Studiengang:	Informatik
Prüfer:	Prof. Volker Diekert
Betreuer:	Dipl.-Inf. Holger Austinat
begonnen am:	14. Juni 2001
beendet am:	14. September 2001
CR-Klassifikation:	C.2.0, C.2.4, D.4.6, E.3, E.4, G.3, H.3.5, J.1, K.6.5

Studienarbeit Nr. 1821

Implementierung eines sicheren Systems zur Vorlesungsumfrage

Stefan Bellon Erik Thiele

Institut für Informatik
Universität Stuttgart
Breitwiesenstraße 20-22
D-70565 Stuttgart

Implementierung eines sicheren Systems zur Vorlesungsumfrage

SPuLe Handbuch

Stefan Bellon Erik Thiele

29. November 2001

Inhaltsverzeichnis

Teil I.

Technische Dokumentation

1. Einleitung

An Universitäten werden Vorlesungsumfragen durchgeführt. Eine Vorlesungsumfrage ist eine Sammlung vorlesungsspezifischer Fragen in Verbindung mit Fragen bezüglich des Vortragenden und generellen Fragen sowie zusätzlichen Vermerken. Bisher wurden diese Vorlesungsumfragen konventionell mit Fragebögen aus Papier durchgeführt. Hierbei traten einige Probleme auf, die mit dem neuen, hier vorgestellten SPULE System gelöst wurden. SPULE ist eine Abkürzung für Secure Practical Universal Lecture Evaluator. Der Name beschreibt kurz und knapp die Eigenschaften des Systems.

1.1. Forderungen an eine Wahl

Eine Wahl muss folgende 5 Kriterien erfüllen:

Wahlberechtigung
: Nur berechtigte Wähler können Stimmen abgeben.

Einmaligkeit
: Jeder Wähler kann nur einmal wählen.

Wahlgeheimnis
: Niemand kann feststellen, wie ein Anderer gestimmt hat.

Fälschungssicherheit
: Niemand kann unbemerkt die Stimme eines anderen ändern.

Verifizierbarkeit
: Jeder Wähler kann sich überzeugen, dass seine Stimme korrekt gezählt wurde.

1.2. Bisheriges papierorientiertes Verfahren

Bisher wurde zur Durchführung der Vorlesungsumfrage (hier exemplarisch am Beispiel der Universität Stuttgart dargestellt) ein auf Papier gedrucktes Formular in der letzten Veranstaltung ausgeteilt, welches dann von den Hörern meistens sofort ausgefüllt und in einen in jedem Saal speziell dafür aufgestellten Karton eingeworfen wurde. In neuerer Zeit wurde für dasselbe Verfahren ein Web-basiertes System erstellt, welches zusätzlich benutzt werden kann. Das gesamte Verfahren hat sowohl in seiner papierlichen als auch in der elektronischen Form gravierende Mängel bezüglich der Forderungen an eine Wahl:

Wahlberechtigung *Nur berechtigte Wähler können Stimmen abgeben.*
: Jeder kann ein Wahlformular ausfüllen oder das Web-basierte System bedienen, unabhängig davon, ob er die Vorlesung besucht hat, also wahlberechtigt ist oder nicht.

Einmaligkeit *Jeder Wähler kann nur einmal wählen.*
: Da man seine Identität beim Wählen nicht preisgeben muss, ist es möglich, beliebig oft zu wählen.

Wahlgeheimnis *Niemand kann feststellen, wie ein anderer gestimmt hat.*
: Diese Eigenschaft ist auch bisher bereits ziemlich gut gewährleistet. Man könnte durch Analyse der Fingerabdrücke auf dem Papier oder durch Abhören des Netzwerkverkehrs jedoch mit geringem Aufwand die Stimme eines Wählers ermitteln.

Fälschungssicherheit *Niemand kann unbemerkt die Stimme eines anderen ändern.*
: In der papierlichen Variante kann mit entsprechenden klassischen Mitteln das Wahlpapier manipuliert werden. Sehr einfach ist es zum Beispiel, ein zusätzliches Kreuz zu setzen. Und in der netzbasierten Variante kann ein Dritter mit entsprechendem Zugang zur Netzwerkhardware das unverschlüsselte Protokoll beliebig manipulieren.

Verifizierbarkeit *Jeder Wähler kann sich überzeugen, dass seine Stimme korrekt gezählt wurde.*
: Es gibt keine Möglichkeit für den Wähler, das tatsächliche Auszählen seiner papierlichen Stimme zu verifizieren. Ebenso verhält es sich bei der netzbasierten Variante. Die Verifikation beruht darauf, dass dem Transport des Papiers bzw. des Web-Formulars vertraut wird sowie den für das Auszählen verantwortlichen Personen.

Aufgrund dieser Unzulänglichkeiten sind Aussagen über die Wahlbeteiligung nicht möglich.

2. Aufgabenstellung

Ziel ist es, ein System zu erstellen und zu implementieren, mit Hilfe dessen die Vorlesungsumfrage auf elektronischem Wege durchgeführt werden kann. Bei der Lösung der gestellten Aufgabe gilt es, verschiedene Aspekte zu berücksichtigen. Die wichtigsten geforderten Punkte sind:

- Kein Student kann für eine Veranstaltung abstimmen, die er nicht selbst besucht hat.
- Kein Student kann mehrfach für dieselbe Veranstaltung abstimmen.
- Das Abstimmungsergebnis kann innerhalb des gesetzten Zeitraums jederzeit geändert werden.
- Keiner kann etwas über das Abstimmverhalten eines anderen herausfinden.

Zum einen sollen die oben genannten Anforderungen realisiert werden, zum anderen muss optional die Möglichkeit bestehen, den bisherigen Modus der Vorlesungsumfrage beizubehalten, das heißt dass wie bisher jeder zu einer Vorlesung abstimmen kann, unabhängig davon, ob er sie je gehört hat. Dadurch sind dann die Vorteile, die SPuLe bietet, nicht mehr nutzbar. Den Modus der Vorlesung soll der Dozent vor dem Beginn der Vorlesung festlegen können.

Im Folgenden wird das von den Autoren entworfene und implementierte SPuLe System vorgestellt, welches die oben genannten Kriterien erfüllt.

3. Datenbank

3.1. Tabellen

Zentrales Element des SPuLe Systems ist eine Datenbank, welche Informationen über Personen, Vorlesungen und Umfrageergebnissen speichert. Folgende Tabellen werden zur Datenerhebung benötigt:

3.1.1. Person

Person (ID, UniqueSecretPassword, ReallifeID, Name, Rank)

Alle Personen im System werden in der Tabelle *Person* gehalten. Im Einzelnen sind dies folgende Attribute:

ID — INTEGER IS PRIMARY KEY
: Dieses Primärattribut wird intern zur Referenzierung von Personen benötigt und ist aus Gründen höherer Performance des Datenbanksystems gewählt worden. An einigen Stellen ist es für einen *Organizer* oder *Admin* auch nötig, Kenntnis von der *ID* einer Person zu haben, um bestimmte Operationen auf diese Person durchführen zu können. Daher wird sie nach dem Einloggen im Hauptmenü eines Einzelnen angezeigt. Ein *Listener* allerdings muss diese *ID* nie benutzen.

UniqueSecretPassword — CHAR(14) IS UNIQUE INDEXED
: Diese eindeutige, zufallsgenerierte Kennung ermöglicht es jedem Benutzer, sich beim System zu authentifizieren. Das Format der Kennung wird in Kapitel 3.2 auf Seite 17 genauer erklärt. Über dieses Attribut wird ein Datenbank-Index erstellt, um eine schnellere Authentifizierung zu ermöglichen.

ReallifeID — VARCHAR(1024) IS UNIQUE
: Mit diesem Attribut wird jeder Benutzer *eindeutig* im System erfasst. Anhand dieser *ReallifeID* kann sich ein Benutzer beim *Trust* authentifizieren. Für Studenten bietet sich hier die Matrikelnummer, für Dozenten und Angestellte der Universität bieten sich Personalausweisnummer o. Ä. an. Man könnte sich sogar vorstellen, den genetischen Fingerabdruck zu speichern.

Name — VARCHAR(256)
: Unter diesem Namen ist die Person dem System bekannt. Hierbei kann es durchaus mehrere Personen mit dem gleichen Namen geben.

Rank — ENUM TYPE IS (Listener, Organizer, Admin)
: Dieses Attribut legt die Rechte der einzelnen Benutzer im System fest. Ein *Organizer* hat implizit die Rechte eines *Listeners* und ein *Admin* hat implizit die Rechte eines *Organizers*. Dieses Feld ist nicht redundant mit der Tabelle *LectureOrganizer*, wie man zuerst meinen möchte, da ein *Organizer* nicht zu jedem Zeitpunkt auch einer *Lecture* zugeordnet sein muss, dennoch aber den Status *Organizer* beibehält.

Eigentlich sollte jede Authentifizierung einer Person immer mit der *ReallifeID* erfolgen. Aus Praktikabilitätsgründen wird allerdings *UniqueSecretPassword* verwendet. Da beide Attribute *unique* sind, d. h. nur einmal vorkommende Werte enthalten dürfen, besteht somit eine Isomorphie zwischen den Attributen, was diese Vorgehensweise überhaupt erst ermöglicht.

3.1.2. Template

Template (ID, FatherID, LeftBrotherID, Type, MetaText, Person_ID)

Diese Tabelle enthält alle Fragen in hierarchisch angeordneter Form:

ID — INTEGER IS PRIMARY KEY
: Mit Hilfe dieser Identifikationsnummer, die das Primärattribut der Tabelle ist, werden die hierarchischen Fragenkataloge aufgebaut und den Fragen die Antworten zugeordnet.

FatherID — IS FOREIGN KEY
: Dieses Attribut referenziert die Identifikationsnummer des Vaterknotens. Referenzen können nur auf vorhandene Datensätze existieren. D. h. beim Löschen müssen alle Referenzen mitgelöscht werden, sonst ist das Löschen eines Datensatzes nicht möglich. Des Weiteren hat ein definierter „Null-Datensatz“ zu existieren, auf welchen als oberstes Element verwiesen werden kann.

LeftBrotherID — IS FOREIGN KEY
: Dieses Attribut referenziert den linken Bruderknoten. Mit diesen zwei Referenzen (*FatherID* und *LeftBrotherID*) kann eine Baum-Hierarchie und eine Reihenfolge innerhalb derselben Hierarchieebene modelliert werden. Wird der „Null-Datensatz“ referenziert, so gibt es keinen linken Bruder.

Type — ENUM TYPE IS (Node, Line, Label, Editable, Radio, Choice)
: Anhand dieses Attributs wird festgelegt, um welchen Knotentyp es sich handelt. Je nach Typ hat das Attribut *MetaText* eine andere Syntax und Semantik.

MetaText — VARCHAR(1024)
: In diesem Attribut wird der Text der Fragen gehalten. Das Format wird im Kapitel 3.2 auf Seite 17 detailliert behandelt.

Person_ID — IS FOREIGN KEY

Dieser Fremdschlüssel referenziert einen Benutzer aus der Tabelle *Person* und legt den Eigentümer der Frage fest. Auf diese Weise wird ein Rechtesystem realisiert, so dass niemand Fragenkataloge anderer, auf die er keinen Zugriff hat, modifizieren kann: Jeder *Organizer* kann nur eigene Fragen modifizieren oder Fragen anderer *Organizer* mit denen zusammen er die entsprechende *Lecture* hält. Auf diese Weise können *Admins* bestimmte Fragen vorgeben, die kein *Organizer* aus seiner *Lecture* entfernen kann.

3.1.3. Lecture

Lecture (Name, State, Template_ID)

In der Tabelle *Lecture* werden die einzelnen Vorlesungen verwaltet.

Name — VARCHAR(256) IS PRIMARY KEY

Der Name der *Lecture* muss eindeutig sein, da er in der Datenbank als Primärschlüssel verwendet wird. Dies bedeutet, dass es ratsam ist, das Jahr an den Vorlesungstitel anzuhängen (z. B. „Programmierkurs SS 2002“).

State — ENUM TYPE IS (initialized, running, terminated)

Mit diesem Attribut wird festgelegt, ob die *Lecture* gerade eingerichtet wird, bereits läuft oder sogar schon beendet ist.

Template_ID — IS FOREIGN KEY

Dieses Fremdschlüssel-Attribut referenziert den Wurzelknoten der hierarchisch modellierten Fragen der Vorlesung.

3.1.4. Event

Event (Lecture_Name, Pos, Name)

Lecture_Name — IS FOREIGN KEY

Anhand dieses Fremdschlüssel-Attributes wird der *Event* der *Lecture* zugeordnet.

Pos — INTEGER

Innerhalb der *Lecture* sind die einzelnen *Events* angeordnet. Dieses Attribut definiert die Ordnung der *Events* untereinander. Das Tupel (Lecture_Name, Pos) ist Primärschlüssel der Tabelle *Event*.

Name — VARCHAR(256)

Dieses Attribut kann ein Text sein, der das Thema des Vortrags für den *Event* beschreibt.

3.1.5. LectureOrganizer

LectureOrganizer (Person_ID, Lecture_Name)

Diese Tabelle beinhaltet Zuordnungen von Personen zu Vorlesungen. Auf diese Weise wird definiert, wer für welche Vorlesung verantwortlich ist. Personen, die für eine Vorlesung verantwortlich sind, dürfen Änderungen an der Konfiguration der Vorlesung vornehmen.

Person_ID — IS FOREIGN KEY
: Dieses Fremdschlüssel-Attribut referenziert eine Person, die für eine *Lecture* verantwortlich ist. Die Person muss in der Tabelle Person als *Organizer* oder als *Admin* geführt sein.

Lecture_Name — IS FOREIGN KEY
: Mit diesem Fremdschlüssel wird eine *Lecture* referenziert.

3.1.6. Legitimation

Legitimation (Code, Lecture_Name, Event_Pos, Number, isDeleted, Person_ID)

In dieser Tabelle werden die einzelnen Legitimationskennungen den Vorlesungen und – nach Benutzung – zusätzlich der entsprechenden Person zugeordnet.

Code — CHAR(7) IS PRIMARY KEY
: Dieser Primärschlüssel beinhaltet Zufallscodes, welche die Benutzung einer *Legitimation* ermöglichen. Das Format des Codes wird in Kapitel 3.2 auf der nächsten Seite erläutert.

Lecture_Name — IS FOREIGN KEY
: Mit diesem Fremdschlüssel-Attribut wird die Zugehörigkeit der *Legitimation* zu einer *Lecture* realisiert.

Event_Pos — IS FOREIGN KEY
: Dieser Fremdschlüssel referenziert einen einzelnen *Event* einer *Lecture*.

Number — INTEGER
: Da es pro *Event* viele *Legitimations* geben muss, wird eine Seriennummer benötigt, anhand der Verwaltungsoperationen an den *Legitimation*-Codes durchgeführt werden können. Pro Tupel (Lecture_Name, Event_Pos) fängt die Nummerierung wieder bei 1 an.

isDeleted — BOOLEAN
: Mit diesem Flag können *Legitimations* markiert werden, die zwar ausgedruckt, aber nicht ausgehändigt worden sind. Somit kann man eine exakte Wahlbeteiligung und eine Kontrolle darüber, ob die Wahl ordnungsgemäß abgelaufen ist, erhalten: Sollte mit einer als gelöscht markierten *Legitimation* abgestimmt worden sein, so hat ein *Organizer* entweder einen Fehler gemacht oder absichtlich die Statistik zu seinen

Gunsten beeinflussen wollen. Dies wird bei der Auswertung berücksichtigt und ein entsprechender Vermerk erscheint.

Person_ID — IS FOREIGN KEY
: Hier wird die Person, welche die *Legitimation* zum Abstimmen benutzt hat, gespeichert.

3.1.7. Answer

Answer (Template_ID, Person_ID, MetaAnswer)

In dieser Tabelle werden die Antworten der Vorlesungsumfragen gespeichert.

Template_ID — IS FOREIGN KEY
: Dieses Fremdschlüssel-Attribut referenziert die Frage, zu der die Antwort gehört. Es werden nur Antworten zu Fragetypen gespeichert, die tatsächlich auch Antworten beinhalten können.

Person_ID — IS FOREIGN KEY
: Dieser Fremdschlüssel gibt die Person an, von welcher die Antworten stammen. Das Tupel (Template_ID, Person_ID) ist Primärschlüssel der Tabelle *Answer*.

MetaAnswer — VARCHAR(1024)
: In diesem Attribut wird die Antwort der Person zur Frage gespeichert. Das Format ist fragetypspezifisch und wird in Kapitel 3.2 behandelt.

3.2. Meta-Formate

3.2.1. Format der Kennungen

Die Authentifizierung mit dem System sowie das Benutzen von Legitimierungen erfordert eine Benutzereingabe. Die Eingabe dieser Kennungen soll einerseits für Benutzer möglichst einfach erfolgen, andererseits muss ein geeignet großer Codewortraum vorliegen, um mit großer Sicherheit gewährleisten zu können, dass durch eine Fehleingabe kein anderes gültiges Codewort getroffen wird. Je größer allerdings die Kennungen, desto größer ist wiederum die Anfälligkeit auf Fehleingaben. Es gilt also den geeigneten Kompromiss für diesen Zweck zu finden.

Damit die Eingabe einfach erfolgen kann und sich keine Probleme bei verschiedenen Tastaturlayouts ergeben, werden nur die Zeichen A – Z und die Ziffern 0 – 9 verwendet, wobei Groß- und Kleinschreibung irrelevant ist. Somit ergibt sich ein Zeichenvorrat von 36 verschiedenen Zeichen (bei Ausdrucken werden immer Großbuchstaben gedruckt, das große O wird jedoch als kleines o gedruckt, um es von der 0 unterscheiden zu können).

Da die Eingabe einer langen Zeichenkette aus diesem Zeichenvorrat für den Menschen dennoch eine hohe Konzentration erfordert und mit zunehmender Länge die Fehlerhäufigkeit steigt, wird der Code in Gruppen gleicher Größe unterteilt, welche mit dem Bindestrich voneinander getrennt werden.

Es ergeben sich also folgende Alternativen:

Gruppengröße	Beispiel	Kommentar
1	A-B-C-3-5	zu umständlich zu tippen
2	AA-BD-84-UV	noch nicht effizient genug
3	AUX-342-Z72	in Erwägung zu ziehen
4	AV84-8823-BDIR	in Erwägung zu ziehen
5	AUSB3-23472-BBUEW	Gruppen wieder unübersichtlich
6	BU3N2G-3HQ64Q	kaum noch durchführbar

Tabelle 3.1.: Vergleich möglicher Gruppengrößen bei Kennungen

Aus Tabelle 3.1 wird deutlich, dass in der Praxis nur Gruppengrößen von 3 oder 4 Buchstaben in Folge verwendet werden können. Da man das *UniqueSecretPassword* seltener als *Legitimations* eingeben muss, wird für das *UniqueSecretPassword* eine Gruppengröße 4 verwendet, für *Legitimations*, die häufiger eingegeben werden müssen, wird die Gruppengröße 3 verwendet.

Nun muss der benötigte Codewortraum berücksichtigt werden, um die Anzahl der Gruppen in der Kennung zu bestimmen.

Anzahl	Beispiel	Codewortraum
1	AU4J	$36^4 \approx 10^{6.23}$
2	JEIT-37JR	$36^8 \approx 10^{12.45}$
3	IUER-2347-CBEJ	$36^{12} \approx 10^{18.68}$
4	AB5S-SH31-NB2S-H1AS	$36^{16} \approx 10^{24.90}$

Tabelle 3.2.: Codewortraum bei Gruppengröße 4

Bei der Auswahl der Gruppenanzahl für das *UniqueSecretPassword* ist entscheidend, dass der Codewortraum groß genug ist, um eventuelle Kollisionen zu vermeiden. Aus den in Tabelle 3.2 gegebenen Möglichkeiten wird in SPuIe die Gruppenanzahl 3 verwendet. Ein gültiges *UniqueSecretPassword* in SPuIe könnte also z. B. `ABCD-EFGH-1234` sein.

Anzahl	Beispiel	Codewortraum
1	A7B	$36^3 \approx 10^{4.67}$
2	BUW-JS7	$36^6 \approx 10^{9.34}$
3	B7W-QMM-38I	$36^9 \approx 10^{14.01}$

Tabelle 3.3.: Codewortraum bei Gruppengröße 3

Bei der Auswahl der Gruppenanzahl für die *Legitimations* muss der Codewortraum nicht mehr ganz so groß sein, da Kollisionen dort nicht direkt sicherheitskritisch sind. Aus diesem Grund verwendet SPuIe die Gruppenanzahl 2. Eine gültige *Legitimation* ist demnach z. B.: `ABC-123`.

3.2.2. Format der Fragen und Antworten in der Datenbank (Meta-Format)

Zur Speicherung der Fragen und deren Antworten muss ein spezielles Format definiert werden, da die verschiedenen Fragetypen in einem Feld der Datenbank untergebracht sind.

Bei der Definition dieses so genannten Meta-Formats werden hier in der Dokumentation folgende Makrodefinitionen benötigt:

$$
\begin{aligned}
\mathit{LABEL} ::= \{&\mathsf{A}-\mathsf{Z}, \mathsf{a}-\mathsf{z}, 0-9, (,), \_, -, /, ., ,, !, ?, ;, :, <, >,\\
&@, \#, *, \%, \&, [,], \{, \}, +, =, \ , \mathsf{NEWLINE}\}\\
\mathit{INT} ::= \{&0-9\}\\
\mathit{BOOL} ::= \{&\mathsf{T}, \mathsf{F}\}
\end{aligned}
$$

Das Zeichen \$ wird als Feldterminator benutzt, $*$ wird in den folgenden Tabellen als der Kleenesche Sternoperator benutzt.

In den Datenbank-Tabellen *Template* und *Answer* haben die verschiedenen *Template*-Typen das in Tabelle 3.4 angegebene Format.

Template_Type	*MetaText*	*MetaAnswer*
Node	NodeName : *LABEL*$*$\$	NULL
Line	LineName : *LABEL*$*$\$	NULL
Label	Text : *LABEL*$*$\$	NULL
Editable	Question : *LABEL*$*$\$ Lines : *INT*$*$\$	Answer : *LABEL*$*$\$
Radio	Question : *LABEL*$*$\$ Anzahl : *INT*$*$\$ Item(0) : *LABEL*$*$\$ ... Item(Anzahl-1) : *LABEL*$*$\$	Answer : *INT*$*$\$
Choice	Question : *LABEL*$*$\$ Anzahl : *INT*$*$\$ Item(0) : *LABEL*$*$\$ ... Item(Anzahl-1) : *LABEL*$*$\$	Answer(0) : *BOOL*\$ ... Answer(Anzahl-1) : *BOOL*\$

Tabelle 3.4.: Meta-Format der Fragen und Antworten

Zur Evaluierung werden noch erweiterte Meta-Formate benötigt, da hier aufsummierte Werte anstelle von Individualwerten vorkommen. Dieses leicht abweichende Meta-Format ist in Tabelle 3.5 auf der nächsten Seite zu sehen.

Template_Type	*EvalMetaAnswer*
Node, Line, Label	n/a
Editable	MetaAnswer : $LABEL*$ (ein Feld-Element pro Antwort)
Radio	NotVoted : $INT*$ Count(0) : $INT*$ ... Count(Anzahl-1) : $INT*$ (ein Feld-Element pro Frage)
Choice	Count(0) : $INT*$ ... Count(Anzahl-1) : $INT*$ (ein Feld-Element pro Frage)

Tabelle 3.5.: Meta-Format der evaluierten Antworten

4. Server

Der Server verwaltet ankommende Netzwerk-Verbindungen von Clients und fungiert als Schnittstelle zur Datenbank. Er ist ebenso wie der Client in JAVA programmiert, ist aber nicht als Applet, sondern als eigenständige JAVA-Applikation ausgelegt. Der Quelltext des Servers ist ebenso wie der des Clients mit Javadoc dokumentiert und kann mittels eines Web-Browsers eingesehen werden. Diese Dokumentation ist im SPuLe Installationspaket in der Datei `javadoc/packages.html` zu finden.

Der Server muss als Applikation laufen, da er mehr Rechte benötigt als in einer Browser-Sandbox zur Verfügung stehen. Da er nur an einer Stelle, nämlich zentral beim *Trust*, läuft, stellt dies keine Einschränkung dar.

Wird der Server gestartet, initialisiert er zuerst seinen Zufallszahlengenerator. Hierfür wird das Linux-Device `/dev/random` benutzt (siehe Kapitel 7.1.3 auf Seite 54). Danach wartet der Server auf eingehende TCP-Verbindungen auf Port 13050 (dieser Port kann bei Bedarf in der Datei `server/Server.java` geändert werden). Für jede ankommende Verbindung wird ein Thread gestartet, der nur für diesen einen Client zuständig ist. Die Verbindung zwischen Client und Server erfolgt verschlüsselt. Aus diesem Grund wird für den Server, welcher unbeaufsichtigt läuft und mit dem keine Benutzerinteraktion stattfindet, eine Zufallszahlenquelle benötigt. Um Zugriff auf `/dev/random` zu bekommen, muss der Server als Applikation laufen.

In jedem einzelnen Thread, den der Server für einen Client anlegt, wird in einer Endlosschleife auf Nachrichten vom Client gewartet. Diese Nachrichten bestehen immer aus zwei Teilen: Zuerst muss der Client die Kennung der Nachricht senden, dann erst wird die eigentliche Nachricht übermittelt. Dies hat den Vorteil, dass der Server bei allen Nachrichten im Voraus weiß, wie groß sie sind. Dadurch wird die Chance verringert, Fehler im Programm durch „Buffer Overflow“-Angriffe ausnutzen zu können. Sobald eine ungültige Nachricht beim Server ankommt, wird die Verbindung zum entsprechenden Client beendet (eine ungültige Nachricht ist in diesem Fall nicht eine, in der nur ein Datum falsch ist, sondern eine Nachricht, deren Aufbau oder Syntax inkorrekt ist). Dieses Verhalten ist aus Sicherheitsgründen so restriktiv.

Die ankommenden Nachrichten werden in einzelnen Prozeduren abgearbeitet. Dabei wird zu Beginn zuerst einmal die Datenbank exklusiv gesperrt. Auf diese Weise kann ein Transaktionsmodell garantiert werden, welches Inkonsistenzen der Datenbank bei intensivem Mehrbenutzerbetrieb vermeidet. Am Ende einer jeweiligen Transaktion wird die Datenbank wieder freigegeben. Andere Threads, die auf die Datenbank zugreifen wollen, sind in diesem Zeitraum blockiert. Diese Verzögerung sollte im Normalfall jedoch so klein sein, dass sie keine Behinderung des Betriebes für andere Benutzer nach sich zieht.

Damit dieses Modell realisiert werden kann, ist es essentiell, dass das Backend der Datenbank Transaktionen und exklusives Sperren unterstützt. Ein weiterer Punkt, der bei der Wahl

der Datenbank eine Rolle spielt, ist die Tatsache, dass der Quellcode der Datenbank öffentlich sein muss, da sonst keine völlige Transparenz im System vorliegt. Aus diesem Grund kamen MySQL und PostgreSQL in Frage. MySQL bietet nur eingeschränkten Transaktions-Support an, daher fiel die Wahl auf PostgreSQL.

Die Ansteuerung des Datenbanken-Backends aus JAVA erfolgt über die JDBC (JAVA DataBase Connectivity). Die Programmierung der Datenbank selbst erfolgt mit SQL. Die Erstinstallation und Initialisierung der Datenbank erfolgt mit mitgelieferten Skripten, die ansonsten unabhängig von Client und Server sind und nur diesem Zweck dienen.

5. Client

Der Client dient als Schnittstelle für *Admins*, *Organizers* und *Listeners*. Bei der Anmeldung beim Server wird durch das Passwort festgestellt, welches der drei Benutzerinterfaces zur Verfügung gestellt werden soll. Hierbei ist die Funktionalität des Interfaces für den *Listener* in der des *Organizers* enthalten. Dieses wiederum ist in der Funktionalität des *Admin*-Interfaces enthalten. Als Implementierungssprache wurde JAVA gewählt, da ein Maximum an Plattformunabhängigkeit gewünscht ist. Ferner sind JAVA Programme direkt im Web-Browser ausführbar, was manuelle Installationen, welche ein Hinderungsgrund gegen die Benutzung wären, überflüssig macht. Der Quelltext des Clients ist ebenso wie der des Servers mit Javadoc dokumentiert und kann mittels eines Web-Browsers eingesehen werden. Diese Dokumentation ist im SPUle Installationspaket in der Datei `javadoc/packages.html` zu finden.

Einer der Hauptgründe, warum Vorlesungsumfragen unter stets mangelnder Beteiligung leiden, ist die Bequemlichkeit der Wähler. Daher muss jede Form von Aufwand für den Wähler minimiert werden und so ist es auch verständlich, dass die Möglichkeit, JAVA Programme direkt aus dem Web-Browser zu starten, für sich allein bereits die Wahl auf JAVA fixiert. Der Client läuft sowohl als JAVA-Applet als auch als JAVA-Applikation. Er benutzt die JAVA-AWT Bibliothek zur Visualisierung.

Um die Sicherheit zu gewährleisten, muss der Nutzer des Clients sicherstellen, dass er genau das Programm vom *Trust* ausführt und nicht eine manipulierte Version. Startet er den Client direkt aus dem Web-Browser, so kann er dies nicht sicherstellen, da ein Angreifer die unsichere Verbindung zwischen dem Web-Browser und dem Web-Server manipulieren könnte. Dennoch wird gerade diese Art, an den Code des Clients zu gelangen, am meisten benutzt werden, da die Bequemlichkeit leider eine große Rolle spielt. Um wirklich sicherzugehen, muss man den Client irgendwie auf sicherem Wege, zum Beispiel per Diskette direkt vom *Trust* beschaffen. Dann muss man ihn jedoch manuell starten und kann ihn nicht mehr direkt aus dem Web-Browser heraus starten.

Nach der Initialisierung des Zufallszahlengenerators und dem Verbindungsaufbau zum Server wird das Hauptfenster angezeigt. Der Client enthält eine Menge von Fenster-Klassen, wobei diese meistens ein ziemlich direktes Frontend für eine spezielle Relation in der Datenbank darstellen. Die Fenster schicken zunächst eine Anfrage an den Server, wie der aktuelle Stand der Daten aussieht. Danach stellen sie diesen dar und bieten Möglichkeiten zur Manipulation der Daten an. Führt der Nutzer eine Änderung durch, wird eine Nachricht zum Server geschickt. Danach wird eine neue Anfrage nach dem aktuellen Stand der Daten geschickt und der Fensterinhalt neu aufgebaut.

Normalerweise läuft der Client in einer ziemlich beschränkten JAVA Sandbox. Um jedoch drucken zu können, benötigt er mehr Rechte und kann daher nicht mehr aus dem Web-Browser heraus gestartet werden. Um *Legitimations* auszudrucken oder die Auswer-

tung einer Vorlesung vorzunehmen, muss der Client als JAVA-Applikation gestartet werden, da externe Programme aufgerufen werden müssen. Die *Legitimations* werden in SPuLe direkt als PostScript generiert und an ein konfigurierbares Programm über `stdout` gestreamt. Um hier Programme, wie z. B. `lpr` oder `gv` angeben zu können, sind die erweiterten Rechte einer JAVA-Applikation nötig. Bei der Auswertung einer Vorlesung wird LaTeX-Code generiert und in eine Datei mit frei wählbarem Namen geschrieben. Aus diesem Grunde sind auch hier Applikationsrechte nötig. Die LaTeX Dateien müssen dann noch von Hand geTeXt werden. Dies ist deshalb sinnvoll, da höchstwahrscheinlich manuelle Nachbearbeitung im Sinne von Zensur unerwünschter Kommentare sowieso stattfinden muss.

Der Client enthält keine besonderen technischen Raffinessen. Er ist im Wesentlichen ein Frontend für die Nachrichten zum Server. Dabei entstand zwar eine große Menge an Quelltext, jedoch ergibt sich dieser quasi als Zwangsläufigkeit. Leider lässt er sich schwierig generieren, da jedes Fenster seine kleinen Eigenheiten hat. Im Nachhinein wäre dennoch zumindest eine teilweise Generierung besser gewesen, als alles von Hand zu programmieren.

Ein großes Problem hingegen stellten die Unzulänglichkeiten von JAVA-AWT und JAVA Applets dar. Die AWT Klassen sind problematisch anzuwenden. Das Hauptproblem ist die Platzierung von Fenstern. Es mussten harte Koordinaten, an die neue Fenster gesetzt werden, einprogrammiert werden, da sie ansonsten stets in der linken oberen Ecke entstehen. Des Weiteren ist die Größe von Fenstern in Pixeln anzugeben, obwohl ihr Inhalt je nach Architektur einmal größer und einmal kleiner ist. Dies erklärt auch die vielen Beispiele von in Web-Seiten eingebetteten JAVA Programmen im Internet, welche je nach Browser ihren Inhalt manchmal abgeschnitten darstellen, da er nicht ins Fenster passt. Leider ist der Client durch diese Unzulänglichkeiten aus ergonomischen Gesichtspunkten sehr schlecht geraten. Bei einer Neuimplementierung sollte eine andere Programmiersprache mit vernünftigen GUI-Tools oder JAVA-Swing verwendet werden, wobei dann wieder das Problem besteht, dass die meisten Browser noch veraltete JAVA Versionen besitzen und somit kein Swing unterstützen. Alternative Programmiersprachen wiederum würden einen höheren Installationsaufwand nach sich ziehen.

6. Nachrichten

Die Kommunikation zwischen dem Server und dem Client erfolgt durch das Versenden von Nachrichten durch den sicheren Kanal (siehe Kapitel 7 auf Seite 53). Zuerst wird eine 32`bit` ID der Nachricht gesendet, erst danach wird das eigentliche Nachrichtenobjekt hinterher geschickt. Dies ermöglicht es dem Server, kontrolliert auf fehlerhafte Nachrichten zu reagieren, da er zum einen bei einer fehlerhaften Nachrichten-ID die Verbindung zum Client beenden kann und zum anderen nach dem Erhalt der ID das Format des Nachrichtenobjektes kennt und somit „Buffer Overflow“-Attacken mit unbekannten Nachrichtenobjekten nicht ermöglicht werden.

Auf jede Nachricht wird exakt eine Antwort gesendet. Im Fehlerfall sendet der Server immer Misc_Result_Send (siehe Kapitel 6.2.15 auf Seite 50). Aus diesem Grund wird im Folgenden unter „Server-Antwort“ nur die Nachricht beschrieben, die im Erfolgsfall gesendet wird.

6.1. Nachrichten vom Client zum Server

6.1.1. Login

Signatur:

Login (string Person_UniqueSecretPassword)

Parameter:

string UniqueSecretPassword
: Die geheime Benutzerkennung des Benutzers, der sich einloggen will.

Server-Bedingungen:

- UniqueSecretPassword muss in der Datenbank eingetragen sein.
- Ist die Person ein *Admin*, so wird sich ihre ID speziell gemerkt, um später Login_SU zu unterstützen.

Server-Antwort:

Login_Successful (siehe Kapitel 6.2.1 auf Seite 45)

6.1.2. Login_SU

Signatur:

Login_SU (int Person_ID)

Parameter:

int Person_ID
: Die ID der Person, zu der sich der *Admin* umwandeln will.

Server-Bedingungen:

- Eingeloggte Person muss ein *Admin* sein.
- ID der eingeloggten Person in Person_ID umwandeln; eigene, „echte“ ID merken, damit man weiterhin auf *Admin*-Status prüfen kann.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.3. Person_Get

Signatur:

Person_Get (int Person_ID)

Parameter:

int Person_ID
: ID, über die Informationen eingeholt werden soll.

Server-Bedingungen:

- Falls Person_ID == 0 gilt und die eingeloggte Person ein *Admin* ist, so werden die Spalten *ID*, *Name* und *Rank* der Tabelle *Person* zurückgeliefert.
- Falls Person_ID != 0 gilt und die eingeloggte Person entweder ein *Admin* ist oder zusammen mit *Person_ID* eine *Lecture* organisiert oder den eigenen Datensatz erfragen will, so werden die Spalten *ID*, *Name* und *Rank* der Tabelle *Person* zurückgeliefert.

Server-Antwort:

Person_Send (siehe Kapitel 6.2.2 auf Seite 45)

6.1.4. Person_ReallifeID_Get

Signatur:

Person_ReallifeID_Get (int Person_ID)

Parameter:

int Person_ID
: ID der Person, deren *ReallifeID* erfragt werden soll.

Server-Bedingungen:

- Wenn die eingeloggte Person ein *Admin* ist, so liefere die *ReallifeID* der angegebenen Person zurück.

Server-Antwort:

Person_ReallifeID_Send (siehe Kapitel 6.2.3 auf Seite 45)

6.1.5. Person_Wildcard_Get

Signatur:

Person_Wildcard_Get (string Person_Name)

Parameter:

string Person_Name
: Suchstring für Personen, deren Daten zurückgeliefert werden sollen.

Server-Bedingungen:

- Wenn die eingeloggte Person ein *Admin* ist, liefere *ID*, *Name* und *Rank* der Personen, die auf den Suchstring matchen, zurück.
- Zur Suche wird die SQL Syntax `LIKE` verwendet. Es werden also Teilstrings gefunden und die folgenden Spezialzeichen sind erlaubt:

Zeichen	Bedeutung
%	matcht jeden String aus 0 oder mehr Zeichen
_	matcht jedes Einzelzeichen

Server-Antwort:

Person_Send (siehe Kapitel 6.2.15 auf Seite 50)

6.1.6. Person_Create

Signatur:

Person_Create (string Person_ReallifeID, string Person_Name, int Person_Rank)

Parameter:

string Person_ReallifeID
: Nachprüfbare Identifikation der Person (z. B. Matrikelnummer, Personalausweisnummer, . . .), die dem System hinzugefügt werden soll.

string Person_Name
: Name der Person, wie er in SPuLe verwendet werden soll.

int Person_Rank
: Rang der einzutragenden Person:

Parameter	Rang
1	*Listener*
2	*Organizer*
3	*Admin*

Server-Bedingungen:

- Trägt die Person mit den gewünschten Daten in SPuLe ein, wenn die eingeloggte Person ein *Admin* ist und weder *Person_Name* noch *Person_ReallifeID* leer sind.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.7. Person_Modify

Signatur:

Person_Modify (int Person_ID, string Person_ReallifeID, string Person_Name,
int Person_Rank)

Parameter:

int Person_ID
: Identifikationsnummer der zu ändernden Person.

string Person_ReallifeID
: Geänderte *ReallifeID*.

string Person_Name
: Geänderter Name der Person wie er in SPuLe verwendet wird.

int Person_Rank
: Geänderter Rang der Person.

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.
- *Person_ID*, *Person_ReallifeID* und *Person_Name* dürfen nicht leer sein, *Person_Rank* muss im definierten Bereich (siehe Kapitel 6.1.6 auf der vorherigen Seite) liegen.
- Eingeloggte Person darf den eigenen Rang nicht verändern (auch nicht, wenn sie mit *LoginSU* als andere Person eingeloggt ist).
- Rang anderer Personen darf immer auf *Organizer* oder *Admin* gesetzt werden.
- Rang anderer Personen darf nur dann auf *Listener* gesetzt werden, wenn diese keine aktiven *Lectures* haben, d. h. als *Organizer* einer *Lecture* eingetragen sind.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.8. Person_Remove

Signatur:

Person_Remove (int Person_ID)

Parameter:

int Person_ID
: Identifikationsnummer der Person, die aus dem System gelöscht werden soll.

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.
- Eingeloggte Person darf sich selbst nicht löschen (auch nicht, wenn sie mit *LoginSU* als andere Person eingeloggt ist).
- Person darf nur gelöscht werden, wenn sie keine aktiven *Lectures* hat, d. h. als *Organizer* einer *Lecture* eingetragen ist.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.9. Person_Print

Signatur:

Person_Print (int Person_ID)

Parameter:

int Person_ID
: Identifikationsnummer der Person, deren *UniqueSecretPassword* neu gesetzt und anschließend auf dem sicheren Drucker im *Trust* ausgedruckt werden soll.

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.10. Lecture_Create

Signatur:

Lecture_Create (string Lecture_Name, int NumberOfEvents)

Parameter:

string Lecture_Name
: Name der zu generierenden *Lecture*, welcher eindeutig sein muss. D. h. bei sich jährlich wiederholenden Vorlesungstiteln ist es ratsam, einen Zusatz wie z. B. „SS01“ an den Namen zu hängen.

int NumberOfEvents
: Anzahl der einzelnen *Events* der *Lecture*, an denen Legitimierungszettel ausgegeben werden. Dieser Wert ist für die restliche Lebensdauer der *Lecture* fix und sollte daher von Anfang an mit Bedacht gewählt werden.

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.
- *NumberOfEvents* muss im Bereich $0 \ldots 16$ liegen.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.11. Lecture_Remove

Signatur:

Lecture_Remove (string Lecture_Name)

Parameter:

string Lecture_Name
: Vorlesungstitel der zu entfernenden *Lecture*.

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.
- *Lecture* darf nicht im Zustand *running* sein (siehe Kapitel 6.1.13 auf der nächsten Seite).
- Daten werden kaskadierend aus den folgenden Tabellen gelöscht, wobei die folgenden Abhängigkeiten berücksichtigt werden:
  Lecture → *LectureOrganizer*
  Lecture → *Event* → *Legitimation*
  Lecture → *Template* → *Answer*

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.12. Lecture_Get

Signatur:

Lecture_Get (string Lecture_Name)

Parameter:

string Lecture_Name
: Vorlesungstitel, über den Informationen eingeholt werden sollen.

Server-Bedingungen:

- Wenn eingeloggte Person ein *Listener* ist, dann werden nur laufende *Lectures* (d. h. *Lectures* mit Zustand *running*, siehe Kapitel 6.1.13 auf der nächsten Seite) zurückgeliefert.
- Ist die eingeloggte Person ein *Organizer* oder ein *Admin*, so werden alle *Lectures* zurückgeliefert.

- Ist *Lecture_Name* leer, so wird eine Liste aller *Lectures* entsprechend dem Rang der eingeloggten Person zurückgeliefert.
- Wenn *Lecture_Name* gesetzt ist, so werden Informationen über diese *Lecture* zurückgeliefert (sofern vorhanden und Rang der eingeloggten Person den Zugriff erlaubt).
- Information, ob die eingeloggte Person ein *Organizer* dieser *Lecture* ist, wird auch zurückgeliefert.
- Information, ob die eingeloggte Person bereits *Legitimations* für diese *Lecture* eingegeben hat.
- *Lectures*, die mit 0 *Events* angelegt sind, werden immer als legitimiert zurückgeliefert.

Server-Antwort:

Lecture_Send (siehe Kapitel 6.2.4 auf Seite 46)

6.1.13. Lecture_Modify

Signatur:

Lecture_Modify (string Lecture_Name, int Lecture_State)

Parameter:

string Lecture_Name
: Name der *Lecture*, deren Zustand geändert werden soll.

int Lecture_State
: Neuer Zustand der *Lecture*:

Parameter	Vorlesungszustand
1	*initialized*
2	*running*
3	*terminated*

Server-Bedingungen:

- Nur möglich, wenn eingeloggte Person ein *Admin* ist.
- *Lectures* dürfen nur in einen „späteren“ Zustand geschoben werden. Ein Ändern in einen „früheren“ Zustand ist nicht möglich.
- Die Änderung von *initialized* nach *running* kann nur durchgeführt werden, wenn die *Lecture* mindestens einen *Organizer* zugeordnet hat.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.14. Lecture_Event_Get

Signatur:

Lecture_Event_Get (string Event_Lecture_Name)

Parameter:

string Event_Lecture_Name
: Name der *Lecture*, deren einzelne *Events* erfragt werden.

Server-Bedingungen:

- Die eingeloggte Person muss entweder ein *Admin* oder ein *Organizer* der *Lecture* oder legitimierter *Listener* sein.
- Die *Events* werden in chronologischer Reihenfolge beginnend mit Nummer 1 gesendet.

Server-Antwort:

Lecture_Event_Send (siehe Kapitel 6.2.5 auf Seite 46)

6.1.15. Lecture_Event_Modify

Signatur:

Lecture_Event_Modify (string Event_Lecture_Name, int Event_Pos,
string Event_Name)

Parameter:

string Event_Lecture_Name
: Name der *Lecture*, bei der eine Beschreibung eines *Events* geändert werden soll.

int Event_Pos
: *Event*-Nummer, deren Beschreibung geändert werden soll.

string Event_Name
: Neue Beschreibung des *Events*.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein und die *Lecture* darf sich nicht im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) befinden.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.16. LectureOrganizer_Add

Signatur:

LectureOrganizer_Add (string LectureOrganizer_Lecture_Name,
int LectureOrganizer_Person_ID)

Parameter:

string LectureOrganizer_Lecture_Name
: Name der *Lecture*, die einen *Organizer* hinzugefügt bekommen soll.

int LectureOrganizer_Person_ID
: Identifikationsnummer der Person, die nun als *Organizer* der *Lecture* hinzugefügt werden soll.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein.

- Die Person, die hinzugefügt werden soll, muss *Organizer* sein und darf nicht bereits *Organizer* dieser *Lecture* sein.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.17. LectureOrganizer_Remove

Signatur:

LectureOrganizer_Remove (string LectureOrganizer_Lecture_Name,
int LectureOrganizer_Person_ID)

Parameter:

string LectureOrganizer_Lecture_Name
: Name der *Lecture*, bei der ein *Organizer* entfernt werden soll.

int LectureOrganizer_Person_ID
: Identifikationsnummer der Person, die als *Organizer* der *Lecture* entfernt werden soll.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein.
- Der letzte *Organizer* der *Lecture* darf nur im Zustand *initialized* entfernt werden.
- Ein *Organizer* darf sich nur dann selber entfernen, wenn er nicht der letzte *Organizer* der *Lecture* ist.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.18. LectureOrganizer_PersonIDs_Get

Signatur:

LectureOrganizer_PersonIDs_Get (string LectureOrganizer_Lecture_Name)

Parameter:

string LectureOrganizer_Lecture_Name
: Name der *Lecture*, deren *Organizer* erfragt werden sollen.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* sein.

Server-Antwort:

LectureOrganizer_PersonIDs_Send (siehe Kapitel 6.2.6 auf Seite 46)

6.1.19. Legitimation_Create

Signatur:

Legitimation_Create (string Legitimation_Lecture_Name,
int Legitimation_Event_Pos, int Count)

Parameter:

string Legitimation_Create
: Name der *Lecture*, für die *Legitimations* erstellt werden sollen.

int Legitimation_Event_Pos
: *Event*-Nummer, für die *Legitimations* erstellt werden sollen.

int Count
: Anzahl der zu erstellenden *Legitimations*.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein und die *Lecture* darf sich nicht im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) befinden.
- Erstelle höchstens die konfigurierbare Maximalanzahl von *Legitimations* pro Nachricht (*MAX_LEGITIMATIONS*; default 500), fange bei 1 an, wenn noch keine vorhanden, sonst höchste vorhandene Nummer plus 1.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.20. Legitimation_Get

Signatur:

Legitimation_Get (string Legitimation_Lecture_Name)

Parameter:

string Legitimation_Lecture_Name
: Name der *Lecture*, deren *Legitimations* erfragt werden sollen.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein.
- Alle *Legitimations* werden zurückgeliefert, auch bereits benutzte!

Server-Antwort:

Legitimation_Send (siehe Kapitel 6.2.7 auf Seite 47)

6.1.21. Legitimation_Modify

Signatur:

Legitimation_Modify (string Legitimation_Lecture_Name,
int Legitimation_Event_Pos, int Legitimation_Number_Begin,
int Legitimation_Number_End, boolean Legitimation_isDeleted)

Parameter:

string Legitimation_Lecture_Name
: Name der *Lecture*, deren *Legitimations* verändert werden sollen.

int Legitimation_Event_Pos
: Nummer der einzelnen *Events*, deren *Legitimations* verändert werden sollen.

int Legitimation_Number_Begin
: Startnummer der zu verändernden *Legitimations*.

int Legitimation_Number_End
: Endnummer der zu verändernden *Legitimations*.

boolean Legitimation_isDeleted
: Flag zum Aktivieren oder Deaktivieren von den spezifizierten *Legitimations*:

Parameter	Bedeutung
true	*Legitimations* werden deaktiviert
false	*Legitimations* werden aktiviert

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein und die *Lecture* darf sich nicht im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) befinden.
- Auch bereits zum Abstimmen benutzte *Legitimations* werden als gelöscht markiert (dies ist gewünschtes Verhalten)!
- *Die Legitimations* eines *Events* sind lückenlos durchnummeriert und beginnen bei 1. *Legitimation_Number_Begin* und *Legitimation_Number_End* sind inklusive.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.22. Legitimation_Enter

Signatur:

Legitimation_Enter (string Legitimation_Code)

Parameter:

string Legitimation_Code
: Kennung der zum Abstimmen zu benutzenden *Legitimation*.

Server-Bedingungen:

- Eingeloggte Person muss mindestens *Listener* sein.
- *Lecture* muss sich im Zustand *running* (siehe Kapitel 6.1.13 auf Seite 32) befinden.
- *Legitimation* muss unbenutzt sein.
- Eingeloggte Person darf nicht *Organizer* der *Lecture* sein.
- Eingeloggte Person darf nicht bereits eine andere *Legitimation* desselben *Events* benutzt haben.
- Implizit werden alle veränderbaren *Templates* der *Lecture* in die Tabelle *Answer* kopiert, wenn sie dort noch nicht für die eingeloggte Person vorhanden sind.

Server-Antwort:

Legitimation_Successful (siehe Kapitel 6.2.9 auf Seite 47)

6.1.23. Legitimation_Entered_Get

Signatur:

Legitimation_Entered_Get (string Legitimation_Lecture_Name)

Parameter:

string Legitimation_Lecture_Name
: Name der *Lecture*, über die benutzte *Legitimations* der eingeloggten Person erfragt werden sollen.

Server-Bedingungen:

- Wenn *Legitimation_Lecture_Name* existiert, werden alle bereits benutzten *Legitimations* der eingeloggten Person für diese *Lecture* zurückgeliefert.

Server-Antwort:

Legitimation_Entered_Send (siehe Kapitel 6.2.8 auf Seite 47)

6.1.24. Template_Get

Signatur:

Template_Get (int Template_ID)

Parameter:

int Template_ID
: Identifikationsnummer der *Template*, deren Daten erfragt werden sollen.

Server-Bedingungen:

- Mindestens eine der folgenden Bedingungen muss für die eingeloggte Person erfüllt sein:
  - *Admin*.
  - Besitzer von *Template_ID*.
  - *Organizer* der *Lecture* zu der *Template_ID* gehört.
  - *Listener* und *Legitimation* für die *Lecture* zu der *Template_ID* gehört.
- *Template_ID* muss größer 0 sein.

Server-Antwort:

Template_Send(ERROR_OK) (siehe Kapitel 6.2.11 auf Seite 48)

6.1.25. Template_Children_Get

Signatur:

Template_Children_Get (int Template_ID)

Parameter:

int Template_ID
: Identifikationsnummer der *Template*, deren Daten der Kinder-*Templates* erfragt werden sollen.

Server-Bedingungen:

- *Template_Type* (Übersicht der verschiedenen Typen in Tabelle 3.4 auf Seite 19) muss *Node* sein.
- Mindestens eine der folgenden Bedingungen muss für die eingeloggte Person erfüllt sein:

  - *Admin*
  - Besitzer von *Template_ID*
  - *Organizer* der *Lecture*, zu der *Template_ID* gehört.
  - *Listener* und *Legitimation* für die *Lecture* zu der *Template_ID* gehört.

- Rückgabe ist Array, welches außer den Daten der Tabelle *Template* noch die Antworten der Tabelle *Answer* für die eingeloggte Person enthält, wenn sie vorhanden sind.

Server-Antwort:

Template_Children_Send(ERROR_OK) (siehe Kapitel 6.2.12 auf Seite 48)

6.1.26. Template_Create

Signatur:

Template_Create (int Template_FatherID, int Template_LeftBrotherID,
int Template_Type, string Template_MetaText)

Parameter:

int Template_FatherID
: Identifikationsnummer der *Template*, die Vater der zu generierenden *Template* werden soll.

int Template_LeftBrotherID
: Identifikationsnummer der *Template*, die der linke Bruder der zu generierenden *Template* werden soll.

int Template_Type
: Typ der zu generierenden *Template*.

string Template_MetaText
: *MetaText* der *Template* (Übersicht der verschiedenen Typen und ihrer Meta-Formate in Tabelle 3.4 auf Seite 19).

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* sein.
- *Template_FatherID* $\neq$ *Template_LeftBrotherID* (Ausnahme: beide 0).
- Wenn *Template_FatherID* $= 0$, dann *Template_LeftBrotherID* $= 0$ und *Template_Type* $=$ *Node*, sonst Fehler.
- Zustand der *Lecture* muss *initialized* sein.

- Eigentümer des Vaterknotens darf nicht *Admin* sein, es sei denn, eingeloggte Person ist *Admin*.
- Alle Bruderknoten müssen denselben Vaterknoten haben.
- Reihenfolge zwischen Bruderknoten muss konsistent bleiben.
- Test auf zyklischen Graphen muss beim Einfügen nicht stattfinden, da durch Einfügen kein zyklischer Graph erzeugt werden kann (auf die nicht-existente Identifikationsnummer des neuen Knotens kann noch kein anderer zeigen).

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.27. Template_Remove

Signatur:

Template_Remove (int Template_ID)

Parameter:

int Template_ID
: Identifikation der *Template*, die entfernt werden soll.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* sein.
- Zusätzlich alle folgenden Bedingungen zum Löschen von *Template_ID*:
  - *Template*-Eigentümer ist nicht 0 oder eingeloggte Person ist *Admin*.
  - *Template* ist keine *Topnode* einer *Lecture*.
  - Zustand der *Lecture* ist *initialized*.
  - Eingeloggte Person ist *Admin* oder eingeloggte Person ist Eigentümer der *Template* oder eingeloggte Person ist *Organizer* der *Topnode* der *Lecture* und Eigentümer der *Template* ist kein *Admin*.
- Beim rekursiven Löschen der Kindknoten muss nur noch der letzte der obigen Punkte erfüllt werden.
- Die korrekte Reihenfolge zwischen Bruderknoten wird beim Löschen erhalten.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.28. Template_Modify

Signatur:

Template_Modify (int Template_ID, int Template_FatherID,
int Template_LeftBrotherID, int Template_Type, string Template_MetaText)

Parameter:

int Template_ID
: Identifikationsnummer der zu ändernden *Template*.

int Template_FatherID
: Identifikationsnummer des neuen Vaterknotens.

int Template_LeftBrotherID
: Identifikationsnummer des neuen linken Bruders.

int Template_Type
: Neuer Typ der *Template*.

string Template_MetaText
: Neuer *MetaText* der *Template* (Übersicht der verschiedenen Typen und ihrer Meta-Formate in Tabelle 3.4 auf Seite 19).

Server-Bedingungen:

- *Template_ID* $\neq 0$ und zusätzlich eine der folgenden Bedingungen für eingeloggte Person erfüllt:
  - *Admin*.
  - *Organizer* und Eigentümer von *Template_ID*.
  - *Organizer* der *Lecture* zu der *Template_ID* gehört.
- *Template_FatherID* $\neq$ *Template_LeftBrotherID* (Ausnahme: beide 0).
- Wenn *Template_FatherID* $= 0$, dann *Template_LeftBrotherID* $= 0$ und *Template_Type* $=$ *Node*, sonst Fehler.
- *Template_Type* kann für folgende Typen geändert werden:
  Line $\leftrightarrow$ *Label*
  Radio $\leftrightarrow$ *Choice*
- Zustand der *Lecture* muss *initialized* sein.
- Eigentümer des Vaterknotens darf nicht *Admin* sein, es sei denn, eingeloggte Person ist *Admin*.
- Alle Bruderknoten müssen denselben Vaterknoten haben.

- Reihenfolge zwischen Bruderknoten muss konsistent bleiben.
- Test, ob der Graph noch azyklisch ist, wenn nicht, Fehler!
- Das Attribut *Person_ID* in der Tabelle *Template* wird beim Verändern auf die Identifikationsnummer der eingeloggten Person gesetzt, es sei denn, die Veränderung bezieht sich nur auf *Template_FatherID* und *Template_LeftBrotherID* und wird nur geändert, weil die Reihenfolge der *Templates* modifiziert wurde.

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.29. Answer_Modify

Signatur:

Answer_Modify (int Answer_Template_ID, string Answer_MetaAnswer)

Parameter:

int Answer_Template_ID
: Identifikationsnummer der *Template*, deren Antwort geändert werden soll.

string Answer_MetaAnswer
: Neue Antwort im Meta-Format (Übersicht der verschiedenen Typen und ihrer Meta-Formate in Tabelle 3.4 auf Seite 19).

Server-Bedingungen:

- Eingeloggte Person muss mindestens *Listener* sein.
- *Lecture*, zu der *Answer_Template_ID* gehört, muss im Zustand *running* (siehe Kapitel 6.1.13 auf Seite 32) sein.
- Falls Antworten für eingeloggte Person noch nicht vorhanden sind, kreiere alle Antworten für die entsprechende *Lecture* für die eingeloggte Person (nur die Typen *Editable*, *Radio* und *Choice*).

Server-Antwort:

Misc_Result_Send(ERROR_OK) (siehe Kapitel 6.2.15 auf Seite 50)

6.1.30. Evaluate_Statistic_Get

Signatur:

Evaluate_Statistic_Get (string Lecture_Name)

Parameter:

string Lecture_Name
: Name der zu evaluierenden *Lecture*.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein und die *Lecture* muss sich im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) befinden.
- Es müssen mindestens *MIN_VOTES* Stimmen (default: 5) für die *Lecture* abgestimmt haben.
- Hat die *Lecture* mehr als 0 *Events*, wird die Besuchsstatistik errechnet und zurückgeliefert.
- Hat die *Lecture* 0 *Events*, wird ein leeres Array als Besuchsstatistik zurückgeliefert.
- Ist mit einer gelöschten *Legitimation* abgestimmt worden, wir ein Flag gesetzt, so dass ein Vermerk auf der Auswertung erscheint.

Server-Antwort:

Evaluate_Statistic_Send(ERROR_OK) (siehe Kapitel 6.2.13 auf Seite 49)

6.1.31. Evaluate_Answers_Get

Signatur:

Evaluate_Answers_Get (int Template_ID)

Parameter:

int Template_ID
: Identifikationsnummer der *Template*, deren *Answers* erfragt werden soll.

Server-Bedingungen:

- Eingeloggte Person muss *Admin* oder *Organizer* der *Lecture* sein und die *Lecture* muss sich im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) befinden.
- Es müssen mindestens *MIN_VOTES* Stimmen (default: 5) für die *Lecture* abgestimmt haben.
- Gibt die *MetaAnswers* der angeforderten *Template* zurück. Bei *Radio* und *Choice* wird die Summe aller Stimmen zurückgeliefert, bei *Editable* wird die Liste aller *Answers* in zufälliger Reihenfolge zurückgeliefert.

Server-Antwort:

Evaluate_Answers_Send(ERROR_OK) (siehe Kapitel 6.2.14 auf Seite 49)

6.2. Nachrichten vom Server zum Client

6.2.1. Login_Successful

Signatur:

Login_Successful (int Person_ID)

Parameter:

int Person_ID
: Liefert die Identifikationsnummer der Person zurück, welche sich mittels ihrer Kennung mit S$^{p}_{u}$le authentifiziert hat.

6.2.2. Person_Send

Signatur:

Person_Send (ARRAY OF (int Person_ID, string Person_Name, int Person_Rank, boolean Person_isDeleted))

Parameter:

int Person_ID
: Identifikationsnummer der Person.

string Person_Name
: Name der Person, wie er in S$^{p}_{u}$le benutzt wird.

int Person_Rank
: Rang der Person.

6.2.3. Person_ReallifeID_Send

Signatur:

Person_Reallife_Send (string Person_ReallifeID)

Parameter:

string Person_ReallifeID
: Authentifizierungsmerkmal der Person (z. B. Matrikelnummer, Personalausweisnummer, genetischer Fingerabdruck, . . .).

6.2.4. Lecture_Send

Signatur:

Lecture_Send (ARRAY OF (string Lecture_Name, int Lecture_State,
int Lecture_Template_ID, boolean LoginIsOrganizer, boolean LoginHasAnswered))

Parameter:

string Lecture_Name
: Name der *Lecture*.

int Lecture_State
: Zustand der *Lecture* (siehe Kapitel 6.1.13 auf Seite 32).

int Lecture_Template_ID
: *Topnode* der *Lecture*.

boolean LoginIsOrganizer
: Parameter, welcher anzeigt, ob eingeloggte Person *Organizer* der *Lecture* ist.

boolean LoginHasAnswered
: Parameter, welcher anzeigt, ob eingeloggte Person *Legitimations* für die *Lecture* eingegeben hat (eine *Lecture* mit 0 *Events* liefert hier immer *true* zurück).

6.2.5. Lecture_Event_Send

Signatur:

Lecture_Event_Send (ARRAY OF (string Event_Name))

Parameter:

string Event_Name
: Beschreibung des *Events*.

6.2.6. LectureOrganizer_PersonIDs_Send

Signatur:

LectureOrganizer_PersonIDs_Send (ARRAY OF (int LectureOrganizer_Person_ID))

Parameter:

int LectureOrganizer_Person_ID
: Identifikationsnummer von *Organizer* der *Lecture*.

6.2.7. Legitimation_Send

Signatur:

Legitimation_Send (ARRAY OF (ARRAY OF(string Legitimation_Code,
boolean Legitimation_isDeleted)))

Parameter:

string Legitimation_Code
: Kennung der *Legitimation*.

boolean Legitimation_isDeleted
: Parameter, welcher angibt, ob die *Legitimation* als gelöscht markiert ist oder nicht.

6.2.8. Legitimation_Entered_Send

Signatur:

Legitimation_Entered_Send (ARRAY OF(string Legitimation_Code))

Parameter:

string Legitimation_Code
: Kennungen der *Legitimations*, die ein Benutzer bereits eingegeben hat.

6.2.9. Legitimation_Successful

Signatur:

Legitimation_Successful (string Legitimation_Lecture_Name)

Parameter:

string Legitimation_Lecture_Name
: Liefert den Namen der *Lecture* zurück, für welche eine *Legitimation* eingegeben wurde.

6.2.10. Template_Created

Signatur:

Template_Created (int Template_ID)

Parameter:

int Template_ID
: Liefert die Identifikationsnummer einer neu angelegten *Template* zurück.

6.2.11. Template_Send

Signatur:

Template_Send (int Template_ID, int Template_FatherID, int Template_LeftBrotherID,
int Template_Type, string Template_MetaText, int Template_Person_ID,
string Answer_MetaAnswer)

Parameter:

int Template_ID
: Identifikationsnummer der *Template*.

int Template_FatherID
: Identifikationsnummer des Vaterknotens der *Template*.

int Template_LeftBrotherID
: Identifikationsnummer des linken Bruderknotens der *Template*.

int Template_Type
: Typ der *Template*.

string Template_MetaText
: *MetaText* der *Template* (Übersicht der verschiedenen Typen und ihrer Meta-Formate in Tabelle 3.4 auf Seite 19).

int Template_Person_ID
: Identifikationsnummer des Eigentümers der *Template*.

string Answer_MetaAnswer
: *MetaAnswer* des eingeloggten Benutzers für die *Template*.

6.2.12. Template_Children_Send

Signatur:

Template_Children_Send (ARRAY OF (int Template_ID, int Template_FatherID,
int Template_LeftBrotherID, int Template_Type, string Template_MetaText,
int Template_Person_ID, string Answer_MetaAnswer))

Parameter:

int Template_ID
: Identifikationsnummer der *Template*.

int Template_FatherID
: Identifikationsnummer des Vaterknotens der *Template*.

int Template_LeftBrotherID
: Identifikationsnummer des linken Bruderknotens der *Template*.

int Template_Type
: Typ der *Template*.

string Template_MetaText
: *MetaText* der *Template* (Übersicht der verschiedenen Typen und ihrer Meta-Formate in Tabelle 3.4 auf Seite 19).

int Template_Person_ID
: Identifikationsnummer des Eigentümers der *Template*.

string Answer_MetaAnswer
: *MetaAnswer* des eingeloggten Benutzers für die *Template*.

6.2.13. Evaluate_Statistic_Send

Signatur:

Evaluate_Statistic_Send (int TotalVotes, ARRAY OF (int UsedLegitimationCount, int HandedOutLegitimationCount), boolean LectureIsValid)

Parameter:

int TotalVotes
: Anzahl der Personen, die für die *Lecture* abgestimmt haben.

int UsedLegitimationCount
: Anzahl der *Legitimations*, die von diesem *Event* benutzt wurden.

int HandedOutLegitimationCount
: Anzahl der *Legitimations*, die für dieses *Event* ausgeteilt wurden.

boolean LectureIsValid
: Parameter, welcher angibt, ob mit gelöschten *Legitimations* gewählt wurde oder nicht.

6.2.14. Evaluate_Answers_Send

Signatur:

Evaluate_Answers_Send (ARRAY OF string Answer_MetaAnswer)

Parameter:

string Answer_MetaAnswer
: *MetaAnswer* auf gegebene *Template*. Je nach *Template*-Typ werden summierte Werte oder eine Liste aller Werte zufällig sortiert zurückgeliefert.

6.2.15. Misc_Result_Send

Signatur:

Misc_Result_Send (int ErrorCode, string ErrorDescription)

Parameter:

int ErrorCode
: Fehlercode, den der Server auf eine Anfrage des Clients zurückmelden muss:

  ERROR_OK
  : Kein Fehler, sondern generische „OK“-Antwort des Servers. Wird benötigt, da jede Anfrage des Clients genau eine Antwort des Servers erfordert.

  ERROR_INCORRECTLOGIN
  : Es wurde versucht, sich mit einer nicht registrierten Benutzerkennung beim System zu authentifizieren.

  ERROR_ACCESSDENIED
  : Es wurde versucht, auf Daten zuzugreifen, auf die eingeloggte Person aufgrund ihres Ranges keinen Zugriff hat.

  ERROR_NOSUCHPERSON
  : Es wurde versucht, eine Operation auf einer nicht existierenden Identifikationsnummer einer Person auszuführen.

  ERROR_NOSUCHTEMPLATE
  : Es wurde versucht, eine Operation auf einer nicht existierenden Identifikationsnummer einer *Template* durchzuführen.

  ERROR_NOSUCHLECTURE
  : Es wurde versucht, eine Operation auf einer nicht existierenden *Lecture* durchzuführen.

  ERROR_NOSUCHEVENT
  : Es wurde versucht, eine Operation auf einem nicht existierenden *Event* durchzuführen.

  ERROR_SUICIDEFORBIDDEN
  : Es wurde versucht, die eingeloggte Person selbst zu löschen oder sie aus der Liste der *Organizers* einer *Lecture* zu entfernen, obwohl sie der letzte *Organizer* der *Lecture* ist.

  ERROR_VOTINGDENIED
  : Es wurde versucht, für eine *Lecture* abzustimmen, obwohl eingeloggte Person *Organizer* der *Lecture* ist.

  ERROR_ALREADYVOTED
  : Es wurde versucht, eine *Legitimation* zu benutzen, die bereits benutzt wurde, oder eine *Legitimation* für einen *Event* einzutragen, für den bereits eine *Legitimation* der eingeloggten Person eingetragen ist.

ERROR_LECTURENOTRUNNING
: Es wurde versucht, eine *Legitimation* für eine *Lecture* einzutragen, die nicht im Zustand *running* (siehe Kapitel 6.1.13 auf Seite 32) ist.

ERROR_LECTURENOTTERMINATED
: Es wurde versucht, eine *Lecture* zu evaluieren, die nicht im Zustand *terminated* (siehe Kapitel 6.1.13 auf Seite 32) ist.

ERROR_RECEIVEDILLEGALVALUE
: Es wurde ein Wert gesendet, den der Server in diesem Kontext nicht verwerten kann (z. B. ein Wert außerhalb des erlaubten Bereichs, . . .).

ERROR_ILLEGALLEGITIMATION
: Es wurde versucht, eine nicht-existente *Legitimation* einzutragen.

ERROR_TOOFEWVOTES
: Es wurde versucht, eine *Lecture* zu evaluieren, für die zu wenig Leute abgestimmt haben.

ERROR_SQLSELECTFAILED
: Eine SQL-SELECT-Anfrage an die Datenbank ist fehlgeschlagen.

ERROR_SQLINSERTFAILED
: Eine SQL-INSERT-Anweisung an die Datenbank ist fehlgeschlagen.

ERROR_SQLUPDATEFAILED
: Eine SQL-UPDATE-Anweisung an die Datenbank ist fehlgeschlagen.

ERROR_SQLDELETEFAILED
: Eine SQL-DELETE-Anweisung an die Datenbank ist fehlgeschlagen.

ERROR_INTERNAL
: Sonstiger interner Fehler (z. Z. nicht benutzt).

string ErrorDescription
: Genaue Beschreibung des Fehlers.

7. Kryptographie

Ziel der Kryptographie in SPuLe ist es, einen abhörsicheren Kanal, der genau die Daten überträgt, die gesendet wurden, zwischen einem Client und dem Server herzustellen. Ferner muss sich der Client sicher sein, mit dem echten Server und nicht mit einem anderen Programm zu kommunizieren. Um dieses zu erreichen, sind zunächst einige Hilfsalgorithmen vonnöten um letztlich den Kanal implementieren zu können.

7.1. Hilfsalgorithmen

7.1.1. Das Modul SHA256

In der Datei `common/SHA256.java` ist der *SHA256* Algorithmus [3] implementiert. Das *SHA256* Verfahren ist ein kryptographisches Hashverfahren. Die Implementierung ist von `http://www.cryptix.org/` und wurde von uns lediglich vereinfacht. Das *SHA256*-Modul enthält die Routine `hash`, welche ein 64`Byte` großes Datenpaket in einen 32`Byte` großen *SHA256*-Hashwert umrechnet.

7.1.2. Das Modul randomsha256

Die Datei `common/randomsha256` implementiert einen Zufallszahlengenerator. Er besitzt den Zustand

$$z_i \in 0,\ldots,255 \quad i = 1,\ldots,64$$

sowie den aktuellen Auslesezeiger $r \in 1,\ldots,64$. Zur Initialisierung wird ein 64`Byte` langer Initialisierungsvektor übergeben, welcher direkt in den Status z_i kopiert wird. Der Auslesezeiger r wird auf 32 gesetzt.

Um ein Zufallsbyte zu erzeugen, wird zunächst überprüft, ob $r = 32$ ist. Ist dies der Fall, wird `rehash` aufgerufen und $r = 0$ gesetzt. z_r ist der generierte Zufallswert. Nun wird noch r um 2 erhöht. Damit ist die Erzeugung eines Zufallswertes abgeschlossen.

Zuerst berechnet die `rehash` Routine $d = \texttt{SHA256}(z)$. Danach werden die Werte aus z_a (mit $a = 1,\ldots,32$) sequentiell nach z_b (mit $b = 33,\ldots,64$) kopiert. Nun wird $z_a := d_a$ (mit $a = 1,\ldots,32$) gesetzt.

Bleibt noch die Frage zu klären, weshalb r um 2 und nicht um 1 erhöht wird. Angenommen, r würde um 1 erhöht, so könnte man 32`Bytes` auslesen, danach würden die gelesenen Bytes in die höherwertige Hälfte des Zustandes kopiert. Die niederwertige Hälfte würde nun auf neue Werte gesetzt. Diese 32 Werte kann man auslesen. Nun hat man 64 Werte,

die den aktuellen Zustand widerspiegeln. Da man nun den kompletten Zustand des Generators kennt, kann man die folgenden Werte voraussagen. Durch eine Erhöhung von *r* um 2 gelangt man jedoch niemals an den Zustand und kann somit die Werte nicht voraussagen.

7.1.3. Das Modul linux_genrandom

Hier wird ein Zufallszahlengenerator implementiert, der nichts weiter tut, als das Linux eigene Zufallszahlengeneratorgerät auszulesen und die Werte zurückzugeben. Die Beschreibung dieses Gerätes kann der Linux Kernel-Dokumentation entnommen werden. Es ist einem selbst programmierten Algorithmus vorzuziehen, da es einer breiteren Überwachung durch mehr Programmierer unterliegt und seinen Zustand permanent von systemweiten Ereignissen wie Tastendrücken aktualisiert wird. Der Grund, überhaupt einen eigenen Zufallszahlengenerator programmiert zu haben (das `randomsha256`-Modul), liegt darin, dass in einer Java Sandbox, wie sie zum Beispiel beim Programmstart aus einem Browser heraus vorzufinden ist, kein Zugriff auf betriebssystemspezifische Funktionalität besteht.

7.1.4. Das Modul Rijndael_Algorithm

In der Datei `common/Rijndael_Algorithm.java` ist der *Rijndael* Algorithmus [4] implementiert. *Rijndael* ist ein vom AES [7] vorgeschlagener symmetrischer Verschlüsselungsstandard, welcher das gealterte *DES* und *Triple-DES* ablösen soll. Die hier verwendete Implementierung stammt von `http://www.cryptix.org/` und wurde vereinfacht. Das `Rijndael_Algorithm`-Modul implementiert den Algorithmus mit 32`Byte` breiten Schlüsseln und 32`Byte` großen Datenblöcken, welche jeweils zu 32`Byte` großen Kryptoblöcken verschlüsselt werden.

7.1.5. Das Modul bigintcoder

Dieses Modul ist in der Datei `common/bigintcoder.java` implementiert. Es dient zum Konvertieren von Java BigInteger Datentypen in Byte Arrays und zurück. Zusätzlich wird in die BigInteger Darstellung ungenutzte Entropie eingefügt.

Die Routine um ein Byte Array in ein BigInteger zu konvertieren, erhält als Parameter das zu konvertierende ByteArray und die Anzahl Bits, die die Maximallänge der zu generierenden Zahl bestimmen. Es wird mit dem BigInteger 0 begonnen. Um ein zusätzliches Byte in den BigInteger zu kodieren, wird dieser zuerst mit 256 multipliziert und danach das Byte addiert. Auf diese Weise wird zunächst der überflüssige Platz (entstehend durch die Maximalzahl an Bits) mit Entropie gefüllt, dann wird das ByteArray Byte für Byte kodiert und danach die Länge des Byte Arrays.

Die Routine zur Decodierung eines BigIntegers erhält als Parameter lediglich einen BigInteger. Um ein Byte nach dem anderen aus dem BigInteger zu decodieren, wird dieser `mod` 256 gerechnet und das Ergebnis ist das gewünschte Byte. Der BigInteger selbst wird nun durch 256 dividiert und der Divisionsrest vergessen. Auf diese Weise wird zunächst die Länge des ByteArrays decodiert. Ist diese negativ oder größer als 50000, so wird ein

Fehler generiert. Ansonsten werden die Werte des ByteArrays rückwärts nacheinander decodiert. Die jetzt noch lesbaren Entropiewerte werden ignoriert. Damit ist die Decodierung abgeschlossen.

7.2. Implementierung des Kanals

Der Kanal ist asymmetrisch, d. h. es gibt einen Server und einen Client. Zunächst baut der Client eine TCP Verbindung zum Server auf. Nun wird zuerst eine Initialisierungsphase durchlaufen und danach schaltet der Kanal in den Kommunikationsmodus. Der Kanal wird in der Datei `common/securestream.java` implementiert. Der Kanal benötigt einen InputStream und einen OutputStream und stellt darauf aufbauend seinerseits wiederum einen InputStream und einen OutputStream zur Verfügung, so dass man ihn leicht in bestehende Kommunikationsmodule einbauen kann.

7.2.1. Initialisierung

Es wird ein RSA Schlüsseltripel n, e, s benötigt. e ist hierbei der geheime Schlüssel. Das Tripel muss groß genug gewählt sein, damit es möglich ist, 1000`bit` Zahlen zu verschlüsseln.

Der Server enthält hart einkodiert die beiden Konstanten

```
server.rsa.n  server.rsa.e
```

Der Client enthält hart einkodiert die beiden Konstanten

```
server.rsa.n  server.rsa.s
```

Hierbei sind n und s öffentlich bekannt. e jedoch ist ein Geheimnis, welches nur der Server kennt. Aufgrund dieses Geheimnisses kann sich der Server gegenüber einem Client als authentisch verifizieren. Bei der Installation des Systems legt der *Trust* ein RSA Schlüsseltripel fest, welches per Zufall generiert wird. Somit ist der Server weltweit einzigartig und kann mittels seines Geheimnisses eindeutig ausfindig gemacht werden.

Im Folgenden wird das ablaufende Protokoll erläutert. Es soll mehrere Dinge gewährleisten:

- Der Client ist sich sicher, dass er mit dem richtigen Server kommuniziert und nicht mit jemand anderem.
- Der Angreifer kann nicht die übertragenen Daten abhören (Die Tatsache, dass Daten übertragen werden sowie eine Obergrenze der übertragenen Datenmenge kann jedoch ermittelt werden).
- Kein Angreifer kann Änderungen am Datenstrom vornehmen.

Es läuft folgendes Protokoll ab:
Der Client setzt seinen symmetrischen Schlüssel auf einen Zufallswert:

$$byte[32]client.S = randomdata$$

Er konvertiert den Schlüssel in eine BigInteger Zahl mittels dem `bigintcoder`-Modul:

$$\texttt{BigInteger client.a} = \texttt{bloat\_to\_1000\_bits\_using\_random}(\texttt{client.S})$$

Danach verschlüsselt er `client.a` mit dem `RSA`-Modul und schickt es an den Server:

$$\texttt{BigInteger client.b} = \texttt{rsa\_encode}(\texttt{client.rsa.n}, \texttt{client.rsa.s}, \texttt{client.a})$$
$$\texttt{send client.b}$$

Jetzt empfängt der Server den Wert

$$\texttt{receive BigInteger server.b}$$

und decodiert die Nachricht mit dem `RSA`-Modul, wodurch er den vom Client verschlüsselten Wert zurück erhält:

$$\texttt{BigInteger server.a} = \texttt{rsa\_decode}(\texttt{server.rsa.n}, \texttt{server.rsa.e}, \texttt{server.b})$$

Er extrahiert nun den symmetrischen Schlüssel mittels dem `bigintcoder`-Modul:

$$\texttt{Byte}[32]\texttt{server.S} = \texttt{unbloat}(\texttt{server.a})$$

Nun berechnet er einen 32`Byte` großen Zufallswert

$$\texttt{byte}[32]\texttt{server.Rklein} = \texttt{random data}$$

und verschlüsselt ihn mittels dem `Rijndael`-Modul:

$$\texttt{byte}[32]\texttt{server.m}_2 = \texttt{rijndael\_encrypt\_32\_blocksize}(\texttt{server.Rklein}, \texttt{server.S})$$

Das Ergebnis wird an den Client gesendet:

$$\texttt{send server.m}_2$$

Nun berechnet der Server eine Hilfsvariable durch Anhängen von Nullen

$$\texttt{byte}[64]\texttt{server.R} = \texttt{server.Rklein}[0\ldots31] + 0[32\ldots63]$$

und bildet daraus den Hashwert mit dem `SHA256`-Modul:

$$\texttt{byte}[32]\texttt{server.hR} = \texttt{sha256\_hash\_64\_to\_32}(\texttt{server.R})$$

Dieser Hashwert wird dann symmetrisch verschlüsselt und an den Client geschickt:

$$\texttt{byte}[32]\texttt{server.m}_3 = \texttt{rijndael\_encrypt\_32\_blocksize}(\texttt{server.hR}, \texttt{server.S})$$
$$\texttt{send server.m}_3$$

Der Client empfängt zunächst die zwei Nachrichten

$$\texttt{receive byte}[32]\texttt{client.m}_2$$
$$\texttt{receive byte}[32]\texttt{client.m}_3$$

und entschlüsselt sie:

$$\texttt{byte}[32]\texttt{client.Rklein} =$$
$$\quad \texttt{rijndael\_decrypt\_32\_blocksize}(\texttt{client.m}_2, \texttt{client.S})$$
$$\texttt{byte}[32]\texttt{client.hR} =$$
$$\quad \texttt{rijndael\_decrypt\_32\_blocksize}(\texttt{client.m}_3, \texttt{client.S})$$

Der Client berechnet nun eine Hilfsvariable durch Anhängen von Nullen

$$\texttt{byte}[64]\texttt{client.R} = \texttt{client.Rklein}[0 \ldots 31] + 0[32 \ldots 63]$$

und bildet daraus den Hashwert mit dem `SHA256`-Modul:

$$\texttt{byte}[32]\texttt{client.myhR} = \texttt{sha256\_hash\_64\_to\_32}(\texttt{client.R})$$

Er überprüft nun, ob

$$\texttt{client.myhR} = \texttt{client.hR}$$

gilt. Falls ja, ist das Verfahren erfolgreich abgeschlossen und die Gleichheit der symmetrischen Schlüssel $\texttt{server.S} = \texttt{client.S}$ ist gewährleistet.

7.2.2. Kommunikationsmodus

Im Kommunikationsmodus ist keine Asymmetrie des Kanals mehr erkennbar. Beide Seiten können sowohl senden als auch empfangen. Daher wird im Folgenden eine beliebige Seite als Sender und die andere als Empfänger bezeichnet.

Der Sender hat einen Datenblock $\texttt{byte}[50]\texttt{sender.data}$, welcher übertragen werden soll. Der Füllgrad des Blockes ist in der Variablen `sender.pos` gespeichert und bewegt sich im Bereich $1 \ldots 50$. Will der Sender also Daten schicken, welche kleiner als ein Block sind, so wird trotzdem ein kompletter Block übertragen. Für den Fall, dass mehr Daten übertragen werden sollen als in einen Block passen, werden mehrere Blöcke gesendet. Als erstes wird der unbenutzte Teil des zu sendenden Blockes mit Zufallswerten gefüllt. Dann teilt der Server den zu sendenden Datenblock `sender.data` in zwei Blöcke auf, wobei der erste noch den Füllgrad enthält. Beide Blöcke sind mit Zufallswerten aufgefüllt.

$$\texttt{byte}[32]\texttt{sender.d}_0\texttt{r}_0 = \texttt{sender.data}[0 \ldots 24] + \texttt{byte}(\texttt{sender.pos}) + 6\texttt{byte random}$$
$$\texttt{byte}[32]\texttt{sender.d}_1\texttt{r}_1 = \texttt{sender.data}[25 \ldots 49] + 7\texttt{bytes random}$$

Die beiden Blöcke werden nun mit dem `Rijndael`-Modul verschlüsselt und danach gesendet:

$$\texttt{byte}[32]\texttt{sender.c}_0 =$$
$$\quad \texttt{rijndael\_encrypt\_32\_blocksize}(\texttt{sender.d}_0\texttt{r}_0, \texttt{server/client.S})$$
$$\texttt{byte}[32]\texttt{sender.c}_1 =$$
$$\quad \texttt{rijndael\_encrypt\_32\_blocksize}(\texttt{sender.d}_1\texttt{r}_1, \texttt{server/client.S})$$
$$\texttt{send sender.c}_0$$
$$\texttt{send sender.c}_1$$

Jetzt wird noch die Konkatenation aus $\texttt{sender.d}_0\texttt{r}_0$ und $\texttt{sender.d}_1\texttt{r}_1$ mit dem `SHA256`-Modul gehasht und versendet:

$$\texttt{byte}[64]\texttt{sender.d}_0\texttt{r}_0\texttt{d}_1\texttt{r}_1 = \texttt{sender.d}_0\texttt{r}_0[0\ldots31] + \texttt{sender.d}_1\texttt{r}_1[32\ldots63]$$
$$\texttt{byte}[32]\texttt{sender.hash} = \texttt{sha256\_hash\_64\_to\_32}(\texttt{sender.d}_0\texttt{r}_0\texttt{d}_1\texttt{r}_1)$$
$$\texttt{send sender.hash}$$

Auf Empfängerseite werden zunächst die zwei verschlüsselten Blöcke empfangen:

$$\texttt{receivebyte}[32]\texttt{recver.c}_0$$
$$\texttt{receivebyte}[32]\texttt{recver.c}_1$$

Sie werden dann mit dem `Rijndael`-Modul entschlüsselt:

$$\texttt{byte}[32]\texttt{recver.d}_0\texttt{r}_0 =$$
$$\quad \texttt{rijndael\_decrypt\_32\_blocksize}(\texttt{recver.c}_0, \texttt{server/client.S})$$
$$\texttt{byte}[32]\texttt{recver.d}_1\texttt{r}_1 =$$
$$\quad \texttt{rijndael\_decrypt\_32\_blocksize}(\texttt{recver.c}_1, \texttt{server/client.S})$$

Jetzt wird noch die Konkatenation aus $\texttt{receiver.d}_0\texttt{r}_0$ und $\texttt{receiver.d}_1\texttt{r}_1$ und ihr Hashwert mit dem `SHA256`-Modul berechnet:

$$\texttt{byte}[64]\texttt{recver.d}_0\texttt{r}_0\texttt{d}_1\texttt{r}_1 = \texttt{recver.d}_0\texttt{r}_0[0\ldots31] + \texttt{recver.d}_1\texttt{r}_1[32\ldots63]$$
$$\texttt{byte}[32]\texttt{recver.myhash} = \texttt{sha256\_hash\_64\_to\_32}(\texttt{recver.d}_0\texttt{r}_0\texttt{d}_1\texttt{r}_1)$$

Jetzt wird der gesendete Hashwert empfangen

$$\texttt{receive byte}[32]\texttt{recver.hash}$$

und mit dem berechneten verglichen:

$$\texttt{recver.hash} = \texttt{recver.myhash}$$

Ist der Vergleich negativ ausgefallen, wurde damit ein Betrugsversuch oder Übertragungsfehler entdeckt und die Verbindung abgebrochen. Nun muss noch der Füllgrad extrahiert werden und der Datenblock aus den zwei empfangenen Blöcken zusammengesetzt werden:

$$\texttt{int recver.fill} = \texttt{recver.d}_0\texttt{r}_0[25]$$
$$\texttt{byte}[50]\texttt{recver.data} = \texttt{recver.d}_0\texttt{r}_0[0\ldots24] + \texttt{recver.d}_1\texttt{r}_1[25\ldots49]$$

Es wird überprüft, dass $1 \leq \texttt{recver.fill} \leq 50$ ist. Falls nicht, wird mit Fehler abgebrochen. Jetzt ist im `recver.data` Feld der ursprünglich gesendete Datenblock enthalten und `recver.fill` enthält den Füllgrad.

8. Sicherheit des Systems

Die Sicherheit von Spule basiert auf der Annahme, dass der *Trust* vertrauenswürdig ist. Der *Trust* besteht hierbei aus den *Admins* und den Menschen, welche UNIX-technisch Rechte auf dem *Trust*-Rechner besitzen. Es wäre technisch möglich, gewisse Teile des Spule Systems weiter zu sichern, so dass sie nicht auf dem Vertrauen auf den *Trust* basieren, jedoch kann der *Trust* ohnehin jederzeit alles tun, was er möchte, denn er hat das Recht, Benutzer zu erzeugen, die gar nicht existieren. Da hieraus folgt, dass man dem *Trust* vertrauen muss, wurde beim Spule Design weiterhin stets unter der Annahme eines vertrauenswürdigen *Trusts* gearbeitet und zum Beispiel die Datenbank mit den Wahlergebnissen unverschlüsselt abgespeichert. Hätte man die Wahlergebnisse theoretisch perfekt verschlüsselt, könnte der *Trust* einen modifizierten Server aufsetzen, welcher wiederum alles abhört. Durch die Fähigkeit des *Trusts*, den Server zu manipulieren, hat er immer irgendwelche Manipulationsmöglichkeiten. Nach dem Alles-oder-Nichts-Prinzip ist daher ein uneingeschränktes Vertrauen in den *Trust* sinnvoll. Es würde sich sonst um False Security handeln.

Des Weiteren wird angenommen, dass der Benutzer, sei es *Listener*, *Organizer* oder *Admin*, auf einem vertrauenswürdigen System arbeitet und den vom *Trust* compilierten Code ausführt und keinen anderen. Niemand kann die Maschine des Benutzers direkt abhören, sei es durch Mitschneiden der Tastendrücke oder Auslesen des Speicherinhaltes.

Das Sicherheitsmodell von Spule geht also von einer in sich sicheren Insel, dem *Trust*, und beliebig vielen anderen, in sich sicheren Inseln, den *Benutzern* aus. Dazwischen befinden sich Netzwerkverbindungen. Diese dürfen abgehört und manipuliert werden. Spule bleibt trotzdem sicher.

8.1. Der Kanal

Nochmals zur Wiederholung die Bedingungen die der Kanal erfüllen muss:

1. Der Client ist sich sicher, dass er mit dem richtigen Server kommuniziert und nicht mit jemand anderem.
2. Der Angreifer kann nicht die übertragenen Daten abhören (Die Tatsache, dass Daten Übertragen werden sowie eine Obergrenze der übertragenen Datenmenge kann jedoch ermittelt werden).
3. Kein Angreifer kann Änderungen am Datenstrom vornehmen.

Aus Punkt 1 folgt, dass der Server ein Geheimnis besitzt, welches niemand sonst auf der Welt besitzt. Ansonsten könnte dieser Jemand sich auch als Server ausgeben. Dieses Geheimnis ist `rsa.e`. Die Nutzdatenübertragung über den Kanal soll letztlich durch ein

symmetrisches Krypto-Verfahren abgewickelt werden, da dieses effizienter ist als ein asymmetrisches Verfahren. Ein solches symmetrisches Verfahren jedoch benötigt einen Key, der beiden Kommunikationspartnern bekannt ist und niemandem sonst. Daraus folgt direkt, dass der Key, falls es sich um mehr als 2 Personen handelt, bei jeder Verbindung neu erzeugt werden muss. Der Server müsste sonst pro Client einen eigenen Key haben. Dann wären aber die Clients alle unterschiedlich, was verhindert werden soll. Daher wird beim Aufbau eines Kanals stets ein neuer symmetrischer Schlüssel generiert. Dies ist die Aufgabe der Initialisierungsstufe des Kanals (siehe Kapitel 7.2.1 auf Seite 55). Der Client erzeugt eine Zufallszahl (welche als symmetrischer Schlüssel verwendet werden wird) und schickt sie RSA-verschlüsselt an den Server. Ein Angreifer kann diesen Wert nicht abhören, da nur der Server das zur Entschlüsselung notwendige Geheimnis `rsa.e` kennt. Der Angreifer könnte jedoch einen anderen Wert als den vom Client erzeugten an den Server schicken. Um dies zu verhindern, wird nach der Übertragung des Wertes eine Kontrolle durchgeführt. Hierfür erzeugt der Server einen Zufallswert und den Hashwert dieses Zufallswertes. Die beiden Werte werden mit dem ausgetauschten symmetrischen Schlüssel verschlüsselt und an den Client geschickt. Nur wer den symmetrischen Schlüssel kennt, ist in der Lage, ein solches Paar aus einem Wert und dessen Hashwert verschlüsselt an den Client zu senden. Aufgrund des `rsa.e` Geheimnisses ist nur der Server in der Lage, den symmetrischen Schlüssel zu besitzen. Daraus folgt, dass sich der Client nun sicher ist, dass er dem einzigen und echten Server seinen symmetrischen Schlüssel korrekt übermittelt hat und ferner keine andere Person diesen Schlüssel besitzen kann.

Damit sind die Voraussetzungen für den Kommunikationsmodus des Kanals (siehe Kapitel 7.2.2 auf Seite 57) erfüllt. Dadurch, dass alle zu übertragenden Daten mit dem Rijndael-Verfahren verschlüsselt sind und hierfür der zuvor ausgetauschte symmetrische Schlüssel verwendet wird, kann niemand den Klartext abhören. Drei Dinge kann ein Angreifer jedoch nach wie vor tun:

1. Erkennen identischer Daten. Der Angreifer erkennt an identischen Datenblöcken, dass die identischen Daten übermittelt wurden. Er weiß zwar nicht, welche dies sind, aber er weiß, dass sie mehrfach gesendet wurden. Um das zu verhindert, werden zu jedem Datenblock Zufallswerte hinzugefügt. Die Wahrscheinlichkeit, dass zweimal derselbe Datenblock übermittelt wird, ist dadurch extrem gering geworden. Die Restwahrscheinlichkeit ist klein genug, um ignoriert werden zu können. Aus praktischen Gesichtspunkten ist damit der Angreifer nicht mehr in der Lage, mehrfach gesendete Daten zu erkennen.

2. Übermitteln defekter Daten. Der Angreifer kann einfach Datenmüll in den Strom einbauen. Da das Rijndael-Verfahren keine Fehlerkorrektur implementiert und die Mächtigkeit der Codemenge gleichgroß der Mächtigkeit der Klartextmenge ist, können die Kommunikationspartner dies nicht erkennen. Daher wird zu jedem Datenblock noch der Hashwert des Klartextes und der Zufallswerte aus Punkt 1 unverschlüsselt mit übertragen. Der Angreifer kann nun zwar immer noch Datenmüll einfügen, jedoch erkennen die Kommunikationspartner diesen Zustand und können nach Ausgabe einer Fehlermeldung abbrechen.

3. Erkennung der Übertragung von Daten. Ein Angreifer erkennt, dass Daten übermittelt werden und wann dies geschieht. Anhand dessen kann er unter Umständen Rückschlüsse auf die Tätigkeiten des Benutzers des Clients ziehen. Er weiß ja, dass zuerst das Login übertragen wird, und aufgrund statistischer Erhebungen kann er wissen, welche Pausen zwischen Paketen auf welches Klickverhalten in welchen Menüs schließen lassen. Um dies zu verhindern, wäre ein kontinuierlicher Datenstrom nötig, welcher ununterbrochen sendet. Wenn keine Nutzdaten vorhanden sind, würden nutzlose Zufallswerte gesendet. Wenn mehr Nutzdaten auf ihre Übertragung warten, als die konstante Datenstromgeschwindigkeit erlaubt, so würde die Übertragung künstlich verzögert werden. Der Angreifer hätte dann keinerlei Anhaltspunkte mehr, außer der Gesamtdauer der Sitzung. Diese könnte man auch noch durch eine konstante Laufzeit des Clients abschaffen. Das hieße, der Benutzer muss immer 10 Minuten am Client sitzen. Wenn er schneller ist, muss er bis zu den 10 Minuten warten. Braucht er länger, so muss er trotzdem abbrechen. Dann könnte man in einem weiteren Schritt noch die Uhrzeiten des Clientlaufes beschränken auf volle Stunden. Es sollte jedoch klar sein, dass die Einschränkungen irgendwann einfach zu groß werden. Gegen die hier aufgeführten Angriffsmöglichkeiten bietet Spule keinen Schutz.

8.2. Sicherheit bezüglich der Bedingungen an eine Wahl

Wie bereits früher erwähnt, muss eine Wahl folgende Bedingungen erfüllen:

Wahlberechtigung *Nur berechtigte Wähler können Stimmen abgeben.*

Herkömmlich: Jeder kann ein Wahlformular ausfüllen oder das Web-basierte System bedienen, unabhängig davon, ob er die Vorlesung besucht hat, also wahlberechtigt ist oder nicht.

Mit Spule: Jeder eingeschriebene Student kann sich beim *Trust* eine Kennung zur Authentifizierung geben lassen. Nur mit dieser kann er sich im System einloggen. Um für eine Vorlesung abstimmen zu können, muss er *Legitimations* der jeweiligen Vorlesung besitzen. Somit wird sichergestellt, dass niemand „unbefugterweise“ für (oder gegen) eine Vorlesung abstimmt.

Einmaligkeit *Jeder Wähler kann nur einmal wählen.*

Herkömmlich: Da man seine Identität beim Wählen nicht preisgeben muss, ist es möglich, beliebig oft zu wählen.

Mit Spule: In der Datenbank, die im *Trust* läuft, werden die Antworten einer Person auf die Fragen einer Vorlesung gespeichert. Dort kann das Tupel (Person_ID, Lecture_Name) nur einmal vorkommen. Daraus ergibt sich automatisch, dass ein und dieselbe Person nur eine Stimme pro Vorlesung hat.

Wahlgeheimnis *Niemand kann feststellen, wie ein anderer gestimmt hat.*

Herkömmlich: Diese Eigenschaft ist auch bisher bereits ziemlich gut gewährleistet. Man könnte durch Analyse der Fingerabdrücke auf dem Papier oder durch Abhören des Netzwerkverkehrs jedoch mit geringem Aufwand die Stimme eines Wählers ermitteln.

Mit SPuLe: Da Client und Server verschlüsselt kommunizieren, kann keine dritte Person den Nachrichtenverkehr abhören. Die Tatsache, dass der Server im *Trust* integriert ist, garantiert auch, dass vom Server aus die Daten nicht in falsche Hände gelangen können.

Fälschungssicherheit *Niemand kann unbemerkt die Stimme eines anderen ändern.*

Herkömmlich: In der papierlichen Variante kann mit entsprechenden klassischen Mitteln das Wahlpapier manipuliert werden. Sehr einfach ist es zum Beispiel, ein zusätzliches Kreuz zu setzen. Und in der netzbasierten Varianten kann ein Dritter mit entsprechendem Zugang zur Netzwerkhardware das unverschlüsselte Protokoll beliebig manipulieren.

Mit SPuLe: Da Client und Server verschlüsselt kommunizieren, kann keine dritte Person den Nachrichtenverkehr verändern. Die Tatsache, dass der Server im *Trust* integriert ist, garantiert auch, dass die Daten auf dem Server vor Veränderungen sicher sind.

Verifizierbarkeit *Jeder Wähler kann sich überzeugen, dass seine Stimme korrekt gezählt wurde.*

Herkömmlich: Es gibt keine Möglichkeit für den Wähler, das tatsächliche Auszählen seiner papierlichen Stimme zu verifizieren. Ebenso verhält es sich bei der netzbasierten Variante. Die Verifikation beruht darauf, dass dem Transport des Papiers bzw. des Web-Formulars vertraut wird sowie den für das Auszählen verantwortlichen Personen.

Mit SPuLe: Das Vertrauen, welches dem *Trust* entgegengebracht wird, beinhaltet das Vertrauen, dass keine Abstimmungsdaten gelöscht werden. Dass dies nicht automatisch passiert, kann jeder anhand des veröffentlichten Quellcodes von Server und Client nachprüfen.

Hiermit ist gezeigt, dass SPuLe die gestellten Anforderungen in vollem Umfang erfüllt.

Teil II.

Administrationshandbuch

9. Installation

Sie benötigen zur Installation des Trusts einen sicheren Rechner. Bitte installieren Sie:

- GNU/Linux Betriebssystem
- PostgreSQL 7.0.3 mit JDBC 1.0 Treiber (oder neuere Versionen falls kompatibel)
- Sun JDK 1.1.8 (oder Java Runtime Environment)

Falls die PostgreSQL Installationsroutine nach einem „default encoding“ fragt, wählen sie SQL_ASCII. Geben Sie dann ein:

```
su - postgres
(root Passwort eingeben)
createuser -d -A spulepg
```

Editieren Sie die Postgres Konfiguration:

```
PGALLOWTCPIP=yes
in /etc/postmaster/postmaster.init
```

Dies ist nicht dazu da, dass fremde Rechner auf die Datenbank zugreifen dürfen, sondern dient nur dem Zweck, dass der lokale Spule Server auf die Datenbank zugreifen kann.

Legen Sie einen neuen Benutzer in ihrem System an. Nennen Sie ihn *spule*. Kopieren Sie in sein Homeverzeichnis die Spule-Installationsdatei spule-1.0.0.tar.gz. Loggen Sie sich als der *spule* Benutzer ein und entpacken Sie das Archiv:

```
tar xzvf spule-1.0.0.tar.gz
cd spule-1.0.0
```

Editieren Sie nun die Datei `HARDPRINTERSTUFF.java` im Verzeichnis `server/`. Dort können Sie das Druckkommando für einen postscriptfähigen Drucker eingeben sowie in der Papiergröße zwischen „a4“ und „legal“ wählen.

Nun muss ein RSA Key generiert werden, welcher den Trust authentifiziert. Dieser muss dann in den Quellcode sowohl des Trusts als auch des Clients eingebaut werden:

```
erik@torgan:~/spule-1.0.0$ ./compile.sh
erasing many many files
compiling all now... stand by...
done.
```

```
erik@torgan:~/spule-1.0.0$ java rsakeygen.rsakeygen
i will now generate two random numbers using Linux /dev/random
entropy source. you can speed up this process by moving
your mouse and or beating on your keys and or creating
harddisk activity and or many other things
see kernel source if you are curious

*** Creating p
generating a random number ...
... done
*** Creating q
generating a random number ...
... done
*** Calculating phi(n)
*** Creating s
*** Calculating e...
n=14701255084252742480108955332639050776065500595626627571421345838692
09022066912861144671725245159826360262475114664499561394660857206591800
23238171992301067334044223795165612381946259556643540462173599314254960
56410004505703982765527818102332842400921919535622519894333001338739980
00617924757595522857640687817810
e=14064836201987065787636306290469336399458345994294378951402679087302
43512612456163853926432644002986519967374253275206777944013259785472490
44710195228081124581966712318787982539606062247265922027799910615608490
71315763121024229714723124791540868650587871077107856691524935627717320
19981708924231269542910979539630
s=100001027
erik@torgan:~/spule-1.0.0$
```

Damit haben Sie *n*, *e* und *s* für die Public Key Verschlüsselung berechnet. Diese Werte müssen nun in die entsprechenden Dateien kopiert werden. Editieren sie dazu die Datei `server/HARDRSAKEY.java`. Tragen Sie hier den Wert der Variablen *e* ein. Editieren sie `client/HARDRSAKEY.java` und tragen Sie den Wert der Variablen *s* ein. Zum Schluss editieren sie `common/HARDRSAKEY.java` und tragen dort *n* ein.

Editieren Sie nun den Default Servernamen in `client/HARDSERVERADDRESS.java`, das Druckkommando in `client/HARDINITIALPRINTCOMMAND.java` und den Namen ihrer Organisation in `HARDORGANIZATIONNAME.java`.

Kompilieren Sie nun nochmals das Paket:

```
./compile.sh
```

Anschließend muss noch die Datenbank für SPULE initialisiert werden. Hierbei werden die Tabellen, welche im „Technischen Handbuch“ beschrieben werden, angelegt und mit Initialwerten belegt. Zu diesen Initialwerten zählen zwei „Null-Datensätze“ für die Tabellen

Person und *Template* sowie einen *Admin*, mit dem möglich ist, den Rest des Systems interaktiv mit dem Spule Client zu konfigurieren. Das *UniqueSecretPassword* dieses *Admins* ist zu Beginn auf `ABCD-EFGH-1234` festgesetzt. Es empfiehlt sich, dieses möglichst schnell zu ändern, oder besser noch, diese Person zu löschen, nachdem weitere *Admins* mit korrekten Werten angelegt worden sind.

Die Initialisierung der Datenbank erfolgt mit den im Verzeichnis `source/db-tools/` zur Verfügung gestellten Skripten. Das Skript `database.sh` dient als Starter für die anderen Skripte, die nie manuell aufgerufen werden müssen. Um die Tabellen zu erstellen und mit Initialwerten zu belegen, wechseln Sie bitte in das Verzeichnis und geben Folgendes ein:

```
./database.sh init
```

Vergessen Sie nicht, ihr `postgresql.jar` in den CLASSPATH einzutragen. Nun können sie den server mit

```
java server.Server
```

starten. Der Client kann mit

```
java client.Spule
```

gestartet werden. Der Code des Clients kann auf einen Web-Server kopiert werden, indem man `Spule.html` kopiert sowie die Unterverzeichnisse `common/` und `client/` anlegt und kopiert.

Für *Organizers* (welche drucken wollen) kann der Client nicht aus einem Browser installiert werden. Kopieren sie hierfür die Unterverzeichnisse `common/` und `client/` auf den Zielrechner und starten sie den Client wie oben angegeben.

10. Systemablauf

Zunächst müssen alle mit SPuLe in Verbindung kommenden Personen in das System eingetragen werden. Dies gilt sowohl für *Admins*, *Organizers* als auch für *Listeners*.

Um nun eine Vorlesung (*Lecture*) samt ihrer Umfrage durchzuführen, muss zuerst ein *Admin* die *Lecture* eintragen. Dabei muss ihr Name und die Anzahl *Events* festgelegt werden, an denen *Legitimations* ausgeteilt werden sollen. Diese Anzahl darf auch 0 sein, was zur Folge hat, dass jeder wählen darf, unabhängig davon, ob er die Veranstaltung besucht hat oder nicht. Der *Admin* wird dann die *Lecture* editieren und einen *Organizer* für sie eintragen. Nun kann dieser *Organizer* die *Lecture* editieren und den einzelnen *Events* Namen geben. Ferner kann er *Legitimations* erzeugen und drucken. Dies kann er auch später noch tun, wenn die *Lecture* bereits läuft. Nun kann der *Admin* der *Lecture* Fragen hinzufügen, welche der *Organizer* nicht löschen kann. Dies lässt sich besonders elegant mittels dem Cut'n'Paste Mechanismus im *TopNode-Editor* erledigen. Ebenso kann der *Organizer* seine eigenen Fragen in der *Lecture* editieren.

Sobald dies abgeschlossen ist, wird der *Admin* die *Lecture* in den *running* Zustand schalten. Nun können die Fragen nicht mehr manipuliert werden. Die *Organizers* können jetzt ihre Veranstaltungen abhalten und *Legitimations* verteilen. Diese *Legitimations* wiederum können von den *Listeners* gesammelt oder in das SPuLe System eingetragen werden, welches Sie sich merkt. So hat der *Listener* die Wahl, entweder die vielen *Legitimations* zu sammeln und am Ende gesammelt einzugeben oder immer wieder einige einzugeben, so dass man sie nicht sammeln muss. Ferner kann ein *Listener*, beginnend mit der ersten abgegebenen *Legitimation*, stets seine Wahl manipulieren und so zum Beispiel nach jedem gerade besuchten *Event* einen Kommentar abgeben.

Nachdem genügend Zeit seit dem letzten *Event* vergangen ist, so dass alle *Listeners* ausreichend Zeit hatten, um ihre Wahl abzugeben, kann der *Admin* die *Lecture* in den *terminated* Zustand schalten. Jetzt ist es einem *Listener* nicht mehr möglich, seine Wahl einzusehen oder zu ändern. Aber der *Admin* hat nun die Möglichkeit, eine *Evaluation* als LaTeX Dokument zu erstellen und nach eventueller manueller Zensierung auszudrucken. Danach kann die *Lecture* gelöscht werden.

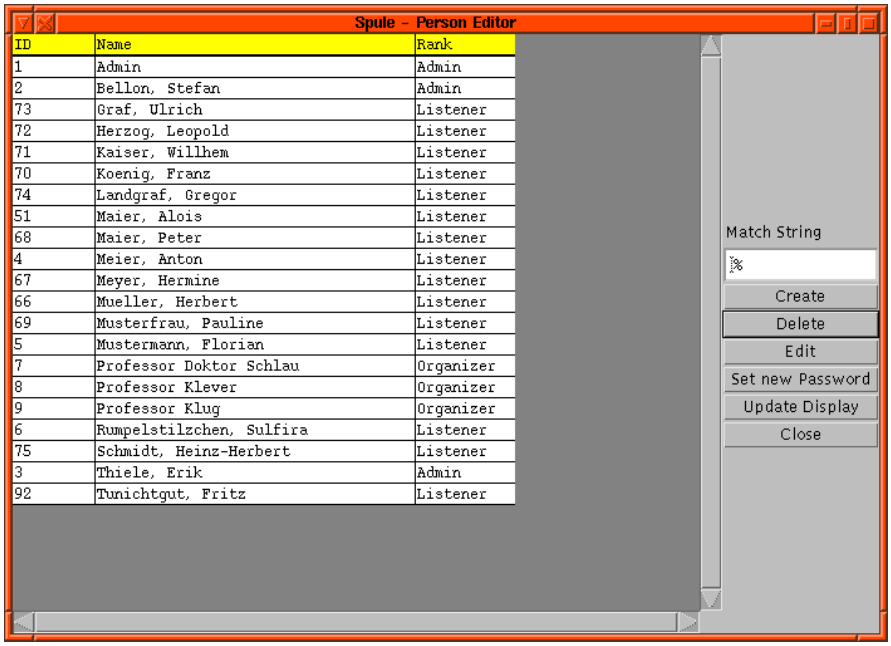
11. Bedienung des Clients

Es wird hier nur die beim *Admin* gegenüber dem *Organizer* hinzugekommene Funktionalität dokumentiert.

Generell unterliegt der *Admin* keinen Zugriffsrechtsbeschränkungen. Das heißt, überall dort, wo an anderen Stellen in der Dokumentation eine Einschränkung vorgenommen wurde, fällt diese nun weg.

11.1. Main Fenster

11.1.1. Edit Persons Fenster



Dieses Fenster zeigt alle dem Spule System bekannten Benutzer an.

Der *Match String* ist eine SQL Suchmaske. Das % Zeichen matcht alles. Das „-“ Zeichen matcht genau ein einzelnes Zeichen. Gibt man hier zum Beispiel „%oo%“ ein, so werden nur diejenigen Personen angezeigt, die zwei „o“ in ihrem Namen haben. Um nur diejenigen Personen anzuzeigen, die ein „s“, gefolgt von einem beliebigen Zeichen und danach ein „t“ in ihrem Namen haben, gibt man „%s_t%“ ein.

Mit *Create* kann man neue Benutzer erstellen. Es öffnet sich ein Fenster zur Eingabe der Daten des neuen Benutzers. Der *Name* sollte einer konsequenten Konvention folgen, da ansonsten mit der Zeit ein Durcheinander entsteht. Es wird empfohlen, Personen in der „Nachname, Vorname“-Notation zu nennen. Professoren sollten ihren Titel vorangestellt bekommen. Dies dient zum leichteren Auffinden der Personen mittels des *Match Strings*. Anhand der ReallifeID kann sich ein Benutzer beim *Trust* authentifizieren. Für Studenten bietet sich hier die Matrikelnummer, für Dozenten und Angestellte der Universität bieten sich Personalausweisnummer o. Ä. an. Man könnte sich sogar vorstellen, den genetischen Fingerabdruck zu speichern. Wichtig ist die einmalige Festlegung einer Konvention, die dann konsequent eingehalten wird. Mit dem *Rank* Feld kann der Status des neuen Nutzers festgelegt werden.

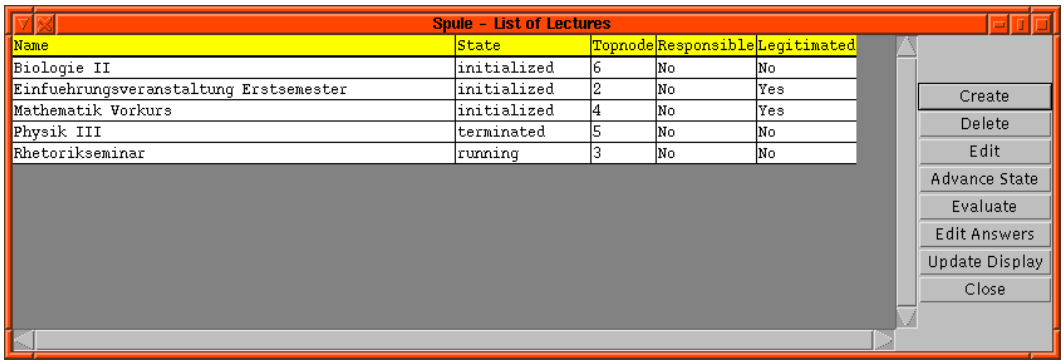
Edit editiert die selektierten Personen, *Delete* löscht sie.

Set New Password gibt den selektierten Personen ein neues zufälliges Passwort und druckt für jede Person ein „Person Unique Secret Data Sheet“ aus. Dieses muss dann an die Personen persönlich übergeben werden, nachdem ihre *ReallifeID*, welche auch auf dem Papier steht, überprüft wurde.

11.1.2. Switch Person ID

Es öffnet sich ein Fenster, in dem der Administrator die *PersonID* der Person angeben kann, deren Identität er annehmen will (Eine Auflistung aller dem System bekannten *PersonIDs* findet sich im *Edit Persons* Fenster). Auf diese Weise kann er Supportleistungen erbringen. Wenn ein Nutzer des Systems ein Problem hat, kann der Administrator ihm die Lösung zeigen, ohne ihn zur Eingabe seines Passworts auffordern zu müssen. Nach dem die *PersonID* gewechselt wurde, entspricht dies einem Neustart des Clients als neuer Benutzer. Einzig der Menüpunkt *Switch Person ID* bleibt erhalten, mit dem der Administrator wieder seine ursprüngliche oder eine andere Identität einnehmen kann.

11.1.3. Edit Lectures / Edit Votes



Hier kann der *Admin* neue *Lectures* mit dem *Create* Knopf erzeugen. Er wird zur Eingabe eines *Namens* und der Anzahl *Events* der neuen *Lecture* aufgefordert. Selektierte *Lectures* können gelöscht werden.

Der Admin darf auch *Lectures* editieren, für die er nicht als Organizer eingetragen ist.

Mit *Advance State* kann der Status einer *Lecture* von *initialized* über *running* auf *terminated* gestellt werden. Bitte beachten Sie, dass dies jeweils ein unumkehrbarer Schritt ist.

Mit *Evaluate* lässt sich eine LaTeX Datei erstellen, welche das Ergebnis der Vorlesungsumfrage repräsentiert. Dies ist nur möglich, wenn die *Lecture* im Status *terminated* ist.

Teil III.

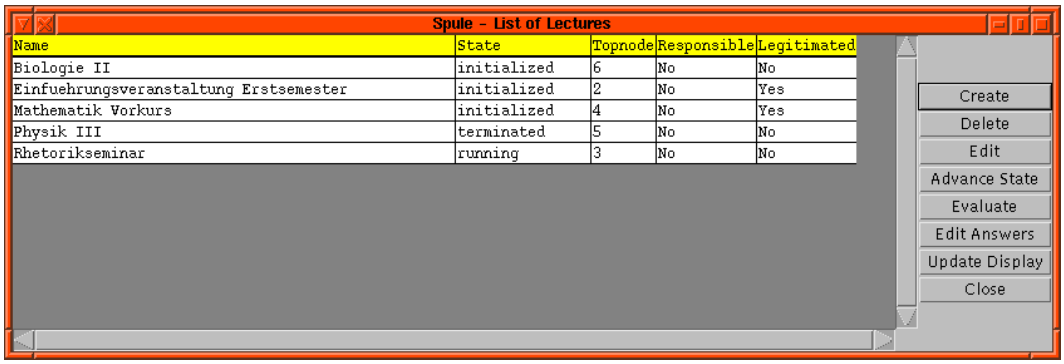
Dozentenhandbuch

12. Bedienung des Clients

Es wird hier nur die beim *Organizer* gegenüber dem *Listener* hinzugekommene Funktionalität dokumentiert.

12.1. Main Fenster

12.1.1. Edit Lectures / Edit Votes



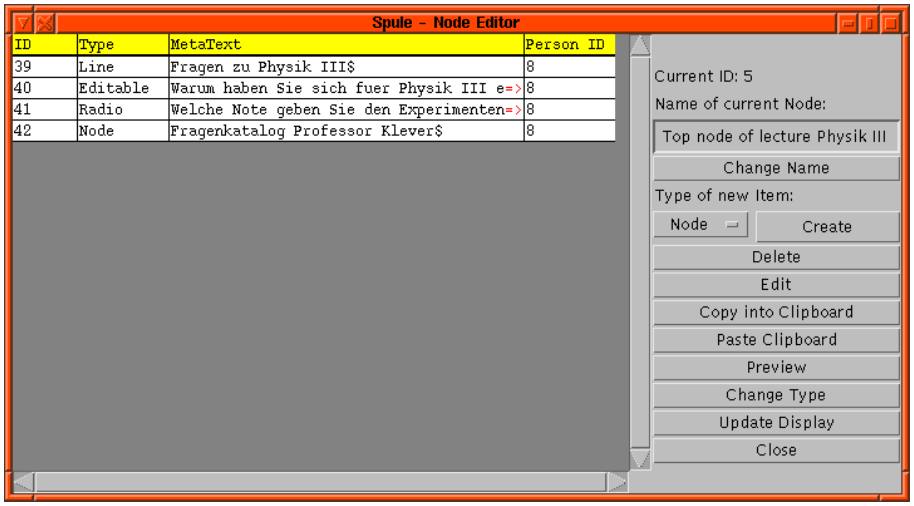
Mit *Edit* kann eine *Lecture* editiert werden. Es öffnet sich ein Fenster. Dieses besteht aus drei Teilen.

Im oberen Teil kann man *Organizers* zu dieser *Lecture* hinzufügen oder entfernen.

Der mittlere Teil dient zum Editieren der Namen der einzelnen *Events* dieser *Lecture*. Mit *Print all* kann man alle *Legitimations* zu den selektierten *Events* ausdrucken. So muss man sie nicht extra im unteren Teil selektieren.

Im unteren Teil kann man *Legitimationen* hinzufügen, löschen oder drucken. Man kann mittels *Undelete* auch gelöschte *Legitimations* wieder herstellen.

12.1.2. Edit Topnodes



Es wird ein *Node-Editor* Fenster geöffnet, welches den *Top Node* darstellt. Der *Node Editor* für das *Top Node* ist gegenüber den *Node Editoren* für andere *Nodes* insofern eingeschränkt, als dass er nur *Nodes* und keine anderen Typen kreieren kann. Im Folgenden wird daher der *Node Editor* für eine allgemeine *Node* beschrieben.

Links erscheint eine Auflistung aller Einträge dieser Node. Auf der rechten Seite wird die *ID* der Node dieses Fensters angezeigt, darunter ihr Name. Mit *Change Name* kann man den Namen ändern.

Darunter kann man den Typ des zu kreierenden Items wählen und durch Druck auf *Create* erzeugen. Das neue Item wird oberhalb der selektierten Zeile eingefügt oder ans Ende der Liste, falls keine Zeile selektiert ist. Mit *Delete* kann ein Item gelöscht und mit *Edit* editiert werden.

Mit der Funktion *Copy into Clipboard* ist es möglich, das selektierte Item in einen Zwischenpuffer zu kopieren. Falls es sich um ein *Node* handelt, wird rekursiv der Inhalt dieses *Nodes* mit abgespeichert. Der Inhalt des Zwischenpuffers bleibt nur während der Sitzung des Clients erhalten. Mittels *Paste Clipboard* wird der Inhalt des Zwischenpuffers vor das selektierte Item oder, falls keines selektiert ist, an das Ende der Liste eingefügt.

Der *Preview* Knopf ermöglicht eine Vorschau für das selektierten Item bzw. das gesamte aktuelle Node, falls nichts selektiert ist.

Change Type ermöglicht das schnelle Wechseln zwischen *Line* und *Label* bzw. zwischen *Radio* und *Choice*.

Teil IV.

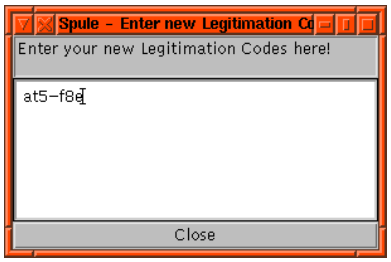
Benutzerhandbuch

13. Bedienung des Clients

13.1. Main Fenster

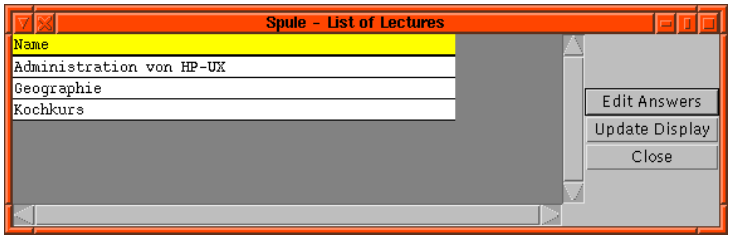
Das Hauptfenster zeigt ihren *Namen*, ihre *ID* und ihren *Rank* an.

13.1.1. Enter new Legitimation Codes

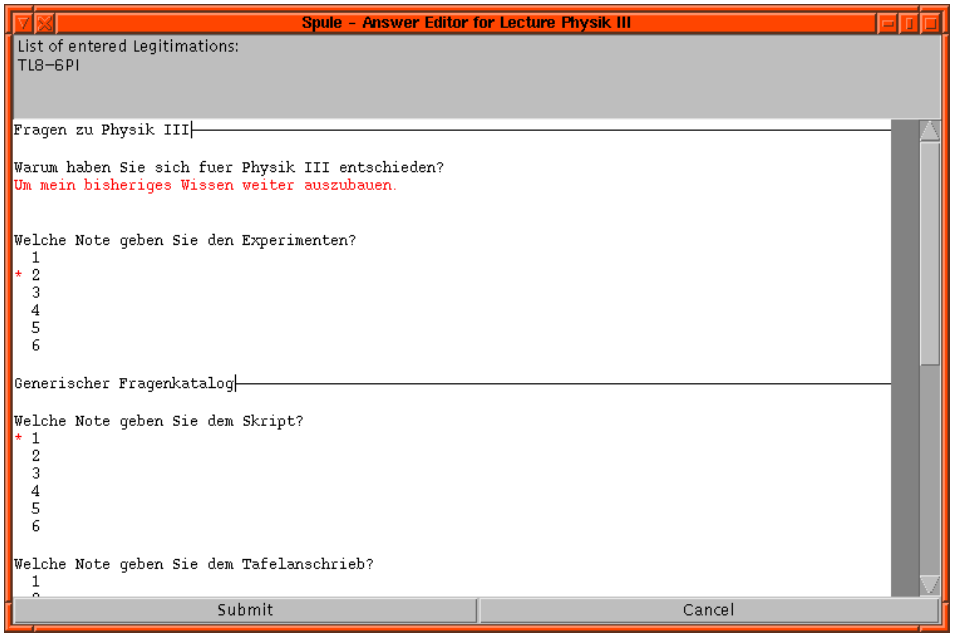


Hier haben Sie die Möglichkeit, ihre gesammelten Legitimation Codes einzugeben. Nach der Eingabe können sie die Zettel wegwerfen.

13.1.2. Edit Votes



Hier können Sie für alle Vorlesungen abstimmen, für die Sie *Legitimation Codes* abgegeben haben.


Spule - Answer Editor for Lecture Physik III
List of entered Legitimations:
TL8-6PI
Fragen zu Physik III
Warum haben Sie sich fuer Physik III entschieden?
Um mein bisheriges Wissen weiter auszubauen.
Welche Note geben Sie den Experimenten?
1
* 2
3
4
5
6
Generischer Fragenkatalog
Welche Note geben Sie dem Skript?
* 1
2
3
4
5
6
Welche Note geben Sie dem Tafelanschrieb?
1
Submit
Cancel

Teil V.

Anhang

A. Studienarbeit-Administrativa

A.1. Arbeitsaufteilung

Die Zuständigkeitsbereiche waren während der Studienarbeit wie folgt aufgeteilt:

Stefan Bellon:
: Datenbank (MySQL [6], PostgreSQL [8], JDBC [10]), Server (Verzeichnis `server/`) sowie Routinen zur Generierung vom LaTeX- [5] und PostScript-Ausgaben [1] im Client (`LaTeXEvaluator.java`, `Evaluator.java`, `Printlegis.java` im Verzeichnis `client/` und `common/StringUtils.java`).

Erik Thiele:
: Client (Großteil im Verzeichnis `client/`), Java-AWT [2, 9] sowie Kryptographie (Großteil im Verzeichnis `common/`).

gemeinsam:
: Design der Nachrichten zwischen Server und Client (`common/Messages.java`).

A.2. Quelltext der Ausarbeitung

Da zur Entwicklung der Studienarbeit die frei verfügbaren Dienste von sourceforge.net genutzt wurden, liegt der Quelltext von SPuLe unter `http://sourceforge.net/projects/spule/` zum Download bereit. Er ist unter die GNU General Public License gestellt, welche im Folgenden abgedruckt ist.

B. The GNU General Public Licence

The following is the text of the GNU General Public Licence, under the terms of which this software is distrubuted.

GNU GENERAL PUBLIC LICENSE
Version 2, June 1991

B.1. Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software—to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by

someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

B.2. Terms and conditions for copying, distribution and modification

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

   Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

   You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

   (a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

(b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

(c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

(a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

(c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

   If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a

whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

   Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and “any later version”, you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. **Because the Program is licensed free of charge, there is no warranty for the Program, to the extent permitted by applicable law. except when otherwise stated in writing the copyright holders and/or other parties provide the program “as is” without warranty of any kind, either expressed or implied, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the Program is with**

you. Should the Program prove defective, you assume the cost of all necessary servicing, repair or correction.

12. **In no event unless required by applicable law or agreed to in writing will any copyright holder, or any other party who may modify and/or redistribute the program as permitted above, be liable to you for damages, including any general, special, incidental or consequential damages arising out of the use or inability to use the program (including but not limited to loss of data or data being rendered inaccurate or losses sustained by you or third parties or a failure of the Program to operate with any other programs), even if such holder or other party has been advised of the possibility of such damages.**

END OF TERMS AND CONDITIONS

B.3. Appendix: How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

```
<one line to give the program's name and a brief idea of what it does.>
Copyright (C) 19yy  <name of author>

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA.
```

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

```
Gnomovision version 69, Copyright (C) 19yy name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type ‘show w’.
This is free software, and you are welcome to redistribute it
under certain conditions; type ‘show c’ for details.
```

The hypothetical commands ‘show w’ and ‘show c’ should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than ‘show w’ and ‘show c’; they could even be mouse-clicks or menu items–whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a “copyright disclaimer” for the program, if necessary. Here is a sample; alter the names:

```
Yoyodyne, Inc., hereby disclaims all copyright interest in the program
‘Gnomovision’ (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice
```

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Glossar

Admin Person, die Bestandteil des Trusts ist. Sie hat volle Rechte im SPuLe-System.

Angreifer Person, die den ordnungsgemäßen Betrieb des Systems in irgendeiner Weise stören oder gar verhindern will.

Answer Antwort einer Person auf eine Frage eines Fragebogens.

Applet Programm, welches in einer Sandbox mit eingeschränkten Rechten innerhalb des Browsers läuft.

Asymmetrischer Kanal Kanal mit unterschiedlichen Enden.

Authentifizierung Prozess des Beweisens, die Person tatsächlich zu sein, die man vorgibt zu sein.

AWT JAVA-Bibliothek zum erstellen und benutzen grafischer Benutzerelemente.

BigInteger JAVA Integer-Datentyp beliebiger Größe.

Bruderknoten Knoten, welcher in der Hierarchie auf gleicher Ebene liegt.

Buffer Overflow Pufferüberlauf, z. B. wenn eine Datenmenge die Größe des vom Programmierer statisch festgelegten Bereichs für die Daten übersteigt.

Choice Template-Typ für ein m-aus-n Auswahlfeld auf dem Fragebogen.

Client Programm, welches Dienste, die von einem Server angeboten werden, wahrnimmt, d. h. Anfragen an ihn stellt.

Clipboard Zwischenablage, in die Knoten abgelegt und aus der sie später wieder herausgeholt werden können.

Codewortraum Menge aller möglichen Codewörter bei gegebener Kodierung.

Dozent Person, die eine Vorlesung hält.

Editable Template-Typ für ein Eingabefeld auf dem Fragebogen.

Entropie Zufallsquelle.

Evaluierung Auswertung einer Vorlesungsumfrage, die veröffentlicht wird (sowohl der Vorgang der Auswertung als auch das Ergebnis).

Event Einzelne Veranstaltung einer Vorlesung.

False Security Irreführender Eindruck von Sicherheit, obwohl überhaupt keine vorhanden ist.

Foreign Key Synonym zu Fremdschlüssel.

Fragebogen Eine Sammlung von Fragen, eine Vorlesung betreffend (siehe auch Web-Formular).

Fragenkatalog Sammlung von Fragen, die in einem Fragebogen verwendet werden können.

Fremdschlüssel Primärattribut einer anderen Tabelle, welches referenziert wird.

Hashverfahren Injektive Funktion.

Hashwert Ergebniswert der Hashfunktion zu einem gegebenen Wert.

Inkonsistenz Fehlerhafter Zustand der Datenbank.

InputStream JAVA Stream, welcher nur lesen kann.

JAVA-Applet Ein in JAVA geschriebenes Applet.

Klartext Unverschlüsselter Text.

Krypto-Verfahren Verfahren zur Verschlüsselung.

Kryptographie Lehre der Verschlüsselung.

Label Template-Typ für eine Bemerkung auf dem Fragebogen.

Lecture In SPuLe synonym zu Vorlesung verwendet.

Legitimation Kennung, welche zum Ausfüllen des zu der entsprechenden Vorlesung gehörenden Fragebogens legitimiert.

Line Template-Typ für eine Trennlinie (optional mit Kommentar) auf dem Fragebogen.

linker Bruder Knoten, welcher in der Hierarchie auf gleicher Ebene liegt und direkt vor dem momentanen Knoten eingeordnet ist.

Listener Person, die zum Wählen berechtigt ist, z. B. jeder Student.

Matrikelnummer Identifikationsnummer, die jeder Student bei seiner Einschreibung erhält und die sich das gesamte Studium über nicht ändert.

Mehrbenutzerbetrieb Betrieb der Datenbank, bei dem mehrere Benutzer gleichzeitig Anfragen an die Datenbank stellen.

Meta-Format Format, in welchem verschiedene für SPuLe relevante Texte in der Datenbank gehalten werden (siehe 3.2 auf Seite 17).

MetaText Format, in welchem verschiedene für SPuLe relevante Texte in der Datenbank gehalten werden (siehe 3.4 auf Seite 19).

Node Template-Typ für einen Knoten, der hierarchisch untergeordnete Templates beinhalten kann.

Null-Datensatz Leerer Datensatz, der referenziert werden kann.

Organizer Person, die für eine Vorlesung verantwortlich ist, meistens der Dozent oder sein Assistent.

OutputStream JAVA Stream, welcher nur schreiben kann.

Primärschlüssel Attribut in einer Datenbank anhand dessen ein Datensatz eindeutig gefunden werden kann.

Primary Key Synonym zu Primärschlüssel.

Radio Template-Typ für ein 1-aus-n Auswahlfeld auf dem Fragebogen.

Rijndael Vom AES verfasster symmetrischer Verschlüsselungsstandard.

RSA Asymmetrisches Verschlüsselungsverfahren.

Sandbox Konzept der Abarbeitung von Programmen in einer sicheren Umgebung mit eingeschränkten Rechten.

Server Programm, welches Dienste zur Verfügung stellt, Anfragen von Clients entgegen nimmt und beantwortet.

SHA256 Kryptographisches Hashverfahren.

Swing Grafische Benutzerumgebung alternativ zu AWT.

Symmetrischer Verschlüsselungsstandard Verschlüsselungsverfahren, das mit nur einem Schlüssel auskommt.

TCP Verbindung Internetverbindung zur streambasierten Datenkommunikation.

Template Element eines Fragenkatalogs.

Template-Typ Typ einer Template, z. B. Node, Label, Line, Editable, Choice oder Radio.

Thread Prozess, welcher zusammen mit anderen Threads auf dem gleichen Speicherbereich abläuft.

Topnode Oberster Knoten einer Vorlesung, d. h. ein Template vom Typ Node.

Transaktion Anfrage an eine Datenbank, die aus mehreren Teilen bestehen kann und entweder ganz oder gar nicht ausgeführt wird.

Transaktionsmodell Datenbankenkonzept, welches sicherstellt, dass Änderungen alle in ihrer Gesamtheit oder gar nicht durchgeführt werden, d. h. Inkonsistenzen treten nicht auf.

Trust Gesamtsystem aus dem SPULE-Server und den SPULE-Admins, dem man uneingeschränkt vertrauen kann und muss.

unique Synonym für ‘einmalig’.

UniqueSecretPassword Geheime Kennung, mit der sich jede Person bei SPULE anmelden muss.

Vaterknoten Knoten, welcher in der Hierarchie eine Ebene höher liegt und dem momentanen Knoten übergeordnet ist.

Vorlesung Eine Reihe von Veranstaltungen, die ein Semester lang dauern.

Vorlesungstitel Name der Vorlesung.

Vorlesungsumfrage Am Semesterende wird für jede Vorlesung eine Umfrage durchgeführt, bei welcher die Qualitäten der Vorlesung, der begleitenden Materialien und der Dozenten festgestellt werden sollen. Eine Vorlesungsumfrage besteht demnach aus verschiedenen Fragen, anhand derer die Bewertung stattfindet.

Web Teil des Internets.

Web-Formular Fragebogen, der über das Web zur Verfügung gestellt wird und ausgefüllt werden kann.

Wurzelknoten Oberster Knoten der Hierarchie.

Literaturverzeichnis

[1] ADOBE SYSTEMS INCORPORATED: *PostScript Language Reference*. Addison-Wesley, Reading, MA, USA, Dritte Auflage, 1999. This new edition defines PostScript Language Level 3. An electronic version of the book is available at the Adobe Web site, and is also included in a CD-ROM attached to the book. http://partners.adobe.com/supportservice/devrelations/PDFS/TN/PLRM.pdf; http://partners.adobe.com/asn/developer/PDFS/TN/PLRM.pdf.

[2] ARNOLD, KEN, JAMES GOSLING und DAVID HOLMES: *The Java Programming Language*. The Java Series. Addison-Wesley, Reading, MA, USA, Dritte Auflage, 2000. http://www.aw.com/cp/javaseries.html; http://www.aw.com/cseng/titles/0-201-63455-4/.

[3] THE CRYPTIX FOUNDATION LIMITED: *Cryptix*. http://www.cryptix.org/.

[4] DAEMEN, J. und V. RIJMEN: *AES proposal: Rijndael*, 1999.

[5] GOOSSENS, MICHEL, FRANK MITTELBACH und ALEXANDER SAMARIN: *Der LaTeX-Begleiter*. Addison-Wesley, Bonn; Paris; Reading, MA, USA, 1994.

[6] MYSQL.ORG: *MySQL*. http://www.mysql.org/.

[7] NATONAL INSTITUTE OF STANDARDS AND TECHNOLOGY: *AES*. http://csrc.nist.gov/encryption/aes/.

[8] POSTGRESQL GLOBAL DEVELOPMENT GROUP: *PostgreSQL*. http://www.postgresql.org/idocs/.

[9] SUN MICROSYSTEMS INCORPORATED: *Java*. http://java.sun.com/.

[10] SUN MICROSYSTEMS INCORPORATED: *Java DataBase Connectivity*. http://java.sun.com/products/jdbc/.

[11] SUN MICROSYSTEMS INCORPORATED: *Javadoc*. http://java.sun.com/j2se/javadoc/index.html.

Erklärung

Hiermit versichern wir, diese Arbeit selbstständig verfasst und nur die angegebenen Quellen benutzt zu haben.

_____

(Stefan Bellon / Erik Thiele)