Ausgewählte Sicherheitsaspekte mobiler Netze

Markus Dolic, Mario Fischer

Hauptseminar „Dienste & Infrastrukturen mobiler Systeme“
Wintersemester 03/04
Institut für Informatik
Ludwig Maximilians Universität München
{dolic,fischerm}@informatik.uni-muenchen.de

Zusammenfassung Die vorliegende Arbeit soll einen Einblick in die Sicherheitsproblematik und ihre besondere Bedeutung im Rahmen der mobilen Kommunikation vermitteln. Da ein vollständiger Überblick bei diesem sehr weit gefassten Thema ohnehin kaum zu erreichen ist, werden nach einer allgemeinen Einführung der grundlegenden Begriffe und Verfahren drei ausgewählte Aspekte ausführlicher behandelt.
Im Abschnitt *Digitales Geld* werden Ansätze untersucht, einen elektronischen Ersatz für physisches Bargeld zu schaffen, und dabei dessen positive Eigenschaften möglichst genau abzubilden. Das Kapitel *Location Management mit Wahrung der Privatsphäre* erläutert den Versuch, die in den heute üblichen Mobilfunknetzen mögliche Ortung der Teilnehmer zu verhindern. Abschließend folgt noch, um auch eine bereits etablierte Anwendung zu berücksichtigen, eine Darstellung der *Sicherheitsmechanismen im Wireless Application Protocol*.
Daß die gezeigten Verfahren oft nicht nur im mobilen Bereich sondern generell in Rechnernetzen eingesetzt werden – mit anderen Worten: nicht nur im „Spezialfall“ Mobilkommunikation –, liegt dabei in der Natur der üblichen, meist auf modularen Systemen oder Schichtenmodellen beruhenden Herangehensweise an die Implementierung größerer Netzwerksysteme.

1 Einleitung

Die stetig steigende Marktdurchdringung und allgemeine Akzeptanz mobiler Endgeräte (v.a. Mobiltelefone, aber auch PDAs, Notebooks etc.) war während der letzten Jahre zweifellos ein prägender Faktor des öffentlichen Lebens, der sich wie kein anderer durch nahezu alle Gesellschafts- und Altersschichten zog. Hohe Investitionen der führenden Unternehmen in diesem Bereich ermöglichten überraschend schnelle technologische Fortschritte, deren Auswirkungen im täglichen Leben beispielsweise in Form von mobilen Geräten zutage treten, deren Funktionsumfang selbst technisch interessierten Menschen noch vor wenigen Jahren utopisch erschien.

Die in den vorhandenen Netzen angebotenen Dienste erwecken bei näherer Betrachtung allerdings den Eindruck, mit dieser Entwicklung nicht ganz Schritt

halten zu können; das bereits vor geraumer Zeit als „Killer Application“ gehandelte Einkaufen und Bezahlen per Handy etwa ist ohne großes Aufsehen wieder fast vollständig von der Bildfläche verschwunden – dabei beinhalten gerade derartige Anwendungen zweifellos ein beträchtliches finanzielles Potential.

Es lässt sich sicherlich eine Reihe von Gründen für diese gegenwärtige Lage finden. Einer davon, und gleichzeitig Thema dieser Arbeit, ist die in mobilen Netzen allgemein als kritisch erachtete Sicherheitsproblematik; da sämtliche Daten über ein grundsätzlich offen zugängliches Medium übertragen werden, gestaltet sich die Abwehr möglicher Angriffe hier deutlich schwieriger als in drahtgebundenen Netzwerken. Gleichzeitig bemängeln Kritiker zu Recht die schleichende Aushöhlung der Privatsphäre durch Rückverfolgbarkeit von Transaktionen, eindeutige Identifizierung der Anwender, durchgehende Protokollierung und generell ein falsches Verständnis des Sicherheitsbegriffs.

Eine vollständige Beschreibung aller Sicherheitsmerkmale, Technologien und Schwächen der derzeit gängigen mobilen Systeme würde selbstverständlich jeden Rahmen sprengen. Die vorliegende Ausarbeitung erläutert daher zunächst die für das Verständnis der folgenden Ausführungen notwendigen Grundlagen, und gibt im Anschluß daran einen Einblick in einige (der bescheidenen Meinung der Autoren nach recht anschauliche und nicht uninteressante) sicherheitsrelevante Verfahren und Protokolle.

2 Kryptographische Grundlagen

2.1 Grundbegriffe

Da der Begriff „Sicherheit“ umgangssprachlich zahlreiche Bedeutungen hat, möchten wir zunächst versuchen, eine Definition dieses Begriffes zu geben. Im Zusammenhang mit Netzwerken (und damit auch im Mobilkommunikations-Kontext) soll **Sicherheit** ein Überbegriff für (unter anderem) folgende Eigenschaften sein:

- **Vertraulichkeit**: Eine Nachricht soll *nur* für den/die Empfänger zugänglich sein.
- **Authentizität**: Es soll eindeutig feststellbar sein, ob eine empfangene Nachricht auch tatsächlich vom vorgeblichen Absender stammt.
- **Integrität**: Eine Veränderung der Nachricht auf dem Weg zwischen Sender und Empfänger soll erkannt werden können.
- **Non-Repudiation**: Der Sender einer Nachricht soll im Nachhinein nicht leugnen können, diese verfaßt zu haben.

Wichtigstes Werkzeug, damit ein Kommunikationssystem für unsere Begriffe als *sicher* gelten darf, ist die Verwendung kryptographischer Algorithmen und Protokolle. Es ist üblich, den daran beteiligten Personen folgende Namen zu geben:
Alice und **Bob** kommunizieren miteinander, manchmal sind **Carol** und **Dave** dabei, **Eve** ist (nur) Lauscherin, während **Mallory** der aktive Angreifer, und **Trent** der vertrauenswürdige Vermittler ist.

2.2 Werkzeuge für Protokolle

Grundwerkzeuge, die wir in den anschließend diskutierten Protokolle benötigen werden, sind hier:

- **symmetrische Verschlüsselung**: Alice und Bob vereinbaren vor Beginn des Nachrichtenaustausches einen **gemeinsamen, geheimen Schlüssel** k. Damit werden dann die Nachrichten ver- und entschlüsselt:

$$C = Enc_k(P), P = Dec_k(C)$$

  Die symmetrischen Verfahren gliedern sich in die **Blockchiffrierung**, bei der Datenblöcke fixer Größe auf einen anderen (gleichgroßen) Datenblock ein-eindeutig abgebildet werden, und die **Stromchiffrierung**, bei der jedes Bit des Klartextes sofort in ein verschlüsseltes Bit verwandelt wird (durch XOR-Verknüpfung mit dem Output eines Schlüsselstromgenerators). Blockchiffrierungen sind zustandslos, d.h. jede Eingabe wird immer auf dieselbe Ausgabe abgebildet. Dadurch kann ein Angreifer ein Wörterbuch aufbauen, was unter Ausnutzung der unterschiedlichen Häufigkeit der Buchstaben einer Sprache sehr leicht sein kann. Abhilfe dagegen sind **Ciphermodes**, die intern eine Blockchiffrierung benutzen, ihren Output aber auch von *früheren* Inputs abhängig machen, wie etwa Cipher-Feedback- oder Output-Feedback-Mode.
- **asymmetrische Verschlüsselung**: Hierbei besteht ein Schlüssel von Alice aus einem öffentlichem Teil (**Public Key** *pubAlice*) und einem privaten Teil (**Private Key** *privAlice*). Mit dem Public Key können Nachrichten nur verschlüsselt und nur mit dem Private Key diese wieder entschlüsselt werden.

$$C = Enc_{pubAlice}(P), P = Dec_{privAlice}(C)$$

  Die asymmetrische Verschlüsselung basiert auf (bislang) *NP-harten* mathematischen Problemen, wie etwa die Zerlegung großer Zahlen in ihre Primfaktoren beim bekannten RSA, oder die sog. diskreten Logarithmen: Findet jemand einen effizienten Algorithmus dafür, werden diese Verfahren nutzlos.
- **digitale Signatur**: Sie ähnelt der asymmetrischen Verschlüsselung. Anschaulich (und stark vereinfacht) gesprochen handelt es sich bei der digitalen Signatur um die Verschlüsselung einer Nachricht M mit dem privaten Schlüssel des Unterzeichners: Jeder kann das unterschriebene Dokument S verifizieren, indem er die signierte Nachricht mit dem öffentlichen Schlüssel des Unterzeichners entschlüsselt und prüft, ob etwas „sinnvolles“ herauskommt. Dies läßt sich z.B. erreichen, indem man fordert, daß jedes Dokument, das man unterzeichnet (genauer: dessen *Hashwert* man unterzeichnet, um Choosen-plaintext-Angriffe (2.3) zu verhindern), immer mit einem persönlichem Header beginnen muß.

$$S = Sig_{privAlice}(M), M = Ver_{pubAlice}(S)$$

– **Hash-** oder **Einwegfunktionen**: Sie bilden jede Nachricht M beliebiger Länge auf eine Zahl h, dem sog. **Hashwert** aus einem festen Wertebereich, ab (typischerweise 128 Bit aufwärts).

$$h = hash(M)$$

Wichtige Eigenschaften guter Hashfunktionen sind unter anderem:

- **Non-Reversibilität**: Für einen gegebenen Hashwert muss es praktisch unmöglich sein, die Ursprüngliche oder auch nur irgendeine passende Nachricht zu finden.[1]
- **Kollisionsfreiheit**: Es muß praktisch unmöglich sein, zwei Nachrichten mit gleichem Hashwert zu konstruieren.
- **Repräsentativität**: Jedes Bit eines Textes muss den Hashwert beeinflussen.

2.3 Angriffstypen

Die Arten der Angriffe, die sich aktiv (Daten nur lesbar machen) oder passiv (Daten manipulieren) gegen kryptographische Algorithmen und Protokolle durchführen lassen, werden (unter Anderem) in folgende Typen eingeteilt:

– **Ciphertext-only**: Eve hat nur die verschlüsselten Texte zur Analyse vorliegen. Dies ist natürlich insbesondere in der Mobilkommunikation immer gegeben.
– **Known-plaintext**: Eve kennt zu (einigen) verschlüsselten Texten auch die dazugehörigen Klartexte. (z.B wenn der Anfang eines Textes immer gleich lautet e-Mail Header, HTTP-Header, etc.).
– **Chosen-plaintext**: Eve kennt zu (einigen) verschlüsselten Texten auch die dazugehörigen Klartexte. Zusätzlich kann noch Eve bestimmen, welche Klartexte sie verschlüsselt haben möchte (etwa um Schwachstellen eines speziellen Algorithmus auszunutzen).
– **Chosen-ciphertext**: Eve kennt zu (einigen) verschlüsselten Texten auch die dazugehörigen Klartexte. Zusätzlich kann noch Eve bestimmen, welche verschlüsselten Texte sie gerne entschlüsselt haben möchte.
– **Man-in-the-Middle**: Wenn Alice mit Bob kommunizieren möchte, steht in Wirklichkeit Mallory zwischen den beiden: Er gibt Alice gegenüber an, Bob zu sein und umgekehrt.
– **Playback**: Mallory zeichnet eine Kommunikation zwischen Alice und Bob auf und spielt im Anschluß daran einem der beidem nocheinmal den aufgezeicneten Text des anderen vor.

[1] Für kryptographische Anwendungen sind Hashfunktionen wie der CRC oder Reed-Solomon-Code ungeeignet, denn diese sind „nur“ auf Fehlererkennung bzw. -korrektur optimiert und keineswegs auf Nicht-Umkehrbakeit!

2.4 Zertifizierung

Mit **Zertifizierung** bezeichnet man generell, daß jemand etwas öffentlich als wahr anerkennt.

Oft werden öffentliche Schlüssel zertifiziert: Hierbei unterschreibt Trent den öffentlichen Schlüssel von Bob. Das ist sinnvoll, damit Alice prüfen kann, daß ein Schlüssel auch *wirklich Bob* gehört, und nicht etwa Mallory's ist: Der könnte ihr seinen öffentlichen Schlüssel als Bob's „andrehen", um lesen zu können was sie eigentlich Bob erzählen wollte. Anhand von Trent's Unterschrift auf Bob's öffentlichem Schlüssel kann Alice erkennen, daß Trent für dessen Echtheit garantiert:

$$BobsCertifiedKey = Sig_{privTrent}(pubBob)$$

Trent's öffentlicher Schlüssel kann wiederum zertifiziert sein, usw. Dadurch entsteht ein Geflecht aus Zertifizierungen, das unter Pretty good Privacy (PGP) oder dem GNU Privacy Guard (gpg) als **Vertrauensgeflecht** bezeichnet wird.

Ähnlich wie ein Protokollstapel bei Computernetzen, so gibt es auch in der Kryptographie Protokolle, die andere Protokolle „aufrufen". Die benötigten Hilfsprotokolle wollen wir kurz erklären:

2.5 Bit-Commitment-Protokolle

Mit einem **Bit-Commitment-Protokoll** wird eine Möglichkeit geschaffen, daß sich Alice zu einem bestimmten Zeitpunkt auf etwas festlegt (ihr **Commitment**, das nicht notwendigerweise nur ein Bit sein muß), Bob es erst später erfährt, und Alice ihre Festlegung nach Abgabe nicht mehr ändern kann.

Das Standard-Beispiel, dies zu illustrieren, ist das Folgende:

Example 1. Börsenmaklerin Alice will Spekulant Bob davon überzeugen, daß sie ein gutes Gespür für erfolgversprechende Aktien hat. Aber wie kann sie es Bob beweisen? Sagt sie ihm die Kurse im Vorraus, wird Bob die Aktien sofort kaufen und Alice nachher nicht bezahlen. Sagt sie ihm im Nachhinein, was sie getippt hatte, wird ihr Bob nicht glauben, daß Alice wirklich schon letzte Woche getippt hatte.
Um Bob zu überzeugen, daß Alice „Bescheid weiss", tun sie Folgendes:

1. Bob gibt Alice einen Zufallsstring R.
2. Alice notiert ihre Vorhersage (ihr Commitment) im String C und hängt daran einen Zufallsstring P (Padding). Dann hängt sie alle Strings aneinander und hasht sie:
$$H = hash(R + C + P)$$
3. Alice übergibt den Hashwert H an Bob.
4. (eine Woche später, nachdem Alice's Vorhersage wertlos geworden ist): Alice gibt Bob ihren Aktientip und das Padding $C + P$.
5. Bob überzeugt sich, daß sich Alice richtig festgelegt hatte, (das Padding interessiert ihn dabei nicht) hängt seinen Zufallsstring an Alice's Festlegung, hasht es, und vergleicht es mit Alice's Hashwert von letzter Woche:
$$H = hash(R + C + P)?$$

Der Zufallsstring R dient dazu, daß Alice nicht im Vorhinein an sinngemäß unterschiedlichen Festlegungen solange herumformuliert, bis die Hashes gleich sind: Schon allein durch Einstreuen von Leerzeichen am Zeilenende oder Variieren von Groß- und Kleinbuchstaben gibt es für jeden Text t $2^{strlen(t)}$ bzw. $2^{countlines(t)}$ Variationen und damit oft etwa ebenso viele unterschiedliche Hashwerte der sinngemäß gleichen Texte.
In diesem Fall könnte sie Bob im Nachhinein eine ihrer Festlegungen geben, die sich zufällig als wahr erwiesen hat. Bob's Zufallszahl nimmt Alice die Möglichkeit, derartige Textlisten weit im Voraus zu berechnen.

Auch Alice's Interessen sind gewahrt: Das Padding P dient dazu, daß Bob nicht während der Woche versuchen kann, Alice's Festlegung C zu erraten, indem er die Hashes aller möglicher Festlegungen mit H durchtestet. Dies ist insbesondere dann wichtig, wenn Alice's Commitment nur wenige Möglichkeiten umfaßt, die Bob schnell durchprobiert hätte: Im Extremfall „ja/nein-Frage“ wären das ganze 2 Versuche.

Für Bob ist der Hashwert H (ohne P) völlig wertlos für seine Aktiengeschäfte und er kann Alice nicht ausnutzen.

2.6 Zero-Knowledge-Protokolle

Will Alice nun Bob beweisen, daß sie ein Geheimnis kennt, ohne daß es Bob *jemals* erfährt, reicht Bit-Commitment nicht aus – am Schluß werden dort immer „die Karten aufgedeckt“. Ein **Zero-Knowledge-Protokoll** leistet das gleiche *ohne* Enthüllung des Geheimnis.

Bob stellt Alice eine Reihe von Fragen, die Alice nur in Kenntnis des Geheimnis sicher beantworten kann. Ohne Kenntnis des Geheimnis kann sie nur raten, hat also bestenfalls eine 50%-Trefferchance, falls sich mit dem Geheimnis nur eine „ja/nein-Frage“ beantworten läßt. Wichtig für Alice ist, daß die Beantwortung der Fragen nicht das Geheimnis selbst kompromittiert („Was kommt heraus, wenn du zu deiner Geheimzahl 2 dazuaddierst?“). Es ist klar, daß Bob Alice nur glaubt, wenn sie *alle* seiner Fragen beantwortet – für Alice also eine Betrugschance von bestenfalls $1/2^{AnzahlFragen}$. Es ist bei diesen Protokollen nicht möglich, jemand Dritten zu überzeugen: Dieser müsste immer annehmen, das die beiden anderen sich vorher abgesprochen hätten.

Solche Beweise funktionieren nur interaktiv: Bob kann Carol im Nachhinein nicht beweisen, daß Alice wirklich das Geheimnis kennt: Carol würde annehmen, daß sich Alice und Bob vorher abgesprochen hätten.

Auf das Beispiel aus dem Bit-Commitment angewandt bedeutet dies, daß Alice und Bob die Aktienvorhersage n mal wiederholen (mit der Annahme, daß das Geheimnis Alice's Methode ist, die Aktien vorherzusagen).

2.7 Secret-Splitting Protokolle

Example 2. Wir wollen annehmen, Fabrikbesitzerin Alice besitzt eine geheime Methode ihre Ware günstiger herzustellen als die Konkurrenz. Da einer ihrer

Ingenieure Bob oder Carol irgendwann einmal doch zur Konkurrenz wechseln könnte, will sie nicht, daß irgendeiner von ihnen das gesamte Geheimnis kennt – jeder soll nur ein Bruchstück davon kennen, das alleine nutzlos ist. Nur zusammen können die Ingenieure das Geheimnis anwenden.

Eine einfache Methode für ein derartiges **Secret-Splitting**, bei dem jeder der Beteiligten nur im Besitz von Datenmüll ist, der sich nur in Summe zu einem Geheimnis zusammensetzt, ist der Einsatz eines **One-Time-Pads**: Das ist ein Schlüssel, der genauso lang ist wie die Nachricht selbst und das einzige Kryptosystem mit mathematisch 100%-iger Sicherheit – aber für die meisten Anwendungen eben wegen der Schlüssellänge unpraktikabel):
Sei das Geheimnis im String S notiert und R ein gleichlanger Zufallsstring (das One-Time-Pad). Dann händigt Alice an Bob R und an Carol $S \oplus R$ aus.
Sowohl R als $S \oplus R$ sind völlig nutzlos, erst zusammengesetzt ergeben sie wieder das Geheimnis:
Bekanntlich gilt immer $S = R \oplus (S \oplus R)$, eine Eigenschaft der XOR-Verknüpfung.

2.8 Blinde Signaturen

Protokolle für blinde Signaturen ermöglichen es, daß Bob ein Dokument unterschreibt, das er nicht lesen kann. Alice kann das Dokument wieder „auspacken“ und Bob's Unterschrift ist immer noch auf dem Dokument.

Praktische Situation, wo so etwas Sinn macht ist zum Beispiel diese hier:

Example 3. Bob ist Notar und unterschreibt alles, was ihm vorgelegt wird. Was in den Dokumenten steht ist ihm egal, denn Bob beglaubigt lediglich notariell, daß ihm das Dokument zu einem bestimmten Zeitpunkt vorgelegt wurde (beispielsweise, indem Bob täglich seine Unterschrift wechselt).
Konkret wird die blinde Signatur durchgeführt, indem das Dokument D mit einem **Blinding Factor** „multipliziert“ wird ($Blind()$), das ganze dann unterzeichnet, und der Blinding Factor anschließend wieder aus dem Dokument herausgerechnet wird ($View()$). Die Unterschrift bleibt trotzdem auf dem Dokument:

$$D \to Blind(D) \to Sig(Blind(D)) \to$$
$$\underbrace{View(Sig}(Blind(D)) = \underbrace{Sig(View}(Blind(D))) = Sig(D)$$

Die Entscheidende Tatsache an dieser Rechnung ist also, daß die Operationen $Sig()$ und $View()$ *kommutativ* sein müssen!

3 Digitales Geld

3.1 Überblick

Wahrscheinlich verbindet man mit diesem Begriff zuerst Dinge wie Online-Banking oder Internet-Bezahl-Dienste. Mit digitalem Geld ist (hier) aber ein möglichst

gutes **Analogon zum Bargeld** gemeint, und das heißt insbesondere, daß es keine zentrale Instanz gibt, die den Geldfluß überwachen kann (so wie die Bank beim Online-Banking oder Bezahlen mit ec-Karte).

Viele Bezahldienste werben zwar mit „Anonymität“, von *technischer* Anonymität kann dort aber nie die Rede sein, da man sich dort immer anmelden muß, und dort für jeden Kunden ein eigenes **Schattenkonto** geführt wird. Derartige Konten werden sogar bei der GeldKarte geführt, die ansonsten dem Anonymitätsbegriff am ehesten entspricht. An „echtes“ digitales Geld stellt man daher die folgenden Anforderungen:

- **Double-spending-Sicherheit**: Digitales Geld kann nicht kopiert und erneut ausgegeben werden.
- **Privatsphäre**: Der Geldfluß ist nicht nachvollzieh- oder überwachbar.
- **Unabhängigkeit**: Die Sicherheit des Geldes hängt nicht von einem physischen Ort ab, das Geld lässt sich über Computernetze übertragen.
- **Transferierbarkeit**: Dig. Geld ist zu anderen Benutzern übertragbar.
- **Offline-Bezahlung**: Eine Bezahlung zwischen Kunden und Händler benötigt keine Online-Verbindung zur Bank während der Bezahlung.
- **Teilbarkeit**: Dig. Geld kann in kleinere Teile gespalten werden (natürlich so, daß sie Summe stimmt).

Für den Mobilkommunikations-Bereich können wir auf die Offline-Forderung verzichten, daher werden im Folgenden nur Protokolle behandelt, die die Einhaltung zumindest der ersten drei Forderungen garantieren.[2]

Von den zahlreichen Protokollen für digitales Geld wollen wir stellvertretend die anonyme Schecks genaur Vorstellen: Sie bieten als einzige die technische Anonymität der Beteiligten und entsprechen somit unsererem Hauptthema „Privacy“ am ehesten.

3.2 Anonyme Schecks

Anonyme Schecks werden auch als **elektronische Münzen** bezeichnet, wohl deswegen weil sie, wie eine einzelne Münze, nicht teilbar sind.

Die folgenden Protokolle dazu kommen zwar allesamt auch nicht ohne zentrale Instanz aus (im Fologenden *die Bank* genannt), aber sie erfährt nichts über den Geldfluß selbst. Die Rolle der Bank beschränkt sich lediglich auf das Verwalten der Konten ihrer Benutzer. Woher das Geld kommt, das diese einzahlen, oder wohin abgehobenes Geld fließt, erfährt die Bank nicht.

Abstrakt formuliert behandeln wir im Folgenden eine Realisierung von **authentifizierten, nicht zurückverfolgbaren Nachrichten**:

[2] Es existieren zwar Systeme, die *alle* dieser Forderungen erfüllen, bislang sind sie aber nur von theoretischem Interesse, denn allein eine Bezahlung benötigt etwa 200 MB Daten.

Privacy beachten. Wir beginnen mit einem einfachen Protokoll, das wir dann so lange erweitern wollen, bis niemand mehr betrügen kann (oder *nur* im Betrugsfall identifiziert werden kann) (Siehe auch Abb. 1):

1. Alice bereitet 100 anonyme Schecks über 1000 $ vor.
2. Jeder Scheck kommt in einen Umschlag zusammen mit einem Stück Kohlepapier.
3. Die Bank öffnet alle Umschläge bis auf einen (Alice weiß natürlich nicht, welcher das sein wird), und überzeugt sich, daß alle über genau 1000 $ notiert sind. Dies geschieht über ein Zero-Knowledge-Protokoll.
4. Den letzten unterschreibt die Bank ungeöffnet durch den Umschlag mit dem Kohlepapier mit ein Protokoll für blinde Signaturen. Die Bank zieht dafür von Alice's Konto 1000 $ ab.
   Damit Alice garnicht auf die Idee kommt, 99 Schecks über 1000 $ und einen über 10000 $ vorzubereiten (in der Hoffnung, dieser werde der blind-signierte und damit 9000 $ ergaunert zu haben) wird man in der Praxis mit mehr als 100 Umschlägen arbeiten und Betrug unter harte Bestrafung stellen.
5. Zu Hause packt Alice den Umschlag aus und hat so einen anonymen Scheck mit der Unterschrift der Bank.
6. Damit kann Alice nun beim Händler bezahlen, er sieht ja die Unterschrift der Bank auf dem Scheck.
7. Der Händler trägt den Scheck zur Bank, sie erkennt Ihre Unterschrift wieder, und schreibt ihm deshalb die 1000 $ auf sein Konto gut.

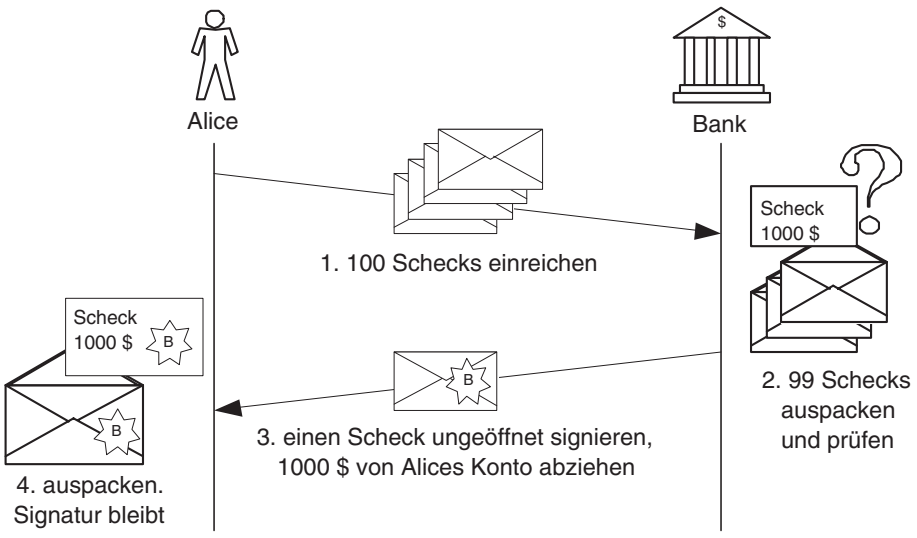

Abbildung 1. Alice erzeugt mit der Bank einen Scheck

Dem Aspekt der Privatsphäre haben wir Rechnung getragen, nicht aber der Sicherheit:
Alice kann natürlich mit dem selben Scheck mehrmals bezahlen oder der Händler ihn mehrmals einlösen, und die Bank (die ein Interesse daran hat, nicht durch Scheckkopien „ausgesaugt“ zu werden) kann es nicht feststellen!

„Double-Spending“ verhindern. Die Idee hierbei ist das Integrieren einer weltweit und zeitlich einmaligen **Eindeutigkeitsfolge** (Abkürzung **guID**) in jeden Scheck; dafür reicht ein etwa einzeiliger Zufallsstring in der Praxis sicher aus: Wir erweitern vorangegangenes Protokoll:
Alice versieht jeden Ihrer 100 anonymen Schecks vor dem Einpacken noch zusätzlich mit einer solchen, in jedem Scheck anderen guID. Wenn die Bank ihre Unterschrift abgibt, sieht sie nur die verschlüsselte guID – und dann nie wieder, denn der Scheck wird danach entschlüsselt und ist damit nicht zurückverfolgbar.

Geht der Händler mit dem Scheck zur Bank, sieht die Bank in ihrer Liste für verbrauchte guIDs nach, ob schon einmal ein Scheck mit dieser guID eingezahlt wurde:
Falls ja, bekommt der Händler das Geld auf seins Konto gutgeschrieben und guID kommt auf die Liste, falls nein verweigert die Bank den Scheck.

Um nicht mit einem solchen **verbrauchten Scheck** betrogen zu werden, müssen wir (vorerst) die Offline-Forderung fallen lassen:
Der Händler muß *während* des Bezahlens im Kontakt zur Bank stehen, und den Scheck auch *sofort* einlösen („Mir wird hier ein Scheck mit guID = „0xf00ba5...“ angeboten – Wenn er gültig ist, löse ich ihn hiermit ein.“).

Betrüger identifizieren. Kommt nun ein Händler zur Bank, um nachträglich einen Scheck einzulösen, und stellt sich dort heraus, daß es ein verbrauchter Scheck ist, kann die Bank nicht feststellen, wer betrügt:
Will der Händler betrügen, indem er versucht den Scheck zum zweiten mal einzulösen, oder hatte Alice (von der die Bank den Namen nicht kennt) den Händler mit einem verbrauchten Scheck bezahlt?

In der folgenden Erweiterung des Protokolls, die auch die Offline-Forderung erfüllt, muß Alice ihren Namen in Form einer **Indentitätsbitfolge** (Abkürzung **IBF**) derart in den Scheck einkodieren, daß sie *nur* lesbar wird, wenn sie betrügt (indem sie mit dem selben Scheck ein zweites mal bezahlt).

Das finale Protokoll als ganzes:

1. Alice bereitet jeden von n Schecks wie folgt vor:
   - Über ein *Secret-Splitting Protokoll* spaltet Alice nun n mal ihre Identitätsbitfolge I in jeweils eine linke und eine rechte Hälfte $[I_{x_L}, I_{x_R}]$ auf. Wichtig: Jedes dieser Paare von IBF-Hälften liefert Alice´s Identität, nicht aber linke und rechte IBF-Hälften aus unterschiedlichen Splits (also z.B $I_{37_L} \oplus I_{37_R} = I$, aber $I_{37_L} \oplus I_{38_R} \neq I$) Siehe Abb. 2.
   - Sie notiert den eigentlichen Scheck mit Betrag, der üblichen guID, sowie den Hashwerten aller $2n$ IBF-Hälften (Abb. 3).

Auf jedem Scheck ist also notiert:

$$[Betrag,\ guID,\ hash(I_{1_L}),\ hash(I_{1_R}),\ \ldots,\ hash(I_{n_L}),\ hash(I_{n_R})]$$

2. Alice packt alle Schecks in Umschläge mit Kohlepapier und bringt sie zur Bank.
3. Die Bank führt die üblichen Überprüfungen durch. Ausserdem verlangt sie alle ursprünglichen IBF-Hälften, testet ob sie zu den entsprechenden Hashes auf dem Scheck passen, und überprüft für alle zusammengehörendern IBF-Hälften, ob sie zusammen Alice's IBF ergeben.
4. Falls alles passt unterschreibt die Bank den letzten Scheck blind, und zieht Alice dafür 1000 $ von ihrem Konto ab.
5. Alice packt den Scheck zu Hause wieder aus.
6. Alice kauft beim Händler ein: Alice zeigt dem Händler den Scheck vor, er prüft die Unterschrift der Bank und hält ihn deshalb für echt.
7. Der Händler verlangt nun von Alice, ihm entweder die linke oder rechte IBF-Hälfte aller n IBF-Hälften zu geben: Er gibt ihr dazu einen zufälligen **Auswahlstring** in der Form „L,R,R,R, ...,R,L"), dessen entsprechende IBF-Hälften Alice ihm dann mitteilen muß (Abb. 4). Der Händler muß jetzt überprüfen, ob die Hashes der IBF-Hälften auch mit denen übereinstimmen, die auf dem signierten Scheck notiert sind!
   Es wird vorrausgesetzt, daß die Wahrscheinlichkeit, daß 2 Händler Alice einmal den selben Auswahlstring geben praktisch 0 ist.[3] Er testet, ob die verlangten IBF-Hälften, die ihm Alice gibt, auch zu den auf dem Scheck notierten Hashes passen.
8. Der Händler speichert seinen Auswahlstring und die ihm übergebenen IBF-Hälften zusammen mit dem Scheck ab.
9. Der Händler löst den Scheck bei der Bank ein: Er übergibt ihr den Scheck, seinen Auswahlstring und alle IBF-Hälften. Die Bank überprüft zuerst ob die guID noch unverbraucht ist, und ob die IBF-Hälften zu den auf dem Scheck notierten Hashes passen.
   Falls ja, lief alles legal und die Bank schreibt dem Händler den Scheckbetrag aufs Konto gut. Die guID kommt zusammen mit dem Auswahlstring und allen IBF-Hälften auf die Liste verbrauchter guIDs.
   Falls nein, findet die Bank den Betrüger wie folgt:
   - Der Händler betrügt: Dann befindet sich in ihrer Liste verbrauchter guIDs ein identischer Auswahlstring. (Der Händler kann beim zweiten Einlöseversuch keinen anderen Auswahlstring vorgaukeln, da die Bank beim verifizieren der entsprechenden Hashes der IBF-Hälften dies bemerken würde). Der Händler ist also überführt und Alice's Identität ist immer noch unbekannt.
   - Alice betrügt (Abb. 5): Dann ist der Auswahlstring des Händlers unterschiedlich zu dem, den die Bank in der guID-Liste mitgespeichert hat.

[3] Mit z.B. 64 IBF-Splits auf einem Scheck ist eine Wahrscheinlihkeit von $1/2^{64}$ für eine Kollission wohl ausreichend klein

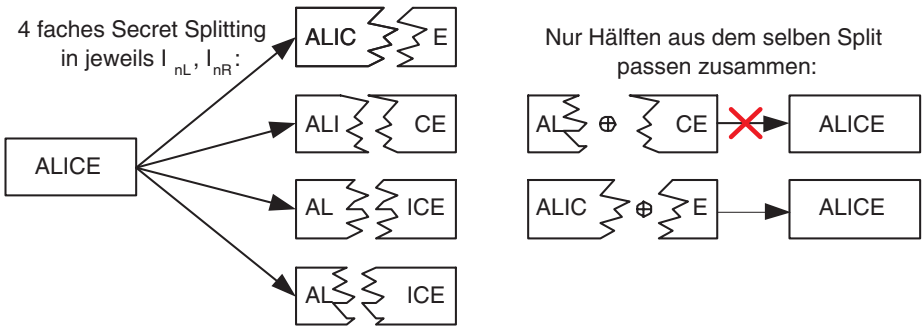

Abbildung 2. Secret-splitting von Alice's Idendität

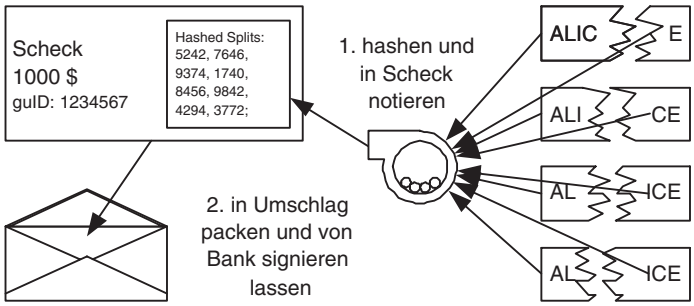

Abbildung 3. Scheck fertigstellen und in Umschlag packen

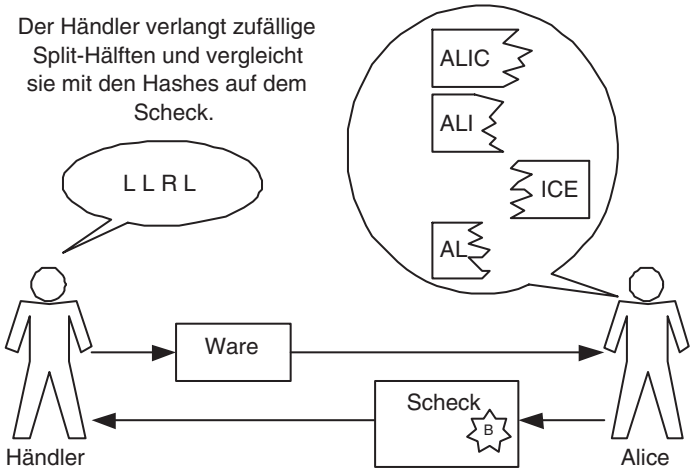

Abbildung 4. IBF-Hälften zum Auswahlstring des Händlers

Diese beiden Strings unterscheiden sich demnach an mindestens einer Position. Es gibt also mindestens ein Paar zusammengehöriger IBF-Hälften! Die beiden IBF-Hälften (die eine aus der guID-Liste von der ersten Einlöse, die andere bringt der Händler gerade mit) zusammengesetzt ergeben nun Alice's IBF: Alice ist als Betrügerin überführt und ihre Identität ist offengelegt!

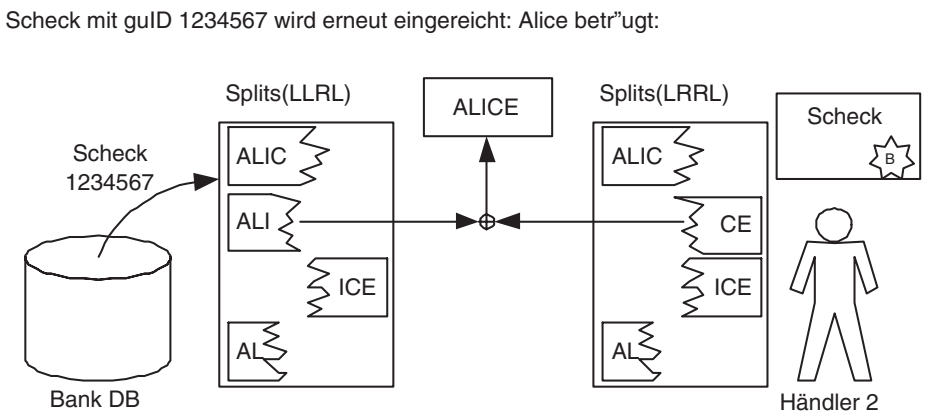

Abbildung 5. Alice betrügt, da sich ihre IBF rekonstruieren lässt.

Daß der Händler nicht betrügen kann, haben wir im letzten Schritt des Protokolls erklärt. Wenn der Händler keine zufälligen Auswahlstrings an seine Kunden stellt, sondern z.B. immer den selben, würde er sich nur selber schaden, da er dann als Betrüger dasteht, wenn ihm ein verbrauchter Scheck untergeschoben wurde.

Alice *kann* den Händler betrügen, aber es wird entdeckt (Abb. 5).

Alice kann die Bank mit vergleichsweise guten Chance von $1/n$ betrügen, wenn sie versucht, der Bank einen Umschlag mit falsch datiertem Scheck zur Unterschrift vorzulegen. Hier muss, wie bereits erwähnt, n hoch gewählt und Betrug hart bestraft werden.

Kann die Bank etwas über den den Weg des Schecks in Erfahrung bringen? Das erste mal bekommt sie eine blind gemachte Version des Schecks vorgelegt[4], beim Einlösen sieht sie auf dem Scheck n Hashes, und n IBF-Hälften: Die sind zwar identisch mit den IBF-Hälften aus dem blind signierten Scheck, können aber nicht zu diesem in Bezug gebracht werden, da die Bank ihn ja nur im Umschlag „gesehen" hat.

[4] die anderen Schecks, deren Öffnung die Bank verlangt hatte, sind natürlich nicht die geöffnete Version des Schecks, der verschlossen bleibt, da Alice in jeden Scheck eine andere guID vergeben hat.

3.3 Gesamtbewertung

Anonyme Schecks erfüllen nicht die Transferierbarkeits-Forderung: Der Weg eines Schecks ist immer von der Bank zum Käufer zum Händler zur Bank. Alice wäre schlecht beraten, wenn sie ihren Scheck an Mallory weiterverkauft: Betrügt Mallory den Händler, dann legt unser Protokoll Alice als „Betrügerin“ offen, da der Scheck ja ihren Namen enthält – und nicht Mallory's!

Für den Mobilkommunikations-Bereich könnte man sicher auf die Offline-Forderung verzichten – dann würde das zweite Protokoll bereits ausreichen. Die Käuferin Alice müsste dann quasi während des Bezahlens (dem sofortigen Einlösen des Schecks bei der Bank) an der Kasse warten.

Eve kann betrügen: Sie muß nur die Scheckübergabe zwischen Alice und dem Händler abhören und vor dem Händler bei der Bank zum Einlösen erscheinen. Der nach ihr erscheinende Händler steht dann als Betrüger in der Bank.

Folglich sollte der Besitzer sein digitales Geld schützen wie Echtes.

3.4 Implementierungen

Die Praxistauglichkeit der vorgestellten Verfahren belegt u.a. das an der University of Southern California entwickelte, vollständig implementierte und lizensierbare *NetCash*-System [5]. Es beruht auf anonymen Schecks (dort als *electronic coins* bezeichnet) und bietet neben der Erfüllung der o.g. Anforderungen auch die für den praktischen Einsatz notwendige Skalierbarkeit durch Verzicht auf eine zentrale Bank-Instanz (stattdessen werden sog. *Currency Server* verwendet). Im Verbund mit *NetCheque* [6] werden je nach Wunsch verschiedene Anonymitätsgrade ermöglicht.

Unter dem Namen *PayMe* wurde ein Protokoll entworfen und als Prototyp implementiert, das die Flexibilität von NetCash mit der vollständigen Anonymität des mittlerweile aus wirtschaftlichen Gründen eingestellten *eCash*-Systems verbindet [7]; zudem bietet es die Möglichkeit, Geld ohne Betrugsgefahr zwischen Anwendern zu transferieren, indem die Schecks bei der Bank anonym „umgetauscht“ werden.

Wenngleich auch diese Ansätze auf das Internet (und damit vorwiegend für die im Vergleich zu Mobiltelefonen rechenstarken PCs) gedacht sind (bzw. waren), so bereitet eine Verwendung der anonymen Schecks im Mobilkommunikationsbereich keine grösseren Probleme:
Die Erzeugung der Schecks selber (Abb. 1) kann zwar auf Grund des massiven Einsatzes von Public-Key-Kryptographie („100 Umschläge“, Blinding-Factor) wohl vorerst nicht im Endgerät des Bankkunden selber vorgenommen werden, doch diese Aufgabe lässt sich einfach auf den PC des Kunden verlagern. Damit könnte Alice dann *auf Vorrat* Schecks erzeugen, und diese im Anschluß einfach auf ihr Handy übertragen: Ein anonymer Scheck ist aus der Sicht des Handys dann nichts anderes, als eine Datei die nur mit Genehmigung des Besitzers ausgelesen werden darf.

Der Bezahlvorgang konkret könnte dann etwa ein verschlüsselter Bluetooth-Dateitransfer zwischen den Handys von Alice und dem Händler sein. Ist der

Händler besonders mißtrauisch, so leitet der den Scheck sofort (online) an die Bank weiter. Im anderen Fall ist das Risiko für ihn immer noch geringer als z.B. bei Kreditkartengeschäften: Zwar bekommt er bei beiden Bezahlformen (ersteinmal) nicht sein Geld, die Identität von Alice kann dann aber bei anonymen Schecks offengelegt werden. Bei einem Kreditkartenbetrug erfährt man im Allgemeinen hinterher nicht den Namen des Betrügers.

4 Location Management mit Wahrung der Privatsphäre

4.1 Motivation

Die heutige Stellung eines Netzbetreibers kommt einem „Big Brother“ sehr nahe: Er weiß nicht nur (im technischen Sinne), *was* seine Kunden am Telefon besprechen (die GSM-Verschlüsselung ist ja bekanntlich keine Ende-zu-Ende-Verschlüsselung sondern nur eine zwischen dem Endgerät des Teilnehmers und seiner Basisstation), – er weiß auch immer, *wo* sie sich gerade Aufhalten, zumindest im Bereich welcher Funkzelle, was in städtischen Gebieten ungefähr einigen Häuserblocks entspricht.
Nachrichten vom „kreativen Einsatz der jungen Fahndungstechnik“ (Spiegel, heise News[4]), mit denen (nicht nur) Strafverfolger über die sog. „stille SMS“ Auflagen der Strafprozessordnung umgehen können (die eine Standortbestimmung nur bei Schwerverbrechern zulassen würden), lassen einen nachdenklich werden, ob der Netzbetreiber wirklich in der Lage ist, die Privatsphäre seiner Teilnehmer zu wahren.
Wir wollen deshalb einen Lösungsansatz auf technischer Ebene für das Problem der Privatsphäre bezüglich des Aufenthaltsortes geben.

4.2 MIX-Netzwerke

Dieses Konzept wurde 1981 von David Chaum (ursprünglich für den e-Mail Versand) entwickelt:
Unter einem **MIX-Netzwerk** versteht man einen Graphen mit 2 Sorten von Knoten, den **Benutzern** und den **Mixen**; die Kanten beschreiben, welcher Knoten an welchen etwas senden kann. Benutzer können unter Verwendung der Mixe untereinander Nachrichten austauschen, und niemand kann den Weg und den Inhalt der durch den Graphen wandernden Nachrichten verfolgen.

Dies geschieht, indem die Nachricht durch eine vom Absender zufällig festgelegte Route über unterschiedliche Mixe zum Empfänger (die Mix-Kaskade) geschickt wird, wobei jeder Mix *nur* weiss, wohin er eine Nachricht (die er nicht lesen kann) als nächstes schicken soll.

Die Verschleierung der Route zum Empfänger geschieht, indem der Sender die Nachricht in umgekehrter Reihenfolge des Routings mit den PublicKeys der zu durchlaufenden Mixe sowie der Adresse des jeweils nächsten Mixes verschlüsselt.

Man kann sich einen Mix somit als ein Postamt vorstellen, an welches man Briefe adressieren kann, es diese dann öffnet, und den Ihnalt (ein weiterer Brief) weiterschickt (an weitere Postämter oder den End-Empfänger).

Vorraussetzung für die Sicherheit eines solchen Netzwerks ist die Annahme, daß ein Angreifer zumindest nicht sehen kann, was innerhalb *eines* Mixes geschieht (es gibt also keinen „Big Brother“; s. S. 219).

Um für Eve Verkehrsfluß-Analysen zu erschweren, gibt es noch ein paar zusätzliche Aufgaben für das MIX-Netzwerk:

- **Padding/Splitting**: Alle Nachrichten werden zu einer festen Größe oder zumindest zu einem ganzzahlig vielfachem einer Blockgröße aufgefüllt oder zerlegt. Damit kann Eve keine Ereignisse der Art beobachten wie „Alice hat 37896 Bytes verschickt und Bob kurz darauf 37896 Bytes emfangen . . . “.
- **Dummy-Traffic**: Alle Mixe müssen ständig (oder zumindest zu jedem Taktzyklus) Nachrichten oder Pseudo-Nachrichten senden. Ansonsten könnte Eve, falls einmal nur eine Nachricht durch das Netz wandert, beobachten welcher Benutzer/Mix gerade sendet, und würde dadurch Quelle bis Ziel der Nachricht kennen. Noch sicherer ist es, wenn auch noch alle Mix-Benutzer ständig zumindest Pseudo-Nachrichten versenden müssen.
- **Repetition-Ignoring**: Ein Mix muss das Senden von Nachrichten verweigern, die er schon einmal versandt hat: Eve könnte sonst durch einen Playback-Angriff (2.3) herausfinden, wohin der Mix die Nachricht tatsächlich gesandt hat, indem sie die Nachricht erneut an den Mix sendet, und seine „jetzigen“ Output-Nachrichten mit seinen „damaligen“ Output-Nachrichten vergleicht.

Das ganze MIX-Konzept lässt sich durch ein Beispiel leicht veranschaulichen:

Example 4. Das MIX-Netzwerk besteht aus Benutzerin Alice, die Benutzer Bob eine Nachricht M schreiben möchte, ohne daß Lauscherin Eve es mitbekommt. Es gibt auch noch einige andere Benutzer, die uns aber nicht interessieren: Entscheidend ist nur, daß es bei 2 Benutzern trivial wäre, wer wem schreibt. Die Mixe $M_1, \ldots, M_4$ stehen Alice dazu zur Verfügung, (das bedeutet, sie kennt deren PublicKeys $pubM_1, \ldots, pubM_4$). Innerhalb des Netzes sei jeder Knoten von jedem erreichbar.
Alice wählt zunächst zufällig eine Route aus, über die sie M transportiert haben möchte. Dazu wählt sie einige (nicht notwendigerweise alle) Mixe aus, wobei wir doppeltes Vorkommen einiger Mixe auf dem Transportweg ausdrücklich zulassen wollen (die Route darf also *Zyklen* enthalten):

Alice verpackt nun ihre Nachricht in einen Umschlag an Bob, sie verschlüsselt also M asymmetrisch zu

$$U_1 = Enc_{pubBob}(M)$$

Diesen Umschlag U_1 verpackt sie nun in einen weiteren Umschlag an den letzten Mix auf dem Weg zu Bob M_1, zusammen mit einem Hinweis, das U_1 an Bob geschickt werden soll:

$$U_2 = Enc_{pubM_1}(Bob, U_1) = Enc_{pubM_1}(Bob, Enc_{pubBob}(M))$$

Dies setzt sie nun solange fort, bis die verschachtelten Umschläge ihre gewünschte Route repräsentieren. An M_1 übergibt sie also

$$Enc_{pubM_1}(M_2, Enc_{pubM_2}(M_3, Enc_{pubM_3}(M_1, Enc_{pubM_1}(Bob, Enc_{pubBob}(M)))))$$

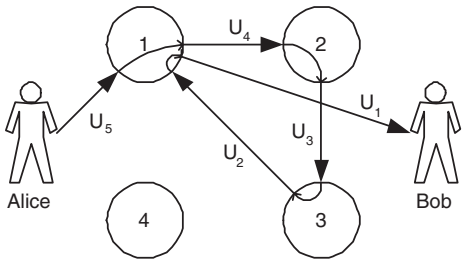

Abbildung 6. Route $Alice \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow M_1 \rightarrow Bob$

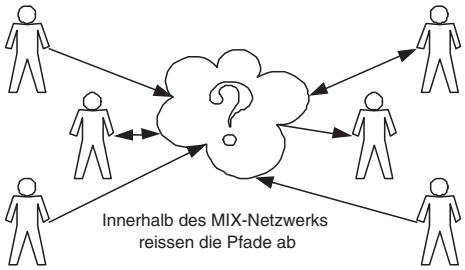

Abbildung 7. Der Traffic in/aus den Mixen für „Zuschauer“

Die Zustellung erfolgt dann einfach, indem jeder Mix die eingehende Nachricht entschlüsselt und den enthalten Umschlag weiterleitet:

$$
\begin{array}{ll}
M_1 : Enc_{pubM_2}(M_3, Enc_{pubM_3}(M_1, Enc_{pubM_1}(Bob, Enc_{pubBob}(M)))) & \rightarrow M_2 \\
M_2 : Enc_{pubM_3}(M_1, Enc_{pubM_1}(Bob, Enc_{pubBob}(M))) & \rightarrow M_3 \\
M_3 : Enc_{pubM_1}(Bob, Enc_{pubBob}(M)) & \rightarrow M_1 \\
M_1 : Enc_{pubBob}(M) & \rightarrow Bob \\
Bob\text{: } Dec_{privBob}(Enc_{pubBob}(M)) = M. &
\end{array}
$$

Anfangs wurde erwähnt, daß ein MIX-Netzwerk dann sicher ist, solange man mindestens *einem* Mix auf der gewählten Route vertrauen kann, in den also Eve nicht „hineinschauen“ kann (= sehen, was sich in dem Umschlag befindet, der innerhalb des Mixes geöffnet wird). Nehmen wir also einmal an, Eve kontrolliert alle Mixe aus unserem Beispiel mit Ausnahme von M_3 (sie kennt also die PrivateKeys der von ihr kontrollierten Mixe).

In Abb. 4 erkennt man, daß aufgrund des Dummy-Traffics, den der Mix M_3 *zusätzlich* zur eigentlich weitergeleiteten Nachricht an die Eve-kontrollierten Mixe M_1, M_2, M_4 sowie einige Benutzer sendet, Eve keinen eindeutigen Pfad der Nachrichten rekonstruieren kann: sie weiß nicht, welche der aus M_3 ausgehenden Nachrichten der eingehenden entspricht.

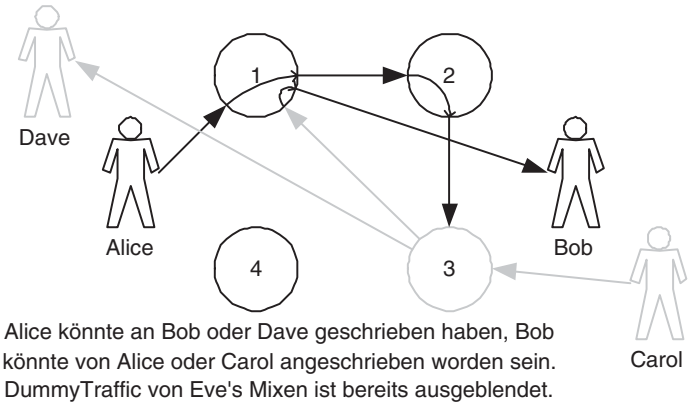

Abbildung 8. Extremfall: Eve kontrolliert alle Mixe bis auf M_3

4.3 Implementierung

In [2] wird ein Vorschlag zum Einsatz eines MIX-Netzwerkes zur Verschleierung der Standortinformation eines Teilnehmers gegeben:
Hierbei legt der MobilTeilnehmer (MS) eine via Mixe verschleierte Route, seine **untraceable Return-Address**, von seinem Home-Location-Register (HLR) zu seinem Location-Area-Identifier (LAI) fest. Auf der MS-Seite wird er auf bekannte Weise über sein sein „vergängliche Pseudonym“, nämlich seiner Temporary Mobile Subscriber-Identity (TMSI) referenziert.
Das MIX-Netwerk wird also nur benutzt um den Ort eines Teilnehmers und nicht die Identität eines sendenden teilnehmers zu verschleiern.

Example 5. Mobilteilnehmerin Alice legt ihre untraceable Return-Address über die MIX-Kaskade $HLR \rightarrow M_1 \rightarrow M_2 \rightarrow M_3 \rightarrow TMSI$ fest.

Alice übergibt dazu an ihr HLR einen Umschlag, der mit der Adresse des ersten Mixes M_1 auf der Route zu ihr beschriftet ist. Darin befindet sich wieder ein Umschlag an den nächsten Mix M_2 sowie ein Schlüssel k_1, mit denen M_1 alle Nachrichten, die er auf diesem Pfad weiterschicken soll, verschlüsseln wird.

Im innersten Umschlag, der vom letzten Mix der gewählten Kaskade M_3 an die Location Area des MS zugestellt wird, befindet sich dann die bekannte $TMSI$, sowie ebenfalls ein Schlüssel k_3 zur symmetrisch verschlüsselten Kommunikation zwischen MS und M_3 (vgl. Abb. 9).

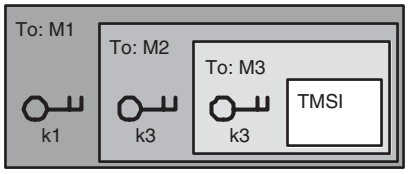

Abbildung 9. Der verschleierte Location Area Identifier $\{LAI\}$

Diesen derart verschleierten LAI wollen wir mit $\{LAI\}$ bezeichnen, und er berechnet sich hier folgendermaßen:

$$\begin{aligned}
U_1 &= Enc_{pubM_3}(k_3, TMSI).\\
U_2 &= Enc_{pubM_2}(k_2, M_3, U_1) =\\
&= Enc_{pubM_2}(k_2, M_3, Enc_{pubM_3}(k_3, TMSI)).\\
U_3 &= Enc_{pubM_1}(k_1, M_2, U_2) =\\
&= Enc_{pubM_1}(k_1, M_2, Enc_{pubM_2}(k_2, M_3, Enc_{pubM_3}(k_3, TMSI))).\\
\{LAI\} &= (M_1, U_3).
\end{aligned}$$

Der Netzbetreiber speichert nun für jeden seiner Teilnehmer einen Eintrag der Form $[MSISDN, \{LAI\}]$ in seinem HLR ab. Damit kann aktiv eine Verbindung zu jedem MS aufbauen, weiß aber nicht, wo sie sich befinden.

Mobile terminated Call Setup. Trifft ein Anruf für den MS (über seine *MSISDN* referenziert) ein, so geht der Netzbetreiber wie folgt vor:
Er schickt eine (noch unverschlüsselte) *call_setup_msg*-Nachricht zusammen mit dem $\{LAI\}$ selbst an den ersten Mix M_1, den er aus dem $\{LAI\}$ ablesen kann. Bei M_1 angekommen, wird der äusserste Umschlag des $\{LAI\}$ ausgepackt, damit M_1 weiß, wohin er die Nachricht überhaupt weiterleiten soll:
Hier ist es der Mix M_2.
Somit erfährt M_1 also, daß er *call_setup_msg* an M_2 weiterleiten soll. Bevor er dies tut, verschlüsselt er aber noch die weiterzuleitende Nachricht mit dem ebenfalls im geöffneten Umschlag sichtbaren Schlüssel k_1 zu $Enc_{k_1}(call\_setup\_msg)$. M_1 schickt also die verschlüsselte *call_setup_msg* zusammen mit dem „inneren Rest“ des $\{LAI\}$, U_3 an M_2 weiter. Dieser verfährt analog: Rest des $\{LAI\}$ auspacken, Nachricht verschlüsseln, diese zusammen dann mit dem inneren des restlichem $\{LAI\}$ an seine Bestimmung weiterschicken, siehe Abb. 10.

Bei Alice angekommen, wird die mehrfach symmetrisch-verschlüsselte Nachricht

$$Enc_{k_3}(Enc_{k_2}(Enc_{k_1}(call\_setup\_msg)))$$

in umgekehrter Reihenfolge wieder ausgepackt: Die dazu erforderlichen, von ihr generierten Schlüssel $k_1, \ldots, k_3$ muß sie gespeichert haben.

Ein $\{LAI\}$ kann immer nur einmal verwendet werden, denn MIX-Netzwerke dürfen bekanntlich keine Nachricht ein zweites mal versenden. Deshalb benötigt

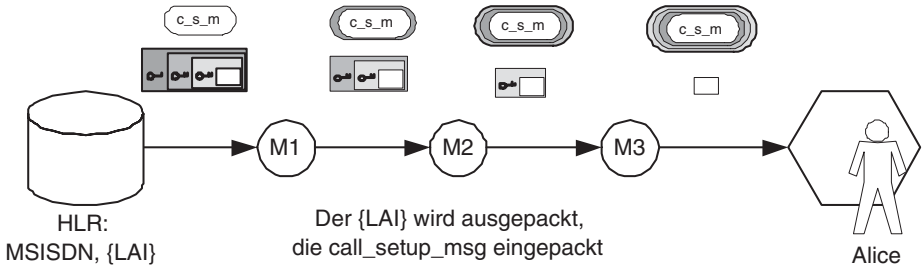

Abbildung 10. Der Weg einer *call_setup_msg*

das HLR nach jeder von ihr ausgegangenen Transaktion einen neuen $\{LAI\}$. Das MS könnte jedesmal unmittelbar nach einer solchen Transaktion einen neuen generieren, oder nach Vorschlag der Autoren von [2], das MS sog. „Sets of $\{LAI\}$" erzeugen zu lassen, die dann über einen kurzen Index angesprochen werden könnten, was auch eine Bandbreiten-Einsparung mit sich brächte.

Mobile originated Call Setup. Diese Richtung ist deutlich einfacher: Der abgehende Nachrichtenversand von Alice erfolgt einfach, wie wir es in 4.2 beschrieben haben.

4.4 Gesamtbewertung

Die Idee des Mix-Netzwerks für ein mobiles Kommunikationsnetz an sich ist gut. Die Autoren von [2] nehmen allerdings nicht dazu Stellung, wie das MS die zahlreichen, „teuren" Public-Key-Operationen zum Generieren neuer $\{LAI\}$ lösen soll. Diese finden nämlich relativ häufig statt: Jeder eingehende Anruf und jedes Location-Update verbrauchen einen $\{LAI\}$.
Somit scheint dieses Konzept zwar interessant, aber lässt neben der schwierigen technischen Machbarkeit noch einige Fragen unbeantwortet, insbesonder die Naheliegendste: „wer betreibt eigentlich die Mixe?"

5 Sicherheitsmechanismen im Wireless Application Protocol

5.1 Motivation

Eine ausführliche Behandlung des Wireless Application Protocol mag angesichts seiner (auch 6 Jahre nach Einführung) eher als gering einzuordnenden Marktbedeutung im Bereich sicherheitskritischer Anwendungen auf den ersten Blick fraglich erscheinen. Allerdings lassen sich anhand der für WAP vorgesehenen Verfahren und Protokolle einige grundlegende Sicherheitsmechanismen und auch die damit einhergehenden Probleme relativ anschaulich darstellen.

Nicht zuletzt existiert auch eine breite Allianz auf Seiten der Industrie [21], die den möglichen Einsatz in neuen, potentiell lukrativen Anwendungsgebieten wie m-Commerce zweifellos gerne unterstützt – es ist daher durchaus realistisch anzunehmen, daß WAP trotz einiger Startschwierigkeiten in Zukunft auch in diesen Bereichen verstärkt zum Einsatz kommen wird.

Im Hinblick auf das Thema der Arbeit soll im folgenden vor allem die für WAP vorgesehene Sicherheitsschicht WTLS genauer dargestellt werden; auf die weiteren Protokollschichten wird nur soweit eingegangen, wie dies zum Gesamtverständnis notwendig ist.

5.2 Grundlagen von WTLS

Ziele und besondere Anforderungen. Die im WAP vorgesehene Sicherheitsschicht *Wireless Transport Layer Security* (WTLS) orientiert sich grundlegend an der im Internet benutzten Schicht *Transport Layer Security* (TLS), die wiederum auf dem *Secure Socket Layer*-Protokoll (SSL) beruht. Sie soll insbesondere die folgenden Eigenschaften bieten (vgl. Abschnitt 2.1, S. 204):

- *Datenintegrität*
- *Vertraulichkeit*
- *gegenseitige Authentifikation*
- *Schutz gegen Denial-of-Service-Angriffe*

Im Gegensatz zu TLS/SSL ist WTLS für drahtlose Netze konzipiert; so sollen etwa Anwendungen die vier genannten Funktionen nach Bedarf (abhängig von Sicherheitsanforderungen oder Einschränkungen des zugrunde liegenden Netzwerks) selektiv ein- und abschalten können. Im Vergleich zu drahtgebundenen Netzen sehr lange *round-trip-delay*-Zeiten, niedrige Bandbreiten (abhängig vom Bearer) und hohe Fehlerraten müssen berücksichtigt werden, weshalb u.a. auch die Verwendung verbindungsloser Transportmedien (Datagramme) vorgesehen ist. Weiterhin spielen die meist eingeschränkte Rechenleistung und Speicherkapazität mobiler Endgeräte sowie nicht zuletzt auch die in vielen Ländern geltenden Im- bzw. Exportbeschränkungen für Kryptographieverfahren eine Rolle.

Einbindung in den WAP-Protokollstack. Die WAP-Protokollarchitektur orientiert sich, wie in Abb. 11 dargestellt, am bekannten ISO/OSI-Schichtenmodell, ist dabei allerdings deutlich flexibler als die im Internet eingesetzte, relativ starre Ausprägung dieses Modells. Einige Besonderheiten werden im folgenden erläutert, wobei hier nicht alle Schichten ausführlich behandelt werden sollen.

Die *Transportschicht* bietet für die darüber liegenden Schichten ein einheitliches Interface zur Nutzung einer ganzen Reihe verschiedener, im Mobilfunk gebräuchlicher Trägertechnologien (in Abb. 11 in der *Netzwerkschicht* dargestellt). Das hier implementierte datagrammorientierte Wireless Datagram Protocol (WDP) entspricht in seiner Funktionalität etwa (bei einigen Trägerdiensten, etwa CSD oder GPRS, sogar genau, vgl. [19]) dem im Internet verwendeten UDP.

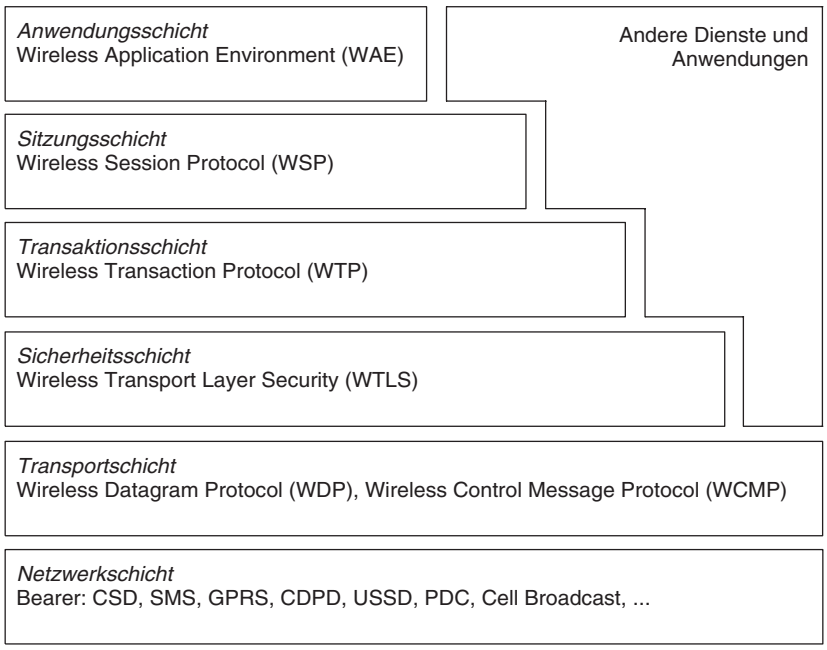

Abbildung 11. Der WAP-Protokollstack [16, S. 15]

Für die im WAP explizit vorgesehene *Sicherheitsschicht* existiert zwar mit TLS ein entsprechendes Gegenstück im Internet, ist dort aber nicht in dieser Form im ursprünglichen Schichtenmodell zu finden. Die Verwendung von WTLS ist allerdings vollständig optional und somit nicht fester Bestandteil jeder WAP-Anwendung. Sind für eine konkrete Applikation keine Verschlüsselung oder andere Sicherheitsmechanismen nötig, leitet die Sicherheitsschicht daher lediglich transparent Nachrichten zwischen Transaktions- und Transportschicht weiter.

Die *Transaktionsschicht* bietet speziell für Endgeräte mit geringen Ressourcen optimierte Transaktionsdienste, die in drei Klassen eingeteilt werden [20, S. 20ff]:

- Klasse 0: nicht verlässlicher, nicht bestätigter Nachrichtenaustausch (*unreliable one-way*; entspricht direkt dem WDP aus der Transportschicht, gedacht für sehr einfache Push-Dienste)
- Klasse 1: verlässlicher, nicht bestätigter Nachrichtenaustausch (*reliable one-way*, für Push-Dienste)
- Klasse 2: verlässlicher, bestätigter Nachrichtenaustausch (*reliable two-way*, für bidirektionalen Datenverkehr)

Verlässlichkeit bedeutet in diesem Zusammenhang verbindungsorientierten Datentransfer über die darunterliegende paketorientierte Transportschicht durch Empfangsbestätigungen, Duplikaterkennung etc.

Das in der *Sitzungsschicht* implementierte Wireless Session Protocol (WSP) bildet das WAP-Äquivalent zu HTTP 1.1, bietet aber darüber hinaus Session-Management-Mechanismen und ermöglicht sowohl verbindungs- als auch datagrammorientierten Betrieb.

Im Wireless Application Environment (WAE) in der *Anwendungsschicht* können schließlich Anwendungen und Dienste wie etwa Wireless Markup Language (WML) implementiert werden.

5.3 Kommunikationsablauf

Der grundlegende Ablauf eines WTLS-geschützten Kommunikationsvorgangs ist recht einfach zu skizzieren. Zunächst einigen sich die Kommunikationspartner über die zu verwendenden kryptographischen Algorithmen und tauschen die dazu benötigten Schlüssel aus; alle folgenden Nachrichten werden danach nur noch nach den vereinbarten Verfahren verschlüsselt ausgetauscht. Wie auch bei TLS sind diese Vorgänge für die Anwendungsschicht (bzw. generell für höhere Protokollschichten) transparent.

Aus Rücksicht auf die in mobilen Systemen oft vorliegenden Einschränkungen (vgl. Abschnitt 5.2, S. 223) werden asymmetrische Verschlüsselungsverfahren in WTLS nur im Rahmen des sog. *Handshake Protocol* verwendet, um geheime Schlüssel und ggf. Zertifikate auszutauschen; der hohe Rechenaufwand ist hier vertretbar, da dieser Vorgang nur relativ selten durchgeführt werden muss.

Nachdem die Kommunikationspartner Algorithmen und Schlüssel vereinbart haben, wird durch das *Change Cipher Protocol* (das nur aus einer einzigen Nachricht innerhalb des Handshake-Protokolls besteht) der eigentliche Datenaustausch eingeleitet; die im folgenden übertragenen Nutzdaten werden dabei symmetrisch verschlüsselt.

Wichtig sind in diesem Zusammenhang auch der Sitzungs- und Verbindungsbegriff (*Session* bzw. *Connection*): Zwischen den beiden Kommunikationsendpunkten wird stets eine sichere Session aufgebaut, die als Umgebung für eine beliebige Anzahl von Connections dient. Dieses Vorgehen hat den wesentlichen Vorteil, daß nicht für jedes zu übertragende Objekt (z.B. Bilder in einer WML-Seite) ein vollständiger neuer Verbindungsaufbau durchgeführt werden muß (vgl. *Abbreviated Handshake* im nächsten Abschnitt).

Treten im Verlauf der Sitzung sicherheitskritische Fehler auf (etwa der Empfang einer verfälschten Nachricht), werden diese dem Partner über das *Alert Protocol* gemeldet. Im Falle einer schwerwiegenden Sicherheitsverletzung wird die Verbindung sofort nach Versand der Alert-Meldung abgebrochen.

Das *Record Protocol* bildet die Basis für die drei bisher genannten Protokolle sowie die übergeordneten Schichten (in der WTLS-Spezifikation vereinfachend als *Application Data Protocol* bezeichnet). Auf dieser „unteren Ebene“ innerhalb der WTLS-Schicht werden sämtliche Informationen über den gegenwärtigen und

zukünftigen Zustand der Verbindung verwaltet und die eigentlichen kryptographischen Operationen durchgeführt.

Im folgenden sind wesentliche Aspekte dieser Protokolle ausführlicher dargestellt, wobei hier – schon aus Platzgründen – kein Anspruch auf Vollständigkeit erhoben wird; auf für das Gesamtverständnis weniger wesentliche technische Details wurde zugunsten einer leichter verständlichen Beschreibung gelegentlich verzichtet.

Handshake Protocol. Wie bereits angedeutet, werden im Handshake-Protokoll (s. Abb. 13) die Parameter für den folgenen gesicherten Kommunikationsvorgang vereinbart. Um ein möglichst breites Spektrum von Endgeräten und Sicherheitsanforderungen abzudecken, ist in WTLS eine ganze Reihe verschiedener Algorithmen vorgesehen, die meisten davon mit verschiedenen möglichen Schlüssellängen [17]:

- symmetrische Verschlüsselungsverfahren (*Bulk Encryption Algorithms*):
  - RC5
  - DES
  - IDEA
  - 3DES
- (asymmetrische) Schlüsselaustauschverfahren (*Key Exchange Suites*):
  - DH
  - RSA*
  - ECDH*
- Hashfunktionen (*Keyed MAC Algorithms*):
  - SHA
  - MD5

Prinzipiell ist sowohl Block- als auch Stream-Verschlüsselung (vgl. Abschnitt 2.2, 205) möglich; derzeit werden jedoch nur Block Cipher-Algorithmen (*Cipher Block Chaining*, CBC) unterstützt [17, S. 82]. In der *ClientHello*-Nachricht übermittelt zunächst der Client die von ihm unterstützte, nach Präferenz sortierte Teilmenge dieser Liste (aufgrund der bereits genannten Einschränkungen möglicherweise nur sehr wenige Verfahren [9, S. 229]). Der Server wählt aus dem Angebot die am besten geeigneten aus und meldet diese dem Client in der *ServerHello*-Nachricht. Daneben enthalten die Client- und ServerHello-Nachrichten noch weitere Daten wie Sitzungskennung (*Session ID*) und Zufallszahlen zur Berechnung des *Master Secret* (s.u.).

Im Anschluß daran können bei Bedarf Zertifikate (*Certificate*/*CertificateRequest*) sowie zusätzliche Schlüsselinformationen (*ServerKeyExchange*/*ClientKeyExchange*) ausgetauscht werden. Nach Ablauf dieser Nachrichtensequenz sind beiden Seiten die nötigen Daten bekannt, um ein sog. „gemeinsames Geheimnis“ (*Master Secret*), später Grundlage für Nachrichtenauthentifizierung und symmetrische Verschlüsselung, zu berechnen: Nach dem zuvor vereinbarten asymmetrischen Verfahren übermittelt zunächst der Client dem Server das

* mit optionaler Authentifizierung durch Zertifikate

Pre-Master Secret. Aus diesem Wert sowie den in den Hello-Nachrichten enthaltenen Zufallszahlen wird dann auf beiden Seiten auf gleiche Weise, aber natürlich unabhängig voneinander, das Master Secret berechnet (s. Abb. 12).

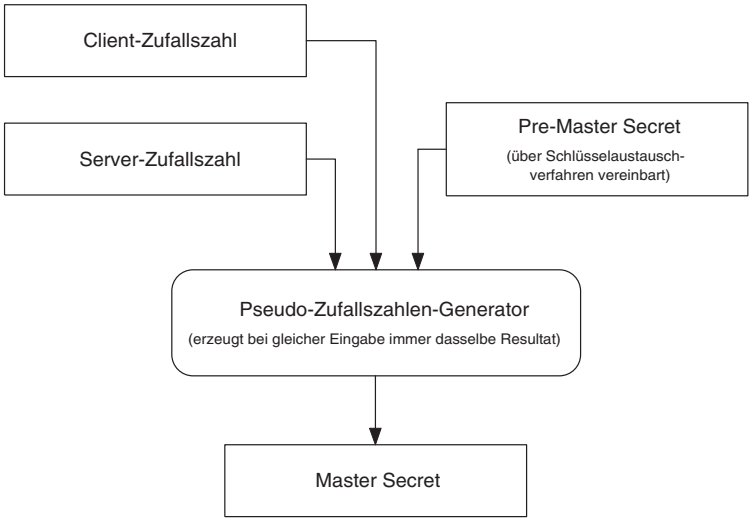

Abbildung 12. Berechnung des Master Secret [8, S. 314]

Damit ist der eigentliche Initialisierungsvorgang beendet und wird durch die *ChangeCipherSpec*- und *Finish*-Messages[5] abgeschlossen. Die Finish-Nachrichten enthalten einen Hashwert über alle bisher versendeten Nachrichten und das Master Secret, den die jeweilige Gegenseite ebenfalls berechnet und so Authentifizierung und erfolgreichen Schlüsselaustausch verifizieren kann; gleichzeitig sind dies die ersten in verschlüsselter Form versendeten Daten.

Wie bereits erwähnt, muß zum Aufbau weiterer Verbindungen in einer bereits bestehenden Sitzung nicht der gesamte Handshake erneut durchgeführt werden. Hierfür wird aus Rücksicht auf die ggf. geringen Ressourcen der sog. *Abbreviated Handshake* verwendet, bei dem nur die Hello-, ChangeCipherSpec- und Finished-Nachrichten gesendet werden (s. Abb. 14). ClientHello enthält dabei die Session ID der Sitzung, in der die neue Verbindung aufgebaut werden soll. Das bisherige Master Secret wird als nächstes Pre-Master Secret verwendet, wodurch auch der beim ersten Ablauf notwendige Austausch mit asymmetrischer Verschlüsselung entfällt.

[5] obwohl ChangeCipherSpec lediglich eine einzelne Nachricht ist, wird diese in der WTLS-Spezifikation aus technischen Gründen als eigenständiges Protokoll innerhalb des Handshake Protocol geführt

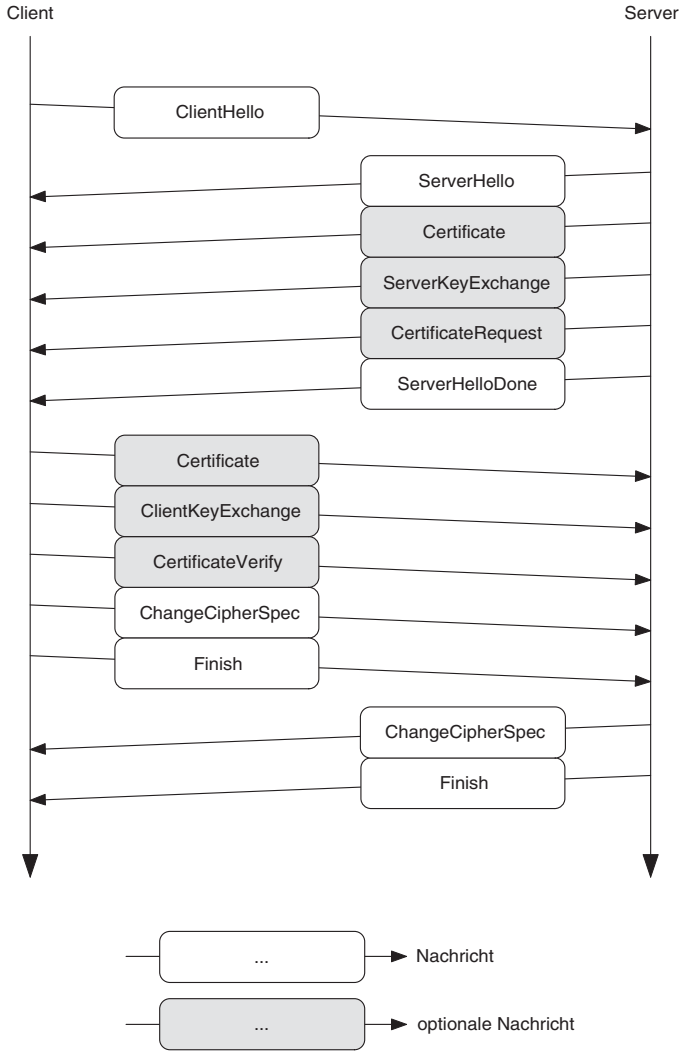

Abbildung 13. Das WTLS Handshake-Protokoll (Full Handshake) [8, S. 311]

Das Abbreviated Handshake-Protokoll kann auch verwendet werden, um im Verlauf einer Sitzung neue Schlüssel zu erzeugen (wodurch eine eventuelle kryptographische Analyse zusätzlich erschwert wird), oder um eine zwischenzeitlich unterbrochene, aber noch aktive Session wieder aufzunehmen.

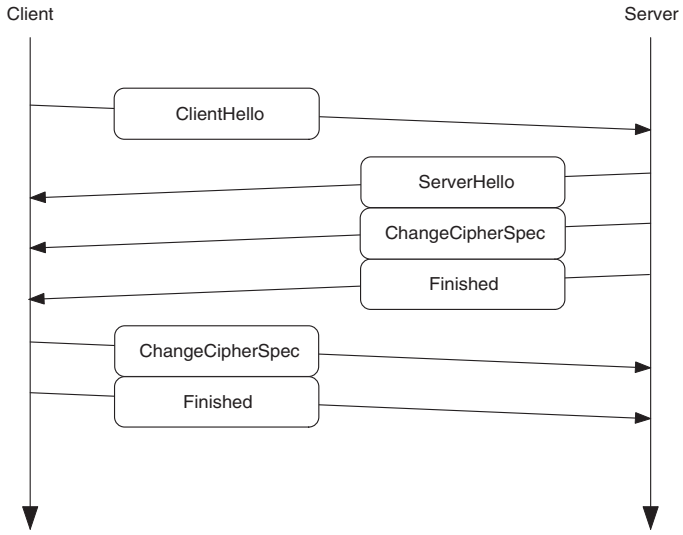

Abbildung 14. Das Abbreviated Handshake-Protokoll [12, S. 8]

Daneben existieren noch zwei weitere, spezialisierte Handshake-Verfahren, die hier aus Platzgründen nur kurz angesprochen werden sollen. Beim *Optimized Full Handshake* bezieht der Server ein Zertifikat des Clients von einem Verzeichnisdienst (oder ggf. aus einer eigenen Liste); der *Shared Secret Handshake* beruht auf einem beidseitig fest implementierten *shared secret* (z.B. bereits in der Hardware), welches beim Verbindungsaufbau als pre-master secret verwendet wird.

Insbesondere die drei verkürzten Verfahren sind – verglichen mit dem TLS-Handshake – sehr kompakt formuliert: Soweit möglich wurde auf Acknowledge-Meldungen und ähnliches verzichtet, da durch den bei jedem Nachrichtenwechsel anfallenden, in mobilen Systemen oft recht hohen round-trip-delay der Verbindungsaufbau sonst stark verzögert würde.

Alert Protocol. Das *Alert-Protokoll* definiert Mechanismen zur Meldung sicherheitsrelevanter Fehler, wie etwa die Verwendung eines ungültigen Zertifikates oder den Empfang verfälschter Daten, an den Kommunikationspartner. Diese Fehlermeldungen werden, je nach Beeinträchtigung der Sicherheit durch das aufgetretene Problem, eingeteilt in *warnings*, *critical* und *fatal errors*, die jeweils unterschiedliche Folgen nach sich ziehen.

Als „fatal“ klassifizierte Meldungen führen zum sofortigen Verbindungsabbruch und zum Verwerfen der aktuellen Session-ID. Der Aufbau neuer Verbindungen innerhalb der momentan aktiven Session wird dadurch unterbunden; bereits bestehende dürfen jedoch zu Ende geführt werden.

Bei „critical errors“ wird ebenfalls die betroffene Verbindung beendet, die Session-ID bleibt allerdings gültig und darf damit auch zur Initialisierung neuer Verbindungen benutzt werden.

„Warnings“ sind aus Protokollsicht lediglich informative Mitteilungen ohne explizit vorgegebene Folgen; eine eventuelle Reaktion des Empfängers ist von der konkreten WTLS-Implementierung und/oder höheren Protokollebenen abhängig, letztendlich also von den Sicherheitsanforderungen der jeweiligen Anwendung.

Auch ein regulärer Abbau der Verbindung erfolgt durch Alert-Nachrichten, sogenannte *Closure Alerts*: das Schließen einer einzelnen Verbindung per *connection_close_notify* (im Sinne der genannten Regeln als critical klassifiziert), die Beendigung der Sitzung durch *session_close_notify* (fatal).

Record Protocol. Das *Record-Protokoll* bildet die Grundlage für die übergeordneten Protokolle; die eigentliche Verarbeitung der Nachrichten, d.h. kryptographische Operationen, Versand/Empfang etc. wird hier implementiert. Dazu ist zunächst die Verwaltung des Protokollzustandes (*Connection State*) notwendig. Aus logischer Sicht existieren zu jedem Zeitpunkt zwei Zustände: der gegenwärtige und der bevorstehende Zustand (*Current* bzw. *Pending State*). Der Nachrichtenverkehr wird stets in der Umgebung des aktuellen Zustands abgewickelt, während durch das Handshake-Protokoll die Parameter für den kommenden Zustand verhandelt werden. Diese Parameter bestehen aus (vgl. [17, S. 39], s. auch Abschnitt 5.3):

- **Connection End**
  die Art des Verbindungsendpunktes (Client oder Server)
- **Bulk Cipher Algorithm**
- **MAC Algorithm**
- **Compression Algorithm**
- **Master Secret**
- **Client Random**
- **Server Random**
- **Sequence Number Mode**
  das verwendete Verfahren zur Vergabe von Sequenznummern
- **Key Refresh**
  Angabe des Intervalls, nach dessen Ablauf neue Sicherheitsparameter vereinbart werden müssen

Nachdem diese Informationen feststehen und die jeweiligen Schlüssel erzeugt wurden, kann der Pending State zum neuen Current State gemacht werden, wobei der Pending State mit Standardwerten (keine Verschlüsselung, Kompression, MAC usw.) neu initialisiert wird. Der Connection State selbst enthält folgende Daten (vgl. [17, S. 40]; jeweils für beide Kommunikationsrichtungen):

- **Compression State**
  der momentane Zustand des Kompressionsalgorithmus (bei Verwendung von Datagrammen keine zustandsabhängige Kompression möglich!)
- **MAC Secret**
  bei der Nachrichtenauthentifizierung verwendetes Geheimnis
- **Encryption Key**
  Schlüssel für symmetrische Verschlüsselung
- **Initialisierungsvektor**
  nötig für die Blockverschlüsselung (CBC, vgl. Abschnitt 2.2)
- **Sequence Number**
  aktuelle Sequenznummer

Der eigentliche Versand der Anwendungsdaten gestaltet sich im Gegensatz zu den Verfahren beim Verbindungsaufbau relativ einfach (s. Abb. 15). Zunächst werden die Daten in Fragmente aufgeteilt und komprimiert, wobei in WTLS derzeit allerdings nur die *Null-Komprimierung* spezifiziert ist (keine Veränderung der Daten). Danach wird mittels der vereinbarten Hashfunktion aus dem komprimierten Fragment und dem Master Secret ein Nachrichtenauthentifizierungscode (MAC) erzeugt und an den Datenblock angehängt. Das Ergebnis dieser Operation wird symmetrisch verschlüsselt, wiederum nach dem vorher festgelegten Algorithmus und mit dem gemeinsamen Geheimnis; das Resultat dieser Verschlüsselung kann schließlich an den Kommunikationspartner versendet werden.

5.4 Angriffsmöglichkeiten und Verbesserungsansätze

Dadurch, daß mit TLS ein etablierter, von vielen Stellen kritisch überprüfter Standard die Grundlage für WTLS bildete, waren „kritischen Bereiche“ des Protokolls schon in der Entwurfsphase bekannt und vorher entdeckte Sicherheitslücken bereits geschlossen. Dennoch entstanden vor allem durch die Fokussierung auf eingeschränkte Ressourcen, insbesondere durch die Unterstützung des in TLS nicht vorgesehenen verbindungslosen Datentransports, neue Probleme und Schwachstellen, die zumindest bei der ersten Fassung des Standards Nachbesserung erforderten. Einige dieser Schwachpunkte sollen in diesem Kapitel dargestellt werden.

WAP Gap. Das wohl bedeutendste Problem bei WTLS ist weniger eine Sicherheitslücke im klassischen Sinn, sondern eher eine systembedingter, oft als „WAP Gap“ bezeichneter Schwachpunkt in der Architektur: da WTLS zwar an TLS angelehnt, aber nicht dazu kompatibel ist, kann es bei der Kommunikation zwischen mobilem Endgerät und Diensteanbieter im drahtgebundenen Internet – vermutlich die bei weitem häufigste Anwendungssituation – keine echte *end-to-end*-Sicherheit bieten. Bei einem solchen Dialog wird zwar Sicherheit zwischen dem mobilen Gerät und dem *WAP-Gateway* durch WTLS, zwischen Gateway und Internet-Server durch TLS gewährleistet; das Gateway selbst aber

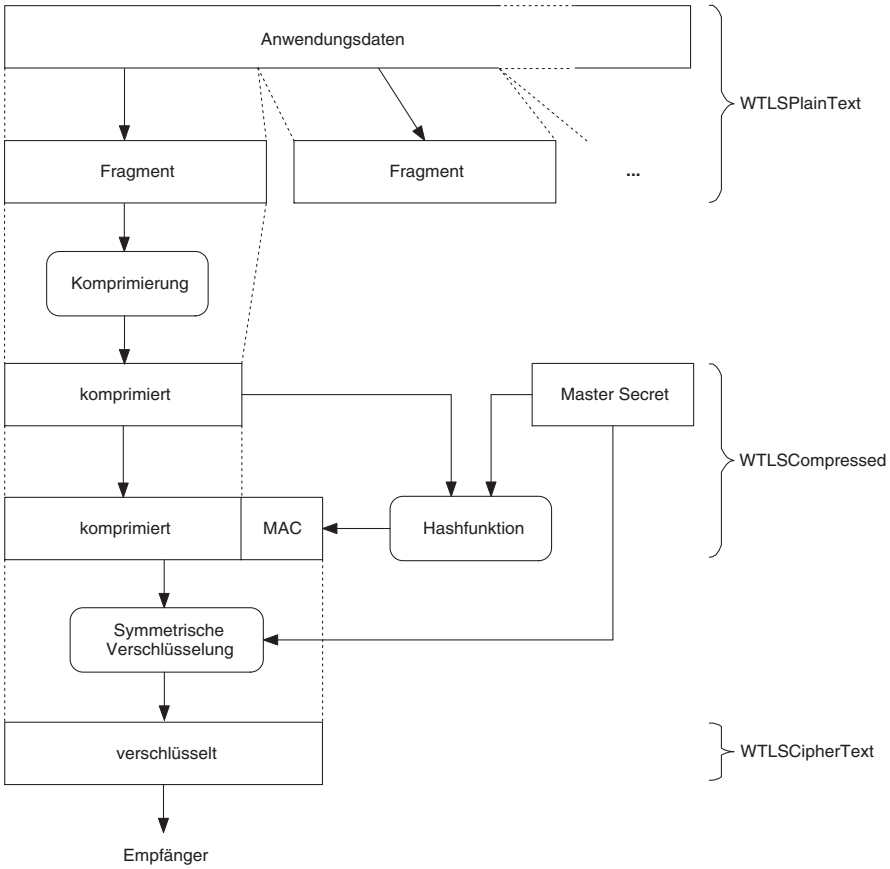

Abbildung 15. Verarbeitung der Anwendungsdaten im Record Protocol [8, S. 313]

muß zwischen den Protokollen „übersetzen“, d.h. die Nachrichten ent- und wieder verschlüsseln (sog. „hop-by-hop“-Verschlüsselung; vgl. Abb. 16).

Da bei diesem Vorgang sämtliche Daten zumindest zeitweise im Klartext vorliegen, sind die Sicherheitsmängel offensichtlich:

- Zugriff Dritter auf das Gateway-System ist unvermeidbar, auch bei bester Absicherung zumindest durch das Administrationspersonal des Betreibers
- ein solcher Knotenpunkt, über den zahlreiche Transaktionen abgewickelt werden, wird fast zwangsläufig zu einem primären Angriffsziel
- end-to-end-Authentifizierung zwischen mobilem Nutzer und Internet-Anbieter ist nicht möglich [10]
- potentielle rechtliche Probleme [11]

Hinzu kommen noch die typischen potentiellen Performance-Probleme derartiger Proxy-Architekturen, da das WAP-Gateway üblicherweise ein einzelnes, beim Netzbetreiber installiertes System ist, das eine große Zahl mobiler Geräte versorgt (*bottleneck*).

In einigen Anwendungen mag dieser Aufbau, sofern man dem Betreiber des Gateways vertraut, noch ausreichende Sicherheit bieten – zumindest für Finanzdienste und ähnliches ist es aber zweifellos inakzeptabel.

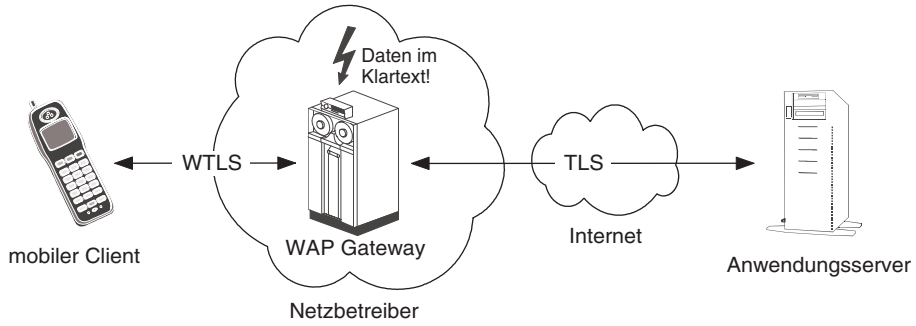

Abbildung 16. Gateway-basierte Verbindung ohne end-to-end-Sicherheit [11]

Ein Ansatz, die Sicherheit im Rahmen der gegebenen Einschränkungen zu erhöhen, ist daher die Verwendung eines dedizierten Gateways beim Dienstbetreiber; ggf. kann dieses sogar gemeinsam mit dem Anwendungsserver auf demselben System implementiert werden [9, S. 232]. Da sich somit der Endpunkt der WTLS-geschützten Verbindung zum Endbenutzer im direkten Einflußbereich des Anbieters befindet, bietet dieses Verfahren im weiteren Sinne end-to-end-Sicherheit (s. Abb. 17).

Der offenkundige Nachteil dieser Lösung besteht in der aufwendigen Infrastruktur, da jeder Anbieter Zugangspunkte für ein oder gar mehrere mobile Netze bereitstellen muß, und der Anwender wiederum unterschiedliche Account-Daten für sämtliche von ihm benutzten Dienste zu verwalten hat.

WAP bietet daher unter der Bezeichnung *Transport Layer End-to-End Security* die Möglichkeit, eine Verbindung zwischen mobilem Endgerät und Gateway/Server des Dienstanbieters (in diesem Fall bezeichnet als „Secure Subordinate Pull Proxy“) automatisiert durch das Gateway des Netzbetreibers („Master/Default Pull Proxy“) aufbauen zu lassen ([18]; s. Abb. 18). Da dieser Ansatz die Zugangshardware auf Seiten des Content-Anbieters überflüssig macht, ist er wesentlich einfacher und damit günstiger zu realisieren als der zuvor genannte. Allerdings ist hier eine Kooperation mit einem oder mehreren Netzbetreibern

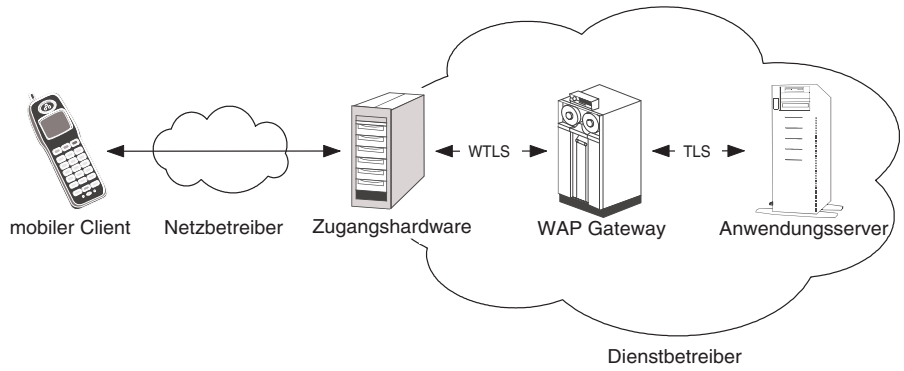

Abbildung 17. Implementierung des Gateways beim Dienstbetreiber (vgl. [18, S. 8])

notwendig, was für private bzw. nicht-kommerzielle Zwecke möglicherweise ein Hindernis darstellt.

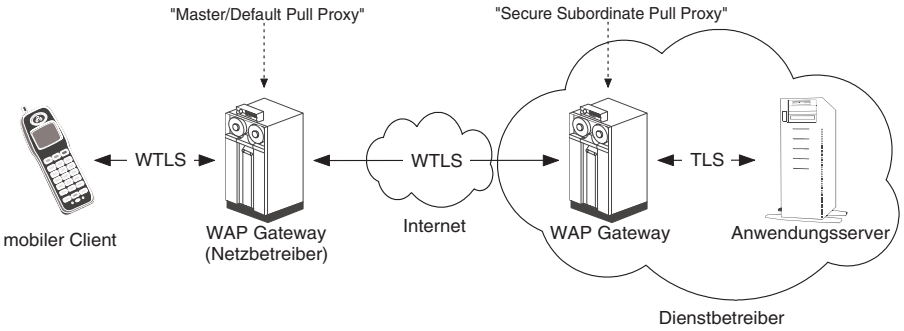

Abbildung 18. WAP Transport Layer End-to-End Security (vgl. [18, S. 8])

Schwache kryptographische Algorithmen. Gelegentlich werden schwache Verschlüsselungsalgorithmen als Mängel der WTLS-Spezifikation angeführt [13, 14]. Insbesondere betrifft dies die DES-Verschlüsselung mit 40 Bit Schlüssellänge und die Nachrichtenauthentifizierung durch SHA_XOR_40.

Da ein DES-Schlüssel in jedem Byte ein Paritätsbit enthält, verkürzt sich seine effektive Länge dadurch um den Faktor 32 auf 35 Bit, was keinen ausreichenden Schutz vor Brute-Force-Angriffen darstellt (vgl. [8, S. 297]). In neueren WTLS-Revisionen wird allerdings ohnehin darauf hingewiesen, daß 40-Bit-DES

allenfalls noch aus Kompatibilitätsgründen implementiert, aufgrund der schwachen Verschlüsselung aber nicht mehr verwendet werden sollte [17, S. 82].

Bei SHA_XOR_40 wird ein 40 Bit langer MAC erzeugt, indem eine Nachricht zunächst in 5-Byte-Blöcke aufgeteilt wird und diese dann per XOR miteinander verknüpft werden. Daß diese Methode keinen ausreichenden Schutz der Integrität gewährleistet, ist offensichtlich: Die Veränderung eines beliebigen Bits n der Nachricht erfordert jeweils nur das Invertieren des entsprechenden Bits (n mod 40) im MAC. Aus diesem Grund wurde SHA_XOR_40 in der dritten WTLS-Revision aus der Liste möglicher MAC-Algorithmen entfernt [17, S. 5].

Beide Kritikpunkte sind (bzw. waren) an sich korrekt, lassen jedoch die Intention von WTLS außer Acht: eine flexible Sicherheitsarchitektur kann schließlich durchaus auch schwache Algorithmen anbieten, solange sich diese je nach Bedarf deaktivieren lassen (was dem Server-Betreiber, wie in Kap. 5.3 beschrieben, freisteht). Allerdings spielen hier wieder die in Abschnitt 5.4 genannten Probleme eine Rolle, da der Diensteanbieter im Allgemeinen keinen Einfluß auf die vom Gateway akzeptierten Sicherheitsparameter hat, sofern er dieses nicht selbst betreibt.

Nicht-authentifizierte Alert-Nachrichten. In der ursprünglichen WTLS-Spezifikation existierten einige Alert-Meldungen (meist „Warnings“, also nicht mit Verbindungsabbruch verbunden), die im Klartext und ohne Authentifizierung verschickt wurden. Da bei verbindungslosem Transport auch solche Nachrichten eine Sequenznummer benötigen, ist es einem Angreifer möglich, beliebige verschlüsselte Pakete durch Versand von Fehlermeldungen mit gleicher Sequenznummer aus dem Datenstrom zu „entfernen“ (*Datagram Truncation Attack*); auch ein Denial-of-Service-Angriff ist denkbar.

Diese Sicherheitslücke wurde mittlerweile behoben: Alert-Nachrichten beinhalten nun eine 4 Byte lange Prüfsumme, generiert aus dem zuletzt empfangenen WTLSCipherText-Block [17, S. 47].

Non-Repudiation. Ein weiterer Nachteil von WTLS ist das Fehlen des Sicherheitsmerkmals *Non-Repudiation* (dt. *Nicht-Anfechtbarkeit*), d.h. die Nachweisbarkeit der Übertragung einer Nachricht. Gerade im Hinblick auf e-Commerce-Anwendungen, etwa elektronische Bezahlvorgänge, wäre diese Eigenschaft sinnvoll.

5.5 Weitere Entwicklung

Alternative Lösungen. Aufgrund der vor allem in der ursprünglichen WTLS-Version vorhandenen Lücken wurden einige alternative Sicherheitskonzepte vorgeschlagen, die diese beseitigen sollten; stellvertretend sollen an dieser Stelle zwei davon angesprochen werden.

Beim in [10] vorgestellten *WAE-Sec* etwa soll durch ein TLS-kompatibles, im Wireless Application Environment implementiertes Protokoll das WAP-Gap umgangen werden; diese neue Sicherheitsschicht kommuniziert dabei an WSP und

WTP „vorbei“ direkt mit der Transportschicht. Auf potentielle Performance-Probleme durch die vollständige TLS-Kompatibilität wird im genannten Artikel allerdings nicht näher eingegangen.

[11] dagegen beschreibt mit *KSSL* („Kilobyte SSL“) eine konkrete Implementierung von TLS auf mobilen Endgeräten auf Grundlage der *Java 2 Mobile Edition* (J2ME). Zunächst wird hier die Annahme, TLS würde sich für den Einsatz in mobilen Netzen grundsätzlich nicht eignen, in Frage gestellt und zumindest teilweise relativiert: Ein vollständiger Handshake-Vorgang beispielsweise kann mit KSSL nach Aussage der Autoren in einem Zeitraum von etwa 10-13 Sekunden abgewickelt werden[6], was bei sicherheitskritischen Anwendungen wie Online-Banking durchaus vertretbar sei.

Neben diesen auf den Sicherheitsbereich beschränkten Ansätzen existiert sogar ein vollständiges Alternativmodell zu WAP: Das *Lightweight and Efficient Application Protocol* (LEAP) [15] umfasst ein komplettes Schichtenmodell zur mobilen Datenübertragung auf der Basis freier Protokolle. Diese sowie weitere Vorschläge verbindet jedoch die Tatsache, daß ihr Einsatz in „freier Wildbahn“ recht unwahrscheinlich bleibt. Dadurch, daß ein Großteil der marktbeherrschenden Unternehmen aus dem Mobilfunkbereich den WAP-Standard aktiv oder passiv unterstützt, wurden mit dem millionenfachen Verkauf WAP-fähiger Endgeräte in den letzten Jahren Fakten geschaffen, die auch durch technisch besser durchdachte Lösungen nicht rückgängig gemacht werden können. Da diese Unterstützung wohl zumindest mittelfristig weiter bestehen wird, soll nun abschließend ein kurzer Blick auf die geplante Weiterentwicklung des Standards geworfen werden.

WAP 2.0. Die wesentliche Neuerung in WAP 2.0 [22] ist die grundsätzliche Orientierung hin zu etablierten Internet-Standards, insbesondere IP, TCP und HTTP. Ermöglicht wird dies vor allem durch die schnell fortschreitende Entwicklung sowohl der Endgeräte (wesentliche verbesserte Rechen- und Speicherkapazität) als auch der Mobilfunknetze (höhere Übertragungsraten). Als logische Konsequenz dieser Entwicklung wird somit auch das bislang notwendige WAP-Gateway beim Übergang zwischen Mobilfunknetz und Internet obsolet. Der existierende „alte“ Protokollstack bleibt aus Gründen der Abwärtskompatibilität und für nicht IP-fähige Trägerdienste erhalten.

Auf der Anwendungsebene wird mit *XHTML Mobile Profile* (XHTMLMP) eine auf dem W3C-Standard XHTML beruhende Markup-Sprache unterstützt; bestehende WML-Daten können per XSLT in das neue Format überführt werden.

6 Schlußbemerkung

Aufgrund der sehr unterschiedlichen thematischen Ausrichtung der hier vorgestellten Lösungen fällt es schwer, ein abschließendes Fazit zu ziehen. Auffällig

[6] auf gebräuchlichen Endgeräten in einem Netz mit 9,6 kBit/s Übertragungsrate

ist vielleicht eine Beobachtung, die sich analog auch in vielen anderen Bereichen machen lässt: auch gut durchdachte, sinnvolle Entwicklungen können sich kaum am Markt durchsetzen, wenn das wirtschaftliche, gesellschaftliche oder politische Umfeld diesen Prozess zum gegebenen Zeitpunkt verhindert. Einfacher ausgedrückt: technologische Ausgereiftheit ist bei weitem keine hinreichende Bedingung für kommerziellen Erfolg (gelegentlich nicht einmal eine notwendige).

Der Ansatz der Mix-Netzwerke ist vielversprechend, allerdings ist es fraglich ob ein so hohes Maß an Anonymität – im Hinblick auf die Unterbindung jeder Verfolgungsmöglichkeit, insbesondere auch bei Straftaten – überhaupt allgemein erwünscht ist.

Das digitale Geld ist ein hervorragendes Beispiel für eine Technologie, die ihrer Zeit weit voraus ist; erst jetzt, Jahre nach ihrer ersten Einführung, lässt die Marktdurchdringung mobiler Endgeräte in vielen Ländern einen breiten Einsatz sinnvoll erscheinen.

WTLS bzw. WAP schließlich zeigt im Gegenzug, daß sich auch Entwicklungen, bei denen die Ausführung gelegentlich Fragen offen und die Akzeptanz zunächst lange auf sich warten lässt, bei ausreichend gegebener Rückendeckung durch wichtige Konzerne und mit teils drastischen „Umbaumaßnahmen“ letztendlich im Markt plazieren lassen.

Literatur

1. Schneier, B. Angewandte Kryptogaphie. Addison-Wesley, 1. Auflage, 1996.
2. Federrath, H., Jerichow, A. und Pfitzmann, A. MIXes in Mobile Communication Systems: Location Management with Privacy. Institut für Informatik, TU Dresden, 1996.
3. Jendricke, U. und Rannenberg, K. A MixDemonstrator for teaching Security in the virtual University. Institut für Informatik, Universität Freiburg, 1996. http://www.iig.uni-freiburg.de/telematik/forschung/projekte/kom_technik/viror/mixDemo.html
4. Heise-News: Staatsanwaltschaft kritisiert „Spitzel-SMS“ der Polizei. 06.04.2003 http://www.heise.de/newsticker/data/tol-06.04.03-001/
5. Medvinsky, G. und Neuman, B.C. NetCash: A design for practical electronic currency on the Internet. *Proceedings of the 1st ACM Conference on Computer and Communication Security*, 1993. http://www.isi.edu/people/bcn/papers/pdf/9311_netcash-medvinsky-neuman-cccs93.pdf
6. Medvinsky, G. und Neuman, B.C. Requirements for Network Payment: The NetCheque Perspective. *Proceedings of IEEE COMPCON'95*, 1995. http://www.isi.edu/people/bcn/papers/pdf/9503_netcheque-neuman-medvinsky-compcon95.pdf
7. O'Mahony, D. und Peirce, M. Scaleable, Secure Cash Payment for WWW Resources with the PayMe Protocol Set. http://www.w3.org/Conferences/WWW4/Papers/228/
8. Roth, J. *Mobile Computing. Grundlagen, Technik, Konzepte.* dpunkt.verlag, 2002.
9. Dalton, C., King, C., und Osmanoglu, E. *Security Architecture: Design, Deployment and Operations.* Osborne-McGraw-Hill, 2001.
10. Ponce, D. und Soriano, M. A Security and Usability Proposal for Mobile Electronic Commerce. *IEEE Communications Magazine*, S. 62–67, 2002.

11. Gupta, S. und Gupta, V. Securing the Wireless Internet. *IEEE Communications Magazine*, S. 68-74, 2001.
12. Do, T.V. WAP Security: WTLS. 2001. http://ece.gmu.edu/courses/ECE636/project/reports/TDo.pdf
13. Saarinen, M.-J. Attacks against the WAP WTLS Protocol. http://www.cc.jyu.fi/~mjos/wtls.pdf
14. Jormalainen, S. und Laine, J. Security in the WTLS. 2000. http://www.hut.fi/~jtlaine2/wtls/
15. Banan, M. LEAP: One Alternative to WAP. 2000. http://www.freeprotocols.org/LEAP/Manifesto/article/LEAP-OneAlternative/main.pdf
16. Open Mobile Alliance. WAP Architecture, Version 30-Apr-1998. http://www1.wapforum.org/tech/terms.asp?doc=SPEC-WAPArch-19980430.pdf
17. Open Mobile Alliance. Wireless Transport Layer Security, Version 06-Apr-2001. http://www.openmobilealliance.org/wapdocs/wap-261-wtls-20010406-a.pdf
18. Open Mobile Alliance. WAP Transport Layer End-to-end Security, Approved Version 28-June-2001. http://www.openmobilealliance.org/wapdocs/wap-187-transporte2esec-20010628-a.pdf
19. Open Mobile Alliance. Wireless Datagram Protocol, Version 14-Jun-2001. http://www.openmobilealliance.org/wapdocs/wap-259-wdp-20010614-a.pdf
20. Open Mobile Alliance. Wireless Transaction Protocol, Version 10-Jul-2001. http://www.openmobilealliance.org/wapdocs/wap-224-wtp-20010710-a.pdf
21. Open Mobile Alliance. Current Members. http://www.openmobilealliance.org/currentmembers.html
22. Open Mobile Alliance. WAP 2.0 Technical White Paper. 2002. http://www.wapforum.org/what/WAPWhite_Paper1.pdf