Fachspezifisches Seminar Kryptologie

Joachim Schiele

23. Februar 2006

Betreuer Professor Hauck.
Dank an Professor Hauck und Dr. Steffanie Reiferscheid.

Inhaltsverzeichnis

Diese Ausarbeitung bezieht sich auf die Publikation von Thomas Sander und Amnon Ta-Shma mit dem Titel „Auditable, Anonymous Electronic Cash - Extended Abstract“[1] und den Vortrag, welchen ich darüber gehalten habe.

1 Vorwort

Die meisten elektronischen Geldsysteme arbeiten mit Signaturen. Bei Signatur basierten Systemen ist allerdings die Möglichkeit vorhanden, dass die Bank bösartigerweise gültiges Geld erstellen kann ohne dies melden zu müssen. Wie in der Vergangenheit gezeigt wurde, könnte dies zu einem Problem größeren Ausmaßes werden falls der geheime Schlüssel der Bank gestohlen wird. Ein weiterer Nachteil ist, dass das oben erwähnte System es nicht ermöglicht den Geldfluss zu überwachen (z.B. mittels Seriennummern auf Münzen). In der vorliegenden Publikation wir nun ein signaturfreies System vorgestellt welches vollständige Anonymität garantiert, prüffähig ist und welches keinen Gebrauch von Kryptographie im Sinne von öffentlichen / privaten Schlüsseln macht. Die Sicherheit in diesem System hängt von der Integrität der Datenbank ab, welche von einem oder mehreren Finanzinstituten gewartet wird.

Wie in der Publikation schon erwähnt ist, darf man nicht erwarten dies sei eine vollständige Lösung aller Probleme mit elektronischem Geld. Das vorgestellte Verfahren ist als theoretisch effizient aber nicht praktikabel eingestuft. Es hat sich aber herausgestellt, dass die intern verwendete Datenstruktur, welche später noch genauer erklärt wird, einige Problemen aufwirft und daher nicht so verwendet werden kann wie sie in der Publikation und hier vorgestellt wird. Dazu aber später mehr.

2 Einführung

Geldsysteme sollten sicher und fehlerfrei sein. Jeder der ein Konto mit Geld darauf besitzt, verlässt sich dabei auf eine Bank, obwohl eine Bank auch bankrott gewirtschaftet werden kann. Dabei spielt auch die Regierung eine wichtige Rolle, da diese Anleger der Bank gegen etwaige Zahlungsunfähigkeit versichert. Als Gegenleistung oder besser Auflage muss die Bank dem Staat fast alle Daten offenlegen. So erhält der Staat Einblick und Kontrolle über das Finanzwesen. Es existieren zwei wichtige Aufgaben für den Staat: Kontrolle über die Arbeitsweise der Bank, werden z.B. alle Überweisungen richtig getätigt. Der zweite Punkt ist die wirtschaftliche Lage der Bank und ob diese Gefahr läuft rote Zahlen zu schreiben indem sie schlechte Kredite vergibt oder schlechte Investitionen macht. Wichtig ist auch die Überwachung der Kriminalität innerhalb der Bank, also von Mitarbeitern. Die Überwachungsfunktion nimmt jedoch an Bedeutung zu, wenn man die Verwendung von elektronischem Geld ermöglicht, da hier gültiges, nicht-registriertes Geld ev. sehr leicht erstellt werden kann ohne das es bemerkbar ist.

Ein System welches **überwachbar** ist, ermöglicht eine einfache Kontrolle des Geldflusses. Es existiert d.h. kein inoffizielles Geld[2], welches der Bank nicht bekannt ist. Es elektronisches Geldsystem welches über diese Eigenschaft verfügt und gleichzeitig den gesamten Geldfuss mitüberwachen kann, verhält sich also genau gleich wie wir es vom Bargeld kennen. Falls man diese Überwachbarkeit im obigen Sinne nicht realisieren kann ergeben sich Probleme wie Inflation, da man nicht weiß woher das Geld kommt, welches ja keinen realen Gegenwert besitzt.

2.1 Geldfälschung

Wie es bei heutigem Bargeld der Fall ist, sollte auch bei elektronischen Geld eine Seriennummer vorhanden sein. Diese kann bei Bedarf jederzeit gesperrt werden, so dass im selben Moment die Münze nicht mehr verwendet werden kann. Die bisherigen Konzepte für elektronisches Geld enthielten Sicherheitslöcher was die Verwendung kritisch gestaltete. Um diese Probleme zu beseitigen ist es Wichtig, dass das Geld einerseits überwachbar ist und andererseits jede Geldmünze gesperrt werden kann, obwohl die Anonymität der Benutzer gewahrt bleibt.

[1] Vgl. [1] im Literaturverzeichnis
[2] ev. sogar Blüten

Im Vorgestellten Modell sind alle Angaben der Bank öffentlich, frei zugänglich. Dies bedeutet, dass Banknoten selbst von allen Benutzern verifiziert werden können. Die Sicherheit dieses Systems hängt in der Integrität der Datenbank seitens der Bank ab.

In signaturbasierten Systemen könnte der private Schlüssel der Bank gestohlen werden, das damit erstellte Geld sieht selbst für die Bank absolut gültig aus. Es kommt dann zur Inflation wobei niemand feststellen kann wer das neue Geld in den Markt bringt was fatal ist. Dies würde das Versagen des gesamten Konzeptes mit signaturbasierten Systemen bedeuten. Im vorgestellten Modell ist diese Schwäche behoben.

2.2 Erpressung

Das vorgestellte System verfügt über die Eigenschaft, dass Erpressung wie wir es im konventionellen Sinne kennen, nicht mehr möglich ist. Ein Erpresser kann zwar Geld erzwingen, jedoch gibt es keine Möglichkeit das Geld ohne Kenntnis der Seriennummer seitens des Zahlenden zu übergeben. Das Geld kann zwar übergeben werden, jedoch kann das Geld jederzeit (z.B. nach Freilassung der Geiseln) ungültig gemacht werden. Eine Konsequenz könnte sein, dass eine Geisel in Zukunft nicht mehr freigelassen wird bevor das Geld vom Erpresser ausgegeben worden ist. „shit happens“.

2.3 Geldwäsche

Natürlich stellt sich die Frage ob Geldwäsche verhindert werden kann, gerade weil ein so hoher Grad an Anonymität gewahrt wird. Wenn man ein Betragslimit einführt und ein nicht übertragbares Zahlungssystem verwendet, so könnte dies gelingen ohne die Anonymität zu verlieren.

Kleine Beträge welche durch das Betragslimit begrenzt sind und nicht übertragbar sind, erschweren die Geldwäsche erheblich da man dann sehr viele Geldeinheiten benötigt dies effizient durchführen zu können. Erpressungen, Banküberfälle und Geldwäsche wären somit nur schwer realisierbar während die Anonymität für die Nutzer gewahrt bleibt.

2.4 Double Spending

Wie schon unter 4.6 hingewiesen wird, lässt sich „Double Spending“ recht einfach verhindern, da die ID der Person zurückverfolgt werden kann, falls eine Münze mehr als ein mal ausgegeben wurde. Da in dieses System Geld nicht übertragbar ist (vgl. wie man es von heutigem Bargeld kennt) kann es zwar anonym verwendet werden jedoch gleichzeitig einen festen Eigener haben.

3 Grundlagen

In einem signaturbasierten System erhält jede Münze eine eindeutige Signatur. Das hier vorgestellte Verfahren funktioniert jedoch etwas anders und wird nicht auf gleiche Weise offline funktionieren wie es das bei signaturbasierten Verfahren der Fall ist. Zur Struktur gab es drei Vorschläge, hier im Detail:

1. Die Bank verwaltet eine List $L = \{x_1, x_2, ..., x_n\}$, nun kann eine Information über eine Münze in der Liste enthalten sein und ist somit gültig oder sie ist nicht in der Liste und kann somit ausgeschlossen werden. Jedoch müsste dann für jeden Teilnehmer die komplette Liste übertragen werden, dies ist ein zu großer Aufwand.

2. Jede Münze erhält eine Signatur, dies sollte aber gerade vermieden werden, da schon oben aufgezeigt wurde welche Folgen ein Diebstahl des privaten Schlüssels haben würde.

3. Der Vorschlag von Merkle ist eine **binäre Baumstruktur** zu verwenden, wobei die Blätter Münzen repräsentieren und die Äste bestehen aus hierarchisch gebildeten Hashwerten. Wie in der Publikation beschrieben, war das die gesuchte Struktur.

Da der Vorschlag von Merkle den Autoren zur Zeit der Erstellung ihrer Arbeit überzeugend vorkam wurde diese Datenstruktur übernommen, die resultierenden Probleme werden im Abschnitt 6 diskutiert. Eine weitere **Anforderung an die Hash-Funktion** ist, dass sie **kollisionsresistent** sein muss. Von diesem Baum werden dann die Wurzeln an alle Teilnehmer verteilt. Da eine Hash-Funktion, also eine gute Einwegfunktion, verwendet wird um diese Information zu erhalten, kann von einem Eintrag aus der Liste keine Rückrechnung stattfinden.

3.1 Die Baumstruktur

Im „paper“ wird vorgeschlagen einen balancierten Baum zu verwenden. Die Vorteile eines balancierten Baumes sind, dass die Höhe O(log n) nicht überschreitet wobei n die Knoten sind. Vorteile bringt dies z.B. bei den Algorithmen welche auf diesen Datenstrukturen operieren, da diese dann schneller sind als bei nicht balancierten Bäumen. Zum Vergleich könnte bei einem nichtbalancierten Baum ein Suchalgorithmus im „worst case“ O(n) benötigen.

Allgemein unterscheidet man **zwei Kriterien** bei balancierten Bäumen:

1. Höhenbalance
   Zitat Wikipedia: „Nach der Höhe balancierte Bäume stellen für jeden Knoten sicher, dass die Höhe der linken Unterbaumes und die Höhe des rechten Unterbaumes nur um ein bestimmtes Verhältnis oder eine bestimmte Differenz voneinander abweichen.“

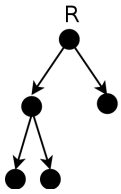

Abbildung 1: Ohne Höhenbalance

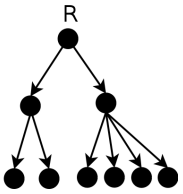

Abbildung 2: Mit Höhenbalance

2. Gewichtsbalance
   Zitat Wikipedia „In einem gewichtsbalancierten Baum (BB-Baum) gilt für jeden Knoten: Die Anzahl der Knoten (das Gewicht) links unter ihm steht in einem gewissen Verhältnis zu der Anzahl der Knoten rechts unter ihm.“

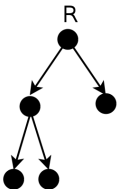

Abbildung 3: Ohne Gewichtsbalance

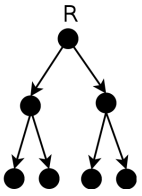

Abbildung 4: Mit Gewichtsbalance

Hier muss sowohl die **Höhenbalance** als auch die **Gewichtsbalance** eingehalten werden. Eine weitere Anforderung ist, dass die **Bäume binäre Strukturen** sein müssen, wie dies u.a. von der Hash-Funktion verlangt wird. Binäre Bäume sind im Allgemeinen Bäume in denen man von der Wurzel ausgehend Richtung Blätter immer nur links oder rechts aus gehen kann, da nie eine Abzweigung mit mehr als zwei Möglichkeiten vorkommt. Abbildung 2 ist z.B. kein binärer Baum, während die Bäume in Abbildung 1, 3 und 4 binäre Bäume darstellen.

3.2 Operationen an der Baumstruktur

Hier soll ein Allgemeiner Überblick über die möglichen Operationen an den Baumstrukturen geben werden, da dies aus der Sicht der Datenverarbeitung wichtig erscheint. Anzumerken ist, dass die Baumstruktur nicht nur einen einzelnen Baum repräsentiert, sondern eine gewisse Menge an Bäumen. Während das System arbeitet, werden alte Bäume immer größer, da sie immer wachsen. Es kommen aber auch neue Bäume hinzu. Wenn zwei Bäume gleich groß sind, werden sie zu einem noch größeren zusammengefügt.

3.2.1 Aktualisierung der Wurzeln

Wie in Abschnitt 4.4 beschrieben steht, muss die Datenstruktur erweitert werden. Falls neue Münzen angefordert werden, so müssen jeweils ein Paar Münzen zu einem „Minutenbaum " verschmolzen werden. Es entsteht, wie auch beim Zusammenfügen von Bäumen gleicher Größe, eine Hash-Kette welche den Weg vom Blatt zur Wurzel beschreibt. Dies ist z.B. für Alice von Bedeutung, da sie nur mit der Kenntnis dieser bezahlen kann. Diese ist aber unter Abschnitt 5.3 noch näher beschrieben.

3.2.2 Zusammenführung von zwei Bäumen zu einem neuen Baum

Wie auch in Abschnitt 4.4 schon beschrieben, ist dies die zweite wichtige Operation am Baum. Da wir oben schon erwähnt Baumstrukturen zusammenfügen müssen.

3.2.3 Geld ungültig machen:

Das Löschen eines Blattes wird dann gebraucht, wenn man ein Blatt (genau: also ein z) aus der Baumstruktur entfernen will. Eigentlich kann man aber keine Blätter löschen, sondern nur durch einen anderen Inhalt ersetzen (ev. einen Dummy-Wert). Dies wird nötig, falls Erpresser Münzen erhalten haben. Diese Münzen müssen gelöscht werden, bevor sie ausgegeben werden können. Falls hingegen ein normaler Benutzer Geld ausgibt, so muss kein Blatt gelöscht werden, da der Aufwand dafür zu groß wäre. Hingegen kann dank der Verhinderung von „double-spending" kein Geld doppelt ausgegeben werden (siehe Abschnitt 4.6). Das Problem ist allerdings, dass ein Erpresser kein Identitätsnachweis hinterlegt hat und somit kein P_A wie Alice besitzt. Falls er das Geld korrekterweise nur ein mal verwendet, so kann dank der im System verfügbaren Anonymität sowieso nicht verfolgt werden.

Nach dem Einsetzen des Dummy-Wertes müssen alle Hash-Werte oberhalb neu berechnet werden, dies geschieht nicht anders als unter Abschnitt 3.2.1 beschrieben steht.

4 Das Protokoll

4.1 Setup

Während des Systemsetups wählen die Bank und der Prüfer (möglicherweise auch Benutzer) gemeinsam folgende Objekte. F_q ist ein Körper der Größe $q = poly(N)$ und N ist die obere Grenze der von der Bank herstellbaren Münzen. G ist eine Primgruppe der Ordnung p für welche der „Diskrete Logarithmus" schwer lösbar bzw unlösbar ist. $|G| \geq q^3$.
Weiter wird eine effiziente 1-1 Einbettung $E : F_q^3 \longrightarrow [0..p-1]$ gewählt (durch Verwendung der Binärdarstellung von Objekten in F_q). $g : [0..p-1] \times [0..p-1] \longrightarrow G$ ist eine kollisionsfreie, verheimlichende Einwegfunktion.

D ist ein großer Definitionsbereich $|D| \geq |G|$, und $h : D \times D \longrightarrow D$ eine kollisionsresistente Hash-Funktion. Ein effizientes 1-1 einbettendes $F : G \longrightarrow D$ wird gewählt. Die Bank verwaltet einen Hash-Baum T über D mit N Blättern. Dieser Hash-Baum bildet sich schrittweise und er muss nicht initialisiert werden.

Jeder Verkäufer erhält eine eindeutige ID und es wird vorausgesetzt, dass ein Zufalls-Orakel existiert welches folgende Abbildung herstellt: $H : TIME \times ID \longrightarrow F_q$, also die Zeit und die UserID auf ein zufälliges Element aus F_q abbildet. Jeder Verkäufer kann pro Zeiteinheit nur eine Transaktion

durchführen, wobei die Zeiteinheit sehr kurz gewählt werden kann. Alternativ kann der Händler zu jeder weiteren Transaktion im selben Zeitintervall eine serielle Nummer hinzufügen, die sich dann je Transaktion im selben Zeitintervall unterscheiden muss.

4.2 Eröffnung eines Kontos

Wenn Alice ein Konto eröffnet muss sie sich bei der Bank identifizieren. Die Bank vereinbart mit ihr eine öffentliche Identität $P_A \epsilon F_q$ welche Alice eindeutig identifiziert.

4.3 Geld abheben

Alice authentifiziert sich selbst bei der Bank. Sie wählt ein $u_1 \epsilon_R F_q$, $serial \epsilon_R F_q$ und berechnet $u_2 = P_A - u_1 \epsilon F_q$ und $x = (u_1, u_2, serial) \epsilon F_q^3$. $serial$ bezeichnet die Seriennummer einer Münze und u_1, u_2 werden dazu verwendet die Identität von Alice zu schützen. Weiter wählt sie $r \epsilon_R [0..p-1]$ und sendet $z = F(g(E(x), r))\epsilon D$ zur Bank. Sie gibt der Bank einen nicht-interaktiven „Zero Knowledge Argument“ kurz ZKA[3], dass sie $u_1, u_2, serial$ und r, also $z = F(g(E(u_1, u_2, serial), r))$ und $u_1 + u_2 = P_A$, und somit den Beweis das die Münze korrekt ist. Die Bank versichert sich, dass die Münze vorher noch nicht ausgegeben wurde.

Im ZKA muss Alice Anfragen[4] richtig beantworten können. Diese Anfragen werden durch das Zufalls-Orakel H während des ZKA erzeugt.

Dann zieht die Bank einen Dollar von Alices Konto ab und fügt dem Baum T an einer noch unbenutzten Stelle den Wert z an (zusammen mit den nötigen Änderungen vom Blatt z zur Wurzel R). Wenn das Zeitintervall beendet ist erstellt die Bank eine Hash-Kette vom Blatt zur Wurzel R und gibt diese an Alice. Weiter gibt die Bank die neue Wurzel R an bzw verschmelzt den neuen „Minutenbaum“mit anderen Bäumen und verteilt die neuen Wurzeln R.

4.4 Updates: Versenden von Roots

Jede Minute wird ein neuer „Minutenbaum“ erstellt und am Ende wird das Resultat der hinzugenommenen Trees verteilt. Falls zwei „Minutenbäume“ existieren werden diese zusammengenommen und zu einem „Stundenbaum“ verbunden (Siehe Abbildung 5). Gleiches gilt auch für zwei „Stundenbäume“, welche zu einem „Tagesbaum“ zusammengenommen werden (Siehe Abbildung 6).

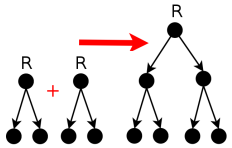

Abbildung 5: Zwei Minutenbäume werden zu einem Stundenbaum verschmolzen

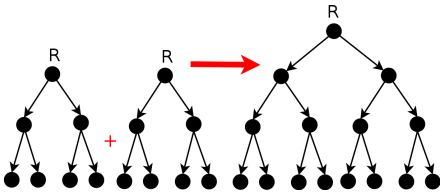

Abbildung 6: Zwei Stundenbäume werden zu einem Tagesbaum verschmolzen

Am Ende einer Zeiteinheit wird eine Broadcast Nachricht an alle Nutzer versendet, welche in diesem Zeitabschnitt eine Münze abgehoben haben. So enthält das „StundenUpdate“ die HashWerte der zwei „Minutenbäume“ welche durch das Hashverfahren verschmolzen wurden. Verkäufer können ihrer eigenen Aktualisierungsmethoden entwickeln wie z.B. alle 2 Stunden ein Update zu machen, allerdings können dann Münzen die in den letzten 2 Stunden erst erstellt wurden beim Händler nicht angenommen werden.

[3] Unter Abschnitt 5.4 näher beschrieben
[4] challenges

4.5 Bezahlung

Alice will dem Händler M (mit ID m_{id}) mit einer Münze $z = F(g(E(u_1, u_2, serial), r))$ bezahlen. Das Protokoll startet mit der Aktion von Händler M, Alice die Wurzeln R des lebenden Baumes zu senden, welche er kennt. Eine Wurzel nennt man lebendig falls sie die Wurzel eines gerade erzeugten „Minutenbaumes“, „Stundenbaumes“, „Tagesbaumes“, usw ist. Alice sendet dem Händler dann ihre $serial, time$ und den Wert $v = (u_1, c*u_2)$, wobei $c = H(time, m_{id})$ ist. c selbst darf dabei nicht 1 sein. Sie beweist dem Händler durch einen ZKA, dass sie $(u_1, u_2, r$ und R) wie auch die Hashkette $((i_1, ..., i_d); w; (y_1, ..., y_d))$ zu R kennt. Näheres zur Hashkette unter Abschnitt 5.3.

1. $R \epsilon WURZELN$
2. $w = F(g(E(u_1, u_2, serial), r))$
3. $v = (u_1 + c * u_2)$

Der Händler überprüft die Richtigkeit des nicht-interaktiven ZKA.

4.6 Geld abheben

Der Käufer transferiert die Informationen welcher er von Alice bei der Bezahlung erhalten hat, zur Bank. Die Bank prüft, dass der Käufer m_{id} das selbe Geld, mit selbigen Parametern, nicht schon früher eingelöst hat. Weiter muss die Bank die Anfragen des ZKA auf Korrektheit prüfen ($c = H(time, m_{id})$), wie die in der Transaktion enthaltenen WURZELN den wirklichen Wurzeln entsprechen und zuletzt ob der ZKA korrekt ist. Wichtig ist auch ob der gegebene serial schon ausgegeben wurde. Wenn nicht überweist die Bank dem Händler das Geld und behält den $serial \epsilon F_q$ und verbindet dies mit $c \epsilon F_q$ und $v(= u_1 + c * u_2) \epsilon F_q$

Doublespending: Wenn dieser *serial* schon zur Bezahlung verwendet wurde, ist die Bank im Besitz über genügend Informationen um folgende lineare Gleichung zu lösen: $v_1 = u_1 + c_1 * u_2$ und $v_2 = u_1 + c_2 * u_2$. Die Bank löst die Gleichung und erhält: $P = u_1 + u_2$ und somit die Identität von Alice.

4.7 Geldinformation löschen

Im „paper“ wird hervorgehoben, dass gerade das ungültig machen bzw Löschen von Geld nicht zu oft vorkommen sollte, da sonst zu viele Nachrichten an alle Beteiligten gesendet werden.

Nun stellt sich folgende Frage: Wird das Geld nun entfernt oder bleibt es einfach im System?

Vermutung: Es bleibt hängen, da ja Alice beim zweiten ausgeben des gleichen Geldes erwischt wird. ES wird nur bei Erpressung gelöscht, dies erfordert dann ein UPDATE an alle Münzenbesitzer des betroffenen Baumes. Verwendete Münzen werden bei der Bank registriert und können dann nicht doppelt verwendet werden, verweilen jedoch für jeden zugänglich im Baum.

Natürlich kann der Verkäufer das Geld nicht doppelt einlösen und Alice kann dank der ID P_A welche die Bank berechnen könnte, falls sie es versuchen würde „Double Spending“, erwischt werden.

5 Werkzeuge

5.1 Hashfunktionen

Mögliche Attacken, welche im Zusammenhang mit Hash-Funktionen wichtig sind wären Kollisionsattacken aber auch Preimage-Attacken. Bei Kollisionsattacken geht es lediglich darum irgendeine Zeichenkette zu finden, welche den gleichen Hash-Wert hat wie eine andere Eingabe. Bei Preimage-Attacken muss zu einer schon gegebenen Eingabe ein Hash-Wert gefunden werden, der Richtig ist.

Was man unter Hash-Funktion versteht, kann man sehr schön bei der Wikipedia nachlesen, darum soll hier nicht weiter darauf eingegangen werden. Erwähnenswert sind jedoch einige weit verbreitete Verfahren wie MD4, MD5 und SHA1. Allerdings kommen um MD5 immer neue schlechte Nachrichten

auf so existieren derzeit zwar keine preimage exploits, es werden jedoch immer mehr Designfehler bekannt. Hash-Algorithmen wie **SHA1 Secure Hash Algorithm - Version 1.0** bzw **AES von Davies Meyer** scheinen momentan das Rennen zu machen.

5.2 Hash-Baum

Der Hash-Baum ist die zentrale Struktur welche die Bank verwaltet. Jeder hat Lesezugriff auf diese Daten. Nur mit der Hilfe des Baumes kann Alice bei einem Händler bezahlen, da der Händler unabhängig von Alice immer die Wurzeln des Baumes bezieht und Alice kann später beweisen, dass sie ein Blatt kennt und von diesem einen überprüfbaren Pfad bis zur Wurzel R. Zu beachten ist, dass es jedoch mehrere Wurzeln R geben kann, abhängig wie viel Bäume gerade verknüpft werden bzw noch im Aufbau sind.

Für einen gegebenen Definitionsbereich D und ein bekannten Hash-Algorithmus $h : D \times D \longrightarrow D$ sowie ein Hash-Baum $Baum(T, val)$ bestehend aus einem balancierten binären Baum T mit Knotenpunkte V. Die Knotenpunkte folgen der Funktion $val : V \longrightarrow D$ was soviel bedeutet wie, dass ein Knotenpunkt mit zwei Unterpunkten v_1 und v_2 durch $val(v) = h(val(v_1), val(v_2))$. Die einzige Operation welche auf einem Hash-Baum durchgeführt werden kann ist UPDATE(Blatt, w) wobei der Wert des Blattes zu w geändert wird und alle Knotenpunkte darüber werden entsprechend der Rechenvorschrift aktualisiert.

Wie viele Münzen kann ein Baum mit tiefe x beherbergen? Mit einem Baum der Tiefe von 26 kann man immerhin 67 Mio. Münzen erstellen. Die Münzen werden nicht gelöscht, da dann die Baum-Struktur sonst ihren ganzen Nutzen verliert. (Siehe Diskussion auf Seite 9, Abschnitt 6). Genauer beschrieben unter Abschnitt: 3.2.3.

5.3 Hash-Kette

Eine Hash-Kette der Länge 1 zu einer Wurzel R ist ein Tripel $(i_1; x; y)$ welches folgende Regel erfüllt: $f^{(i_1)}(x, y) = R$, wobei $f^{(0)}(x, y) = h(x, y)$ und $f^{(1)}(x, y) = h(y, x)$.

Eine Kette der Länge $d > 1$ zu einer Wurzel R ist ein Tripel $(i_1, ..., i_d); x; (y_1, ..., y_d))$
wobei $((i_1, ..., i_{d-1}); f^{(i_d)}(x, y_d); (y_1, ..., y_{d-1}))$ eine Hash-Kette der Länge d-1 ist. Eine Hash-Kette startet beim Wert x und geht bis zur Wurzel R.

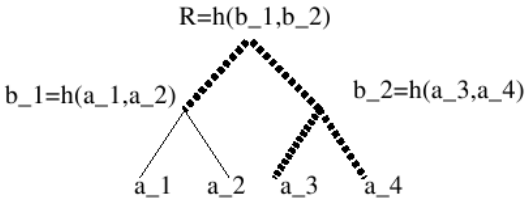

Abbildung 7: Eine Hashkette

5.4 Nichtinteraktiver ZKA

Ein nichtinteraktives „Zero Knowledge Argument “ wird so im Protokoll verwendet, dass der Beweis ohne Vorkenntnisse nachvollziehbar ist. Solche Verfahren wurden im Laufe des Seminars behandelt, darum werde ich hier nicht weiter darauf eingehen. Wichtig ist im Prinzip zu wissen, dass wir einen solchen verwenden.

6 Datenstruktur Problematik

Bei der Konstruktion der „minute trees“, etc müssen die Blätter gleich mit Elementen aus D (siehe oben) bezeichnet werden, damit die Wurzel eines „minute trees“ tatsächlich auch als Hashwert h(x,y) der beiden Werte der Blätter berechnet werden kann.

Da die hierarchische Datenstruktur immer nur in gewissen Zeitintervallen vergrößert wird, stellt sich die Frage ob genügend Geld erzeugt werden kann, wenn die Zahl der neu zu erstellenden Münzen zu gering ist bzw wenn zu viele Nutzer gleichzeitig Münzen erstellen wollen.

Im ersten Fall will die Bank einen neuen „minute tree“ erstellen und braucht dazu zwei Anfragen nach neuen Münzen. Falls nur eine Anfrage kommt muss entweder ein Dummywert eingefügt werden, welcher einen ganzen Platz verschwendet oder die Erstellung der Münze verzögert sich, bis es genug andere Anfragen, aber mindestens noch eine weitere Anfrage gibt. Erst dann kann die Bank die Werte F(g(E(u1,u2,serial),r)) der beiden Nutzer an den Blättern anbringen und h(x,y) berechnen.

Im zweiten Fall wollen mehr als zwei Benutzer Münzen abheben und es entstehen Wartezeiten bis genügend Minuten für die Erstellung neuer „minute tress “ vergangen sind. Sollen in einer Minute 1000 Münzen abgehoben werden, so dauert dies 500 Minuten, bis genügend viele minute trees erstellt sind (plus die nicht mitgerechneten neuen Anfragen).

Man könnte sich nun vorstellen, dass bei einer Anfragenflut auf Münzen einfach die Dummywerte ersetzt werden, das resultierende Problem macht die Vorteile der hierarchischen Datenstruktur aber kaputt, da sich nun alle Hashwerte ändern würden, falls man eine Münze (also ein Blatt) ändert und genau das war ja der Vorteil der hierarchischen Datenstruktur, dies nicht zu müssen. Updates bestehen nur aus der Mitteilung von Wurzeln (R) von zwei „minute trees“ die zu einem „hour tree“ zusammengefasst werden, entsprechend für andere „trees“.

6.1 Datenstruktur Problematik - ev. Lösungsansätze

1. Man könnte pro Zeiteinheit statt nur 2 Münzen gleich mehrere Münzen, also Blätter, zulassen. Dies würde jedoch bedeuten, dass die **Hashfunktion angepasst** werden müsste und das man **keinen binären Baum** mehr hat. Zur Berechnung des Hash-Wertes gab es vorher nur Links (0) und Rechts (1) während nun genaue Positionsbeschreibungen für einzelne Münzen erforderlich werden. So würde nicht mehr nur $DxD \rightarrow D$ sondern z.B. $DxDxDxD \rightarrow D$ bzw beliebig viele D paare (bis zu einer **Obergrenze**). ..... Abbildung 8 veranschaulicht dies.

2. „minute trees“ sind nicht nur binäre Bäume mit zwei Blättern der Höhe 1, sondern binäre Bäume größerer Höhe (abhängig davon, wie viele Münzen in diesem Zeitraum abgehoben werden sollen). Dann erholt man beim Zusammenfügen von „minute trees“, „hour trees“, usw allerdings **keine balancierten Bäume** mehr und die Höhe der Bäume kann größer werden (als etwa 26 wie im „paper“ vorgerechnet). ..... Abbildung 9 veranschaulicht dies.

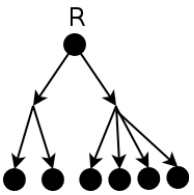

Abbildung 8: kein binärer Baum; benötigt neue Hash-Funktion

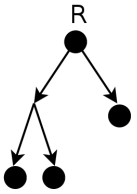

Abbildung 9: Kein balancierter Baum, Höhe größer als 26 vgl. siehe Paper

7 Literaturhinweis

[1] Auditable, Anonymous Electronic Cash Extended Abstract:
http://www.cs.tau.ac.il/~amnon/Papers/ST.crypto99.pdf

Weiterführende Arbeiten (Tomas Sander and Amnon Ta-Shma):
[2] Flow Control: A new Approach for Anonymity Control in Electronic Cash Systems
[3] Blind, Auditable Membership proofs
http://www.cs.tau.ac.il/~amnon/

[4] Wikipedia, http://de.wikipedia.org/wiki/Balancierter_Baum

[5] http://www.doxpara.com/md5_someday.pdf

[6] http://www.heise.de/newsticker/meldung/57038

Wertvolle Hinweise zur Problematik mit der Datenstruktur stammen von Prof. Hauck, Tübingen.

Fazit:

Wie man im Folgenden sieht, ergeben sich aus der sehr komplizierten Arbeit von T.Sander und A.Ta-Shma doch wichtige Erkenntnisse. Wie in Abbildung 10 zu sehen ist ergaben sich anfangs noch einige Schwierigkeiten. In Abbildung 11 ist es fast gelungen alle Blätter zu Gold und damit Geld zu machen, an Haltbarkeit und Materialeigenschaften wird noch gearbeitet. Bisher kann man leider nur abhängig von der Jahreszeit Münzen (und somit Blätter) entfernen. Im Frühling kommt es dann zu einer spontanen Inflation, da nicht vorhersehbar ist, wie viele neue Münzen erzeugt werden.

Abbildung 10: kein balancierter Baum

Abbildung 11: Der Durchbruch: Laburnum anagryroides