Berner Fachhochschule
Technik und Informatik

Bachelor Thesis 2013

Simulation eines Road-Pricing Systems

Schutz der standortbezogenen Privatsphäre mit VPriv

Fachbereich Informatik
Studierende Michael Gautschi
Daniel Brönnimann
Professor Prof. Dr. Eric Dubuis

Management Summery

Unter dem Begriff *Road-Pricing* versteht man das Erheben von Gebühren für die Benützung von Strassen. In bereits vorhandenen Mautsystemen werden die Fahrzeuge erfasst und die Daten zentral ausgewertet. Die standortbezogene Privatsphäre ist bei den eingesetzten Systemen nicht gewährleistet.

In der Schweiz wird aufgrund des zunehmenden Verkehrs, insbesondere in urbanen Gebieten, über die Einführung eines Road-Pricing Systems politisch diskutiert.

Ein Road-Pricing System erfasst die zurückgelegte Strecke eines Fahrzeuges und berechnet die entsprechenden Mautgebühren. Einerseits muss der Staat die Gebühren korrekt berechnen können, andererseits darf er keine Kenntnisse haben über Zeitpunkt und zurückgelegten Weg eines Fahrzeuges. Mit Hilfe moderner Kryptographie lässt sich diese Diskrepanz lösen. Die standortbezogene Privatsphäre ist gewährleistet.

Das *VPriv Verfahren* bietet eine mögliche Lösung diese Anforderungen zu erfüllen. VPriv wurde am Massachusetts Institute of Technology MIT entwickelt und in der Fachliteratur veröffentlicht.

Das Verfahren kann in drei Phasen erklärt werden:

1. Der Autofahrer führt auf seinem persönlichen Computer eine Clientapplikation aus. Der Client generiert Zufallswerte (genannt *Tags*) und übermittelt diese dem Fahrzeug.
2. Die Positionsdaten werden vom Fahrzeug während der Fahrt aufgezeichnet. Sie sind jedoch nicht an das Fahrzeug selbst gebunden, sondern an die generierten Tags.
3. Die aufgezeichneten Positionsdaten werden nach einer Verrechnungsperiode, beispielsweise nach einem Monat, anonym einem Server übermittelt. Der Server weist mittels der Kostenfunktion jedem Tag eine Gebühr zu. Anschliessend sendet er alle Daten dem Client. Jeder Client erhält alle Daten, also auch jene der anderen Systemteilnehmer. Der Client wählt seine Tags aus und berechnet die totalen Mautgebühren.

Der Client muss nun mittels des *Rundenprotokolls* dem Server beweisen, dass er bei der Kostenberechnung nicht betrogen hat. Dazu legt der Client, in zwei im Rundenprotokoll definierten Fällen, gewisse Informationen offen. Diese zwei Fälle sind nötig, damit die standortbezogene Privatsphäre weiterhin gewährleistet bleibt. Im Fall 0 beweist der Client, dass jedem Tag die korrekten Kosten zugewiesen wurden. Im Fall 1 wird überprüft, ob der Client seine eigenen Tags benutzt hat und die totalen Kosten korrekt berechnet wurden. Fall 0 oder 1 wird zufällig gewählt. Durch das mehrmalige Durchlaufen des Protokolls wird die Wahrscheinlichkeit erhöht, dass der Server den Betrug erkennt. Das Verfahren ermöglicht einen Betrugsnachweis von 99.9 Prozent.

Auf dem Strassennetz werden zusätzlich *Spot Checks* installiert, welche die durchfahrenden Fahrzeuge erfassen. Wird ein Fahrzeug von einem solchen Spot Check aufgezeichnet, muss der Fahrzeughalter beweisen, dass er zu gegebener Zeit an diesem Ort war. Damit wird verhindert, dass der Client aufgezeichnete Positionsdaten dem Server vorenthält. Der Fahrzeughalter könnte so Kosten sparen. Die Spot Checks verletzten zwar die standortbezogene Privatsphäre, jedoch ist deren Verteilung nur sehr gering.

Im Rahmen der Bachelor Thesis wurde das VPriv Verfahren studiert und mittels einer Verkehrssimulation mit integriertem Road-Pricing System umgesetzt. Die einfache Verkehrssimulation ermöglicht es, über eine bestimmte Dauer Positionsdaten einer definierten Anzahl von Fahrzeugen aufzuzeichnen. Diese Daten bilden die Basis des VPriv Verfahrens.

Die programmierte Simulationsanwendung dient zur detaillierten Demonstration von VPriv. Der Schutz der Privatsphäre des Autofahrers wird aufgezeigt, indem die auf dem Server vorhandenen Daten denjenigen des Clients gegenübergestellt werden.

Das VPriv Verfahren zeigt interessante Ansätze um den Schutz der standortbezogenen Privatsphäre in einem Road-Pricing System zu gewährleisten, ist jedoch rechen- und zeitintensiv in der Ausführung. Die fehlende Benutzerfreundlichkeit und Praxistauglichkeit zeigt sich vor allem hinsichtlich der Benutzerinteraktion am Ende der Verrechnungsperiode. Die Akzeptanz möglicher Nutzer eines Road-Pricing Systems nach dem VPriv Verfahren ist deshalb eher als gering einzuschätzen.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Ausgangslage

Aufgrund des Bevölkerungswachstums und der zunehmenden Mobilität insbesondere in urbanen Gebieten, können unterschiedlichste Verkehrsprobleme wie überlastete Strassen, Staus oder verstärkter Verschleiss der Infrastruktur auftreten. Eine verursachergerechte Kostenbeteiligung steht auch in der Schweiz vermehrt im Zentrum politischer Diskussionen.

Ein möglicher Lösungsansatz zur Reduktion des Strassenverkehrs ist die Einführung eines Road-Pricing Systems. Ein solches Gebührensystem hat insbesondere zum Ziel, die Verkehrsteilnehmer verursachergerecht an den Kosten zu beteiligen, den aufkommenden Verkehr zu verringern und den Verkehrsfluss zu verbessern. Das Road-Pricing ist demzufolge als Lenkungsabgabe für den Individualverkehr zu betrachten. Mautsysteme sind in einigen grösseren Städten bereits heute im Einsatz.

Bei den zurzeit verwendeten Systemen werden die Positionsdaten der Fahrzeuge zentral erfasst und ausgewertet. Die standortbezogene Privatsphäre der Autofahrer ist bei dieser Lösung nicht gewährleistet. Die vergleichsweise strengen Datenschutzanforderungen der Schweiz lassen ein solches Road-Pricing System hierzulande nicht zu.

VPriv ist ein Verfahren, welches die standortbezogene Privatsphäre der Autofahrer mit Hilfe der Kryptografie schützt. Das System erfasst anonym den zurückgelegten Weg eines Fahrzeuges und berechnet die entsprechenden Mautgebühren.

VPriv wurde am Massachusetts Institute of Technology MIT in Boston entwickelt und in der Fachliteratur im Jahr 2009 veröffentlicht. Der Protokollablauf und die Funktionsweise von VPriv wurden im Rahmen der Projektarbeit 2 beschrieben und untersucht. Die in VPriv verwendeten kryptografischen Verfahren wurden in der Vorstudie thematisiert. Dazu gehören insbesondere das Commitment-Verfahren sowie die Pseudorandom Funktionen. Offene und unklare Punkte, die während der Vorstudie nicht geklärt werden konnten, werden im Rahmen dieser Bachelor Thesis bearbeitet.

2 Ziele

Das Ziel der Bachelor Thesis ist die Simulation eines Road-Pricing Systems nach dem Verfahren von VPriv mit folgenden Anforderungen:

Demonstration des VPriv Verfahrens

Das VPriv Verfahren wird in der Simulationsanwendung umgesetzt. Bei der Initialisierung der Anwendung werden, ohne Eingaben des Simulationsanwenders, die Fahrzeuge auf dem Server registriert und die nötigen Daten übertragen. Die Demonstration des Verfahrens erfolgt während einer simulierten Verrechnungsperiode. Während dieser Zeit fahren die Fahrzeuge über das Strassennetz und sammeln Positionsdaten in einem vorgegebenen Rhythmus. Die gesammelten Positionsdaten werden schlussendlich anonym an den Server übermittelt.

Die aufgezeichneten zurückgelegten Wegstrecken, die daraus resultierenden Strassennutzungsgebühren und das Gesamttotal sind am Ende eines Simulationszyklus für den Anwender ersichtlich. Mit dem VPriv Verfahren, kann die Korrektheit der Kostenberechnung überprüft werden. Die Vorgänge und Berechnungen werden dem Simulationsanwender detailliert ausgegeben.

Demonstration des Schutzes der standortbezogenen Privatsphäre

Am Ende eines Simulationszyklus sind für den Simulationsanwender die dem Server bekannten Informationen ersichtlich. Der Server besitzt die Registrationsdaten, kennt die total zu bezahlenden Mautgebühren und die gesammelten Positionsdaten aller Fahrzeuge. Daraus ist ersichtlich, dass der Server eine grosse Menge von Positionsdaten besitzt, diese aber keinem bestimmten Fahrzeug zuweisen kann. Er ist nicht in der Lage Wegprofile eines bestimmten Fahrzeuges zu erstellen. Die standortbezogene Privatsphäre ist somit durch das VPriv Verfahren gewährleistet.

Simulation eines Verkehrssystems

Um das VPriv Verfahren zu simulieren, wird ein einfaches Verkehrssystem realisiert. Während einer bestimmten Zeit fahren Fahrzeuge autonom auf dem vorhandenen Strassennetz und wählen ihren Weg eigenständig. Auf die Simulation von Geschwindigkeit, Trödelfaktor und Bremsen bzw. Beschleunigung wird verzichtet. Im Strassennetz gibt es unterschiedliche Strassentypen, welche mehr oder weniger durch die Fahrzeuge befahren werden. Die Art der Strasse beeinflusst die Wahl des Weges der einzelnen Fahrzeuge und die Gebühr.

Technische Anforderungen

Die Simulation wird in der Programmiersprache Java umgesetzt. VPriv beinhaltet eine Client- und eine Serverkomponente, sowie einen Transponder im Fahrzeug. Alle Komponenten und die Simulations-Engine laufen in derselben Java Virtual Machine (Java VM), das heisst auf demselben Computer und in derselben Programminstanz.

Um die korrekte Funktionsweise von VPriv zu gewährleisten, sind kryptographische Verfahren notwendig. Einige werden mit Hilfe der Unicrypt Bibliothek der BFH umgesetzt.

3 Projektplan

Vorgang	**bis**
Planung	Mi 27.02.13
Ausgangslage und Anforderungsspezifikation	Di 26.02.13
Projektplan	Mi 27.02.13
Studium und Konzept	Di 26.03.13
Unklarheiten zu VPriv und kryptografischen Funktionen	Di 12.03.13
Voranalyse und Konzept	Di 26.03.13

Realisation	**Meilenstein**
Simulation	
Strassennetz	Di 23.04.13
Verkehrssimulation und Wegfindung	Di 21.05.13
VPriv	
Registration	Di 09.04.13
Driving (Serverseitig)	Di 16.04.13
Reconciliation	Di 23.04.13
Rundenprotokoll (Client und Server)	Di 14.05.13
Transponder	Di 21.05.13
Integration VPriv in Simulation	Di 04.06.13

Abschluss	**bis**
Projektreflektion	Fr 14.06.13
Abgabe Dokumentation	Fr 14.06.13

Tabelle 1: Projektplan

Die Realisation ist in zwei Teilbereiche aufgeteilt: die Realisation der Verkehrssimulation und die Realisation von VPriv. Beide Vorgänge laufen parallel ab (ersichtlich im Gantt-Diagramm auf der folgenden Seite). In der Realisationsphase sind Meilensteine definiert, welche abschliessend realisiert werden. In einer solchen iterativen Phase wird die Detailspezifikation erstellt, das Modul implementiert, getestet und anschliessend mit den anderen Modulen vereint. Als letzte Realisationsphase werden VPriv und die Verkehrssimulation zusammengeführt.

Gantt-Diagramm

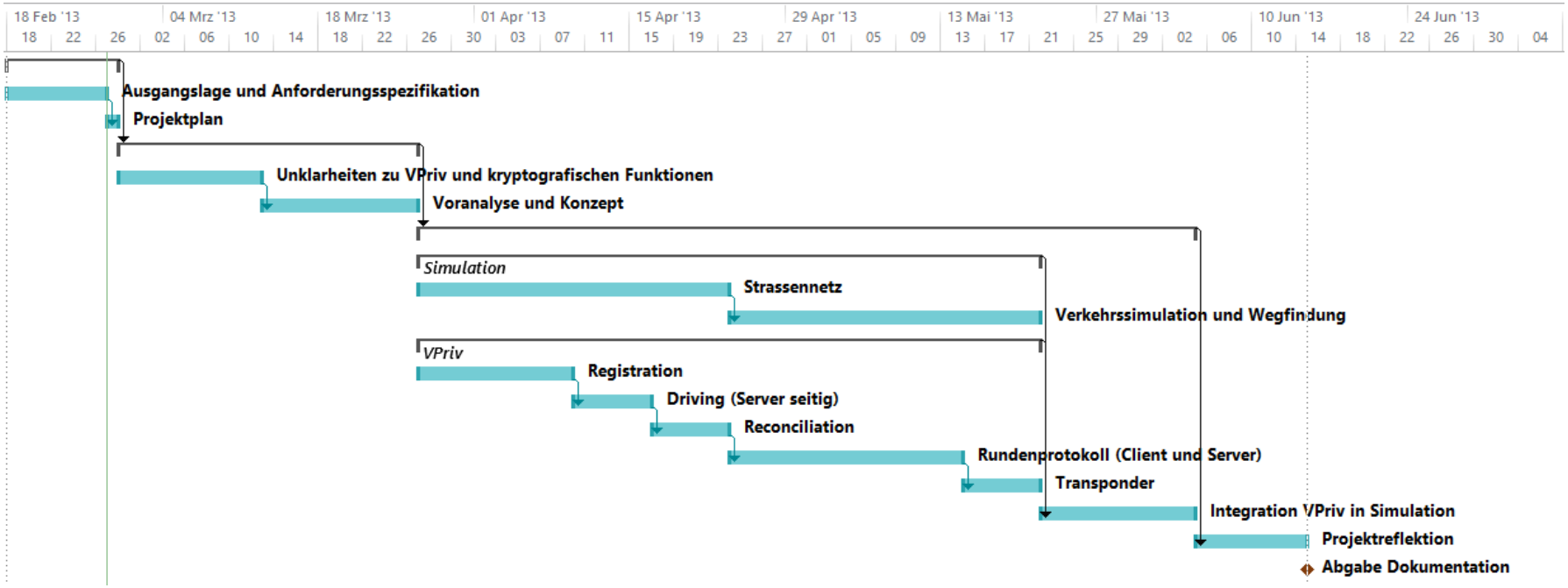


Abbildung 1: Projektplan Gantt-Diagramm

4 Protokoll VPriv

Das VPriv Protkoll wurde im Rahmen der Projektarbeit 2 bereits beschrieben. Aufgrund der Vollständigkeit wird der Protokollablauf hier korrigiert, leicht abgeändert und nochmals aufgezeigt. Die Sicherheitsanalyse des VPriv Verfahrens wurde durch die VPriv Autoren durchgeführt und wird detailliert im VPriv Paper beschrieben.

Das VPriv System besteht aus zwei Softwarekomponenten: der Clientapplikation, welche auf dem persönlichen Computer des Fahrzeughalters ausgeführt wird und der Serversoftware. Der *Transponder* ist im Fahrzeug eingebaut und zeichnet während der Fahrt die Positionsdaten auf.

4.1 Protokoll-Phasen

Registration

Das Fahrzeug wird mit dem Nummernschild beim Server registriert. Der Client generiert danach eine Sammlung von Zufallswerten (genannt *Tags*), damit sie aufgrund des Schutzes der Privatsphäre nicht mit einem Fahrzeug in Verbindung gebracht werden können. Diese Tags werden auf den Transponder im Fahrzeug übermittelt. Der Server kennt diese Tags nicht. Der Client legt sich aber durch *Commitments* (Erklärung siehe Anhang) auf die ausgewählten Tags fest. Die Commitments werden dem Server gesendet und mit der Identität des Autofahrers verknüpft. Die Commitments erlauben keine Rückschlüsse auf die Tags.

Driving

Während der Fahrt generiert der Transponder *Point Tuples*, bestehend aus `<tag, time, location>`. Diese Tuples werden auf den Server übermittelt. Jedes Point Tuple ist eindeutig, da die zufällig generierten Tags nur einmal verwendet werden. Der Server weiss jedoch nicht, von wem die hochgeladenen Tuples sind.

Reconciliation

Diese Phase läuft am Ende einer Rechnungsperiode ab. Der Server berechnet mittels der *Kostenfunktion* zu jedem Tag die dazugehörenden Strassennutzungsgebühren.
Der Server hat in seiner Datenbank eine grosse Anzahl von Tuples der verschiedenen Fahrzeuge gespeichert. Der Server weiss aber nicht, welche Tuples zu welchem Fahrzeug gehören. Mit dieser Tuple-Datenbank und den Commitments ist es möglich, eine korrekte Rechnung zu erstellen: Der Client berechnet die Kosten. Danach beweist die Clientapplikation dem Server (mittels *Zero-Knowledge-Beweis*), dass er die Kosten richtig berechnet hat.

4.2 Protokollablauf [1]

	Client	Server
Registration	(a) **Wählt zufällige Fahrzeug-Tags** v_i. Für jede Runde im Rundenprotokoll (siehe nächste Seite) wird zufällig ein Schlüssel k_r erzeugt. Es wird pro Runde ein Schlüssel benötigt, damit keine Informationen zwischen den Runden offen gelegt werden. (i = Index Tag, r = Index Runde)	
	(b) **Verschlüsselt alle gewählten Fahrzeug-Tags** mit jedem Rundenschlüssel k_r indem $f_k(v_i)$ berechnet wird.	
	(c) Berechnet das Commitment der Schlüssel $c(k_r)$ und der verschlüsselten Fahrzeug-Tags $c(f_k(v_i))$. **Damit legt sich der Client auf die verwendeten Schlüssel sowie die ausgewählten Fahrzeug-Tags fest.** Speichert die zugehörigen Decommitment Schlüssel $d(k_r)$ und $d(f_k(v_i))$.	
	(d) Sendet $c(k_r)$ und $c(f_k(v_i))$ zum Server. Dies **verhindert, dass andere Tags vom Fahrzeug verwendet werden.**	Bindet die Werte an das Fahrzeug.
Driving	Der Transponder **produziert während der Fahrt Point Tuples** und sendet diese zum Server.	Speichert v_i als s_j
Reconciliation		**Berechnet** für alle erhaltenen Fahrzeug-Tags s_j **die dazugehörigen Mautgebühren** t_j
		Sendet alle Tuples ***(s_j, t_j)*** **an den Client**, bei welchen die Kosten $t_j>0$. Der Client erhält alle Tuples der Systemteilnehmer, da der Server nicht weiss, welche Tuples wem gehören.
	Wählt seine Tuples aus und berechnet die **totalen Mautgebühren** und sendet den Wert an den Server.	

Tabelle 2: VPriv Protokollablauf

Das Rundenprotokoll zur Überprüfung der korrekten Kostenberechnung beginnt (siehe nächste Seiten).

[1] Protokollablauf wurde aus der Projektarbeit 2 zur Vollständigkeit übernommen und korrigiert

Rundenprotokoll

Das Rundenprotokoll, zur Überprüfung der korrekten Kostenberechnung, ist Teil der Reconciliation Phase. Der Client legt dazu in zwei Fällen gewisse Informationen offen. Die zwei Fälle sind nötig, damit die standortbezogene Privatsphäre weiterhin gewährleistet ist. Durch das mehrmalige Durchlaufen des Protokolls erhöht sich die Wahrscheinlichkeit, dass der Server Betrug durch den Client feststellt.

	Client	**Server**
Initialisierung	(a) **Mischen der Paare** (s_j, t_j) Das heisst er sendet sie nicht in der Abfolge wie sie vom Fahrzeug aufgezeichnet wurde. So kann der Server kein Wegprofil erstellen. (b) **Verschlüsselt alle vom Server empfangenen Tags** mit dem aktuellen Rundenschlüssel k, $f_k(s_j)$. (c) **Berechnet das Commitment** $c(t_j)$ (d) **Speichern der Decommitments** $d(f_k(s_j))$, $d(t_j)$	
	$f_k(s_j)$, $c(t_j)$ →	
		Der Server **wählt zufällig** ***b = 0 oder b = 1***
	← b	
wenn ***b = 0***	Schlüssel k der aktuellen Runde, Decommitment $d(k)$, Decommitment $d(t_j)$ →	
		(1) Server **überprüft**, ob das gesendete k mit dem Decommitment $d(k)$ übereinstimmt. Das **Commitment** $c(k)$ hat der Server bei der Registrierung erhalten. REVEAL($c(k)$, k, $d(k)$) (2) Server verschlüsselt alle seine s_j (von allen Fahrzeugen) mit dem erhaltenen k. Diese gleicht er mit den vom Client bei der Initialisierung (oben) übermittelten $f_k(s_j)$ ab. Server **überprüft** $c(t_j)$ (vom Client festgelegt) mittels dem erhaltenem $d(t_j)$ und den gespeicherten **Kostentags** t_j auf dem Server. REVEAL($c(t_j)$, t_j, $d(t_j)$)

<table>
<tr><td rowspan="3">wenn b = 1</td><td>Berechnung des Decommitments D der totalen Kosten, welcher der Client zu bezahlen hat.

Dazu wird die Schnittmenge der Client und der Servertags gebildet. $I = \{v_i\} \cap \{s_i\}$.
Das bedeutet nur die Kostentags t_j des Clients werden genutzt.

$COST = \sum t_j$

$D = \sum d(t_j)$</td><td></td></tr>
<tr><td colspan="2">Totale Kosten COST,
Decommitment der totalen Kosten D,
$f_k(v_i)$,
$d(f_k(v_i))$
⟶</td></tr>
<tr><td></td><td>(1) Es wird überprüft, ob $d(f_k(v_i))$ ein Decommitment von $f_k(v_i)$ ist.

Der Cipher $c(f_k(v_i))$, welches zur Überprüfung dieses Decommitment benötigt wird, wurde bereits bei der Registration an den Server übermittelt.

(2) Es wird geprüft, ob der Client die Summe COST korrekt berechnet hat.

Server berechnet:
$\prod_{j,k,f_k(v_i)=f_k(s_j)} c(t_j) \ mod\ p$

Server überprüft mittels des erhaltenen D, ob diese berechnete Zahl ein Commitement von COST ist:
$c = g^{COST} h^D$ muss gleich sein wie $\prod c(t_j)$

Oder anders ausgedrückt: $c(\sum t_j) = \prod c(t_j)$

Dadurch hat der Server COST überprüft. Die homomorphe Eigenschaft des Pederson Commitments (Erklärung siehe Anhang) erlaubt dies ohne jemals die einzelnen t_j erfahren zu haben.</td></tr>
</table>

Tabelle 3: VPriv Rundenprotokoll

4.3 Unklarheiten

Die offenen und unklaren Punkte im VPriv Verfahren, bezüglich Kosten- und Pseudorandom-Funktion, werden hier erläutert.

Kostenfunktion

In der Vorstudie der Projektarbeit 2 wurde eine Lösung aufgezeigt bezüglich der Unklarheiten der Kostenfunktion. Folglich wird die Mautgebühr pro Wegstrecke berechnet. Dazu werden die Point Tuples wie folgt angepasst: `<tag, time1, location1, time2, location2>`

Fügt man jedoch einem Point Tuple eine zweite Location zu, ist es dem Server möglich, ein Wegprofil zu erstellen. Die standortbezogene Privatsphäre ist nicht mehr gewährleistet. Die Point Tuples können vom Server zu einer Kette gereiht werden, da die Location 2 eines Point Tuples 1 stets mit der Location 1 eines Point Tuple 2 übereinstimmt.

Aufgrund der vorgängigen Erklärung, handelt es sich bei der vorgeschlagenen Lösung um eine Fehlüberlegung. Die Kostenzuweisung kann nicht auf der Basis einer gefahrenen Strecke auf einer Mautstrasse getätigt werden.

Deshalb wird die ursprüngliche Kostenfunktion der VPriv Autoren verwendet. Einem vorhanden Point Tuple weist der Server einen Kostenwert zu, der gemäss dem Strassennetz vordefiniert ist. Das Point Tuple wird nicht modifiziert. Somit generiert der Transponder bei der Fahrt auf einer Strasse mehrere Point Tuples, welchen der Server jeweils die Streckenkosten zuweist. Die Gesamtkosten der gefahrenen Strecke setzen sich aus mehreren kleineren Beträgen zusammen. Bleibt ein Fahrzeug stehen, generiert der Transponder folglich auf derselben Stelle mehrere Point Tuples mit derselben Location. Die Kosten erhöhen sich dabei solange das Fahrzeug stehen bleibt.

Pseudorandom Funktionen

Die vom Client generierten Tags werden während der Registrationsphase verschlüsselt. Im VPriv Paper wird dafür die Verwendung einer Pseudorandom Funktion angegeben. Nach Rücksprache mit den Autoren von VPriv, kann dafür eine AES Verschlüsselung verwendet werden. Es ist allerdings darauf zu achten, welcher Blockmodus mit AES verwendet wird, damit die verschlüsselten Werte die Eigenschaften einer Pseudorandom Funktion (PRP) aufweisen. Die verschlüsselten Tags müssen so verschlüsselt werden, dass sie wie Zufallswerte aussehen.

Werden die Tags mittels AES verschlüsselt, kann man sie auch mit dem entsprechenden Schlüssel wieder entschlüsseln. Dies ist aber in VPriv nicht gewollt, da sonst Informationen preisgegeben werden. Aus diesem Grund fiel der Entscheid auf eine Einwegfunktion anstelle von AES. Der eingesetzte Algorithmus ist *HMAC SHA 256 Bit*.

Damit keine Informationen zwischen den Runden offen gelegt werden, muss pro Runde ein anderer Schlüssel verwendet werden. Bei der Registration werden die Tags mit jedem Rundenschlüssel verschlüsselt. In einer Runde wird dem Server der jeweilige Rundenschlüssel offen gelegt, womit der Server die Tags verschlüsselt und den gleichen Wert erhält, welcher der Client bei der Registration verschlüsselt hat.

5 Konzept

Die Applikation wird durch den Simulationsanwender über ein einfaches und technisches User Interface bedient.

Die folgende Liste stellt einen vereinfachten Ablauf der Simulationsanwendung dar. Auf bestimmte Punkte wird in den nachfolgenden Kapiteln detaillierter eingegangen.

1. Konfigurationseinstellungen werden durch den Anwender gesetzt.

2. Der Simulationsdurchgang wird gestartet.
   a. Fahrzeuge generieren Point Tuples während der Fahrt auf dem Strassennetz.
   b. Alle generierten Point Tuples werden an den Server übermittelt.

3. Start von VPriv (Reconciliation Phase).
   a. Der Simulationsanwender wählt ein bestimmtes Fahrzeug aus.
   b. Die totalen Kosten für das ausgewählte Fahrzeug werden berechnet und angezeigt.
   c. Die Korrektheit der berechneten Kosten wird angezeigt.
   d. Überprüfung auf Betrug mittels Spot Checks.
   e. Der Schutz der Privatsphäre wird aufgezeigt.

4. Die Applikation detektiert einen möglichen Betrug.

5.1 Applikationsübersicht

Die Simulationsanwendung ist in zwei Teile geteilt. In der Verkehrssimulation generieren die Fahrzeuge Point Tuples, indem sie über das Strassennetz fahren. Diese Daten werden im zweiten Applikationsteil verwendet, um das Konzept von VPriv anzuwenden und aufzuzeigen.

VPriv Teilapplikation [2]

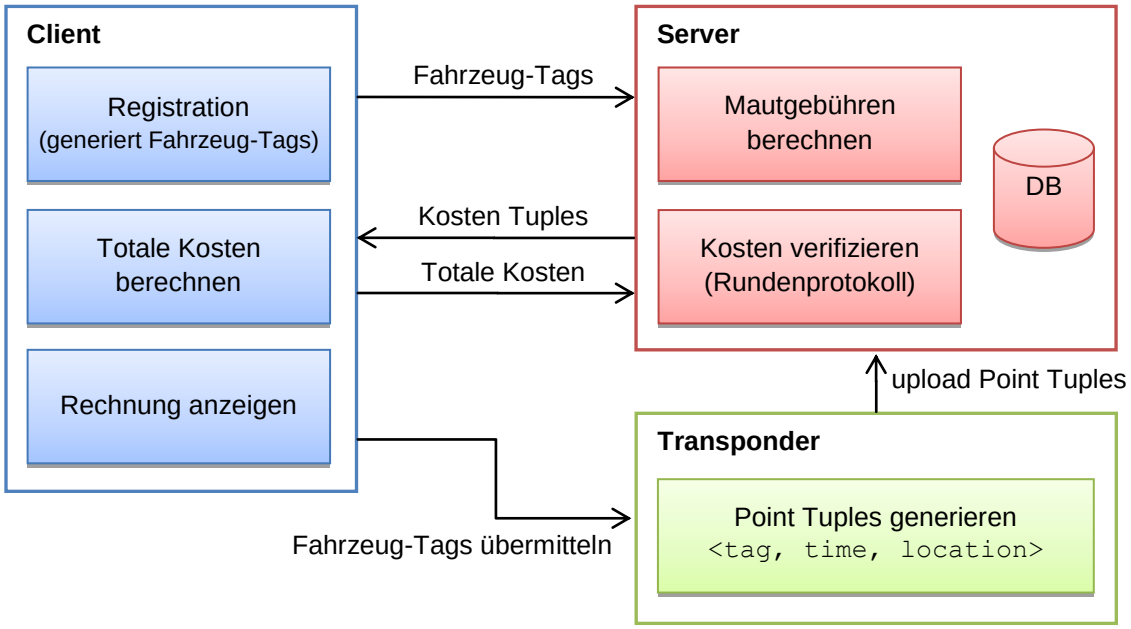


Abbildung 2: VPriv Teilapplikation

Verkehrssimulation Teilapplikation

Mehrere Fahrzeuge bewegen sich auf dem Strassennetz. Während der Fahrt generiert der Transponder Point Tuples, die am Ende der Simulation auf den Server übertragen werden. Jedes Fahrzeug besitzt nur einen Transponder.

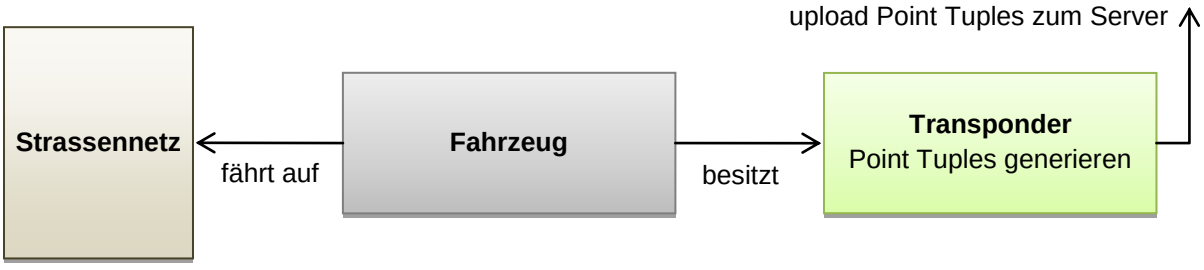


Abbildung 3: Verkehrssimulation Teilapplikation

[2] Auszug aus der Projektarbeit 2, Konzept Simulation

5.2 Strassennetz

Eine Strasse ist eine Verbindung zwischen zwei Kreuzungen und ist in mehrere Teilabschnitte (Abkürzung: *T*) aufgeteilt. Die Länge einer Strasse wird durch die Anzahl Teilabschnitte definiert. Die Teilabschnitte besitzen keine Eigenschaften, können aber zum Beispiel um die Kapazität als Eigenschaft erweitert werden. In dieser Simulation wird die Strassenkapazität nicht berücksichtigt. Das heisst die Fahrzeuge wechseln ihre Position ohne die anderen Fahrzeuge zu berücksichtigen oder auf Kapazitäten zu achten.

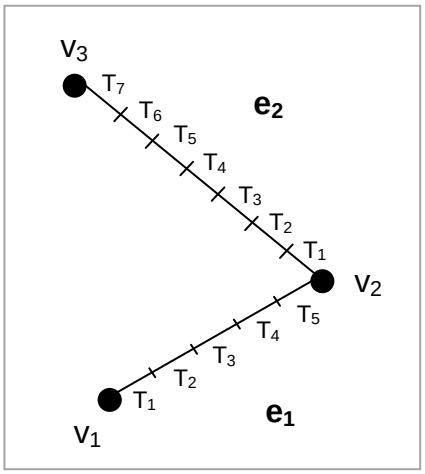


Abbildung 4: Strassennetz als Graph

Das Strassennetz wird als Graph dargestellt. Eine Kreuzung wird im Graph als Knote (engl. Vertex, Abkürzung: *v*) und eine Strasse als Kante (engl. Edge, Abkürzung: *e*) abgebildet. Das Gewicht der Kante ist nicht nur wie üblicherweise eine Zahl, sondern eine Referenz auf ein Objekt mit mehreren Eigenschaften. Die Eigenschaften einer Kante sind der Strassentyp, die Kosten sowie die Teilabschnitte in welche die Kante unterteilt ist.

$$e_i = \begin{cases} \text{Strassentyp (für die Wegfindung)} \\ \text{Kosten (für die Kostenfunktion)} \\ \text{Strassenabschnitte } T_i \end{cases}$$

Der Graph wird im Programm durch eine Adjazenzmatrix repräsentiert. Die Länge und Breite der Matrix ist gleich der Anzahl Knoten im Graphen. Gibt es eine Verbindung zwischen den Knoten v_i und v_j, so ist der Inhalt in der Matrix *A[i][j]* die dazugehörige Kante. Gibt es keine Verbindung ist das Matrixelement NULL.

Die Adjazenzmatrix zum Beispielgraph oben sieht wie folgt aus:

$$A = \begin{array}{c} \begin{matrix} v_1 & v_2 & v_3 \end{matrix} \\ \begin{pmatrix} - & e_1 & - \\ e_1 & - & e_2 \\ - & e_2 & - \end{pmatrix} \end{array} \begin{matrix} v_1 \\ v_2 \\ v_3 \end{matrix}$$

Positionsbestimmung

Die Knoten besitzen keine Koordinaten. Eine Position wird eindeutig über die Kante und den Teilabschnitt definiert. Beispiel: Ein Fahrzeug befindet sich auf der Strasse e_1 im 3. Teilabschnitt. Die Position ist gegeben durch $<e_1, T_3>$. Bewegt sich das Fahrzeug entlang der Strasse fort, so ist die Position im nächsten Simulationsschritt $<e_1, T_4>$ (genauer im Abschnitt Simulationsdurchgang). Dieses vereinfachte Strecken- und Positionsbestimmungssystem reicht für diese Anwendung aus. Die Verkehrssimulation dient dazu, Daten für die anschliessende Verarbeitung mit VPriv zu generieren. Dadurch ist es dem Simulationsanwender möglich die Funktionsweise von VPriv aufzuzeigen.

5.3 Simulationseinstellungen

Vor dem Start des Simulationsdurchganges kann der Simulationsanwender folgende Parameter anpassen:

- Anzahl Fahrzeuge: Die Anzahl Fahrzeuge, die sich auf dem Strassennetz bewegen.
- Anzahl Simulationsschritte: siehe 5.4 Simulationsdurchgang

Für jedes Fahrzeug generiert die Simulationsapplikation automatisch zufällige Tags welche dem Transponder sowie verschlüsselt dem Server übermittelt werden.

Der Simulationsanwender sieht in einer Textausgabe am unteren Ende des GUI, welche Schritte die Applikation im Moment durchführt.

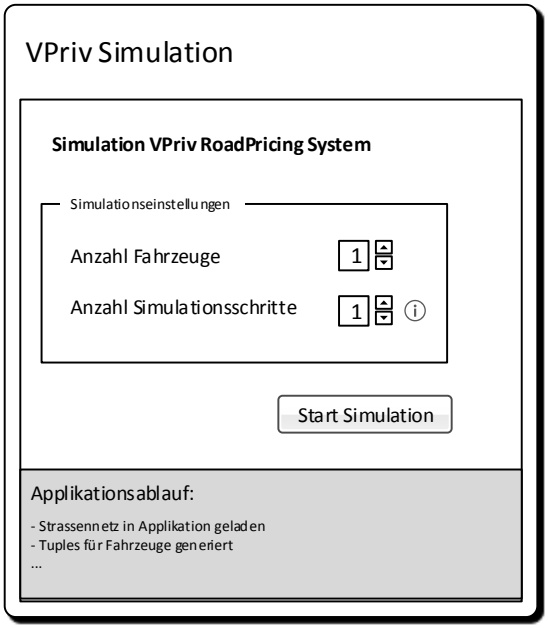


Abbildung 5: Entwurf GUI, Simulationseinstellungen

5.4 Simulationsdurchgang

Während des Simulationsdurchganges bewegen sich die Fahrzeuge auf dem Strassennetz und die Transponder generieren Point Tuples. Diese Point Tuples werden am Ende des Simulationsdurchganges dem Server übermittelt. Der Simulationsdurchgang ist für den Simulationsanwender nicht ersichtlich.

Simulationsschritt

Die Simulation besteht aus einer Anzahl von Simulationsschritten, welche der Simulationsanwender vor dem Start der Simulation festlegt. Eine bestimmte Anzahl von Simulationsschritten entspricht der Verrechnungsperiode.

Während eines Simulationsschrittes werden folgende Aktionen durchgeführt:

- Das Fahrzeug bewegt sich um einen Teilabschnitt weiter. Je nach Situation wechselt das Fahrzeug nicht nur einen Teilabschnitt, sondern auch die Strasse. Aufgrund der Erkenntnisse zur Kostenfunktion aus Abschnitt 4.3, ist es in der Simulation nicht nötig, dass ein Fahrzeug bei einem Simulationsschritt zufällig stehen bleibt und damit zwei Point Tuples mit derselben Location generiert.

- Der Transponder generiert ein „location“ Element mit der aktuellen Position des Fahrzeuges. Es wird ein Point Tuple nach folgendem Schema generiert. `<tag, time, location>`.

Mit einer Durchschnittsgeschwindigkeit von 50 km/h ergeben sich 15 Stunden Fahrzeit pro Monat und Fahrzeug. Wird alle 20 Sekunden eine Location aufgezeichnet ergibt dies total 2‘700 aufgezeichneten Locations. Da pro Simulationsschritt eine Location aufgezeichnet wird, entspricht eine Verrechnungsperiode von 30 Tagen, 2‘700 Simulationsschritten. Um genügend Tags für ein Jahr zu besitzen, muss ein Transponder mindestens über 32‘400 Tags verfügen.

5.5 VPriv Reconciliation Phase

Nach dem Simulationsdurchgang wählt der Simulationsanwender ein beliebiges Fahrzeug aus. Für dieses Fahrzeug berechnet die Applikation die totalen Mautgebühren. Die Simulationsapplikation zeigt die Korrektheit sowie den Schutz der standortbezogenen Privatsphäre auf, indem die Daten vom Client und die Daten vom Server gegenüber gestellt werden. Der Client kennt den Weg, welchen er zurückgelegt hat und sieht welche Kosten der Server den jeweiligen Strecken zugewiesen hat. Auf der Seite des Servers sind alle Daten vorhanden, welche keinen Schluss auf die Wegprofile einzelner Fahrzeuge zulassen.

Die während des Rundenprotokolls durchgeführten Schritte werden auf der Textausgabe am unteren Ende des GUI angezeigt.

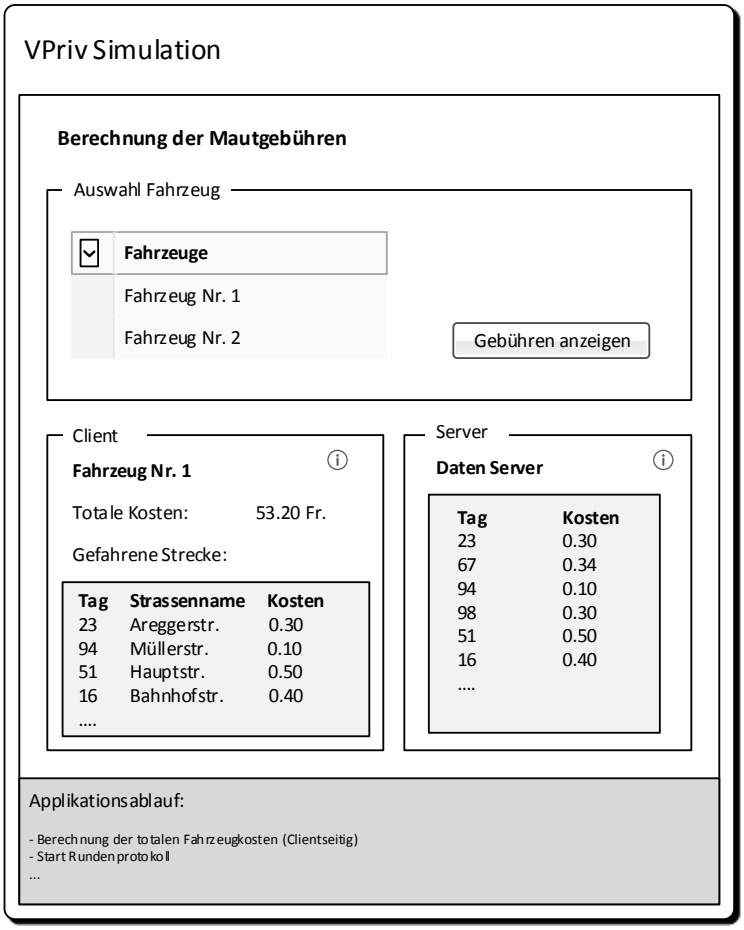


Abbildung 6: Entwurf GUI, Berechnung der Mautgebühr

5.6 Spot Check

Während der Driving Phase gibt es verschiedene Möglichkeiten zu betrügen. Um zu erkennen, ob der Fahrzeughalter den Transponder ausgeschaltet hat oder falsche Point Tuples dem Server übermittelt (Zeit/Location modifizieren), werden am Strassenrand zufällig Spot Checks platziert, welche `<license plate, time, location>` aufzeichnen.

In der Simulation kann ein Spot Check einem Teilabschnitt einer Strasse zugewiesen werden. Alle Fahrzeuge, die diesen Teilabschnitt passieren, werden aufgezeichnet. Diese Daten werden auf den Server übertragen.

Nach der Durchführung des Rundenprotokolls wird geprüft, ob das Fahrzeug an einem Spot Check vorbeigefahren ist. Der Server sendet dem Client dazu alle Zeiten, zu welchen das Fahrzeug den Spot Check passiert hat. Der Client übermittelt dem Server als Beweis das richtige Point Tuple. Anschliessend wird überprüft, ob der Client zu der aufgezeichneten Zeit am rechten Ort war. Zusätzlich wird kontrolliert, ob der Client ein Tag verwendet hat, welches bei der Registration festgelegt wurde. Wenn der Beweis fehlschlägt, gibt der Server eine entsprechende Meldung aus.

Nur der Server kennt die Standorte der Spot Checks. Der Client weiss bloss, wo er zu welchem Zeitpunkt durchgefahren ist. Er kennt den korrekten Ort nur dann, wenn er auch wirklich einen Spot Check passiert hat. Falls der Fahrzeughalter betrügen möchte, so müsste er den entsprechenden Ort kennen. Da sich die Spot Checks fortwährend an anderen Orten befinden, ist dies nahezu unmöglich.

Das Tag im Point Tuple wird dem Server verschlüsselt und committed übertragen, da der Server das Tag des Clients nur in dieser Form kennt. Nur so kann überprüft werden, ob der Client ein Tag verwendet hat, welches er bereits bei der Registration auf den Server übertragen hat.

In der Simulation gibt es nur eindeutige Positions- und Zeitangaben. Wird ein Point Tuple auf einem Teilabschnitt einer Strasse generiert, gibt es für diesen Teilabschnitt eine fest definierte Positionsangabe. Die Zeitangabe ist in der Simulation an den Simulationsschritt gebunden. Toleranzbereiche für den Vergleich der Ort- und Zeitangeben fallen in der Simulation weg.

6 Realisation

Im Rahmen der Bachelorarbeit wurde eine Verkehrssimulation mit integriertem Road-Pricing System programmiert. Die Simulation dient zur detaillierten Demonstration des VPriv Verfahrens. Der Schutz der Privatsphäre des Fahrzeughalters wird in der Simulation aufgezeigt.

6.1 Übersicht

Die Abbildung zeigt eine Gesamtübersicht der Objekte und deren Assoziationen. In den folgenden Kapiteln werden die Simulation, das Strassennetz sowie Client und Server detaillierter spezifiziert.

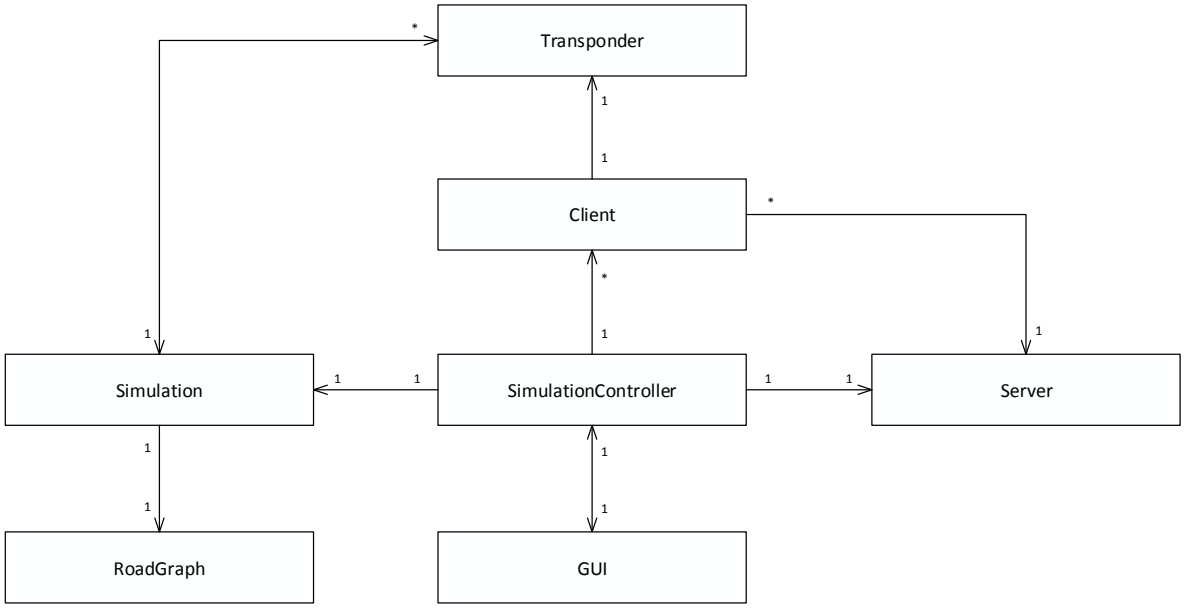


Abbildung 7: Gesamtübersicht der Applikation

Grundsätzliches Programmdesign

Um das GUI von den Programmdaten und der Programmlogik zu trennen, wurde ein Controller eingefügt. Dieser nimmt die Eingaben aus dem GUI entgegen und führt die Aktionen an der Simulation aus. Aktionen aus dem GUI sind der Start der Verkehrssimulation oder der Start von VPriv mit einem ausgewählten Client. Der Controller wird aus der ausführbaren Main Methode des Programms erzeugt. Der Controller beinhaltet wiederum das GUI und macht dieses sichtbar. Die resultierenden Daten aus der Simulation werden vom Controller abgerufen und am GUI angezeigt.

Die Verkehrssimulation und die Ausführung von VPriv nehmen einige Zeit in Anspruch. Daher können die Operationen nicht im Swing-Thread ausgeführt werden, da ansonsten das GUI einfriert und auf keine Benutzereingaben mehr reagiert. Als Lösung dieses Problems werden die zeit- und rechenintensiven Operationen der Simulation als separater Thread ausgeführt. Der Swing-Thread bleibt somit frei für das GUI und behält die Fähigkeit auf Benutzereingaben zu reagieren und Ausgaben anzuzeigen. Änderungen am GUI aus den Threads heraus, beispielsweise beim Hinzufügen einer Zeile im Output, werden von Java selbst erkannt und die Komponente wird mit den aktualisierten Inhalten dargestellt.

6.2 Strassennetz und Wegfindung

Das Strassennetz wurde wie im Konzept beschrieben umgesetzt.

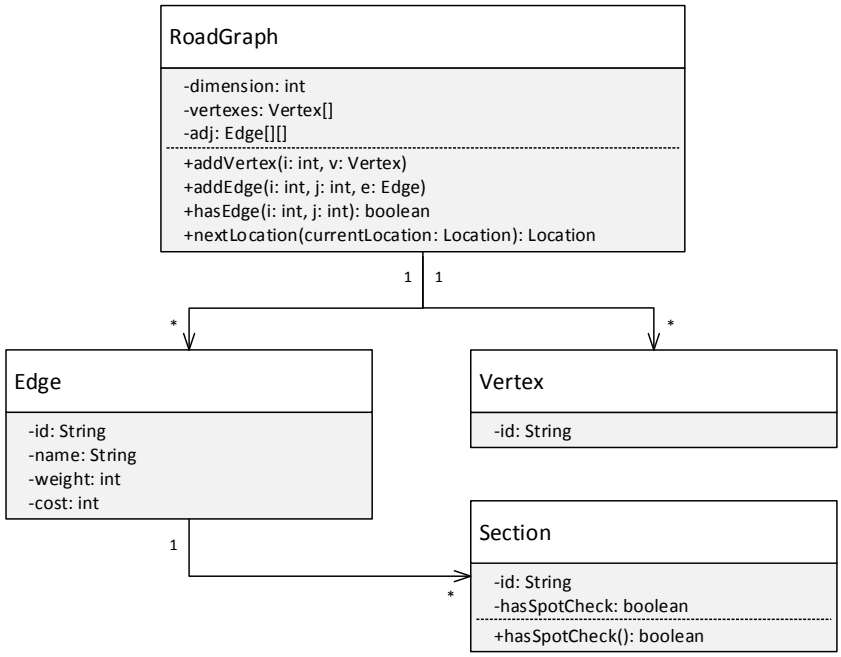


Abbildung 8: Klassendiagramm Strassennetz

Zur Positionsbestimmung wird eine Location erstellt, welche aus der Kante (Edge) und dem Teilabschnitt (Section) besteht. Damit ist die Position im Strassennetz eindeutig definiert. Um die Richtung eines Fahrzeuges zu bestimmen, wird der letzte besuchte Vertex gespeichert.

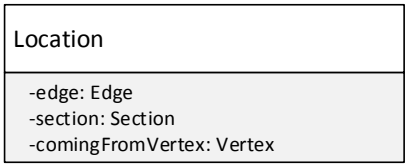


Abbildung 9: Location Objekt

Zur Identifikation von Edges, Sections und Vertexes werden die Java-Objektreferenzen verwendet und mittels der equals()-Methode verglichen.
Eine Location *l1* ist gleich einer Location *l2*, wenn *l1.Edge=l2.Edge* und *l1.Section=l2.Section*

Alle Elemente erhalten eine ID als String. Da die Elemente über die Java-Objektreferenz verglichen werden, dienen diese aber vor allem zur verständlichen Benutzerausgabe. Ein Edge erhält typischerweise die ID e_i, ein Vertex v_i und eine Section $t_i\_s_i$.

Die Membervariable *hasSpotCheck* definiert ob ein Teilabschnitt einen Spot Check beinhaltet.

Die Methode *nextLocation* liefert den nächsten Teilabschnitt der Kante, ausgehend von der aktuellen Location. Besitzt die Kante keinen weiteren Teilabschnitt wird ein Knoten (Kreuzung) erreicht. Die Methode liefert dann zufällig eine nächste Kante, welche vom Knoten ausgeht. Das Gewicht der Kante beeinflusst dabei die Wahrscheinlichkeit, welche Kante gewählt wird.

6.3 Simulation

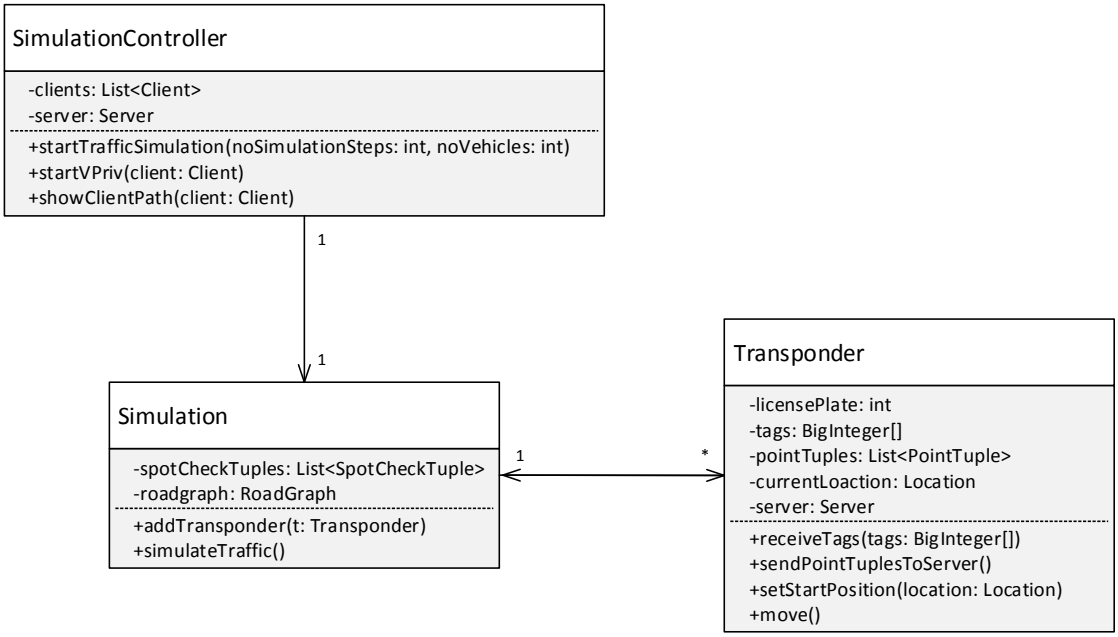


Abbildung 10: Klassendiagramm Simulation

Die Simulationsanwendung nimmt zwei Parameter entgegen: die zu simulierende Anzahl Fahrzeuge sowie die Anzahl Simulationsschritte. Die Parameter werden vom Simulationsbenutzer über das GUI definiert.

Der Controller erzeugt vor dem Start der Simulation einen Server. Für jedes zu simulierende Fahrzeug wird ein Client erzeugt und ein Transponder zugewiesen. Dieser Transponder repräsentiert das eigentliche Fahrzeug in der Simulation. Die Tags erhält der Transponder vom Client.

Die Simulation beinhaltet das Strassennetz sowie eine Liste mit den zu simulierenden Transpondern. Jeder Transponder wird der Simulation bei der Initialisierung hinzugefügt.

Die Simulation wird durch den Controller mittels *simulateTraffic* ausgelöst. Die Simulationslogik bewegt die Transponder über das Strassennetz. Die Transponder generieren bei jedem Simulationsschritt ein Point Tuple mit der aktuellen Location. Der Simulationsschritt wurde wie im Konzept beschreiben umgesetzt. Zusätzlich wird für jedes Fahrzeug, welches einen Spot Check passiert, ein Tuple erzeugt und am Ende der Simulation dem Server übermittelt. Damit erhält der Server alle Spot Check Daten, die er zur Überprüfung benötigt.

Der Transponder benötigt die Referenz zum Server, damit er am Ende des Simulationsdurchganges die erzeugten Point Tuples zum Server senden kann. Die Referenz zur Simulation benötigt der Transponder, damit er auf das Strassennetz zugreifen und die Position bestimmten kann. In der Realität würde der Transponder GPS Positionsdaten ermitteln, welche unabhängig vom Strassennetz sind.

VPriv wird für einen ausgewählten Client nach der Verkehrssimulation gestartet. Dazu werden die erzeugten Point Tuples des Transponders verwendet.

6.4 Statusausgaben

Während der Laufzeit von VPriv werden Ausgaben an der Konsole des GUIs angezeigt. Damit der Client einen Status ausgeben kann, wird das Observer Pattern angewendet. Der Simulationscontroller implementiert dabei das Interface OutputObserver und registriert sich beim Client (Subject) als Observer. Ändert der Client seinen Zustand (beispielsweise wenn eine Runde im VPriv Rundenprotokoll zu Ende ist) werden die Observer (in diesem Fall ist nur der Simulationscontroller registriert) mit der Statusausgabe informiert. Der Simulationscontroller implementiert eine Methode um die empfangene Benachrichtigung an der GUI-Konsole auszugeben.

Zur Formatierung der Ausgabe wird ein Enum erstellt (OutputFormat). Die Ausgabe kann als Header, Error, VPriv oder Normal formatiert werden. Über eine Checkbox kann der Benutzer detaillierte VPriv Protokoll-Ausgaben einstellen. Ausgaben vom Typ VPRIV_DETAIL werden nur ausgegeben, wenn die Checkbox gesetzt ist. Ausgaben vom Typ VPRIV_SIMPLE nur wenn die Checkbox nicht gesetzt ist.

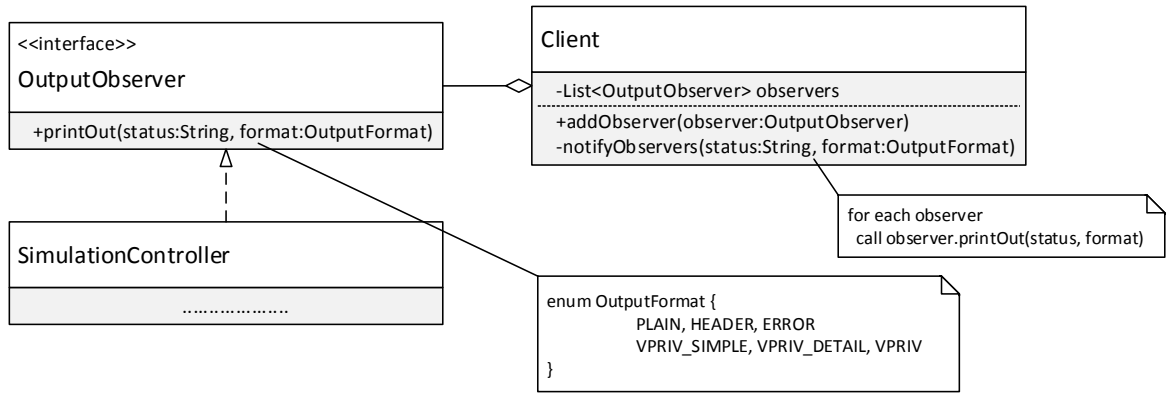


Abbildung 11: Klassendiagramm Statusausgabe

6.5 Point Tuple und Spot Check Tuple

Das Point Tuple wird während der Fahrt durch den Transponder generiert und besteht aus den Elementen `<tag, time, location>`.

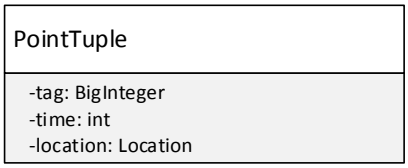


Abbildung 12: Point Tuple

Das Spot Checkt Tuple wird von einem Spot Check bei der Durchfahrt eines Fahrzeuges generiert und besteht aus den Elementen `<license plate, time, location>`.

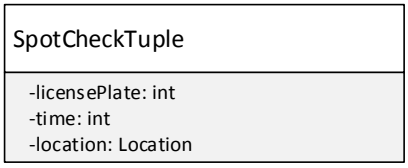


Abbildung 13: Spot Check Tuple

6.6 Client

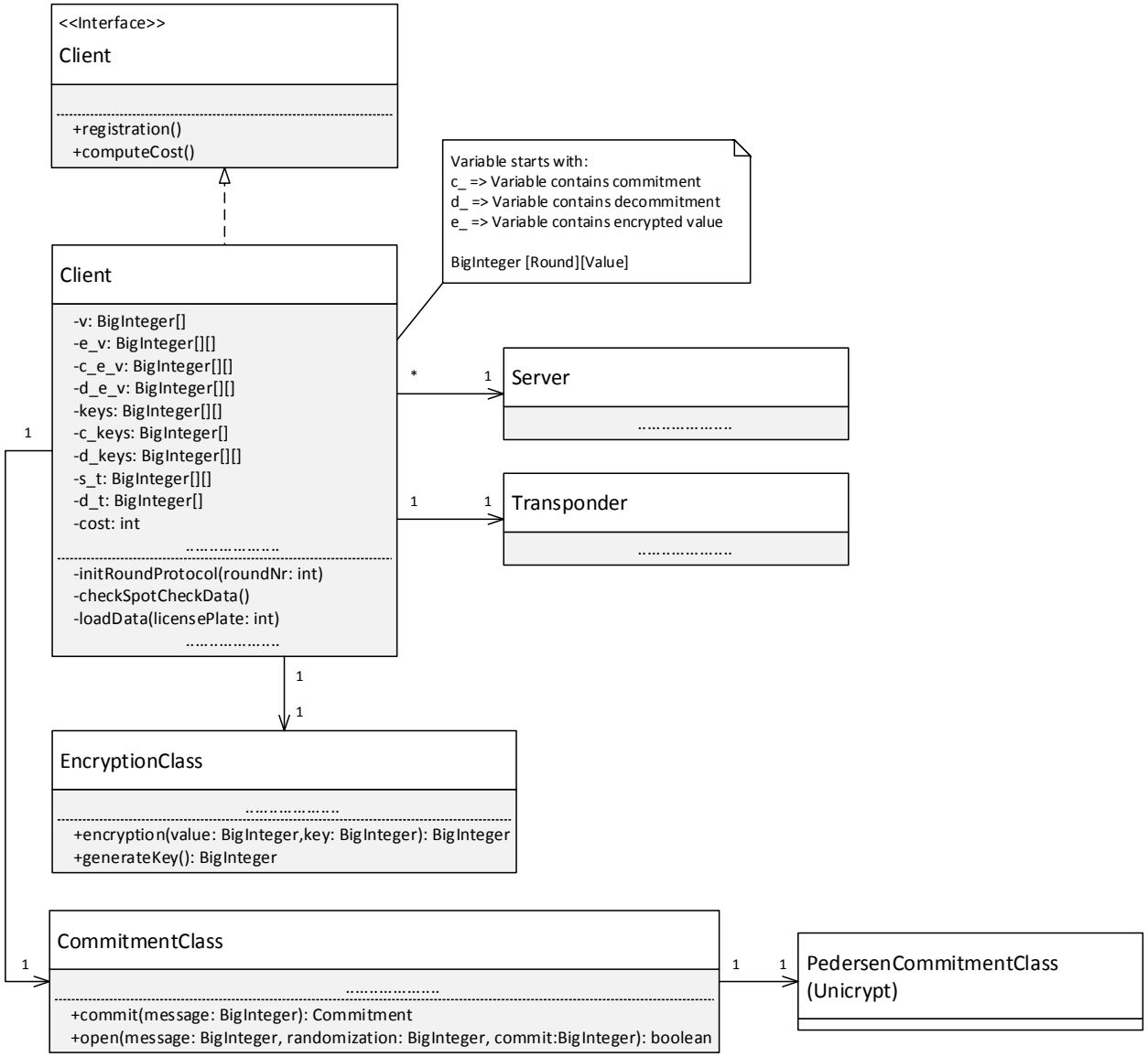


Abbildung 14: Klassendiagramm Client

Erklärungen des Klassendiagramms befinden sich auf der folgenden Seite. Die Variablen des Clients werden im Kapitel 6.8 Programmvariablen beschrieben.

Jedem Client kann nur ein Transponder zugewiesen werden. Die Simulation sieht nicht vor, dass pro Benutzer mehrere Fahrzeuge verwendet werden können. In einem produktiven Road-Pricing System müsste man den Client erweitern, um mehrere Fahrzeuge verwalten zu können. Für jedes Fahrzeug würden die entsprechenden Tags generiert und separat abgespeichert.

Der Client hat zwei Hauptaufgaben: das Generieren von Tags und die Berechnung der korrekten Mautgebühren. Die Details sind im VPriv Protokollablauf nachzulesen.

Die Methode *registration* entspricht der Registration Phase im VPriv Protokollablauf. Zufällige Tags und Schlüssel werden generiert. Danach werden die verschlüsselten Tags und Schlüssel dem Server gesendet. Zusätzlich werden die generierten Tags auf den Transponder übertragen.

Die Methode *computeCost* entspricht der Reconciliation Phase. Der Client berechnet die Mautkosten für die gefahrene Strecke. Mittels des Rundenprotokolls beweist der Client dem Server, dass er die totalen Mautgebühren richtig berechnet hat. Zusätzlich beweist der Client dem Server, dass er bei allfälligen Spot Checks durchgefahren ist.

Kryptographie

Das VPriv Protokoll erfordert zwei kryptographische Funktionen. Eine Commitment- sowie eine Verschlüsselungsfunktion. Wobei es sich bei der Verschlüsselungsfunktion nicht um eine eigentliche Verschlüsselung handelt, sondern um einen Message Authentication Code (MAC).

Die Klasse *CommitmentClass* ermöglicht es ein Commitment zu erstellen sowie ein Commitment zu überprüfen. Als Schema wird das Pederson Commitment verwendet. Die Funktionalität wird aus der BFH Unicrypt Bibliothek importiert.

Die Domain Parameter können entweder automatisch zufällig generiert werden oder sie werden über den Konstruktor mitgegeben. Jeder Client besitzt unterschiedliche Domain Parameter.

Wird ein Commitment von einem Wert erstellt, wird das Resultat in einem Objekt vom Typ *Commitment* gespeichert. Dieses ist notwendig, weil zu jeder Commit Operation auch ein Decommitment (Randomization) generiert wird und abgespeichert werden muss.

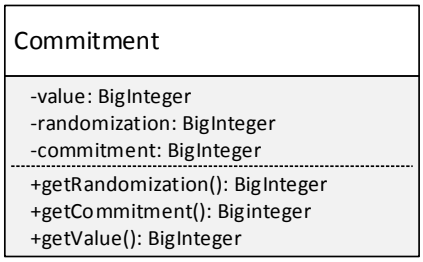


Abbildung 15: Commitment Objekt

Die Klasse *EncryptionClass* generiert aus einem Wert und einem geheimen Schlüssel einen Mac. Der verwendete Algorithmus ist HMAC-SHA 256. Als Bibliothek wurde die Java Security-API, mit dem Standard Cryptographic Service Provider, verwendet.

Die Methode *generateKey* erstellt zufällige 256 Bit lange Schlüssel.

6.7 Server

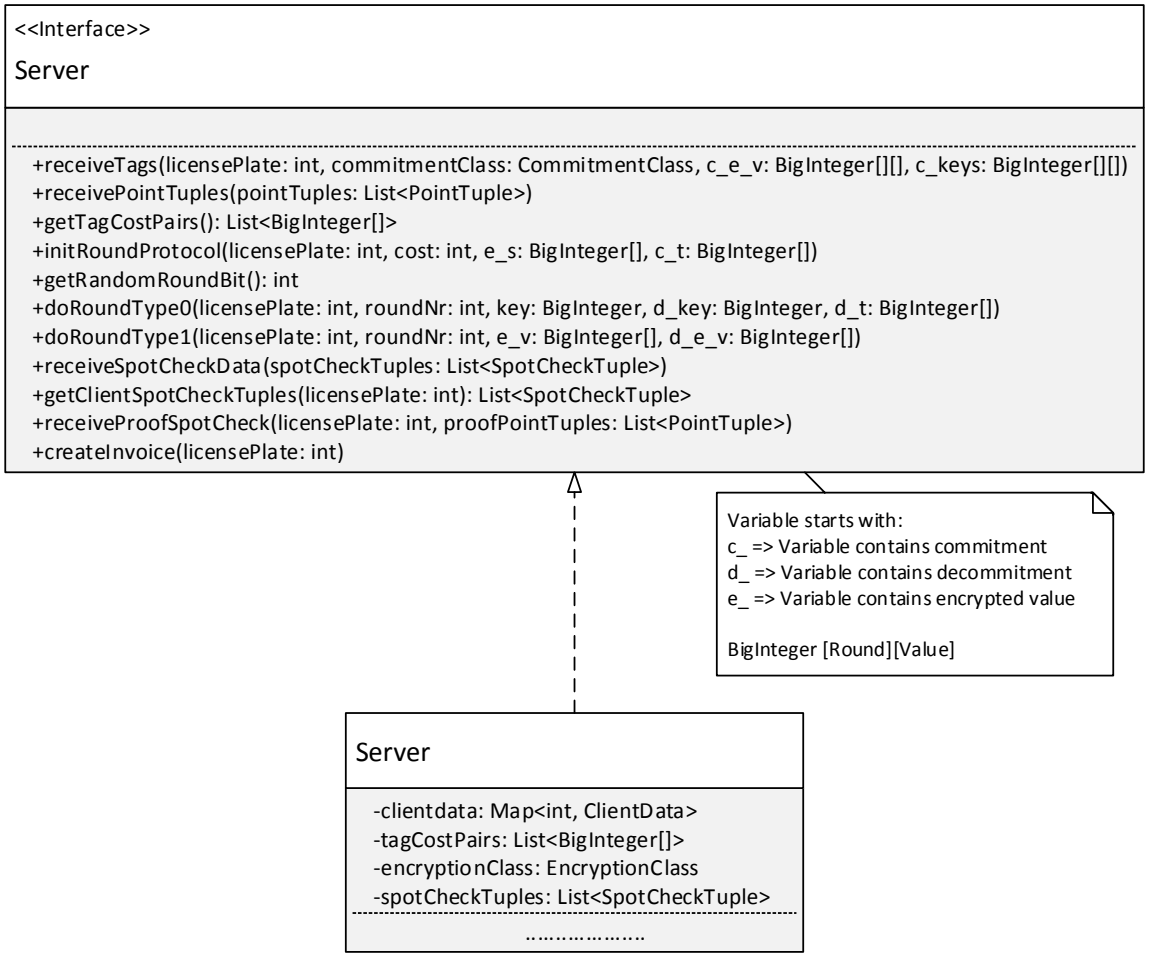


Abbildung 16: Klassendiagramm Server

Der Server prüft, ob die verschiedenen Clients die Mautgebühren richtig berechnet haben, damit nach einer Verrechnungsperiode eine korrekte Rechnung erstellt werden kann. Dazu sind diverse Schritte notwendig. Der Aufruf dieser Schritte erfolgt durch den Client. Bevor der Server die Rechnung ausstellt, prüft er, ob der Client alle notwendigen Schritte durchgeführt hat. Nur so wird eine Rechnung erstellt (Methode *createInvoice*).

Ruft der Client eine Methode auf dem Server auf, identifiziert er sich mittels des Nummernschildes.

Die Variablen des Servers werden im Kapitel 6.8 Programmvariablen beschrieben.

6.7.1 Methoden in Verbindung zum VPriv Protokollablauf

Die folgende Tabelle zeigt auf, welche Methode in welcher Phase des Rundenprotokolls aufgerufen wird.

	Methode		Beschreibung
Registration	receiveTags		Der Sever erhält vom Client die erzeugten Tags und Rundenschlüssel. Zusätzlich sendet der Client dem Server ein Objekt mit den verwendeten Domain Parametern der Commitment Funktion. Die Daten werden pro Benutzer separat abgespeichert.
Driving	receivePointTuples		Die aufgezeichneten Point Tuples erhält der Server vom Transponder. Jeder Location ordnet der Server den korrekten Gebührenwert zu. Auf dem Server werden nur Point Tuples abgespeichert, für welche eine Mautgebühr bezahlt werden muss.
Reconciliation	getTagCostTuples		Gibt eine Liste der Tag-Cost Paare zurück.
	Round Protocol	initRoundProtocol	Initialisierung des Rundenprotokolls. Der Server speichert die empfangenen Daten vom Client ab.
		getRandomBit	Der Server wählt zufällig, ob der Client den Rundenfall b=0 oder b=1 ausführt. Die Entscheidung gibt er dem Client bekannt und speichert sie zusätzlich in den Benutzerdaten ab.
		doRoundType0	Rundenprotokoll Fall b=0.
		doRoundType1	Rundenprotokoll Fall b=1.
	Spot Check	receiveSpotCheckData	Der Server erhält vom Spot Check eine Liste mit allen Spot Check Tuples.
		getClientSpotCheckTuples	Zu einem gegebenen Nummernschild gibt der Server alle zugehörigen Spot Check Tuples zurück.
		receiveProofSpotCheck	Der Server empfängt vom Client eine Liste mit Point Tuples die beweisen, dass der Client einen Spot Check passiert hat. Der Server überprüft, ob der Client wirklich durchgefahren ist.

Tabelle 4: Übersicht der Server Methoden

Da alle Methoden auf dem Server durch den Client aufgerufen werden, muss der Server prüfen, ob der Client nicht betrügt. Theoretisch könnte der Client selber entscheiden, ob er den Rundenfall 0 oder 1 aufruft. Weiter könnte er Runden einfach überspringen. Aus diesem Grund wird vor jedem Rundendurchgang geprüft, ob der Client den richtigen Fall aufgerufen hat. In den Benutzerdaten ist zudem hinterlegt, welche Runde der Client als letztes erfolgreich durchgeführt hat. So ist ein Überspringen von Runden unmöglich.

6.7.2 Speicherung von Benutzerdaten auf dem Server

Der VPriv Server verarbeitet Daten von mehreren Fahrzeugen. Auf dem Server müssen die empfangenen Daten der verschiedenen Clients gespeichert werden. Dazu wird das *ClientData* Objekt verwendet.

Die Abspeicherung der ClientData Objekte erfolgt in einer Map <Nummernschild, ClientData>. Das Nummernschild dient als eindeutige Identifikation des Fahrzeuges. Dies ermöglicht dem Server einen schnellen Zugriff auf die Daten.

Im *ClientData* Objekt werden die vom Client empfangenen Tags und Schlüssel gespeichert. Zudem wird das *CommitmentClass* Objekt des entsprechenden Benutzers hinterlegt. Dies ist notwendig, da jeder Client unterschiedliche Domain Parameter verwendet. Nur so hat der Server die Möglichkeit ein vom Client empfangenes Commitment zu öffnen.

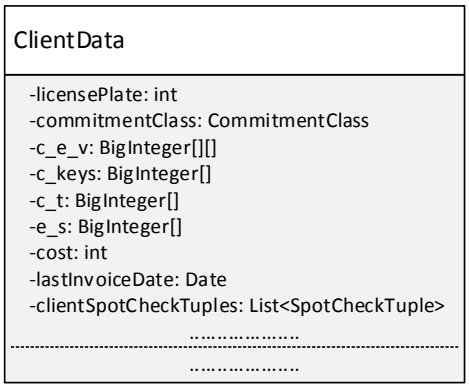


Abbildung 17: Klassendiagramm ClientData

6.8 Programmvariablen

Die VPriv Applikation generiert eine grosse Menge von Daten. Diese Information müssen in diversen Variablen gespeichert werden. Im nachfolgenden Kapitel werden die wichtigsten Variablen und ihr Aufbau beschrieben.

Für die generierten Tags *v* und die Schlüssel *k* wird der Datentyp BigInteger verwendet. Dieser Datentyp ermöglicht beliebig grosse Zahlen. Dies wird vor allem bei den kryptographischen Funktionen wie der Commitment- oder der Verschlüsselungsfunktion wichtig.

Allgemeine Namenskonvention für Variablen

Eine Variabel beginnt mit:
c_ => Variabel enthält ein Commitment
d_ => Variabel enthält ein Decommitment
e_ => Variabel enthält einen verschlüsselten Wert

Client

Die nachfolgende Tabelle gibt eine Übersicht über die wichtigsten Variablen in der Client Klasse.

Name	Datentyp	Beschreibung
v	BigInteger[]	Enthält die zufällig generierten **Fahrzeug-Tags** v_i. Array: [#Tag]
e_v	BigInteger[][]	Enthält die **verschlüsselten Tags** v_i. Die Tags werden pro Runde mit dem Rundenschlüssel verschlüsselt $f_k(v_i)$. Mehrdimensionales Array: [#Runde][#Tag]
c_e_v	BigInteger[][]	Enthält das **Commitment der verschlüsselten Fahrzeug Tags** $c(f_k(v_i))$. Mehrdimensionales Array: [#Runde][#Tag]
d_e_v	BigInteger[][]	Enthält das **Decommitment der verschlüsselten Fahrzeug Tags** $d(f_k(v_i))$. Mehrdimensionales Array: [#Runde][#Tag]
keys	BigInteger[]	Enthält zufällig generierte **Rundenschlüssel** k_r. Array: [#Runde]
c_keys	BigInteger[]	Enthalt das **Commitment der Rundenschlüssel** $c(k_i)$. Array: [#Runde]
d_keys	BigInteger[]	Enthalt das **Decommitment der Rundenschlüssel** $d(k_i)$. Array: [#Runde]

s_t	List<BigInteger[]>	Enthält eine Liste mit den eigenen **Tuples** (s_j, t_j). Es wird eine Liste verwendet, weil zu Beginn nicht klar ist, wie viele Tuples, aus der vom Server empfangenen Menge, dem Client gehören. Array: [0]= s_j; [1]= t_j
d_t	BigInteger[]	Enthält das **Decommitment des Kostentags** $d(t_j)$ Array: [#Kostentag]
e_s	BigInteger[]	Enthält die **verschlüsselten vom Server empfangenen Tags** $f_k(s_j)$. Array: [#Servertag]
c_t	BigInteger[]	Enthält das **Commitment des Kostentags** $c(t_j)$ Array: [#Kostentag]

Tabelle 5: Übersicht über die Variablen des Clients

Server

Die Tabelle gibt eine Übersicht über die wichtigsten Variablen in der Server Klasse.

Name	Datentyp	Beschreibung
clientdata	Map<Integer, ClientData>	Enthält alle vom **Client empfangenen Daten**. Diese Daten sind in einem Objekt vom Typ ClientData gespeichert. Für die Speicherung der Daten wird eine Map verwendet. Die Map ermöglicht einen schnellen Zugriff auf die Benutzerdaten über das Nummernschild des Fahrzeuges. Map: <Nummernschild, Client Daten>
tagCostTuples	List<BigInteger[]>	Enthält alle **von den Transpondern erhaltenen Tuples** (s_j, t_j). Es wird eine Liste verwendet, weil die Transponder periodisch Daten dem Server senden. Die genaue Anzahl von Tuples ist bei der Initialisierung noch nicht bekannt. Array: [0]= s_j; [1]= t_j
spotCheckTuples	List<SpotCheckTuple>	Enthält eine Liste mit **Spot Check Tuples** <Nummernschild, Zeit, Ort> Eine Liste wird verwendet, da periodisch neue Spotcheckdaten hinzugefügt werden.

Tabelle 6: Übersicht über die Variablen des Servers

6.9 Konfigurationsdatei

Die Einstellungen für die Simulationsanwendung sind in einer Konfigurationsdatei (Java-Properties-Datei) abgelegt. Dies ermöglicht es dem Benutzer einfache Anpassungen am Programm vorzunehmen, ohne den Code anpassen zu müssen.

Parameter	**Beschreibung**
weight_StreetCat1=60 weight_StreetCat2=30 weight_StreetCat3=10	Gewichte der verschieden Strassentypen. Die Zahl gibt die Wahrscheinlichkeit an, zu welcher eine Strasse befahren wird. Beispiel: Strassenkategorie 1: 60% Die Summe der Gewichte muss 100 sein.
vehicleMinimum=1 vehicleMaximum=20 vehicleDefault=15	Spinner Einstellungen für die Anzahl Fahrzeuge in der Simulation.
simulationStepsMinimum=30 simulationStepsMaximum=1000 simulationStepsDefault=150	Spinner Einstellungen für die Anzahl Simulationsschritte.
tagLength=32	Länge der Tags in Bit.
numberOfTags=2000	Anzahl Tags, die pro Client generiert werden.
numberOfRounds=10	Anzahl Runden im Rundenprotokoll.
generationTags=false	Gibt an, ob die Tags während der Registration generiert oder aus einer Datei gelesen werden.
tagsFile=generatedClientData/tags_	Speicherort und Dateiname der abgelegten Client Tags. Aus diesen Daten werden die Tags gelesen, wenn generationTags auf false gesetzt ist. Aufbau Dateiname: tags_[Nummernschild].txt
macAlgorithm=HMACSHA256	Verwendeter MAC Algorithmus. Der Algorithmus muss durch die Java Security API unterstützt werden.
keyLength=256	Länge des MAC Schlüssels in Bit.
savePrimLength=258	Länge der Primzahl in Bit, welche für die Commitment Funktion automatisch generiert wird.

Tabelle 7: Beschreibung der Parameter in der Konfigurationsdatei

6.10 Geschwindigkeitsprobleme

Die Berechnung der vielen kryptographischen Funktionen benötigt viel Zeit. Besonders die Commitmentfunktion ist sehr rechenintensiv. Dies haben entsprechende Tests gezeigt.

6.10.1 Registrationsphase

Während der Client Registrationsphase wird für jede Runde im Rundenprotokoll ein Schlüssel generiert. Jedes generierte Tag *v* wird für jede Runde mit dem entsprechenden Schlüssel verschlüsselt und zusätzlich wird ein Commitment davon erstellt.

Beispiel

Anzahl generierter Tags *v*:	10‘000
Anzahl Runden:	10
Anzahl Schlüssel:	10

Total generierter verschlüsselter/committeter Tags: 10*10‘000 = 100‘000 Tags.
Dies entspricht je 100‘000 Commitment- und HMAC-Operationen. Die durchschnittliche Dauer für die Generierung ist 120 Sekunden.

Eine Dauer von ca. 2 Minuten ist für einen einzelnen Benutzer tragbar. In der Simulation sollen aber mehrere Fahrzeuge simuliert werden. Für jedes Fahrzeug müssen die entsprechenden Tags generiert werden. Die Dauer summiert sich mit der Anzahl Fahrzeuge. Aus diesem Grund werden die Tags im Vorfeld der Simulation generiert und in einer Datei abgelegt. Um Zeit zu sparen, werden während der Simulation die Tags aus der Datei ausgelesen.

Die für die Simulationsanwendung verwendeten Dateien werden mittels eines eigenen Programms generiert. Dazu wird ein Client Objekt verwendet und die verschiedenen Tags werden automatisch erzeugt. Die Client Klasse besitzt eine öffentliche Methode, welche es erlaubt, die generierten Daten in eine Textdatei abzuspeichern. Mit den in der Simulation eingesetzten Parametern besitzt eine solche Datei eine Grösse von ca. 4.5 MB. Über die Konfigurationsdatei wird angegeben, ob der Client die Tags während der Laufzeit generiert oder ob er sie aus einer Datei einliest.

6.10.2 Rundenprotokoll

Während des Rundenprotokolls werden viele Commitment- und Vergleichsoperationen durchgeführt. Besonders der Fall b=0 ist sehr zeitaufwändig.

Fall b = 0

Im Fall b=0 beeinflusst die Anzahl Fahrzeuge und insbesondere die Anzahl vom Server empfangener Tuples *(s_j, t_j.)*, die Geschwindigkeit. Es handelt sich bei den Tuples nur um die eigenen Tuples, die einen Kostenwert grösser 0 besitzen.

Beispiel

Anzahl Tuples *(s_j, t_j)* pro Fahrzeug: 1‘000
Anzahl Fahrzeuge: 100

Dies entspricht 100*1‘000*1‘000 = 100‘000‘000 Vergleichsoperationen (If-Anweisung) sowie 1‘000 Commitmentoperationen. Die Zeit wächst linear zur Anzahl der verwendeten Tuples und der Anzahl Fahrzeuge, die in der Simulation simuliert werden.

Um für den Benutzer eine angemessene Ausführungszeit der Simulationsanwendung zu gewährleisten, musste die Anzahl der Tags und Fahrzeuge reduziert werden. 15 Fahrzeuge und 150 Simulationsschritte ergeben eine annehmbare Ausführungszeit.

Fall b = 1

Im Fall b=1 beeinflusst die Anzahl der generierten Tags *v* die Geschwindigkeit. Werden zum Beispiel 10‘000 Tags verwendet, entspricht dies 20‘000 Commitmentoperationen.

Zeitmessung für eine Runde b=1 in der VPriv Simulation mit unterschiedlicher Anzahl Tags:

Anzahl Tags v	Zeit [ms]	Zeit pro Tag [ms]
10	16	1.60
20	31	1.55
50	63	1.26
100	125	1.25
200	266	1.33
500	671	1.34
1‘000	1‘391	1.39
2‘000	2‘719	1.36
5‘000	6‘875	1.38
10‘000	15‘142	1.51
Mittelwert		**1.40**

Tabelle 8: Zeitmessung für eine Runde b=1
Computer: Windows 8, Core i7, 4GB RAM, Java 7 32 Bit

Die Zeit wächst linear zur Anzahl der verwendeten Tags. Die Zeitmessung hat gezeigt, dass 2'000 Tags mit einer Rundendauer von ca. 2.7 Sekunden eine angemessene Anzahl Tags ist. Aus diesem Grund werden pro Fahrzeug in der Registrationsphase 2‘000 Tags generiert.

6.10.3 Verwendete Parameter in der Simulationsanwendung

Folgende Parameter gewährleisten eine angemessene Ausführungszeit der Simulationsanwendung. Diese Werte entsprechen nicht den in der Konzeptphase vorgeschlagenen Parametern. Aber nur durch das Anpassen dieser Werte kann eine für den Simulationsbenutzer angenehme Ausführungsdauer der Simulation gewährleistet werden.

Anzahl Runden: 10 [3]
Anzahl generierte Tags *v*: 2‘000

Folgende Parameter sind Standardwerte, sie können aber vor dem Start der Simulationsanwendung angepasst werden:

Anzahl Fahrzeuge: 15 (max. 20)
Anzahl Simulationsschritte: 150

[3] *Dies entspricht einem Betrugsnachweis von 99.9%. Pro Runde kann der Server mit einer Wahrscheinlichkeit von ½ feststellen, ob der Client die Kosten korrekt berechnet hat. Nach s Runden ist die Wahrscheinlichkeit mindestens 1-(1/2)^{s}, dass der Server den Betrug erkennt.*
VPriv: Protecting Privacy in Location-Based Vehicular Services, Seite 8

6.11 Tests

Für die korrekte Funktion der Simulationsanwendung ist ein Zusammenspiel verschiedener Klassen notwendig. Die meisten Methoden verlangen für die Ausführung Daten, welche in anderen Klassen und Methoden teils zufällig generiert werden. Aus diesem Grund sind die Tests komplex aufgebaut und involvieren verschiedene Klassen und Methoden.

6.11.1 Client und Server

Bei der Registrierung erzeugt der Client die Tags und Schlüssel zufällig. Bei einem Test müssen aber bekannte Daten verwendet werden. Aus diesem Grund wurde auf die implementierte Funktion zurückgegriffen, welche es erlaubt aus einer Textdatei vorgenerierte Tags zu laden. Ein spezieller Test-Client mit genau zwei Tags und einem Rundenschlüssel wurde generiert. Alle Client- und Servertests basieren auf diesen Testdaten.

Client Klasse

Beim Clienttest wird nicht geprüft, ob der Client die Tags korrekt erstellt, verschlüsselt und committet hat. Die Verschlüsselungs- sowie die Commitment-Funktion wird in eigenen Tests geprüft. Aus diesem Grund ist eine Überprüfung der Clientdaten nicht mehr notwendig.

Getestet wird, ob es möglich ist die Mautkosten zu berechnen, ohne vorher die verschiedene Tags generiert zu haben.

Server Klasse

Der Servertest prüft die verschiedenen Funktionen auf Korrektheit. Zusätzlich wird überprüft, ob der Server einen Betrug durch den Client richtig detektiert.

Folgende Betrugsmöglichkeiten werden in Tests abgedeckt:

- Der Client sendet dem Server ein falsches Total der Mautkosten (Rundenfall b=1).
- Der Client sendet einen falschen Rundenschlüssel (Rundenfall b=0).
- Der Client sendet ein verschlüsseltes Tag s, welches nicht ihm gehört oder welches nicht auf dem Server existiert (Rundenfall b=0 und b=1).
- Der Client sendet ein verschlüsseltes Tag v, welches nicht ihm gehört (Rundenfall b=1).
- Der Client sendet falsche Kosten t (Rundenfall b=0).

In einem separaten Test wir überprüft, ob der Ablauf der Funktionsaufrufe durch den Client mit dem VPriv Protokollablauf übereinstimmt.

6.11.2 Verkehrssimulation

Die Verkehrssimulation kann nur als Gesamtes getestet werden. Das bedeutet, zu einer Simulation werden einige Transponder hinzugefügt und diese über eine gewisse Anzahl an Simulationsschritten simuliert. Läuft die Simulation ohne Fehler ab, gilt der Test als korrekt. Es wird auch getestet, ob die Transponder während der Fahrt bei jedem Simulationsschritt ein Point Tuple erzeugt haben. Die Location jedes Point Tuple kann dabei aber nicht getestet werden, da die Verkehrssimulation alle Simulationsschritte in einem Durchgang in der Methode *simulateTraffic* durcharbeitet und die Simulation es nicht ermöglicht in die Daten eines einzelnen Simulationsschrittes Einsicht zu erhalten.

Das Bewegen eines einzelnen Fahrzeugs innerhalb eines Simulationsschrittes wird im Test des Strassennetzes geprüft (siehe Abschnitt unten).

Strassennetz und Wegfindung

Das Strassennetz wird in der Applikation als Graph repräsentiert. Getestet werden das Erstellen eines Graphen und das Hinzufügen von Knoten und Kanten sowie das Abrufen von Informationen aus dem Graphen. Auf Tests von Knoten und Kanten an sich wurde verzichtet, da die Objekte keine Logik sondern nur Membervariablen beinhalten.

Die Wegfindung konnte nur bedingt getestet werden, da der Zufall entscheidet, welche Kante als nächstes befahren wird. Es wird getestet, ob die korrekte nächste Location ausgehend von der aktuellen Position gefunden wird, also ob der nächste Teilabschnitt befahren wird. Gibt es keinen nächsten Teilabschnitt, kommt das Fahrzeug an eine Kreuzung. Somit wird getestet, ob der Algorithmus eine nächste Kante findet, die befahren werden kann.

7 Konklusion

Der Vorteil des VPriv Verfahrens gegenüber bisher eingesetzten Mautsystemen ist, dass die Positionsdaten der Fahrzeuge anonym an den Road-Pricing Server übermittelt werden. Aus den zentral anfallenden Daten können ausschliesslich Kosten berechnet werden. Das System ist nicht in der Lage, Wegstrecken einer bestimmten Person zuzuordnen. Somit können keine Bewegungsprofile einzelner Verkehrsteilnehmer an Dritte weitergegeben werden, was Missbrauch ausschliesst. Die standortbezogene Privatsphäre ist jederzeit gewährleistet.

Um das Road-Pricing System nutzen zu können, muss der Fahrzeughalter auf seinem persönlichen Computer, mittels der entsprechenden Software, Tags generieren und auf den Transponder im Fahrzeug übertragen. Zusätzlich muss er jeweils am Ende einer Verrechnungsperiode die Clientapplikation ausführen, um die totalen Mautgebühren zu berechnen und die Korrektheit dieser gegenüber dem Server zu gewährleisten. Durch die Benutzerinteraktionen, die einige Zeit in Anspruch nehmen, ist das System wenig benutzerfreundlich und praxistauglich und es ist daher mit einer geringeren Akzeptanz der Nutzer zu rechnen.

Zahlreiche kryptographische Operation führen dazu, dass der interaktive Prozess zwischen Client und Server, zur Überprüfung der Mautkosten, äusserst rechen- und zeitintensiv ist. Aufgrund der Interaktivität muss der Client auf das Ende der Berechnung des Servers warten. Die Performance des Servers ist daher aus Sicht der Bedienbarkeit und Benutzerfreundlichkeit für den Fahrzeughalter von entscheidender Bedeutung.

Die Zeitdauer zur Überprüfung der Mautkosten ist abhängig von der Anzahl registrierter Fahrzeuge und gebührenpflichtiger Strassen. Diese Faktoren vergrössern die Datenmenge, welche der Server verarbeiten muss und folglich auch die Zahl der rechenintensiven kryptographischen Operationen. Dies besagt, dass ein effizientes Road-Pricing System nach dem VPriv Verfahren nur möglich ist, wenn die Anzahl Mautstrassen und diejenige der registrierten Fahrzeuge gering ist.

Die Simulationsanwendung erlaubt die Einstellung der performanceabhängigen Parameter wie der Anzahl Fahrzeuge und der Anzahl Simulationsschritte. Damit zeigt die Anwendung die Einschränkungen und Grenzen des VPriv Verfahrens auf.

Das VPriv Verfahren ist eine mögliche Alternative zu den heute im Einsatz befindlichen Road-Pricing Systemen, hat jedoch den Nachteil, dass der gewonnene Schutz der standortbezogenen Privatsphäre ein Reduktion der Benutzerfreundlichkeit zur Folge hat.

Im Rahmen der Bachelor Thesis wurde für das VPriv Verfahren eine Simulationsanwendung programmiert. Diese umfasst eine Verkehrssimulation mit integriertem Road-Pricing System. Die Umsetzung erforderte eine intensive Auseinandersetzung mit dem VPriv Protokoll. Die Ziele, die detaillierte Demonstration des VPriv Verfahrens und das Aufzeigen des Schutzes der Privatsphäre des Fahrzeughalters, konnten erreicht werden.

Im Verlauf der Arbeit entstanden nur geringe zeitliche Abweichungen zum anfänglich erstellten Projektplan. Zu Beginn wurde mehr Zeit als geplant beansprucht damit Unklarheiten im VPriv Protokollablauf geklärt werden konnten. Die Gliederung der Arbeit in einzelne Teilaufgaben ermöglichte eine strukturierte und produktive Arbeitsweise. Durch die Aufteilung der Simulationsanwendung in die Teile VPriv und Verkehrssimulation konnte parallel und effizient programmiert werden.

Die Arbeit an der Bachelor Thesis bestätigte uns die Notwendigkeit des Schutzes der Privatsphäre in einem Road-Pricing System. Durch die Auseinandersetzung mit dem VPriv Verfahren gewannen wir die interessante Erkenntnis, dass es dank Kryptographie möglich ist, die standortbezogene Privatsphäre zu schützen. Wir erhoffen uns, dass dieses Wissen in zukünftige Debatten um ein Road-Pricing System in der Schweiz einfliessen wird.

Wir bedanken uns bei Reto König, Dozent an der Berner Fachhochschule und Mitarbeiter im Institut *Research Institute for Security in the Information Society* (RSIS), für die hilfreichen Informationen im Bereich der Kryptographie. Für die Unterstützung und Betreuung bedanken wir uns bei Prof. Dr. Eric Dubuis. Die konstruktiven Gespräche und Anregungen waren für unsere Arbeit sehr wertvoll. Ein weiterer Dank gilt Stefan Berner unserem Experten.

Glossar

Road-Pricing
: Das Erheben einer Gebühr für die Benützung von Strassen.

VPriv
: Protecting Privacy in Location-Based Vehicular Services. Verfahren um die standortbezogene Privatsphäre in einem Road-Pricing System zu schützen.

Client
: Clientsoftware von VPriv, die auf dem persönlichen Computer des Fahrzeughalters ausgeführt wird.

Server
: Serversoftware von VPriv, welche die Kosten überprüft.

Transponder
: Im Fahrzeug eingebaut. Zeichnet während der Fahrt Positionsdaten auf und sendet sie anonym zum Server.

Simulationsanwendung
: Beinhaltet die Verkehrssimulation inkl. der Umsetzung von VPriv.

Rundenprotokoll
: Der Client beweist dem Server die Korrektheit der Mautkostenberechnung.

Standortbezogene Privatsphäre
: Standortinformationen können nicht gezielt einer Person zugeordnet werden. Das Road-Pricing System ist nicht in der Lage, aufgezeichnete Wegstecken einem bestimmten Fahrzeug/Person zuzuordnen.

Kostenfunktion
: Der Server weist den aufgezeichneten Positionsdaten die entsprechenden Mautgebühren zu.

Zero-Knowledge-Beweis
: Der Client beweist dem Server die gefahrene Wegstrecke, ohne dabei die Wegstrecke explizit offen zu legen.

Tag
: Zufallswert

Point Tuple
: Das Point Tuple wird während der Fahrt durch den Transponder generiert und besteht aus den Elementen <Tag, Zeit, Location>.

Spot Check Tuple
: Das Spot Check Tuple wird durch den Spot Check generiert und besteht aus den Elementen <Kontrollschild, Zeit, Location>.

MAC
: Message Authentication Code. Die MAC Funktion bildet aus einem Zahlenwert und einem geheimen Schlüssel eine Prüfsumme. Es handelt sich um eine Einwegfunktion.

Cryptographic Service Provider
: Der Provider implementiert kryptografischen Algorithmen. Zugriff auf den Provider erfolgt über die Java Security-API. Neue Provider können in Java integriert werden.

Swing
: Eine Programmierschnittstelle (API) und Grafikbibliothek zum Programmieren von grafischen Benutzeroberflächen in Java.

Literaturverzeichnis

Michael Gautschi, Daniel Brönnimann, Bericht Projekt 2 Road-Pricing System mit VPriv, Berner Fachhochschule, 19.01.2013

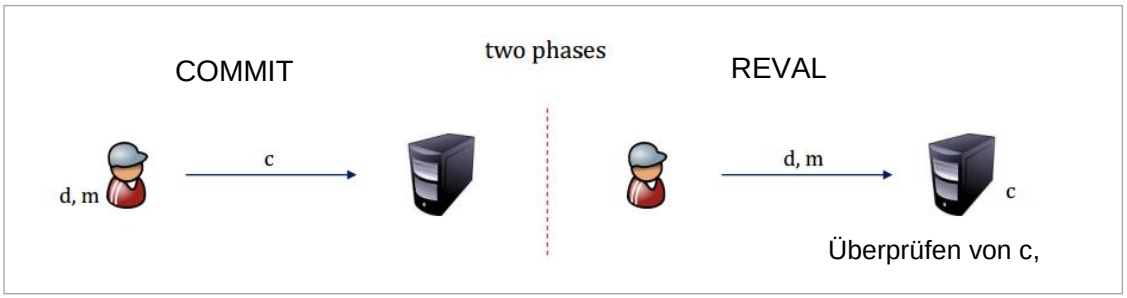
P Popa, Raluca Ada, Hari Balakrishnan and Andrew J. Blumberg, VPriv: Protecting Privacy in Location-Based Vehicular Services, USENIX Association, 2009

Lexikon: Begriff Road-Pricing, http://www.vimentis.ch/d/lexikon/454/Road-Pricing.html, Stand 17. Mai 2013

Anhang

Commitment Verfahren [4]

Ein Commitment-Verfahren ermöglicht es einer Partei, sich gegenüber der anderen Partei auf einen Wert festzulegen, ohne etwas über diesen Wert zu verraten. Später kann dieser Wert dann aufgedeckt werden.



Das Commitment Verfahren besteht aus zwei Algorithmen. COMMIT: das Commitment wird festgelegt und REVEAL: das Commitment wird offen gelegt, in dem man eine zusätzliche Information veröffentlicht. Bei einem Commitment darf es nicht möglich sein, einen Rückschluss auf den ursprünglichen Wert *m* zu machen. Nach dem COMMIT kann der Wert *m* nicht mehr geändert werden.

[4] Auszug aus der Projektarbet 2

Pederson Commitment

c = Commitment, d = Reveal, m = Nachricht

Das Pederson Commitment besitzt die homomorphe Eigenschaft:
*c(m) * c(m') = c(m+m')* und *d(m+m')=d(m)+d(m')*

SETUP

G_q ist eine Untergruppe von Z_p^* mit der Ordnung q.

G_q ist eine zyklische Gruppe.

- Wähle grosse Primzahl p
- Wähle grosse Primzahl q (q muss p *-1* dividieren)
- Wähle die zwei Generatoren g und h aus der Gruppe G_q (g und h dürfen nicht 1 sein)

Hilfe um p und q zu generieren: $p = 2q + 1$

Prüfen ob ein Element a ∈ Z_p^* in G_q ist:
$a \in G_q \leftrightarrow a^q = 1 \, mod \, p$

Domain Parameter (p, q, g, h) sind öffentlich

COMMIT (m, h)

- Wähle zufällig $r \in Z_q$
- Berechne $c = g^m h^r \, mod \, p$

$r \in \{0, .., q-1\}$

c = Commitment
m = Nachricht ($m \in Z_q$)

REVEAL (c, m, r)

- Überprüfen ob die vom Client enthaltene Nachricht *m* mit dem Commitment *c* übereinstimmt.
  $c' = g^m h^r \, mod \, p$
- Überprüfen ob $c' = c$

Sicherheit

Die Sicherheit ist gegeben, weil das DLOG Problem bei grossen Zahlen in der zyklischen Gruppe schwer zu lösen ist. Aus dem Commitment *c* lässt sich nicht die Nachricht *m* berechnen.

$h = g^x$ (g, h sind öffentlich)

$x = DLOG_g(h)$ Wenn DLOG einfach wäre, könnte man x berechnen.

Zahlenbeispiel

G_q ist eine Untergruppe (Zyklische Gruppe) von Z_p^*

SETUP

- Wahl der grossen Primzahlen p und q
  $p = 11, q = 5$
- Wahl der zwei Generatoren g und h
  $g = 3,\ h = 5$

q dividiert p-1: *(11-1)/5=2*

Prüfen ob die Elemente g, h ∈ Z_p^* in G_q sind:

$3^5 = 1\ mod\ 11$

$5^5 = 1\ mod\ 11$

COMMIT (m, h)

$m = 2$

- Bestimmen von r zufällig
  $r = 4$
- Berechne $c = g^m h^r\ mod\ p$
  $c = 4 = 3^2 5^4\ mod\ 11$

REVEAL (c, m, r)

Überprüfen, ob die vom Client enthaltene Nachricht m mit dem Commitment c übereinstimmt.

$c' = 4 = 3^2 5^4\ mod\ 11$
$c' = c = 4$

Erklärung der Diplomandinnen und Diplomanden

Rechte an der Bachelor Thesis

Ich bin damit einverstanden, dass die BFH-TI alle Güter inklusive der Immaterialgüter der durch die BFH-TI geleiteten Arbeiten grundsätzlich nutzen darf. Besondere Abmachungen zwischen Dozenten und beteiligten Unternehmungen und Institutionen bleiben vorbehalten.

Selbständige Arbeit

Ich bestätige mit meiner Unterschrift, dass ich meine Bachelor Thesis selbständig durchgeführt habe. Alle Informationsquellen (Fachliteratur, Besprechungen mit Fachleuten, usw.), die wesentlich zu meiner Arbeit beigetragen haben, sind in meinem Arbeitsbericht im Anhang vollständig aufgeführt.

Datum	12.06.2013
Name Vorname	Michael Gautschi
Unterschrift	.....
Name Vorname	Daniel Brönnimann
Unterschrift	.....