Hauptseminar Kryptographische Verfahren
Digitales Bargeld

Peter Palfrader (palfrade@cs.tum.edu)
Betreuer: Ingo Rohloff (rohloff@cs.tum.edu)

Inhaltsverzeichnis

1 Einleitung

Der Tausch Ware gegen Ware war sicherlich nach der Eigenversorgung der Anfang vom Handel der Menschen untereinander. Da jedoch der Wert der Ware von verschiedenen Parteien unterschiedlich beurteilt wurde, gingen die Menschen ziemlich schnell dazu über, Gegenstände, die allgemeinhin als wertvoll angesehen wurden, wie seltene Felle, Muscheln, Salz und später wertvolle Metalle, als Zahlungsmittel einzusetzen.

In der heutigen Zeit, welche geprägt ist durch weltumspannende Netzwerke, ist es nicht mehr möglich Gegenstände physikalisch auszutauschen um so etwas zu bezahlen. Es braucht neue Mittel und Wege um den Zahlungsvorgang auch in diesen Medien sicher und anonym durchführen zu können.

2 Grundlagen

2.1 Blinde Signaturen

Im Allgemeinen macht es Sinn, dass er Unterschreiber einer Nachricht diese Lesen kann. Es gibt aber auch Ausnahmefälle, in denen das genaue Gegenteil gewünscht ist, nämlich dass der Unterschreiber die Nachricht gar nicht kennen soll.

Ein Verfahren zur Erzeugung von blinden Signaturen ist ein Protokoll, in dem eine Person A einem Unterzeichner B ein Dokument vorlegen kann, so dass

- B die Nachricht nicht sieht und
- A eine gültige Unterschrieft von B unter das Dokument erhält

Beispiel 2.1 *Bob ist Notar. Alice will, dass dieser ein Dokument für sie signiert, will aber nicht, dass Bob weiss was er da eigentlich unterschreibt. Bob interessiert sich nicht für den Inhalt des Dokuments, er zertifiziert ja nur, dass er das Dokument zu einer bestimmten Zeit in Händen hielt.*

1. Alice multiplizert ihr Dokument mit einem Blindfaktor (*blinding factor*).
2. Alice schickt das unkenntlich gemachte Dokument an Bob.
3. Bob unterschreibt das Dokument.
4. Alice dividiert den Blindfaktor wieder aus dem unterschriebenen Dokument heraus und bekommt das von Alice unterschriebene Original.

Dies ist natürlich nur sinnvoll wenn Signatur und Multiplikation kommutativ sind, also nach dem herausdividieren des Blindfaktors die Signatur immer noch gültig ist.

Die Idee von blinden Signaturen stammt von David Chaum, welcher auch deren erste Implementation entwickelte. Diese beruht auf RSA:

Bob hat den öffentlichen Schlüssel e, einen privaten Schlüssel d und den öeffentlichen Modul n. Alice will, dass Bob ein Dokument m blind unterschreibt:

1. Alice wählt eine zufällige Zahl k, sodass $1 < k < n$, und macht ihr Dokument unkenntlich:

   $$t = mk^e \mod n$$

2. Bob signiert t

   $$t^d = (mk^e)^d) \mod n$$

3. Alice macht t^d wieder sichtbar indem sie

$$s = \frac{t^d}{k} \mod n$$

berechnet.

4. Das Ergebnis ist

$$s = m^d \mod n$$

Dass s wie gewünscht gleich $m^d \mod n$ ist, kann leicht gezeigt werden:

$$t^d \equiv (mk^e)^d \equiv m^d k \pmod n$$

also ist

$$\frac{t^d}{k} = \frac{(mk^e)^d}{k} = m^d$$

2.2 Aufteilen von Geheimnissen – Secret Splitting

Beispiel 2.2 *Alice hat ein Geheimnis und möchte es unter zwei Personen aufteilen. Jeder soll ein Stück bekommen, das für sich alleine gesehen absolut wertlos ist. Kombiniert man jedoch die zwei Teile, so soll das Geheimnis sichtbar werden.*

Eine einfache Methode dies zu machen ist folgende:

1. Man erzeugt einen zufälligen Bitstring r der genau so lang ist wie m selbst.
2. Dann berechne man $s = m \oplus r$.
3. s und r sind die beiden Teile des Geheimnisses.

Um m aus s und r zu bekommen berechnet man einfach wieder $s \oplus r$.

Die Sicherheit dieses Systems ist perfekt. Es entspricht der Verschlüsselung mit einem One-Time-Pad, wobei sowohl Kryptogramm als auch Schlüssel als Teile des Geheimnisses verwendet werden.

Wie leicht zu sehen ist funktioniert dieser Algorithmus leicht angepasst auch fuer mehr als zwei Personen.

2.3 Bit Commitment

Sehr häufig steht man in der Praxis vor dem Problem, dass man im Nachhinein beweisen muss, dass man sich vor dem Zeitpunkt t auf einen gewissen Wert b festgelegt hat. Legt man diesen Wert offen, ist das sicher kein Problem. Schwieriger wird es, wenn man den Wert nicht preisgeben will.

Bei physischer Anwesenheit könnte man den Wert in einem Umschlag versiegeln und bei einem Notar hinterlegen und darauf hoffen, dass der Notar ehrlich ist. Die Kryptographie bietet aber bessere Protokolle die ein Betrügen praktisch unmöglich machen.

Beispiel 2.3 *Alice arbeitet an der Wall Street und möchte Bob davon überzeugen, dass sie ein gutes Gespühr für Aktienkurse hat. Bob verlangt, dass Alice fünf Aktien auswählt, damit er anschliessend die Kursentwicklung analysieren kann. Legt nun Alice ihr Wahl offen, koennte Bob diese kaufen und Alice ginge für ihre Beratung leer aus. Wir benötigen also ein Verfahren, welches es Alice nicht ermöglicht, ihre Entscheidung im Nachhinein zu ändern und welches es für Bob unmöglich macht herauszufinden für was sich Alice entschieden hat, bevor der Zeitpunkt t erreicht ist.*

Protokolle, welche uns dies ermöglichen, werden *Commitment* Protokolle genannt. Es gibt mehrere Varianten dieser Protokolle wobei ich hier nur kurz zwei verschiedene vorstellen will.

2.3.1 Commitment mit symetrischen Cyphern

1. Bob generiert einen zufälligen Bitstring R und schickt ihn an Alice

2. Alice erzeugt eine Nachricht aus Bobs Zufallsstring und einem Bit b auf das sie sich festlegen möchte (praktisch können das dann auch mehrere Bits sein).

   Diese verschlüsselt sie mit einem zufälligen Schlüssel K und schickt das Ergebnis zurück an Bob.

   $$C = E_K(b, R)$$

Nach diesen zwei Schritten ist der Festlegungsteil abgeschlossen. Alice kann ihre Wahl b nicht mehr ändern. Ist der Zeitpunkt gekommen, zu dem Alice ihr Commitment offenlegen soll, geht das Protokoll weiter:

3. Alice schickt K an Bob

4. Bob entschlüsselt die Nachricht und somit das Bit b. Er überprüft auch seinen zufälligen String R um zu sehen, dass Alice nicht betrogen hat.

Die zufällige Kette R ist sehr wichtig in diesem Protokoll. Gäbe es sie nicht, könnte Alice problemlos nach einem anderen K suchen, das bei der Entschlüsselung in Schritt 4 für das gesendete C das von ihr gewünschte Bit b ergibt.

Da es das R jedoch gibt muss Alice, wenn sie betrügen will ein K finden, welches bei der Dechifrierung nicht nur das gewünschte b, sondern auch das R korrekt erzeugt. Ist der verwendete Verschlüsselungsalgorithmus gut, ist die Wahrscheinlichkeit dass Alice ein solches K findet vernachlässigbar klein.

2.3.2 Commitment mit Einwegfunktionen

Anstelle von symetrischer Verschlüsselung kann man auch eine kollisionsfreie Einwegfunktion f verwenden.

1. Alice generiert zwei zufällige Bitstrings R_1 und R_2.
2. Alice erzeugt eine Nachricht aus den Zufallsstrings und b:

   $$(R_1, R_2, b)$$

3. Alice berechnet den Wert der Funktion und sendet diesen zusammen mit einer der beiden zufälligen Ketten an Bot

   $$f(R_1, R_2, b), R_1$$

Hier ist der Teil des Commitments abgeschlossen. Alice kann nicht mehr zurück. Die Einwegfunktion in 3 hindert Bob daran, b durch probieren zu bekommen.

Wenn der Zeitpunkt gekommen ist, zu dem Alice ihr Bit offenlegen muss, wird das Protokoll weitergeführt.

4. Alice schickt Bob die ursprüngliche Nachricht.

   $$(R_1, R_2, b)$$

5. Bob verifiziert, dass R_1 mit dem in 3 empfangenen üebereinstimmt und berechnet $f(R_1, R_2, b)$. Stimmt auch das mit dem empfangenen üeberein, so ist b gültig.

2.4 Blum- und Williamszahlen

Definition 2.1 *N wird* Blumzahl $[B1]$ *genannt, wenn $N = PQ$ (mit $P, Q \in \mathbb{P}$) und $P = 3 \mod 4$ und $Q = 3 \mod 4$.*

Definition 2.2 *N wird* Williamszahl $[W]$ *genannt, wenn $N = PQ$ (mit $P, Q \in \mathbb{P}$) und $P = 3 \mod 8$ und $Q = 7 \mod 8$. Man beachte dass Williamszahlen spezielle Blumzahlen sind und somit alle Eigenschaften die für Blumzahlen gelten insbesondere auch für Williamszahlen gelten.*

Definition 2.3 *Sei $N = PQ$. Man kann $\mathbb{Z}_N^*$ nun in 4 Klassen teilen:*

$$\begin{array}{lcl}
\mathbb{Z}_{(1,1)} & = & \{x \in \mathbb{Z}_N^* \mid \left(\frac{x}{P}\right) = 1 \wedge \left(\frac{x}{Q}\right) = 1\} \\
\mathbb{Z}_{(1,-1)} & = & \{x \in \mathbb{Z}_N^* \mid \left(\frac{x}{P}\right) = 1 \wedge \left(\frac{x}{Q}\right) = -1\} \\
\mathbb{Z}_{(-1,1)} & = & \{x \in \mathbb{Z}_N^* \mid \left(\frac{x}{P}\right) = -1 \wedge \left(\frac{x}{Q}\right) = 1\} \\
\mathbb{Z}_{(-1,-1)} & = & \{x \in \mathbb{Z}_N^* \mid \left(\frac{x}{P}\right) = -1 \wedge \left(\frac{x}{Q}\right) = -1\}
\end{array}$$

Die Klasse $\mathbb{Z}_{(1,1)} := QR_N$ *ist die Klasse der Quadratischen Reste.* $SetZ_{(1,-1)}$, $SetZ_{(-1,1)}$, $SetZ_{(-1,-1)} := QNR_N$ *sind die Klassen der Quadratischen Nichtreste.*

Satz 1 *Sei* N *eine Blumzahl und* $x \in QR_N$*. Dann gibt es vier Zahlen* y_1, y_2, y_3, y_4, *sodass* $y_i^2 \equiv x (mod\ N)$ *und* $y_1 \in \mathbb{Z}_{(1,1)}$, $y_2 \in \mathbb{Z}_{(1,-1)}$, $y_3 \in \mathbb{Z}_{(-1,1)}$, $y_4 \in \mathbb{Z}_{(-1,-1)}$.

Es gilt weiters: $y_1 \equiv -y_4 (mod\ N)$, $y_2 \equiv -y_3 (mod\ N)$, $\left(\frac{y_1}{N}\right) = \left(\frac{y_4}{N}\right) = 1$, *und* $\left(\frac{y_2}{N}\right) = \left(\frac{y_3}{N}\right) = -1$.

Man beachte, dass aus Satz 1 unmittelbar folgt, dass die Quadratwurzel y einer Zahl x mit einer 2 Bit Information einteutig bestimmt ist, nämlich ob $\left(\frac{y}{N}\right) = 1$ oder -1, und ob $y < \frac{N}{2}$ oder nicht.

Satz 2 *Sei* $N = PQ$ *eine Williamszahl. Dann ist für jedes* $x \in \mathbb{Z}_N^*$ *entweder* x, $-x$, $2x$, *oder* $-2x$ *in* QN_R.

Weiters für $a, b \in \{-2, -1, 1, 2\}$ *und* $a \neq b$: $ax \in QN_R \Rightarrow bx \notin QN_R$.

Aus Satz 2 folgt, dass man aus jedem x durch Multiplikation mit entweder 1, 2, oder -2 einen quadratischen Rest modulo N bekommt, wenn nicht x selbst schon einer ist.

Definition 2.4 *Sei* N *eine Williamszahl und* $x \in QR_N$.

$$\left[x^{\frac{1}{2}} \mod N\right] = y$$

mit $y^2 = x \mod N$ *und* $y \in QR_N$.

$$\left[x^{\frac{1}{2}} \mod N\right]_1 = y'$$

mit $y'^2 = x \mod N$, $\left(\frac{y'}{N}\right) = 1$ *und* $0 < y' < \frac{N}{2}$.

$$\left[x^{\frac{1}{2}} \mod N\right]_{-1} = y''$$

mit $y''^2 = x \mod N$, $\left(\frac{y''}{N}\right) = -1$ *und* $0 < y'' < \frac{N}{2}$.

Definition 2.5 *Sei* N *eine Williamszahl und* $x \in QR_N$.

$$\langle z \rangle_{QR} = dz \mod N$$

mit $d \in \{\pm 1, \pm 2\}$ *und* $dz \mod N \in QR_N$.

$$\langle z \rangle_1 = d'z \mod N$$

mit $d' \in \{1, 2\}$ *und* $\left(\frac{d'z}{N}\right) = 1$.

$$\langle z \rangle_{-1} = d''z \mod N$$

mit $d'' \in \{1, 2\}$ *und* $\left(\frac{d''z}{N}\right) = -1$.

$\langle z \rangle_{QR}$, $\langle z \rangle_1$, $\langle z \rangle_{-1}$ sind also nur Schreibweisen, um aus z einen quadatischen Rest zu machen, ein Ergebnis mit Jacobisymbol 1 zu erhalten und ein Ergebnis mit Jacobisymbol -1 zu erhalten.

3 Digitales Bargeld

3.1 Anforderungen

Damit digitales Geld überhaupt funktioniert, muss es einige Eigenschaften erfüllen:

Unabhängigkeit Die Sicherheit von elektronischen Bargeld darf nicht von physikalischen Gegebenheiten abhängen. Dann kann man digitales Geld über Netzwerke transportieren.

Sicherheit Digitales Bargeld darf weder kopiert noch mehrmals verwendet werden können. Des weiteren müssen Fälschungen ausgeschlossen werden.

Privatsphäre Die Privatsphäre des Benutzers ist gesichert. Niemand kann eine Verbindung zwischen einer Person und ihrer getätigten Einkäufe herstellen.

Bezahlung Off-Line Ein Geldtransfer vom Kunden zum Händler soll ohne Online-Verbindung zur Bank möglich sein.

Übertragbarkeit Es soll möglich sein, Geld an andere Benutzer weiterzugeben.

und eventuell noch:

Teilbarkeit Einen Geldbetrag soll man in kleinere Beträge aufteilen können (Die Summe dieser Beträge soll natürlich gleich dem Ausgangsbetrag sein).

Herkömmliches Bargeld erfüllt bis auf *Unabhängigkeit* alle diese Anforderungen mehr oder weniger gut.

4 Ein einfaches Protokoll

Ein einfaches Protokoll könnte zum Beispiel so aussehen:

1. Alice fertigt eine Zahlungsanweisung über 100 Euro an.
2. Die Bank unterschreibt diese und bucht 100 Euro von Alice' Konto ab.
3. Alice bezahlt mit der Zahlungsanweisung bei einem Händler Bob.
4. Dieser überprüft die Signatur der Bank.
5. Bob bringt die Zahlungsanweisung zur Bank.
6. Die Bank überprüft die Signatur und schreibt Bob die 100 Euro gut.

Dieses Protokoll hat eine grosse Schwäche: Es ist sowohl Alice als auch Bob möglich, die Zahlungsanweisung mehrmals zu verwenden.

4.1 Double Spending

Um die mehrfache Verwendung des Geldes auszuschliessen kann man obiges Protokoll so modifizieren, dass eine Eindeutigkeitsfolge Teil jeder Zahlungsanweisung ist.

1. Alice fertigt eine Zahlungsanweisung über 100 Euro an. Diese Zahlungsanweisung bekommt eine zufällige und genügend lange (um eine Mehrfachverwendung sehri unwahrscheinlich zu machen) Eindeutigkeitsfolge.
2. Die Bank unterschreibt diese und bucht 100 Euro von Alice' Konto ab.
3. Alice bezahlt mit der Zahlungsanweisung bei einem Händler Bob.
4. Dieser überprüft die Signatur der Bank.
5. Bob bringt die Zahlungsanweisung zur Bank.
6. Die Bank überprüft die Signatur.
7. Die Bank überprüft, ob eine Zahlungsanweisung mit der Eindeutigkeitsfolge bereits eingegangen ist. Wenn nicht, wird dem Händler das Geld gutgeschrieben und die Eindeutigkeitsfolge in einer Datenbank gespeichert. Anderenfalls geht der Händler leer aus.

Dieses Protokoll verhindert erfolgreich, dass der Händler ein und dieselbe Zahlungsanweisung mehrfach bei einer Bank geltent macht. Leider ist es Alice aber weiterhin möglich zu betrügen: Sie kann mit dem selben Geld in mehreren Geschaften einkaufen. Der Händler der als erstes zur Bank geht, bekommt das Geld.

Des weiteren ist die *Anonymität* des Kunden nicht mehr gewährleistet. Die Bank weiss eindeutig, wem welche Zahlungsanweisungen mit welcher Eindeutigkeitsfolge ausgegeben wurden.

4.2 Anonymität

Um die Privatsphäre des Kunden sicherzustellen, muss verhindert werden, dass die Bank die Eindeutigkeitsfolge mit dem Kunden in Verbindung bringen kann. Dazu bedienen wir uns den in 2.1 beschriebenen Blinden Signaturen:

1. Alice fertigt n Zahlungsanweisungen über jeweils den gleichen Betrag an und versieht jede mit einer anderen zufälligen Eindeutigkeitsfolge.

2. Alice macht alle n Zahlungsanweisungen mit einem Protokoll für Blinde Signaturen unkenntlich.

3. Die Bank wählt zufällige $n-1$ Anweisungen aus, lässt diese von Alice sichtbar machen und überprüft den Betrag. Stimmen alle Beträge überein, ist die Bank davon überzeugt, dass auch die übriggebliebene Zahlungsanweisung über diesen Betrag ist.

4. Die Bank unterschreibt diese und bucht den Betrag von Alice' Konto ab.

5. Alice macht die Zahlungsanweisung mit ihrer Eindeutigkeitsfolge wieder sichtbar und bezahlt bei Bob.

6. Bob überprüft die Signatur der Bank.

7. Er bringt die Zahlungsanweisung zur Bank.

8. Die Bank überprüft die Signatur.

9. Die Bank überprüft, ob eine Zahlungsanweisung mit der Eindeutigkeitsfolge bereits eingegangen ist. Wenn nicht, wird dem Händler das Geld gutgeschrieben und die Eindeutigkeitsfolge in einer Datenbank gespeichert. Anderenfalls geht der Händler leer aus.

Da es für die Bank unmöglich ist, die Eindeutigkeitsfolge der unterschriebenen Zahlungsanweisung mit Alice in Verbindung zu bringen, ist die Privatsphäre von Alice geschützt.

Betrag	EUR 100
Eindeutigkeitsfolge	`94C07C9F`
Identitätsstrings	$I_1 = (I_{1_L}, I_{1_R})$
	$I_1 = (I_{2_L}, I_{2_R})$
	$\vdots$
	$I_m = (I_{m_L}, I_{m_R})$

Abbildung 1: Zahlungsanweisung: Anonym und trotzdem sicher vor Double Spending.

4.3 Double Spending, II

Das einzig verbleibende Problem besteht darin, dass Alice das Geld noch immer mehrmals ausgeben kann und dadurch den Händler schädigen kann.

Dies in einem offline Protokoll zu verhindern dürfte ziemlich schwer sein – wenn nicht unmöglich. Deshalb begnügt man sich damit in einem solchen Falle die Identität des Betrügers zu bekommen ohne natürlich die Anonymität von korrekten Nutzern zu beeinträchtigen.

1. Alice fertigt n Zahlungsanweisungen über jeweils den gleichen Betrag an und versieht jede mit einer anderen zufälligen Eindeutigkeitsfolge.

   Auf *jede*r Zahlungsanweisung befinden sich weiters m Paare von Identitätsbitfolgen $I_1 \dots I_m$. Jedes dieser Paare wird wie folgt generiert: Alice erzeugt eine Zeichenfolge die ihren Namen, Ihre Adresse und alle weiteren Informationen enthält, für die sich die Bank im Falle eines Betrugs interessieren könnte. Dann teilt sie diese Information mit Hilfe eines Secret-Splitting Protokolls. Anschliessend legt sie sich mit einem Bit-Commitment Protokoll auf jedes der Teile fest.

   Zum Beispiel besteht I_{23} aus zwei Teilen: I_{23_L} und I_{23_R}. Jeder Teil ist ein Packet auf das sich Alice festgelegt hat und sie kann gebeten werden dieses zu öffnen. Die korrekte Öffnung kann sofort verifiziert werden. Jedes Paar (z.B. I_{23_L} und I_{23_R} nicht jedoch I_{23_L} und I_{42_R}) verrät Alice' Identität.

   Eine Zahlungsanweisung würde also wie in Abbildung 1 gezeigt aussehen.

2. Alice macht alle n Zahlungsanweisungen mit einem Protokoll für Blinde Signaturen unkenntlich.

3. Die Bank wählt zufällige $n-1$ Anweisungen aus, lässt diese von Alice sichtbar machen und überprüft den Betrag. Stimmen alle Beträge überein, ist die Bank davon überzeugt, dass auch die übriggebliebene Zahlungsanweisung über diesen Betrag ist.

4. Die Bank unterschreibt diese und bucht den Betrag von Alice' Konto ab.

5. Alice macht die Zahlungsanweisung sichtbar und bezahlt bei Bob.

6. Bob überprüft die Signatur der Bank.

7. Bob bittet Alice zufällig jeweils die linke oder rechte Hälfe jedes Identitätsstrings zu öffnen. Praktisch funktioniert das so, dass Bob Alice einen zufälligen m bit langen String gibt und Alice die rechte oder die linke Seite I_k öffnet je nachdem ob b_k 0 oder 1 ist.

8. Er bringt die Zahlungsanweisung zur Bank.

9. Die Bank überprüft die Signatur.

10. Die Bank überprüft, ob eine Zahlungsanweisung mit der Eindeutigkeitsfolge bereits eingegangen ist.

    - Wenn nicht, wird dem Händler das Geld gutgeschrieben und die Eindeutigkeitsfolge mit allen offenen Identitätsstrings gespeichert.
    - Ist die Eindeutigkeitsfolge bereits vorhanden, vergleicht sie, welche Seiten der I_k bei der bereits gespeicherten und der neuen Zahlungsanweisung geöffnet wurden. Sind es dieselben Seiten in beiden Anweisungen, so hat Bob betrogen und ein und dasselbe Geld zweimal zur Bank gebracht. Sind sie unterschiedlich so hat Alice betrogen. In diesem Fall existiert ein x sodass bei der ersten Zahlungsanweisung I_{x_L} und bei der zweiten I_{x_R} oder umgekehrt offen sind. Die beiden Hälften lassen sich nun kombinieren und die Bank hat die Identität des Betrügers: von Alice.

Betrachten wir dieses letzte Protokoll und mögliche Angriffspunkte.

Kann Alice betrügen? Da digitales Bargeld im Grunde ja nichts anderes ist als ein paar Bits und Bytes kann sie es problemlos kopieren. Das erste mal mit diesem Geld zu bezahlen ist auch kein Problem. Alles läuft glatt. Beim zweiten Mal aber wird der Händler ihr einen anderen Selektorstring in 7 geben. Wenn der Händler das Geld dann zur Bank bringt, kommt heraus, dass Alice das Geld doppelt verwendet hat. Die Wahrscheinlichkeit, dass sie denselben Selektorstring bekommt, ist verschwindend gering, nämlich $\frac{1}{2^m}$.

Des weiteren könnte Alice versuchen, eine gültige Zahlungsanweisung zu bekommen, auf der gar nicht ihr Name steht. Die Wahrscheinlichkeit dafür ist gar nicht so gering: $frac1n$. Macht man jedoch n gross genug ist auch das kein Problem.

Bobs Chancen zu betrügen sind noch viel schlimmer. Das einzige was er tun kann, ist ein und dieselbe Zahlungsanweisung mehrmals zur Bank zu bringen. Da aber immer dieselben Seiten der Identitatsstrings offengelegt wurden, ist sofort klar, dass er betrügt.

Eve, die den Zahlungsvorgang in 7 belauscht hat, oder aber nachher bei Bob eingebrochen ist und das Geld kopiert hat, kann betrügen indem sie das Geld vor Bob zur Bank bringt. Was die Sache noch verschlimmert ist, dass wenn Bob das digitale Bargeld zur Bank bringt, er plötzlich als der Betrüger dasteht.

Kann die Bank betrügen und eventuell die Privatsphäre von Alice verletzen? Das geht wiederum nicht, da ja die Eindeutigkeitsfolge, welche jede Zahlungsanweisung eindeutig identifiziert, in 4 durch das Blinde Signatur Protokoll geschützt ist.

Einige Nachteile hat dieses Protokoll jedoch immer noch: Erstens werden die Zahlungsanweisungen durch die vielen Identitätsstrings ziemlich groß. Des weiteren ist es nicht möglich dass Alice das Geld an einen anderen Benutzer weitergibt und dieser dann damit bezahlen kann. Auch ist es nicht möglich, dass Alice den Betrag auf der Zahlungsanweisung aufteilt und in mehreren Einkäufen ausgibt. Der gesamte Betrag muss immer auf einmal und bei einem Händler ausgegeben werden.

Ein Protokoll das verspricht diese Probleme zu lösen wurde von Tatsuaki Okamoto und Kazuo Ohta unter dem Namen *Universal Electronic Cash*[2] vorgestellt:

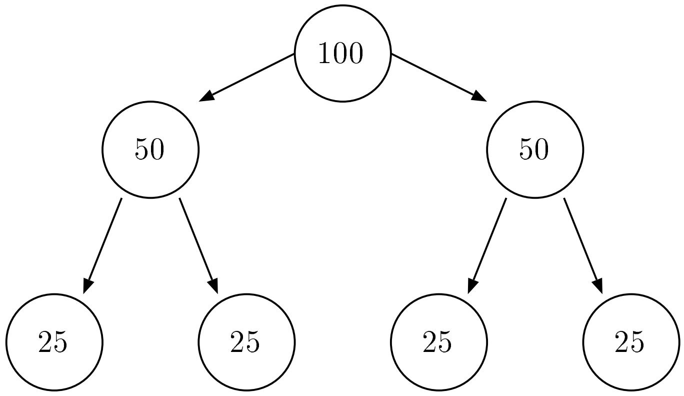

Abbildung 2: Hierarchical Structure Table

5 Universal Electronic Cash

Einer der Hauptvorteile von Universal Electronic Cash (UEC) ist die Teilbarkeit von Bargeld. Das heisst, dass ein Betrag von C Euro in viele kleine Beträge kleiner C aufgeteilt werden kann, sodass die Summe aller dieser Beträge wieder genau C ist.

5.1 Hierarchical Structure Table

Die Hierarchical Structure Table ist ein binärer Baum der Tiefe t. Ein einfaches Beispiel ist in Abbildung 2 zu sehen, welche ein HST der Tiefe 3 üeber den Wert von 100 Euro darstellt. Ein Koten der i-ten Ebene entspricht einem Wert von $\frac{100}{2^{i-1}}$. Da dieser Baum nur eine Tiefe von 3 hat, ist die kleinste Einheit die damit ausgegeben werden kann $25 = \frac{100}{2^{3-1}}$ Euro.

Damit dieses System überhaupt funktioniert, muss der Baum folgende Eigenschaften besitzen:

1. Der Wert eines Knoten N entpricht exakt der Summe der Werte aller Söhne von N.
2. Wurde ein Knoten einmal verwendet (sprich, es wurde damit gezahlt), kann weder einer der Väter dieses Knotens noch einer seiner Söhne jemals wieder verwendet werden.
3. Kein Knoten kann zweimal verwendet werden.

Später werde wir in den Protokollen Γ und Ω Werte eines Knoten brauchen. Diese berechnen sich wie folgt:

$$\Omega_{i,0} = \langle f_\Omega(C \circ 0 \circ N_i) \rangle_1$$

$$\Omega_{i,j_1\ldots j_t} = \langle f_\Omega(C \circ j_1 \circ \ldots \circ j_t \circ N_i)\rangle_1$$

$$\Gamma_{i,0} = \langle f_\Gamma(C \circ 0 \circ N_i)\rangle_{QR}$$

$$\Gamma_{i,00} = \left[\Gamma_{i,0}^{\frac{1}{2}} \mod N_i\right]_{QR}$$

$$\Gamma_{i,01} = \left[\Omega_{i,0}\Gamma_{i,0}^{\frac{1}{2}} \mod N_i\right]_{QR}$$

rekursiv.

$$X_{i,j_1\ldots j_t} = \left[\Gamma_{i,j_1\ldots j_t}^{\frac{1}{2}} \mod N_i\right]_{-1}$$

5.2 Protokolle

Als Vorbereitung für die Protokolle hat die Bank A mehrere RSA Schlüsselpaare generiert: (e_A, n_A, d_A), (e'_A, n'_A, d'_A), (e''_A, n''_A, d''_A), ..., wobei (e_A, n_A), (e'_A, n'_A), ... die öffentlichen Schlüssel sind und d_A, d'_A, ... die privaten.

Die Bank hat alle öffentlichen Schlüssel bekanntgegeben. Jeder Schlüssel hat eine bestimmte Funktion: (e_A, n_A) ist für *Lizenzen* gedacht, während (e'_A, n'_A, d'_A), (e''_A, n''_A, d''_A), ... für *Geldscheine* je nach Wert (100 Euro, 500 Euro, etc.).

Des weiteren setzt die Bank auch einen Sicherheitsparameter $K = O(|n_A|) = O(|n'_A|) = \ldots$ fest (zum Beispiel 40).

Zusätzlich werden noch drei Hashfunktionen publiziert: f_Γ, f_Λ, f_Ω. Diese sollten möglichst gut streuen. Eigenschaften wie Unumkehrbarkeit oder Kollisionsfreiheit sind nicht gefordert.

Ein Kunde P hat eine Kontonummer ID_P und generiert ein RSA Schlüsselpaar (e_P, n_P, d_P) wobei wieder (e_P, n_P) zum Zwecke digitaler Signaturen veröffentlicht werden.

5.2.1 Kontoeröffnung

Wenn ein Kunde P ein Konto bei der Bank A eröffnet stellt diese dem Kunden eine *Lizenz* B zur Verwendung im digitalen Zahlungsverkehr aus. Um B zu bekommen führt P folgendes Protokoll mit A durch:

1. Kunde P wählt eine zufällige Zahl a_i und die Williamszahl N_i mit zwei grossen Primfaktoren P_i und Q_i ($N_i = P_iQ_i$), wobei $P_i \equiv 3 (\mathrm{mod}\ 8)$ und $Q_i \equiv 7 (\mathrm{mod}\ 8)$ für alle $i = 1 \ldots K$.

2. P bildet und sendet K blinde Kandidaten W_i zur Bank A.

$$W_i = r_i^{e_A} g(I_i \circ N_i) \mod n_A \qquad \text{für } 1 \leq i \leq K$$

wobei $r_i \in \mathbb{Z}_{n_A}$ eine Zufallszahl ist, g eine angemessene Hashfunktion und

$$S_i = ID_p \circ a_i \circ (g(ID_p \circ a_i))^{d_P} \mod n_P = S_{1,i} \circ S_{2,i},$$

$$I_{1,i} = S_{1,i}^2 \mod N_i, I_{2,i} = S_{2,i}^2 \mod N_i,$$

$$I_i = I_{1,i} \circ I_{2,i}$$

3. A wählt zufällige $\frac{K}{2}$ Kandidaten und lässt sich von P jeweils a_i, P_i, Q_i, $(g(ID_p \circ a_i))^{d_P} \mod n_P$, ID_p, und r_i zur Überprüfung offenlegen. Sind die überprüften W_i korrekt signiert die Bank die restlichen W_i. Andernfalls ist P als Betrüger entlarvt und A beendet das Protokoll (o.B.d.A. nehmen wir an, A habe $\frac{K}{2}+1$, $\frac{K}{2}+2$, $\cdots$, K überprüft).

4. A sendet an P

$$\left(\prod_{i=1}^{\frac{K}{2}} W_i\right)^{d_A} = \left(\prod_{i=1}^{\frac{K}{2}} r_i^{e_A} g(I_i \circ N_i)\right)^{d_A} \mod n_A$$

5. P macht seine Lizenz B wieder sichtbar und erhählt

$$B = \left(\prod_{i=1}^{\frac{K}{2}} g(I_i \circ N_i)\right)^{d_A} \mod n_A$$

5.2.2 Abheben von Geld

Nach der Kontoeröffnung kann nun P einen elektronischen Geldschein C von der Bank abheben.

Das Schlüsselpaar (e'_A, n'_A) der Bank sei das Paar für den gewünschten Betrag C.

1. P wählt eine zufällige Zahl b und bildet und sendet Z an A

$$Z = r^{e'_A} g(B \circ b) \mod n'_A$$

2. A signiert Z, gibt also $Z^{d'_A} \mod n'_A$ an P und zieht den Betrag von P's Konto ab.

3. P macht den Gelschein wieder sichtbar und bekommt

$$C = (g(B \circ b))^{d'_A} \mod n'_A$$

5.2.3 Bezahlen

1. Um nun sein frisch erworbenes Bargeld wieder unter das Volk zu bringen, muss P erst einmal wissen, welchen Knoten er ausgeben will. Für diesen Knoten $(j_i \dots j_t)$ berechnet P für $i \in 1 \dots \frac{K}{2}$ das X, wie in 5.1 beschrieben.

2. Dann schickt P $(I_i,\ N_i,\ X_{i,j_1 \dots j_t})$ (für alle $i \in 1 \dots \frac{K}{2}$) und (B,C) an den Händler V.

3. Dieser verifiziert die Signatueren der Bank für B (die Lizenz des Kunden) und C (das Bargeld). Ausserdem überprüft V, ob

   $$\left(\frac{X_{i,j_1 \dots j_t}}{N_i}\right) = 1$$

   und ob

   $$X_{i,j_1 \dots j_t}^{2^t} = d_i \Omega_{i,j_1 \dots j_t}^{2^{t-1} j_t} \Omega_{i,j_1 \dots j_{t-1}}^{2^{t-2} j_{t-1}} \dots \Omega_{i,j_1}^{2 j_2} f_\Gamma(C \circ 0 \circ N_i) \mod N_i$$

   mit $d_i \in \{\pm 1, \pm 2\}$. V überprüft also, ob P korrekt Wurzeln gezogen hat und somit die Faktorisierung von N kennt (Wie haben ja angenommen, dass Wurzeln ziehn schwer ist, wenn die Faktorisierung $N = PQ$ nicht bekannt ist). Ist dies der Fall, ist P der rechtmäßige Besitzer des Geldes.

4. V wählt zufällige Bits $E_{i,j_1 \dots j_t}$ und schickt sie an P.

5. P berechnet nun

   $$Y_{i,j_1 \dots j_t} = \left[\Lambda_{i,j_1 \dots j_t}^{\frac{1}{2}} \mod N_i\right]_{(-1)^{E_{i,j_1 \dots j_t}}}$$

   mit $\Lambda_{i,j_1 \dots j_t}$ ähnlich wie $\Omega_{i,j_1 \dots j_t}$ definiert:

   $$\Lambda_{i,j_1 \dots j_t} = \langle f_\Lambda(C \circ j_1 \circ \dots \circ j_t \circ N_i) \rangle_{QR}$$

6. Schlussendlich verifiziert V noch, dass

   $$\left(\frac{\Lambda_{i,j_1 \dots j_t}}{N_i}\right) = (-1)^{E_{i,j_1 \dots j_t}}$$

   also ob die verlangten Jacobiwerte $E_{i,j_1 \dots j_t}$ auch wirklich geliefert wurden, und

   $$\Lambda_{i,j_1 \dots j_t} = d_i' f_\Lambda(C \circ j_1 \circ \dots \circ j_t \circ N_i) \mod N_i$$

   also die Wurzeln richtig gezogen wurden.

   Ist V zufrieden wird die Zahlung akzeptiert.

5.2.4 Gutschrift

Nach Ende des Bezahlungsprotokolls schickt V ein Log eben dieses (H) an die Bank A. Diese speichert H, überprüft es und schreibt, so alles in Ordnung ist, das Geld gut.

5.2.5 Korrektheit

Das System erfüllt die Bedingungen Unabhängigkeit, Privatsphäre und Bezahlung Off-Line.

UCE ist teilbar, wenn alle Anforderungen an die Hierarchical Structure Table erfüllt werden können. Dass der Wert eines jeden Knoten der Summe der Werte seiner Söhne entspricht, ist kein Problem und durch Protokolle lösbar. Interessant sind die beiden anderen Anforderungen: dass kein Knoten doppelt verwendet werden kann und kein Knoten zusammen mit einem seiner Söhne.

Gibt ein Teilnehmer P des Systems ein und denselben Knoten doppelt aus, so erhält die Bank schlussendlich zwei komplette Tabellen Y_i.

Ähnlich den Identitätsstrings im ersten Protokoll, wo jeweils die linke oder rechte Hälfte der Identität sichtbar war, hat man hier entweder einen Wert mit Jacobi Symbol 1 oder -1. Welches Jacobisymbol jeder einzelne Wert hat, wird auch hier wieder vom Händler bestimmt (Schritt 4).

Mit höchster Wahrscheinlichkeit gibt es ein i, sodass einmal Y_i mit Jacobisymbol 1 und einaml Y_i mit Jacobisymbol -1 vorhanden ist. Deshalb ist laut Annahme N leicht faktorisierbar und durch die Kenntnis der Faktoren kennt man laut Annahme die Identität der Person.

Der andere Fall der noch nicht betrachtet wurde ist Spending eines Knotens und einem seiner Söhne. Nehmen wir als Beispiel an, P hätte mit den Knoten Γ_{00} und Γ_{000} gezahlt. Dann hätte P im Laufe der Protokolle X_{00} und X_{000} bekannt gegeben. Da nun $X_{00} = \left[\Gamma_{00}^{\frac{1}{2}} \mod N\right]_{-1}$ und $X_{000}^2 = \left[\Gamma_{00}^{\frac{1}{2}} \mod N\right]_1$ kann N wieder faktorisiert werden. Der Betrüeger ist überführt.

Wie man sieht verhindert auch dieses Protokoll nicht double-spending, aber es deckt den Vorfall ebenfalls befriedigend auf.

5.2.6 Übertragbarkeit

Dieses Protokoll erlaubt es noch nicht, dass Teilnehmer des Systems untereinander Geld austauschen. In [2] beschreiben Tatsuaki Okamoto und Kazuo Ohta auch eine Möglichkeit UCE übertragbar zu machen.

Literatur

[1] Bruce Schneier. *Applied Cryptography* Second Edition; John Wiley & Sons, Inc., 1996

[2] Tatsuaki Okamoto, Kazuo Ohta *Universal Electronic Cash*

[3] Michael O. Rabin *Digitalized Signatures and Public-Key Functions as Intractable as Factorization* Massachusetts Institute of Technology, Laboratory for Computer Science, 1979

[4] Albrecht Beutelspacher, Jörg Schwenk, Klaus-Dieter Wolfenstetter *Moderne Verfahren der Kryptographie: von RSA zu Zero knowledge* Zweite Auflage; Bieweg, 1998

[5] Johannes Buchmann *Einführung in die Kryptographie*
Zweite Auflage; Springer Verlag, 2001