{1}{g}\right)^{r_\beta} = t_4
\end{aligned}
$$

□

Korrektheit: Der Beweis zur Korrektheit basiert auf [CL02, Theorem 4]. Der Knowledge-Extractor extrahiert wie bisher zwei verschiedene Antworten $(s_\alpha, s_\beta, s_\gamma, s_\delta, s_\varepsilon, s_\zeta, s_\eta, s_\varphi, s_\psi)$ und $(s'_\alpha, s'_\beta, s'_\gamma, s'_\delta, s'_\varepsilon, s'_\zeta, s'_\eta, s'_\varphi, s'_\psi)$ des Beweisers mit unterschiedlichen Challenges c und c'. Es sei:

$$
\begin{aligned}
\Delta c &= c' - c\\
\Delta\alpha &= s_\alpha - s'_\alpha = r_\alpha - c\mathbf{c} - (r_\alpha - c'\mathbf{c}) = \mathbf{c}\Delta c\\
\Delta\beta &= s_\beta - s'_\beta = r_\beta - cr_2\mathbf{c} - (r_\beta - c'r_2\mathbf{c}) = r_2\mathbf{c}\Delta c\\
\Delta\gamma &= s_\gamma - s'_\gamma = r_\gamma - \frac{c}{\mathbf{c}-1} - \left(r_\gamma - \frac{c'}{\mathbf{c}-1}\right) = \frac{\Delta c}{\mathbf{c}-1} \mod q\\
\Delta\delta &= s_\delta - s'_\delta\\
\Delta\varepsilon &= s_\varepsilon - s'_\varepsilon\\
\Delta\zeta &= s_\zeta - s'_\zeta\\
\Delta\eta &= s_\eta - s'_\eta\\
\Delta\varphi &= s_\varphi - s'_\varphi \mod q\\
\Delta\psi &= s_\psi - s'_\psi\\
\Delta\sigma &= s_\sigma - s'_\sigma\\
\Delta\xi &= s_\xi - s'_\xi
\end{aligned}
$$

Wir rechnen nun:

$$\begin{aligned} \frac{\mathfrak{C}^{c'}}{\mathfrak{C}^{c}} &= \frac{\left(\mathfrak{g}^{\mathbf{c}}\mathfrak{h}^{\varphi}\right)^{c'}}{\left(\mathfrak{g}^{\mathbf{c}}\mathfrak{h}^{\varphi}\right)^{c}} = \frac{\mathfrak{g}^{c'\mathbf{c}}\mathfrak{h}^{c'\varphi}}{\mathfrak{g}^{c'\mathbf{c}}\mathfrak{h}^{c'\varphi}} \\ \Leftrightarrow \qquad \mathfrak{C}^{\Delta c} &= \mathfrak{g}^{c'\mathbf{c}-c\mathbf{c}}\mathfrak{h}^{c'\varphi-c\varphi} = \mathfrak{g}^{\Delta\alpha}\mathfrak{h}^{\Delta\varphi} \end{aligned}$$

Mittels dieser Methode erhalten wir für (1) bis (7):

$$\mathfrak{C}^{\Delta c} = \mathfrak{g}^{\Delta\alpha}\mathfrak{h}^{\Delta\varphi} \tag{8}$$

$$\mathfrak{g}^{\Delta c} = \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{\Delta\gamma}\mathfrak{h}^{\Delta\psi} \tag{9}$$

$$\mathfrak{g}^{\Delta c} = \left(\mathfrak{g}\mathfrak{C}\right)^{\Delta\sigma}\mathfrak{h}^{\Delta\xi} \tag{10}$$

$$C_r^{\Delta c} = h^{\Delta\varepsilon}g^{\Delta\zeta} \tag{11}$$

$$C_{\mathbf{c}}^{\Delta c} = h^{\Delta\alpha}g^{\Delta\eta} \tag{12}$$

$$v^{\Delta c} = C_u^{\Delta\alpha}\left(\frac{1}{h}\right)^{\Delta\beta} \tag{13}$$

$$1 = C_r^{\Delta\alpha}\left(\frac{1}{h}\right)^{\Delta\delta}\left(\frac{1}{g}\right)^{\Delta\beta} \tag{14}$$

Wir zeigen zuerst, dass $\mathfrak{C}$ ein Commitment für eine ganze Zahl ungleich 1 ist. Es sei:

$$\begin{aligned} \tilde{\alpha} &:= \Delta\alpha\Delta c^{-1} \mod q \\ \tilde{\gamma} &:= \Delta\gamma\Delta c^{-1} \mod q \\ \tilde{\varphi} &:= \Delta\varphi\Delta c^{-1} \mod q \\ \tilde{\psi} &:= \Delta\psi\Delta c^{-1} \mod q \\ \tilde{\sigma} &:= \Delta\sigma\Delta c^{-1} \mod q \\ \tilde{\xi} &:= \Delta\xi\Delta c^{-1} \mod q \end{aligned}$$

Dann ergibt sich aus (8) und (9):

$$\begin{aligned} \mathfrak{C} &= \mathfrak{g}^{\tilde{\alpha}}\mathfrak{h}^{\tilde{\varphi}} \\ \mathfrak{g} &= \left(\frac{\mathfrak{C}}{\mathfrak{g}}\right)^{\tilde{\gamma}}\mathfrak{h}^{\tilde{\psi}} \\ &= \left(\mathfrak{g}^{\tilde{\alpha}}\mathfrak{h}^{\tilde{\varphi}}\mathfrak{g}^{-1}\right)^{\tilde{\gamma}}h^{\tilde{\psi}} \\ &= \mathfrak{g}^{(\tilde{\alpha}-1)\tilde{\gamma}}\mathfrak{h}^{\tilde{\varphi}\tilde{\gamma}}\mathfrak{h}^{\tilde{\psi}} \end{aligned}$$

Aufgrund des diskreten Logarithmus-Problems muss gelten

$$1 \equiv \left(\tilde{\alpha}-1\right)\tilde{\gamma} \mod q$$

und damit

$$\tilde{\alpha} \neq 1 \mod q$$

Aus (8) und (10) ergibt sich

$$\begin{aligned}\mathfrak{g} &= (\mathfrak{g}\mathfrak{C})^{\tilde{\sigma}}\, \mathfrak{h}^{\tilde{\xi}} \\ &= \left(\mathfrak{g}\mathfrak{g}^{\tilde{\alpha}}\mathfrak{h}^{\tilde{\varphi}}\right)^{\tilde{\sigma}}\, \mathfrak{h}^{\tilde{\xi}} \\ &= \mathfrak{g}^{(\tilde{\alpha}+1)\tilde{\sigma}}\mathfrak{h}^{\tilde{\varphi}\tilde{\sigma}}\mathfrak{h}^{\tilde{\xi}}\end{aligned}$$

woraus wieder aufgrund des diskreten Logarithmus-Problems folgt:

$$\tilde{\alpha} \neq -1 \mod q$$

Wir wollen nun zeigen, dass $\tilde{\alpha}$ akkumuliert ist in v. Dafür müssen wir erst folgende Behauptung zeigen:

Beh.:

$$\Delta c \mid \Delta\varepsilon \quad \wedge \quad \Delta c \mid \Delta\zeta$$

Bew.: Wir gehen davon aus $\Delta c \nmid \Delta\varepsilon$ oder $\Delta c \nmid \Delta\zeta$. Es sei $k := \log_g h$, also $g^k = h$. Damit können wir (11) umformen zu:

$$C_r^{\Delta c} = h^{\Delta\varepsilon} g^{\Delta\zeta} = h^{\Delta\varepsilon + k\Delta\zeta}$$

Da Δc zufällig ist, ist es sehr wahrscheinlich, dass es nicht $\Delta\varepsilon + k\Delta\zeta$ teilt. Sollte dies doch der Fall sein, beginnt der Knowledge-Extractor von vorne. Es sei nun

$$d := \mathrm{ggT}\left(\Delta c, \Delta\varepsilon + k\Delta\zeta\right)$$

und wir bestimmen mit dem erweiterten euklidischem Algorithmus r, s, sodass:

$$r\Delta c + s\left(\Delta\varepsilon + k\Delta\zeta\right) = d$$

Wir erhalten:

$$\begin{aligned}h^d &= h^{r\Delta c + s(\Delta\varepsilon + k\Delta\zeta)} \\ &= \left(h^r \left(h^{s(\Delta\varepsilon+k\Delta\zeta)}\right)^{\Delta c^{-1}}\right)^{\Delta c} \\ &= \left(h^r \left(h^{\Delta\varepsilon+k\Delta\zeta}\right)^{s\Delta c^{-1}}\right)^{\Delta c} \\ &= \left(h^r \left(C_r^{\Delta c}\right)^{s\Delta c^{-1}}\right)^{\Delta c} \\ &= \left(h^r C_r^s\right)^{\Delta c}\end{aligned}$$

Es sei:

$$\tilde{b} = (h^r C_r^s)^{\frac{\Delta c}{d}} \, h^{-1}$$

Dann ist

$$\tilde{b}^d = (h^r C_r^s)^{\Delta c} \, h^{-d} = h^d h^{-d} = 1$$

und:

$$h\tilde{b} = (h^r C_r^s)^{\frac{\Delta c}{d}} \tag{15}$$

1. Fall: $\tilde{b} = 1$
Dann haben wir mit

$$h = \underbrace{(h^r C_r^s)}_{x}^{\overbrace{\frac{\Delta c}{d}}^{t}}$$

die starke RSA-Annahme verletzt. $\lightning$

2. Fall: $\tilde{b} \neq 1 \quad \wedge \quad \tilde{b}^2 \neq 1$
Da gilt $\tilde{b}^d = 1$, können wir mit ggT $\left(\tilde{b} - 1\right)$ n faktorisieren. $\lightning$

3. Fall: $\tilde{b} \neq 1 \quad \wedge \quad \tilde{b}^2 = 1 \quad \wedge \quad \frac{\Delta c}{d}$ ist ungerade
Dann gilt:

$$\tilde{b}^{\frac{\Delta c}{d}} = \tilde{b}$$

Wenn wir dies in (15) einsetzen, haben wir die starke RSA-Annahme verletzt. $\lightning$

4. Fall: $\tilde{b} \neq 1 \quad \wedge \quad \tilde{b}^2 = 1 \quad \wedge \quad \frac{\Delta c}{d}$ ist gerade
In diesem Fall muss $(g^r c^s)^{\frac{\Delta c}{d}}$ ungerade Ordnung haben, was ord $\left(h\tilde{b}\right) = 2\text{ord}\,(h)$ widerspricht. $\lightning$

[DF01, §4.1]

Zusammengefasst: Wenn $\Delta c \nmid \Delta\varepsilon + k\Delta\zeta$, haben wir gezeigt, dass dies zum Widerspruch geführt werden kann. Das heißt, $\Delta c \mid \Delta\varepsilon + k\Delta\zeta$. Dies ist allerdings nur wahrscheinlich für den Fall, dass $\Delta c \mid \Delta\varepsilon$ und $\Delta c \mid \zeta$. □

Mit (12) lässt sich auf dieselbe Weise zeigen, dass $\Delta c \mid \Delta\alpha$ und $\Delta c \mid \eta$. Wir setzen nun neu:

$$\hat{\alpha} := \frac{\Delta\alpha}{\Delta c} \qquad \hat{\eta} := \frac{\Delta\eta}{\Delta c} \qquad \hat{\varepsilon} := \frac{\Delta\varepsilon}{\Delta c} \qquad \hat{\zeta} := \frac{\Delta\zeta}{\Delta c}$$

Aus (11) erhalten wir

$$C_r = a h^{\hat{\varepsilon}} g^{\hat{\zeta}} \tag{16}$$

für ein a mit $a^2 = 1$. Dabei gilt $a = \pm 1$, da sonst

$$\begin{aligned} & a^2 \equiv 1 \mod n \\ \Rightarrow \quad & a^2 - 1 \equiv 0 \mod n \\ \Rightarrow \quad & (a+1)(a-1) \equiv 0 \mod n \end{aligned}$$

und wir n faktorisieren könnten, was dem Faktorisierungsproblem [Fak13] widersprechen würde. Wir setzen (16) in (14) ein:

$$1 = a^{\Delta\alpha} \underbrace{h^{\Delta\alpha\hat{\varepsilon}} g^{\Delta\alpha\hat{\zeta}} \left(\frac{1}{h}\right)^{\Delta\delta} \left(\frac{1}{g}\right)^{\Delta\beta}}_{H}$$

Es muss gelten $H \neq -1$, da $g, h \in QR_n$ und $n = pq$ mit $p = 2p' + 1$ und $q = 2q' + 1$, was bedeutet, dass -1 keine Wurzel in $\mathbb{Z}_n$ hat. Da $H \neq -1$ muss gelten $a^{\Delta\alpha} = 1$. Aufgrund des diskreten Logarithmus-Problems muss nun gelten:

$$\Delta\alpha\hat{\zeta} \equiv \Delta\beta \mod \operatorname{ord}(g)$$

Dies setzen wir in (13) ein:

$$\begin{aligned} v^{\Delta c} &= C_u^{\Delta\alpha} \left(\frac{1}{h}\right)^{\Delta\alpha\hat{\zeta}} \\ &= \left(\frac{C_u}{h^{\hat{\zeta}}}\right)^{\Delta\alpha} \\ \Rightarrow \qquad v &= b \left(\frac{C_u}{h^{\hat{\zeta}}}\right)^{\hat{\alpha}} \end{aligned}$$

mit b, sodass $b^2 = 1$. Wie bei a gilt $b = \pm 1$, da wir sonst n faktorisieren könnten. Es sei $s := \operatorname{sign}(\hat{\alpha})$. Dann haben wir $v = u^{|\hat{\alpha}|}$ mit

$$u = \begin{cases} \left(b\frac{C_u}{h^{\hat{\zeta}}}\right)^s & \text{falls } \hat{\alpha} \text{ ungerade} \\ \left(\frac{C_u}{h^{\hat{\zeta}}}\right)^s & \text{sonst} \end{cases}$$

Da $v \in QR_n$ und $-1 \notin QR_n$, kann b nicht -1 sein. Das heißt, wir müssen nur noch zeigen, dass $|\hat{\alpha}|$ im gültigen Wertebereich liegt. Da

$$s_\alpha, s'_\alpha \in \left[-B2^{k'+k''+1}, B2^{k'+k''+1}\right]$$

ergibt sich

$$\Delta\alpha, \hat{\alpha} \in \left[-B2^{k'+k''+2}, B2^{k'+k''+2}\right]$$

Da $B2^{k'+k''+2} < \frac{q}{2}$, folgt, dass

$$\hat{\alpha} = (\Delta\alpha\hat{c} \text{ rem } q)\,(\tilde{\alpha} \text{ rem } q)$$

wobei rem ein ähnlicher Operator wie mod darstellt, der bedeutet:

$$c \text{ rem } q = c - \left\lceil \frac{c}{q} \right\rfloor q$$

Damit ist der absolute Wert von $\hat{\alpha}$ des Commitments $\mathfrak{C}$ tatsächlich akkumuliert in v. Da $B2^{k'+k''+2} < A^2 - 1$, $\hat{\alpha} \not\equiv \pm 1 \mod q$ und $\hat{\alpha} \neq 0$, muss gelten $|\hat{\alpha}| \in \mathbf{C}$. Da $|\hat{\alpha}|$ also im gültigen Wertebereich liegt und da unser Akkumulator sicher ist (siehe 15.5.2), muss $|\hat{\alpha}|$ also im Akkumulatorwert v enthalten

sein und stellt den extrahierten Wert $\mathbf{c}$ dar.

□

Honest-verifier zero-knowledge: Die Werte $\mathfrak{C}$, $C_{\mathbf{c}}$, C_u und C_r sind statistisch unabhängig von u und e. Damit lässt sich leicht ein Simulator erzeugen, dessen Signatur statistisch nicht von einer echten Signatur zu unterscheiden ist. [CL02, Theorem 4]

15.7 Konkrete Implementierung

Wie wir gesehen haben, wird für die beiden Signatures-of-knowledge jeweils ein neues Commitment benötigt. Es kann nicht ein einziges Commitment verwendet werden, da es unterschiedliche Sicherheitsanforderungen bei beiden Signaturen gibt. Eine dritte Signature-of-knowledge, die beweist, dass beide Commitments sich auf $\mathbf{c}$ festlegen, muss erstellt werden. Die genauen Details dieser Signatur lagen leider zum Zeitpunkt unserer Bachelorarbeit noch nicht vor. In der bisher einzigen Veröffentlichung zu Zerocoin [MGGR13] wird sie noch nicht erwähnt, ihr Aufbau kann nur aus dem Quelltext einer ersten Implementierung von Zerocoin [Lib13] entnommen werden. In der Dokumentation des Quelltextes wird allerdings ausdrücklich darauf hingewiesen, dass die Implementierung sich noch in einer experimentellen Phase befindet und Änderungen erwartet werden.

In den nächsten Monaten wollen die Zerocoin-Erfinder eine weitere Abhandlung veröffentlichen, welche die bisherigen Signatures-of-knowledge genauer erklärt und deren Gültigkeit beweist, sowie die dritte Signatur zum Beweis der Gültigkeit der Hilfs-Commitments vorstellt. Leider hatten wir bis zur Abgabe dieser Bachelor-Arbeit trotz E-Mail-Verkehr mit den Erfindern keinen Zugriff darauf und mussten daher viele Details aus dem vorläufigen Quelltext [Lib13] entschlüsseln. Nach Veröffentlichung dieser ausführlicheren Zerocoin-Abhandlung sollten viele Fragen, die die konkrete Implementierung des Zerocoin-Protokolls betreffen, genauer geklärt werden können.

Die momentane Implementierung von Zerocoin beinhaltet außerdem noch keine Integration in den Bitcoin-Client. In der Abhandlung von Zerocoin [MGGR13] wird allerdings erwähnt, dass solch eine Integration existiert und von den Autoren bereits getestet wurde. Bis Zerocoin allerdings in das wirkliche Bitcoin-Netzwerk integriert werden kann, muss es zunächst ausreichend getestet und weiter erklärt werden. Denn eine Änderung des Bitcoin-Protokolls erfordert das Einverständnis der Mehrheit der Miner, da diese die für Zerocoin benötigten speziellen neuen Transaktionen akzeptieren und überprüfen müssen.

Teil V

Fazit

Man erkennt, dass Bitcoin tatsächlich eine neuartige Währung bildet, die ihren Ansprüchen gerecht wird. Sie schafft es, Unabhängigkeit von jeglichen Instanzen zu schaffen, den potenziell misstraut werden kann. So wäre ein Abschöpfen der Ersparnisse durch den Staat, wie es vor Kurzem in Zypern stattgefunden hat [Zyp13], nicht möglich gewesen, wenn die Sparer ihr Geld in Bitcoins angelegt hätten. Des Weiteren führt die klar nachvollziehbare und planmäßige Erhöhung der Währungsmenge zu hoher Planungssicherheit für Besitzer von Bitcoins und durch die obere Grenze der Bitcoinanzahl wird eine Inflation langfristig vermieden.

Jedoch bestehen mit Bitcoin auch Probleme. Zum einen unterliegt ihr Wert, verursacht durch die relativ geringe Geldmenge und den Spekulationen, starken Kursschwankungen [Gox13], wodurch die Planungssicherheit zurzeit nicht existiert. Zum anderen können auch die zu speichernden Private-Keys gestohlen werden. Gelingt dies, kann der Dieb diese vollkommen problemlos nutzen und die zugehörigen Bitcoins unbehelligt ausgeben. Solch ein Diebstahl wäre im Nachhinein nicht mehr zu verfolgen, da der Besitz von Bitcoins nur über den Private-Key definiert ist.

Ein weiteres Problem, dass mit steigender Verbreitung von Bitcoin immer aktueller wird, ist die Größe der Blockchain. Da in ihr alle Transaktionen gespeichert werden müssen, steigt ihre Größe linear mit allen jemals getätigten Transaktionen. Über die Zeit sammelt sich damit eine riesige Datenmenge an, die besonders von Clients auf Smartphones schwer zu bewältigen ist.

Auch das Problem der Anonymität, wie es hier beschrieben wurde, ist wahrscheinlich den wenigsten Nutzern bewusst. Zerocoin löst dieses Problem zwar und schafft es tatsächlich Anonymität für Bitcoin zu gewährleisten, doch diese Erweiterung muss nachträglich noch im Netzwerk eingeführt werden. Die Funktionsweise von Zerocoin ist teilweise noch komplizierter als die von Bitcoin selbst. Daher glauben wir, dass es noch viele Jahre dauern wird, bis Zerocoin allgemein akzeptiert wird. Ein weiteres Hindernis für die Aufnahme von Zerocoin ist, dass die notwendigen Signatures-of-knowledge zu einer noch größeren Blockchain führen. Des Weiteren muss diskutiert werden, ob eine komplette Anonymisierung gerade im Blick auf Geldwäsche und illegale Aktivitäten, überhaupt erwünscht ist. Lösbar wäre dies mit einem zusätzlichen Schlüssel, der alle Commitments öffnen kann, und ausschließlich von der Justiz eingesetzt werden dürfte.

Probleme könnte es auch für die Signaturen mittels EK geben. Hier ist nicht gewährleistet, dass diese langfristig genügend Sicherheit bieten. Auch das Erhöhen des Sicherheitsniveaus durch z. B. das Einsetzen einer neuen Kurve ist schwer nachträglich im Netzwerk einzuführen. Dieses Einführen ist auch potenziell gefährlich, da ein Verändern der Clientsoftware z. B. zu Double-Spends führen kann.

Das größte Problem von Bitcoin ist jedoch die allgemeine Akzeptanz. Die Funktionsweise von Bitcoin ist ohne kryptografische Grundlagen schwer nachzuvollziehen. Aber gerade in Ländern, die eine instabile Währung besitzen, in denen aber Internetzugang verbreitet ist, kann Bitcoin sehr attraktiv sein. Auch wird sich im Bereich der Online-Bezahlsysteme Bitcoin weiter durchsetzen, da es besonders in diesem Sektor seine Vorteile ausspielen kann. Wir sind gespannt, wie sich Bitcoin insgesamt weiter entwickeln wird.

Literatur

[Adr13] *Technical background of version 1 Bitcoin addresses - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Technical_background_of_Bitcoin_addresses. Version: 15.07.2013

[Bas13] *Base58Check encoding - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Base58Check_encoding. Version: 20.07.2013

[BGZ08] Behrends, E. ; Gritzmann, P. ; Ziegler, G.M.: Pi und Co.: Kaleidoskop der Mathematik, Springer, 2008, S. 292

[BM94] Benaloh, Josh ; Mare, Michael de: One-way accumulators: a decentralized alternative to digital signatures. In: *Workshop on the theory and application of cryptographic techniques on Advances in cryptology*. Secaucus, NJ, USA : Springer-Verlag New York, Inc., 1994 (EUROCRYPT '93), 274–285

[Bos] Bosselaers, Antoon: *The hash function RIPEMD-160*. http://homes.esat.kuleuven.be/~bosselae/ripemd160.html

[BP97] Baric, Niko ; Pfitzmann, Birgit: Collision-Free Accumulators and Fail-Stop Signature Schemes Without Trees, Springer-Verlag, 1997, S. 480–494

[Bru13] Brunner, Grant: *The Bitcoin network outperforms the top 500 supercomputers combined - ExtremeTech.com*. http://www.extremetech.com/extreme/155636-the-bitcoin-network-outperforms-the-top-500-supercomputers-combined. Version: 23.05.2013

[But13] Buterin, Vitalik: *Bitcoin Network Shaken by Blockchain Fork - Bitcoin Magazine*. http://bitcoinmagazine.com/bitcoin-network-shaken-by-blockchain-fork/. Version: 22.07.2013

[Cam98] Camenisch, Jan: Group Signature Schemes and Payment Systems Based on the Discrete Logarithm Problem. (1998). ftp://ftp.inf.ethz.ch/pub/crypto/publications/Cameni98.pdf

[Chn13] *Change - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Change. Version: 03.09.2013

[CL02] Camenisch, Jan ; Lyasyanskaya, Anna: *Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials*. 2002

[Dbl13] *Bericht über Double Spend im Bitcoin-Forum*. https://bitcointalk.org/index.php?topic=152348.0. Version: 22.08.2013

[DF01] Damgard, Ivan ; Fujisaki, Eiichiro: An Integer Commitment Scheme based on Groups with Hidden Order, Springer, 2001, 125–142

[Dis13] *Diskriminante - Deutsche Wikipedia*. http://de.wikipedia.org/wiki/Diskriminante. Version: 01.09.2013

[ECC13] *Elliptic Curve DSA - Deutsche Wikipedia.* http://de.wikipedia.org/wiki/ECDSA. Version: 27.06.2013

[EG85] El Gamal, Taher: A public key cryptosystem and a signature scheme based on discrete logarithms. In: *Proceedings of CRYPTO 84 on Advances in cryptology.* New York, NY, USA : Springer-Verlag New York, Inc., 1985, S. 10–18

[Fak13] *Faktorisierungsproblem - Deutsche Wikipedia.* http://de.wikipedia.org/wiki/Faktorisierungsproblem. Version: 03.09.2013

[Fee13] *Transaction fees - Bitcoin Wiki.* https://en.bitcoin.it/wiki/Transaction_fees. Version: 24.08.2013

[Fog13] *Bitcoin fog company.* http://www.bitcoinfog.com. Version: 23.07.2013

[FS87] Fiat, Amos ; Shamir, Adi: How To Prove Yourself: Practical Solutions to Identification and Signature Problems, Springer-Verlag, 1987, S. 186–194

[GFD09] Gallagher, Patrick ; Foreword, Deputy D. ; Director, Cita F.: FIPS PUB 186-3 FEDERAL INFORMATION PROCESSING STANDARDS PUBLICATION Digital Signature Standard (DSS). (2009). http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf

[GLV99] Gallant, R. ; Lambert, R. ; Vanstone, S.: *Faster Elliptic Curve Cryptography using Efficient Endomorphisms.* 1999

[Gox13] *Mt.Gox - Bitcoin Exchange.* https://www.mtgox.com/. Version: 24.08.2013

[HMV03] Hankerson, Darrel ; Menezes, Alfred J. ; Vanstone, Scott: Guide to Elliptic Curve Cryptography. (2003)

[Hoh07] Hohenberger, Susan: Lecture 10: More on Proofs of Knowledge. (2007). http://www.cs.jhu.edu/~susan/600.641/scribes/lecture10.pdf

[Lib13] *Quelltext von libzerocoin auf GitHub.* https://github.com/Zerocoin/libzerocoin. Version: 02.09.2013

[Mau09] Maurer, Ueli: Unifying Zero-knowledge Proofs of Knowledge. In: Preneel, B. (Hrsg.): *Advances in Cryptology - AfricaCrypt 2009*, Springer-Verlag, Juni 2009 (Lecture Notes in Computer Science)

[MGGR13] Miers, Ian ; Garman, Christina ; Green, Matthew ; Rubin, Aviel D.: Zerocoin: Anonymous Distributed E-Cash from Bitcoin. (2013). http://zerocoin.org/media/pdf/ZerocoinOakland.pdf

[Nak] Nakamoto, Satoshi: Bitcoin: A Peer-to-Peer Electronic Cash System. http://bitcoin.org/bitcoin.pdf

[NSA10] Suite B Implementer's Guide to FIPS 186-3 (ECDSA). (2010). http://www.nsa.gov/ia/_files/ecdsa.pdf

[Prt13] *Protocol specification - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Protocol_specification. Version: 02.09.2013

[Pub13] *Public-Key-Verschluesselungsverfahren - Deutsche Wikipedia*. http://de.wikipedia.org/wiki/Public-Key-Verschlüsselungsverfahren. Version: 12.07.2013

[Res00] Research, Certicom: SEC 2: Recommended Elliptic Curve Domain Parameters. (2000). http://www.secg.org/collateral/sec2_final.pdf

[Sch91] Schnorr, Claus-Peter: Efficient Signature Generation by Smart Cards. In: *J. Cryptology* 4 (1991), S. 161–174. http://dx.doi.org/10.1007/BF00196725. – DOI 10.1007/BF00196725

[Sec12] *Secp256k1 - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Secp256k1. Version: 10.12.2012

[Sha13] *SHA-2 - Wikipedia*. http://de.wikipedia.org/wiki/SHA-2. Version: 19.07.2013

[Sig13] *Digitale Signatur - Deutsche Wikipedia*. http://de.wikipedia.org/wiki/Digitale_Signatur. Version: 06.06.2013

[Tra13] *Transactions - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Transactions. Version: 18.07.2013

[Wal13] *Wallet - Bitcoin Wiki*. https://en.bitcoin.it/wiki/Wallet. Version: 19.07.2013

[Wer02] Werner, A.: *Elliptische Kurven in Der Kryptographie*. Springer-Verlag GmbH, 2002 (Springer-Lehrbuch). http://books.google.de/books?id=dFV_bVUFre0C

[ZKB13] *Zero-Knowledge-Beweis - Deutsche Wikipedia*. http://de.wikipedia.org/wiki/Zero_Knowledge. Version: 11.08.2013

[Zyp13] *Zwangsabgabe: Reiche Bankkunden auf Zypern verlieren mehr als erwartet - SPIEGEL ONLINE*. http://www.spiegel.de/wirtschaft/unternehmen/zypern-reiche-bankkunden-verlieren-mehr-als-erwartet-a-891793.html. Version: 30.03.2013