RWTH AACHEN
RHEINISCH-WESTFÄLISCHE TECHNISCHE HOCHSCHULE AACHEN
Research Group IT-Security
Prof. Dr. Ulrike Meyer

Diplomarbeit

Erhoehung des Schutzes vertraulicher Daten bei entfernter Datenhaltung
-
Systemdesign und Implementierung

(Enhancing Privacy On Remote Storage
-
System Design and Implementation)

eingereicht von

Peter Schmitz
Matrikelnummer: 235822

Gutachterin:	Prof. Dr. Ulrike Meyer
Zweitgutachter:	Prof. Dr. Thomas Seidl
Betreuerin:	Prof. Dr. Ulrike Meyer
Abgegeben am:	04.01.2010

Hiermit versichere ich, daß ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Datum, Unterschrift

Danksagung

Mein ganz besonders herzlicher Dank gilt Professorin Ulrike Meyer. Es war ein Privileg, durch sie direkt betreut zu werden. Ich weiß die vielen Stunden, die sie für mich Zeit hatte, sehr zu schätzen und bin besonders dankbar für ihre Aufgeschlossenheit, meine eigenen Ideen in einer Diplomarbeit umsetzen zu können. Sie gab mir viele gute Ratschläge, die mir halfen, in die richtige Richtung zu denken.
Ein lieber Dank geht an meine Freunde und Familie, die mir während der Diplomarbeit in einer schwierigen Zeit zur Seite standen.
Außerdem mag ich mich bei Mark Schlösser bedanken, der durch seine energievolle Art Vorbereitungen getroffen hat, die ich erst im Nachhinein wirklich zu schätzen wußte.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Derzeit ist zu beobachten, daß viele Anwendungen auf eine Datenverarbeitung und Datenhaltung auf Seiten des Serviceanbieters ausgerichtet sind. (z.B. Google Docs) Die Gründe liegen auf der Hand: Softwarewartungen und Aktualisierungen sind einfacher durchzuführen, der Datenfluß zwischen Haltung und Verarbeitung ist kürzer, damit auch effizienter, und die Schnittstelle zwischen beiden einfacher. Hinzu kommt neuerdings ein Konzept namens Cloud Computing, also das Verlagern der Ausführung von rechenintensiven Anwendungen auf eine entfernte Rechnerinfrastruktur (Cloud). Außerdem lassen sich durch die Kopplung von Datenverarbeitung und -haltung leicht Abrechnungsmodelle für die Nutzung von Software einführen, z.B. Software as a Service (SaaS). Das Endgerät des Benutzers tendiert dazu, als sogenannter Thin Client aufzutreten, also nur noch die Anzeige und Eingabe zu übernehmen. Aber trotz abgesicherter (verschlüsselter) Übertragung zwischen Client und Server besteht ein Vertrauensproblem in den Serviceanbieter. Dieser beteuert durch eine sogenannte Privacy Policy seinen sachgemäßen Umgang mit den ihm anvertrauten Daten. Selten besteht bei den Nutzern ein Bewußtsein für dieses Problem. Verschlüsselung der Daten als Lösung liegt nahe; diverse Ansätze zur Verarbeitung von verschlüsselten Daten beim Serviceanbieter lassen dennoch nicht die nötige komplexe Anwendungslogik zur Serviceerfüllung zu.
Als Lösungsansatz wird in der vorliegenden Arbeit eine Trennung von Datenverarbeiter und Datenhaltung durch einen dazwischenstehenden sogenannten „Türsteher“ vollzogen. Der Türsteher läuft zusammen mit dem Datenverarbeiter auf der vertrauensvollen lokalen Maschine, wohingegen der Datenhalter eine entfernte, zentrale Datenhaltung darstellt. Der Datenverarbeiter wird durch eine Sandbox in seinen Kommunikationsmöglichkeiten auf die mit dem Türsteher eingeschränkt und so wird der Datenverarbeiter-Datenhalter-Datenfluß über den Türsteher gelenkt, der sicherheitsrelevante Anpassungen wie die Verschlüsselung vornimmt. Im Gegensatz zu Closed Source Projekten wie Wuala [wua] wird alles sicherheitsrelevante im Türsteher vereint und von jeglicher Anwendungslogik getrennt. Damit wird das Vertrauen in einen Serviceanbieter auf das Vertrauen in die lokale Ausführung der Türsteherprogrammversion verlagert, welcher durch die Plattformmechanismen der für die Beispielimplementierung verwendeten Programmiersprache Java überprüfbar von Dritten vertraut wird.

2

Problemstellung und Einordnung

2.1 Begrifflichkeiten

Dieses Kapitel beschreibt grundlegende Begrifflichkeiten, auf die in den folgenden Kapitel aufgebaut wird. Es werden einerseits kryptographische Werkzeuge wie die symmetrische Verschlüsselung mittels AES, einen Zufallszahlengenerator, eine sichere Hashfunktion und Begriffe wie Perfect Forward Secrecy eingegangen, als auch kryptographische Protokolle wie TLS und SRP erläutert.

2.1.1 Kryptographische Werkzeuge/Begriffe

Symmetrische Blockverschlüsselung AES

In der modernen Kryptographie wird zur Sicherheit eines Kryotosystems das so genannte *Kerckhoffs'schen Prinzip* zu Grunde gelegt. Dabei beruht die Sicherheit eines Verschlüsselungsverfahrens auf der Geheimhaltung des Schlüssels. Dem *Kerckhoffs'schen Prinzip* wird oft die sogenannte „Security by Obscurity" gegenübergestellt. Dort basiert die Sicherheit der Verschlüsselung auf der Geheimhaltung des Verschlüsselungsalgorithmus.
In der vorliegenden Arbeit werden *symmetrische Kryptosysteme* genutzt, die im Gegensatz zu *asymmetrischen Kryptosystemen* den gleichen Schlüssel zur Ver- und Entschlüsselung verwenden. Die symmetrischen Verfahren lassen sich nach [MvOV01] in *Blockverschlüsselung* und *Stromverschlüsselung* aufteilen. Bei der Stromverschlüsselung wird der Klartext Zeichen für Zeichen verschlüsselt, um den Geheimtext zu erhalten, bzw. entschlüsselt, um den Klartext zu erhalten. Die Blockverschlüsselung arbeitet mit einer festen Blockgröße und ver- bzw. entschlüsselt mehrere Zeichen in einem Schritt.
Der *Advanced Encryption Standard (AES)* ist ein symmetrisches Blockchiffren- Verfahren, das im Jahre 2000 als Nachfolger für *DES* bzw. *3DES* vom National Institute of Standards and Technology (NIST) als Standard bekannt gegeben wurde. AES arbeitet auf Blockgrößen von 128 Bit und besitzt eine variable Schlüssellänge von 128,

192 oder 256 Bit. Jeder Block wird zunächst in eine zweidimensionale Tabelle mit vier Zeilen geschrieben, deren Zellen ein Byte groß sind. Je nach Blockgröße variiert die Anzahl der Spalten von 4 (128 Bits) bis 8 (256 Bits). Anstatt jeden Block einmal mit dem Schlüssel zu verschlüsseln, werden verschiedene Teile des erweiterten Originalschlüssels nacheinander auf den Klartext-Block angewendet, die sog. *Runden-Schlüssel*. Die Anzahl der Runden variiert in Abhängigkeit von der Schlüssellänge zwischen 10 (128 Bits), 12 (192 Bits) und 14 (256 Bits) Runden. Jeder Block wird in mehreren Runden nacheinander bestimmten Transformationen unterzogen: Substitution, Zeilenverschiebungen, Spaltenvermischung und Addition des Rundenschlüssels.

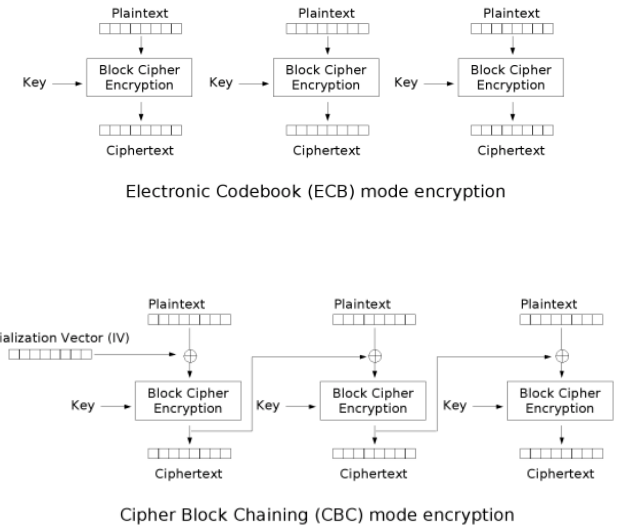


Abbildung 2.1: Verschlüsselung im ECB- und im CBC- Verfahren.

Zur Verarbeitung längerer Nachrichten werden die Daten zunächst in Blöcke unterteilt. Für die weitere Vorgehensweise gibt es verschiedene Betriebsarten. Zu nennen seien der *Electronic Code Book Mode (ECB)* und *Cipher Block Chaining Mode (CBC)*, die ganze Blöcke voraussetzen, so dass der letzte Block mit Fülldaten aufgefüllt werden muss (*Padding*). Die Blöcke werden anschließend nacheinander verschlüsselt.
Das ECB-Verfahren überführt die Klartextblöcke nacheinander und unabhängig voneinander in Geheimtextblöcke, vgl. [Sch05]. Dies bedeutet aber auch gleichzeitig, dass gleiche Klartextblöcke bei gleichem Schlüssel auch immer den gleichen Geheimtextblock ergeben. Durch hinreichend viele Geheimtextblöcke und partielle Annahmen über den Klartext können somit Rückschlüsse auf den geheimen Schlüssel gezogen werden.
Beim häufig eingesetzten CBC-Verfahren wird das Ergebnis des zuvor verschlüsselten Blocks mit dem folgenden Nachrichtenblock per XOR[1] verknüpft (siehe Abbildung 2.1). Für den ersten Block wird ein Initialisierungsvektor verwendet, der zur Generierung beispielsweise einen Zeitstempel oder eine zufällige Zahlenfolge benutzt. Durch die Verkettung der vorhergehend erzeugten Geheimtextblöcke werden Klar-

[1] exklusives Oder

textmuster zerstört, was ein Vorteil gegenüber dem ECB-Verfahren ist, da identische Klartextblöcke unterschiedliche Geheimtexte ergeben.

Zufallszahlengenerator (PRNG)

Als Grundlage eines *Pseudo Random Number Generator*s (PRNG) kann ein Pseudo Random Bit Generator (PRBG) genutzt werden, indem eine hinreichend lange Bitfolge erzeugt wird und diese dann zu einer Zahl konvertiert wird, vgl. [MvOV01], Chapter 5. Daher genügt es sich auf PRBGs zu beschränken:
Ein PRBG ist ein deterministischer Algorithmus, der bei einer gegebenen wirklich zufälligen binäre Sequenz der Länge k, eine binäre Sequenz der Länge $l \gg k$ ausgibt, die zufällig erscheint. Als Eingabe dient ein sogenannter *Seed*. Die Algorithmuseigenschaft deterministisch meint, daß bei gleichem Seed auch gleiche Ausgaben produziert werden. Die Ausgabe eines PRBG ist nicht zufällig, sie erscheint nur zufällig. Tatsächlich ist die Menge der Ausgaben nur eine kleine Teilmenge aller produzierbaren Ausgaben der Länge l, nämlich ${}^{2^k}/_{2^l}$. Die Idee ist, eine kleine zufällige Sequenz (Seed) auf eine länger Sequenz zu strecken.
Es gibt zwei äquivalente Definitionen für einen kryptographisch sicheren PRBG. Zum einen erfüllt ein PRBG den „Next Bit Test“, falls es keinen Algorithmus mit polynomieller Laufzeit gibt, der bei Kenntnis von den ersten l Bits einer Ausgabesequenz s das $(l+1)$te Bit von s mit einer deutlich höheren Wahrscheinlichkeit als $\frac{1}{2}$ vorhersagen kann. Zum anderen erfüllt ein PRBG alle statistischen Tests mit polynomieller Laufzeit in Länge l, falls kein Algorithmus mit polynomieller Laufzeit existiert, der zwischen einer beliebigen Ausgabe eines PRBG und einer wahren zufälligen Sequenz der gleichen Länger unterscheiden kann.
Es gibt verschiedene Algorithmen, sogenannte Generatoren, um Zufallssequenzen zu erzeugen. So zum Beispiel den ANSI X9.17 Generator oder den FIPS 186 Generator. Interessante Generatoren finden sich unter [unc], so zum Beispiel XORShiftRNG, AESCounterRNG oder CMWC4096RNG, von denen zumindest AESCounterRNG kryptographisch sicher ist, ähnlich dem Fortuna RNG von Bruce Schneier und Niels Ferguson.

Kryptographische Hashfunktion SHA

Eine Hashfunktion ist eine effizient berechenbare funktionelle Abbildung einer binären Eingabe beliebiger Länge auf eine binäre Ausgabe mit fester Länge, dem Hashwert. Die Grundidee ist, daß ein Hashwert als eine kompakte Darstellung der Eingabe dient, als eine Art Fingerabdruck. In Anlehnung an [MvOV01] ist eine gewünschte Eigenschaft einer Hashfunktion mit n-Bit Hashwerten, daß die Wahrscheinlichkeit bei einer zufälligen Eingabe für eine bestimmte Ausgabe 2^{-n} ist. Eine kryptographisch Hashfunktionen h wird so ausgewählt, daß es nicht möglich (berechenbar) ist, zwei verschiedenen Eingaben zu finden, die einen gleichen Hashwert besitzen. Weiterhin soll es unmöglich sein, zu einem gegebenen Hashwert y ein Urbild x zu finden, so daß $h(x) = y$ gilt.
Das National Institute of Standards and Technology (NIST) entwickelte im Secure Hash Standard (SHS) den Secure Hash Algorithm (SHA) mit einem Hash-Wert

von 160 Bit. Eine Beschreibung des Algorithmus ist unter FIPS PUB 180-1 [fipa] abgelegt. (FIPS ist die Abkürzung für Federal Information Processing Standards Publication.) Der Secure-Hash-Standard ist Teil des Digital-Signature-Standards (DSS), und dieser wiederum Teil des Capstone-Projekts. Dieses Projekt des US-Ministeriums definiert Standards für öffentlich verfügbare Kryptographie. Es besteht aus vier Hauptkomponenten. Dazu gehören ein symmetrischer Verschlüsselungsalgorithmus (Skipjack, auch Clipper genannt), ein Schlüsselaustauschprotokoll (bisher nicht öffentlich, ein Diffie-Hellman-Verfahren), ein Hash-Algorithmus (SHA) und ein Algorithmus für die digitale Unterschrift (DSA, die SHA benutzt). Die Auswahl der Algorithmen trifft das NIST und die NSA.

Perfect Forward Secrecy (PFS)

Man unterscheidet verschiedene Schlüsselarten, die kompromittiert werden können: Zum einen langlebige (symmetrische oder asymmetrische) (Haupt-)Schlüssel und zum anderen kurzlebige (Sitzungs-)Schlüssel, sowohl für vergangene als auch für zukünftige Sitzungen.
In Anlehnung an [MvOV01] Kapitel 12 wird Perfect Forward Secrecy (PFS) folgendermaßen definiert: Ein Protokoll ist PFS, falls das Bekanntwerden eines langlebigen Schlüssels nicht einen Sitzungschlüssel in der Vergangenheit kompromittiert. Die Idee dahinter ist, daß alte Sitzungen, für die ein Sitzungsschlüssel mit Hilfe eines langlebigen Hauptschlüssels ausgehandelt worden sind, sicher in der Vergangenheit liegen. Insbesondere Diffie-Hellman-ähnliches Vorgehen für die Aushandlung eines Sitzungsschlüssel gewährleistet obige Unabhängigkeit vom Hauptschlüssel. Selbstverständlich bedeutet eine Kompromittierung des Hauptschlüssel weiterhin Zugriff auf zukünftige Sitzungen.

2.1.2 Kryptographische Protokolle

Secure Remote Password Protocol (SRP)

Das Secure Remote Password Protocol (SRP) aktuell in Version SRP-6a (vgl. [srpa] ist ein passwortbasiertes Authentifizierungs- und Schlüsselaustauschprotokoll. Es löst speziell das Problem der Clientauthentifikation gegenüber einem Server, wobei der Client sich nur ein kleines Geheimnis (wie ein Passwort) merken muss. Der Server hingegen merkt sich einen Verifizierer für Benutzerpasswort je Benutzer. Dieser Verifizierer erlaubt es dem Server den Client zu authentifizieren, aber es ist nicht möglich, sich als der Client auszugeben, falls ein Angreifer im Besitz des Verifizierers ist. Außerdem haben der Client und der Server nach Abschluss der Authentifikation einen kryptographisch starkes gemeinsames Geheimnis, aus dem sich ein Schlüssel für einen anschließenden sicheren Kommunikationskanal ableiten läßt. Das Protokoll ist sicher gegen bekannte aktive und passive Angriffe während des Authentifizierungsvorgangs. Der Server ist authentifiziert, indem er durch einen Abgleich des entstandenden geheimen Schlüssels nachweist, daß er im Besitz des korrekten Verifizierers ist. Nachfolgend wird die letzte Version SRP-6a des Protokolls beschrieben:

```
N     Eine große "safe" Primzahl
      (N = 2q+1, wobei q eine (Sophie-Germain) Primzahl ist)
      Alle Rechenoperationen werden modulo N durchgeführt
g     Ein Generator modulo N
k     Parameter (k = H(N, g) in SRP-6a, k = 3 für SRP-6)
s     Salt (Salz) des Benutzers
I     Benutzername (vgl. Identifizierer)
p     Passwort im Klartext
H()   Einweg Hashfunktion
^     (Modulare) Potenz
u     Zufälliger Verwürfelungsparameter
a,b   Geheime,     flüchtige Werte
A,B   Öffentliche, flüchtige Werte
x     Privater Schlüssel (aus p und s abgeleitet)
v     Passwort-Verifizierer
```

Mit obiger Legende werden Vorbereitungen getroffen, damit mit der eigentlichen Authentifizierung und dem Schlüsselaustauch begonnen werden kann. Zunächst berechnet der Client mit seinem Passwort den Passwort-Verifizierer wie folgt:

```
x = H(s, p)               (s zufällig gewählt)
v = g^x                   (Passwort-Verifizierer)
```

Daraufhin übergibt der Client den Verifizierer v samt Salt s an den Server in einer sicheren Weise (was nicht Bestandteil dieses Protokolls ist) und der Server hinterlegt das Tupel (I, s, v) für einen Benutzer I. Das Protokoll kann dann durch Austausch von insgesamt vier Nachrichten durchgeführt werden. Die ersten beiden Nachrichten (eine vom Client zum Server, die andere umgekehrt) stellen sicher, daß beide den einen gemeinsamen geheimen Schlüssel besitzen. Darauf wird noch verifiziert, ob beide den gleichen Schlüssel haben.

```
Client -> Server:  I, A = g^a           (Benutzername, a = zufällige Zahl)
Server -> Client:  s, B = kv+g^b        (Salt senden,  b = zufällige Zahl)

          Beide :  u = H(A,B)

          Client:  x = H(s,p)           (Benutzer gibt Passwort ein)
          Client:  S = (B-kg^x)^(a+ux)  (berechnet Sitzungsschlüssel)
          Client:  K = H(S)

          Server:  S = (Av^u)^b         (berechnet Sitzungsschlüssel)
          Server:  K = H(S)
```

Nun haben beide Parteien einen gemeinsamen Sitzungschlüssel K, den nur sie kennen. Um die Authentifizierung abzuschließen, müssen beide nachweisen, daß sie im Besitz des gleichen Schlüssels sind. Ein mögliches Vorgehen ist:

```
Client -> Server:  M = H(H(N) xor H(g), H(I), s, A, B, K)
Server -> Client:  H(A, M, K)
```

Dabei ist auf folgendes zu achten:

- Der Client bricht den Vorgang ab, falls er $B \equiv 0 \bmod N$ oder $u \equiv 0$ empfängt.
- Der Server bricht den Vorgang ab, falls er $A \equiv 0 \bmod N$ erhält.
- Der Client muss seinen Beweis für K zuerst zeigen. Falls der Server ermittelt, daß dieser Beweis ungültig ist, so muss er den Vorgang abbrechen ohne seinen Beweis zu übermitteln.

SRP hat folgenden Eigenschaften:

- Keine brauchbare Information über das Passwort p oder den privaten Schlüssel x wird während eines erfolreichen Protokolllaufs offenbart. Insbesondere, wird ein Angreifer davon abgehalten, Passwörter auf Basis von abgefangenen Nachrichten zu erraten oder zu verifizieren.
- Keine brauchbare Information über den Sitzungsschlüssel K wird einem Angreifer zugänglich während eines erfolgreichen Laufs. Da K im Gegensatz zu einem Passwort mit begrenzter Entropie kryptographisch stark ist, bestehen keine Bedenken, daß K erraten werden kann, solange K nicht direkt von einem Eindringling berechnet werden kann.
- Selbst ein aktiver Angreifer mit der Fähigkeit Nachrichten zu ändern oder zu erstellen und sie als die vom Client oder Server erscheinen zu lassen wird davon abgehalten, Zugang zum Server, Kenntnis vom Passwort oder Sitzungsschlüssel zu erlangen. Im schlimmsten Fall ist es einem Angreifer möglich, die Authentifikation abzubrechen (vgl. Denial-of-Service).
- Falls ein Angreifer den Passwort-Verifizierer v kennt, kann er sich dennoch nicht als Benutzer ausgeben ohne eine aufwendige Wörterbuchattacke durchzuführen.
- Falls ein Sitzungschlüssel einer vergangenen Sitzung kompromittiert wird, so hilft dies einem Angreifer nicht, das Benutzerpasswort zu erlangen.
- Selbst für den Fall, daß das Benutzerpasswort kompromittiert wird, so sind vergangene Sitzung nicht entschlüsselbar und laufende Sitzungen zumindest gegen passive Attacken sicher. (vgl. PFS)

Transport Layer Security (TLS)

Das *Transport Layer Security (TLS)* Protokoll [TLSa] hat als Hauptziel die Sicherstellung von Geheimhaltung und Integrität der Daten einer Sitzung zweier kommunizierender Parteien (Client und Server). Das Protokoll ist aus zwei Schichten

zusammengesetzt: das auf einem Transportprotokoll wie zum Beispiel TCP aufsetzende *TLS Record Protocol* und einer darüberliegenden Schicht, die unter anderem das *TLS Handshake Protocol* enthält. Das TLS Record Protocol bietet zwei Eigenschaft für eine sichere Verbindung:

- Die Verbindung ist abhörsicher: Dafür wird symmetrische Verschlüsselung eingesetzt, wobei die Schlüssel für jede Verbindung einmalig neu per TLS Handshake Protocol erzeugt werden.
- Die Integrität der Verbindung wird sichergestellt: Über einen *Message Authentication Codes (MAC)* wird jede Nachricht auf Veränderung überprüft.

Außerdem ist das Record Protocol für die Fragmentierung der Daten zuständig. Das TLS Handshake Protocol erlaubt es, entweder nur die Identität des Servers, des Servers und des Clients oder von keinen der beiden zu authentifizieren, indem asymmetrische Verschlüsselung mit Zertifikaten ins Spiel kommt (RSA, DSA, usw.) Außerdem steht nach dem Handshake beiden Parteien ein gemeinsames Geheimnis (pre-master-secret) zur Verfügung, aus dem Schlüssel für die anschließende Verschlüsselung und den Integritätsschutz beim Record Protocol abgeleitet werden. Im Falle einer Verbindung mit Authentifikation ist es nicht einmal einem Angreifer zwischen den beiden Parteien möglich dieses gemeinsam Geheimnis zu erlangen, ohne daß dies auffallen würde und zu einem Verbindungsabbruch führt.
Ein Vorteil von TLS ist die Anwendungsprotokollunabhängigkeit; so können höhere Protokolle transparent auf dem TLS aufbauen. TLS definiert nicht wie der TLS Handshake initiiert wird und wie Zertifikat für Authentifikationen ausgetauscht und überprüft werden. Dies liegt in der Hand der darüberliegenden Anwendung. Es werden vier verschiedene Möglichkeiten des Schlüsselaustausches untersützt:

- RSA: Das von Client erzeugte pre-master-secret wird mit dem öffentlichen Schlüssel des Servers verschlüsselt und der Server kann nach Erhalt dies mit seinem privaten Schlüssel entschlüsseln.
- Anonymes Diffie-Hellman (DH): Client und Server tauschen unsignierte öffentliche DH-Werte aus. Beide erzeugen daraus das pre-master-secret mit ihren privaten DH-Werten. Dieser Schlüsselaustausch ist anfällig für sogenannte Man-in-the-Middle Angriffe, da keinerlei Authentifizierungsinformationen verwendet werden.
- Diffie-Hellman Ephemeral: Client und Server erzeugen kurzlebige (ephemeral) private DH-Werte und signieren die zugehörigen öffentlichen DH-Werte mit RSA oder DSS.
- Diffie-Hellman Fixed: Client und Server benutzen feste öffentliche DH-Werte, die von einer Certificate Authority signiert und somit überprüfbar sind. Beide berechnen das pre-master-secret aus den öffentlichen DH-Werten des anderen und ihrem eigenen privaten DH-Wert.

Der Handshake selbst kann in vier Phasen unterteilt werden, in denen Nachrichten zwischen Client und Server ausgetauscht werden. Abbildung 2.2 zeigt dies nochmal graphisch:

1. Phase 1: Der Client schickt zum Server eine Nachricht, und der Server antwortet dem Client. Die Parameter der Nachrichten sind die höchste vom Client unterstützte SSL-Protokoll-Version, eine 32 Byte lange Zufallszahl, die später verwendet wird, um das pre-master-secret zu bilden, eine Session-ID und die die zu verwendenden Algorithmen für Schlüsselaustausch, Verschlüsselung und Authentifizierung, auch *Cipher Suite* genannt.
2. Phase 2: Der Server identifiziert sich gegenüber dem Client durch Übermittlung von Zertifikaten.
3. Phase 3: Der Client identifiziert sich gegenüber dem Server und generiert ein gemeinsames *pre-master-secret*.
4. Phase 4: Aus dem pre-master-secret wird das master-secret abgeleitet und aus diesem der einmalige symmetrische Sitzungsschlüssel, der während der Verbindung zum Ver- und Entschlüsseln der Daten genutzt wird. Die Nachrichten, die die Kommunikationspartner sich nun gegenseitig zusenden, werden nur noch verschlüsselt übertragen.

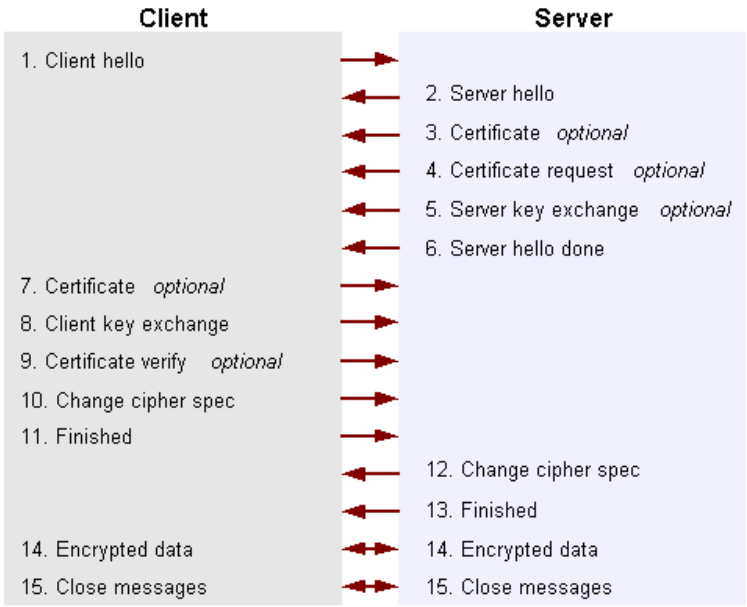


Abbildung 2.2: TLS Handshake Nachrichten

Da das TLS Protokoll flexibel für Erweiterungen ist, lassen sich auch andere Authentifizierungsmöglichkeiten, die zum Beispiel nicht zertifikatsbasiert sind benutzen. Ein Beispiel dafür ist die Umsetzung des Secure Remote Password Protocol (SRP) für TLS, siehe [TLSb].

2.2 Problemstellung

Bei der Datenverarbeitung benötigt der Datenverarbeiter Zugriff auf die Daten, da sonst eine Verarbeitung nicht möglich ist. D.h. im Falle von verschlüsselten Daten müssen die Daten für den Datenverarbeiter entschlüsselbar sein, denn Verar-

beitung von verschlüsselten Daten ist nur sehr eingeschränkt möglich (vgl. Kapitel 2.5). Grundsätzlich ist die Datenverarbeitung lokal durch den Besitzer oder entfernt durch einen Serviceanbieter möglich.
Ein Problem ensteht, wenn die Datenverarbeitung entfernt, also nicht beim Besitzer (besser Zugriffsberechtigtem), sondern bei einem Serviceanbieter stattfindet. Denn dann hat der Serviceanbieter auch Datenzugriff, was im Widerspruch zur kommenden Zielsetzung steht, da der Besitzer keine vollständige Kontrolle über seine Daten hat. Offensichtlich ist der Datenverarbeiter automatisch in der Lage, die Daten zu mißbrauchen, also für nicht serviceerfüllungsrelevante Absichten einzusetzen. Außerdem erhält der Besitzer keine Rückmeldung über einen Mißbrauch des Serviceanbieters. D.h. ein Mißbrauch kann auch unentdeckt bleiben. Erschwerend kommt noch hinzu, daß ein Mißbrauch zeitlich versetzt stattfinden kann, wenn die Daten beispielsweise aus Benutzersicht gesehen schon gelöscht erscheinen.
Anders betrachtet: Es entsteht ein Vertrauensproblem in den Serviceanbieter. Denn oftmals verspricht dieser einen sicheren Umgang mit den Daten, aber der Nutzer muß auf das sachgemäße Verhalten der zuständigen Administratoren des Serviceanbieters *vertrauen*. (vgl. Kapitel 2.5 Privacy Policy)

2.3 Zielsetzung

Eine *entfernte, zentrale Datenhaltung*, d.h. eine globale (weltweite), zentrale Anlaufstelle, die für die Aufbewahrung von Daten zuständig ist[2], bietet den Vorteil, daß der Datenbestand konsistent ist, da nur eine gültige Datenversion auf Grund der zentralen Stelle existiert. Darüberhinaus sind die Daten von überall erreichbar, solange eine Verbindung zu dieser zentralen Stelle vorhanden ist, was auch gleichzeitig einen Nachteil eines zentralen Ansatzes darstellt.

Motiviert durch Datenschutzfragen und dem Verlangen nach einer Lösung für sensible, eigene Daten in den Händen von Serviceanbietern, z.B. Online-Email, kristallisiert sich als Zielsetzung der *sichere Umgang mit Daten* heraus. Dabei bedarf es einer Erklärung und Abgrenzung der Begriffe *„sicher“*, *„Umgang“* und *„Daten“*, da insbesondere der Begriff *„sicher“* oft in ambivalenten Bedeutungen verwendet wird.

Sicherer Umgang mit Daten

Die folgende Auflistung erkläutert die verwendeten Begriffe der Zielsetzung:

- Unter dem Begriff *„Daten“* sind die eigenen Inhaltsdaten (z.B. Dokumente, Bilder, usw.) im Gegensatz zu personenbezogenen Daten (z.B. Kontaktdaten, Informationen zur Person, usw.) gemeint.
- *„Umgang“* meint die Verarbeitung, Haltung und Übertragung der Daten.

[2] Eine Einschränkung und Abkapselung auf z.B. Unternehmengrenzen ist möglich und manchmal sinnvoll.

- Der ambivalente Begriff *„Sicher“* zielt im Folgenden auf den durch den Besitzer bestimmten Zugriff auf die eigenen Inhaltsdaten im Klartext ab. D.h. der Besitzer hat volle Kontrolle darüber, wer wann auf seine Daten zugreift und dies betrifft sowohl die Haltung, als insbesondere auch die Verarbeitung der Daten.

Die Zielsetzung hat selbstverständlich direkte Auswirkungen auf die Anforderungen an einen Lösungsansatz, auf die im folgenden Kapitel eingegangen werden.

2.4 Grundsätzliche Lösungsanforderungen

Ausgehend von der Zielsetzung *„Sicherer Umgang mit Daten“* stellen sich im Rahmen der Problemstellung folgende Anforderungen an einen Lösungsansatz:
Zunächst bedarf es einer grundsätzlichen Trennung von Daten an sich und Daten in zugriffsgeschützter Form, um die volle Datenzugriffskontrolle des Besitzers zu erreichen. Die zugriffsgeschütze Form meint die Verschlüsselung von Daten, um Datenzugriffskontrolle (Zielsetzung) bei entfernter Datenhaltung zu erreichen. Dadurch verschiebt man den Zugriff von einem direktem Zugriff auf einen Zugriff auf die bei der Verschlüsselung verwendeten Schlüssel.
Weiterhin müßte eine entfernte Datenverarbeitung auf verschlüsselten Daten stattfinden, da ansonsten die geforderte selbstbestimmte Zugriffskontrolle nicht gewährleistet ist. Da eine Verarbeitung von verschlüsselten Daten nur sehr eingeschränkt möglich ist, muß die Verabeitung lokal, d.h. immer im vom Benutzer kontrollierten Programmausführungsbereich, stattfinden. Um einen ordnungsgemäßen Umgang gewährleisten zu können, bedarf es einer *lokalen Kontrollinstanz*, die auf Benutzerseite sicherstellt, daß keine Daten außerhalb eines vom Benutzer kontrollierten Bereiches im Klartext zugreifbar sind. Insbesondere soll so der Anbieter der datenverarbeitenden Software vom Zugriff ausgeschlossen werden. D.h. dieser stellt nur noch die Software zur Verfügung, die aber lokal beim Benutzer läuft.
Die Kontrollinstanz übernimmt als Hauptaufgabe also die Kontrolle der Kommunikationsmöglichkeiten des Datenverarbeiters, wodurch Seitenkanäle zwischen Datenverarbeitung und Datenhaltung unterbunden werden. Dabei stellt die Verschlüsselung der Daten das offensichlichste Mittel dar. Im Gegensatz zu einer Verschlüsselung des Übertragungsweges geht es hierbei auch um den Zugriff beim Ziel. Außerdem muß die versprochene Funktionalität überprüfbar sein, um Vertrauen zu schaffen (vgl. Vertrauensproblem). Die Überprüfbarkeit übernimmt die Plattform der Programmiersprache, in der die Kontrollinstanz umgesetzt wird.

2.5 Einordnung der aktuellen Situation

Viele online Webanwendungen, die Benutzerdaten auf den Servern des Anwendungsanbieters ablegen, leiden unter dem beschriebenen Problem, daß die Daten des Benutzers zur Serviceerfüllung für den Anbieter einsehbar sind. Der Anwendungsanbieter beteuert oftmals durch eine Datenschutzbestimmung (Privacy Policy) den sachgemäßen Umgang mit den Daten, der bei ihm registrierten Benutzer. Ein pro-

minentes Beispiel sind die Webanwendungen von Google, z.B. Google Docs. Auf den Hilfeseiten von Google Docs [gooa] heißt es: „Many Google Docs users add personal information to their documents, spreadsheets and presentations, and this information is safely stored on Google's secure servers." Um eine Vorstellung für die Begriff „safely" und „secure" in Hinblick auf die Zielsetzung zu bekommen, findet man in der Datenschutzrichtlinie von Google vom 11. März 2009 [goob] folgendes: „We restrict access to personal information to Google employees, contractors and agents who need to know that information in order to operate, develop or improve our services. These individuals are bound by confidentiality obligations and may be subject to discipline, including termination and criminal prosecution, if they fail to meet these obligations." Also sind die Daten, der bei Google registrierten Benutzer, zumindest für einige Google Mitarbeiter einsehbar. Weitere Beispiele sind, um nur einige zu nennen, online Email-Dienste, wie GMX [gmx], Hotmail [hot], Gmail [gma]. Hier sieht man beispielhaft das beschriebene Vertrauensproblem. Dies steht im Widerspruch zur Zielsetzung aus Kapitel 2.3.
Andererseits gestaltet es sich schwierig, die Anwendungen serverseitig auf verschlüsselten Daten operieren zu lassen. Denn die komplette Anwendungslogik läßt sich nicht auf die folgenden derzeit möglichen Operationen auf verschlüsselten Daten abbilden: Boneh et al. [BGN05, BW07] zeigen die Möglichkeiten grundlegender Operationen auf, die weit entfernt von der Umsetzung einer Anwendungslogik sind. Diese sind die öffentliche Auswertung des verschlüsselten Ergebnisses von 2-DNF Formeln mit verschlüsselten Eingangsvariabelbelegungen und die sichere Auswertung von Anfragen mit Konjunktionen von Gleichheitstests, Vergleichen und Teilmengenbeziehungen über verschlüsselten Werten. Desweiteren sei auf den Beitrag von Sahai [Sah08] verwiesen, der die Frage angeht, ob man verschlüsselte Daten verarbeiten kann, ohne sie vorher entschlüsseln zu müssen.
Ein viel beachtetes Feld ist die Suche auf verschlüsselten Daten. Beispielsweise bringt das Schema von Song, Wagner und Perrig [SWP00] die Daten in eine besondere verschlüsselte Form (Searchable Symmetric Key Encryption), um diese mit einem Suchwort abgleichbar zu machen; nachteilig ist die schlechte Laufzeit der sequentiellen Überprüfung. Andere Ansätze wie der von Goh [Goh03] nutzen Bloomfilter. Baumartige Ansätze werden von Brinkman [BDJ04] diskutiert und insbesondere auf XML als Anwendung [BFD$^+$04] eingegangen wird. Ucal [Uca] bietet einen guten Überblick über verschiedene Ansätze zur Suche auf verschlüsselten Daten.
Da die (serverseitige) Anwendungslogik nicht ausreichend auf verschlüsselten Daten arbeiten kann, wird zumindest die sichere Datenhaltung serverseitig, also entfernt bewältigt. Frühe kryptographische Dateisysteme [Bla93, CCDSP97] setzen dort an und bilden eine Sicherheitsschicht über bestehenden Dateisystemen. Diese wurden um Konzepte wie Access Control List und später Lazy Key Revocation und Key Rotation [KRS$^+$] (durch Key Regression [FKK06] verfeinert) erweitert. Auf diesen Konzepten baut ein interessanter Ansatz namens Wuala [wua] auf, der die eingangs beschriebenen Probleme angeht. Wuala basiert auf einer Technologie, die größtenteils an der ETH Zürich (Eidgenössische Technische Hochschule) erforscht und entwickelt wurde. Wuala ist ein verteiltes Online-Speichersystem, das alle Dateien schon auf Clientseite verschlüsselt und erst dann sowohl auf dem Server von Wuala, als auch in einem Peer-To-Peer-Netz ablegt. Den kryptographischen, theoretischen Hintergrund bildet der Cryptree [GMSW06]. Dabei bestimmt jeder Nutzer, wer Zugriff auf seine

Daten hat und schließt somit selbst die Betreiber von Wuala aus. Von Nachteil ist, daß Wuala Closed Source ist und somit die Funktionsweise des Programms bezüglich der versprochenen Sicherheit (Verschlüsselung und wer Kenntnis des Schlüssels hat) nicht durch ein Audit bestätigt werden kann. Das Vertrauensproblem in die versprochene Funktionsweise (vgl. Spyware) besteht also dennoch. Somit erfüllt Wuala nicht alle Anforderungen, damit ein Benutzer volle Kontrolle über seine Daten inne hat, denn es leidet an besagtem Vertrauensproblem, genauer an der Unüberprüfbarkeit der sicherheitsrelevanten Funktionalität.
Keine dieser aktuellen Anwendungsausrichtungen erfüllt alle beschriebenen Anforderungen an einen Lösungsansatz. Im Vergleich zu kryptographischen Dateisystemen geht der im folgenden beschriebene Ansatz einen Schritt weiter und offenbart keine dateisystemähnliche Struktur. Außerdem wird ein Szenario einer bösartigen, mit dem Datenhalter zusammenarbeitenden Anwendung zugrundegelegt, welche eine Dateisystemstruktur zur ungewollten Informationsübermittlung an die Datenhaltung ausnutzen könnte.

3 Lösungsansatz

Ausgehend von der beschriebenen Problemstellung des vorherigen Kapitels wird im Rahmen der festgestellten grundsätzlichen Anforderungen an einen Lösungsansatz in diesem Kapitel ein konkreter Lösungsansatz beschrieben. Daraufhin wird dieser Ansatz in Bezug auf eine anschließende Umsetzung konkretisiert. Dieses Kapitel schließt mit einer genaueren Betrachtung der verwendeten Sicherheitsmechanismen ab.

3.1 Konkreter Lösungsansatz

Der entscheidende Kern dieses Lösungsansatzes ist eine *Trennung der Ausführungseinheiten nach ihren Zuständigkeiten*. D.h. die Datenhaltung wird von der Datenverarbeitung getrennt, da die beiden Zuständigkeitsbereiche fachlich gesehen disjunkt sind und somit nicht vermischt werden sollten. Wie bei den grundsätzlichen Lösungsanforderungen in Kapitel 2.4 erwähnt kommt eine Kontrollinstanz ins Spiel, die diese Trennung vornimmt und gewährleistet. Im Folgenden wird diese Instanz *Türsteher* genannt. Alle sicherheitsrelvanten Angelegenheiten fallen in den Zuständigkeitsbereich des Türstehers; somit werden die Ausführungseinheiten (Datenverarbeiter, Datenhalter und Türsteher) implizit auch nach Sicherheitsrelevanz getrennt, da alles sicherheitsrelevante nur im Türsteher vonstatten geht.

3.1.1 Türsteher

Der Türsteher ist ein lokal, also auf Benutzerseite, ausgeführtes Programm. Zu den Aufgaben des Türstehers gehören die Anpassung der IDs, eine Art Kennzeichnung für Daten (genauer, siehe Kapitel 3.1.2), und der Daten, also inbesondere die Verschlüsselung. Außerdem wird der Türsteher als Vermittler zwischen Datenhaltung und Datenverarbeiter eingesetzt, um die direkte Kommunikation von Datenverarbeiter mit anderen Programmen, speziell der Datenhaltung, zu unterbinden.
Ein weiterer Kommunikationskanal des Datenverarbeiters könnte den Türsteher und damit die Verschlüsselung umgehen und Daten an Dritte übertragen und diesen zu-

gänglich machen. Daraus folgt, daß der Türsteher volle Kontrolle über die Kommunikationsmöglichkeiten der Datenverarbeitung besitzen muss. Dabei bietet der Türsteher die im Abschnitt Datenhaltung beschriebene Datenschnittstelle dem Datenverarbeiter an und nutzt dabei die der Datenhaltung. Jede Kommunikation der beiden soll über den Türsteher erfolgen. Dies ist nötig, um sicherzustellen, daß die Daten von der Datenverarbeitung kommend immer verschlüsselt werden, bevor sie zur Datenhaltung anschließend übertragen werden. Umgekehrt entschlüsselt der Türsteher die angeforderten Daten, bevor sie dem Datenverarbeiter übermittelt werden. Die Ver- und Entschlüsselung ist benutzerspezifisch, das bedeutet, die Daten werden vom Türsteher bezüglich eines Datenverarbeiters für einen einzelnen Benutzer ver- bzw. entschlüsselt. Dafür authentifiziert sich ein Benutzer zunächst beim Türsteher bzgl. einer anstehenden Datenverarbeitungssitzung. Jeder Benutzer hat einen geheimen symmetrischen Schlüssel, der bei der Registrierung einmal bestimmt wird und fest bleibt. Dieser wird (im Falle einer passwortbasierten Authentifizierung) mit dem Passwort des Benutzers verschlüsselt in der Datenhaltung abgelegt, beim Authentifizierungsvorgang ausgelesen und für die Ver- und Entschlüsselung in der aktuellen Datenverarbeitungssitzung verwendet.

3.1.2 Offene Datenhaltung

Im Rahmen der in Kapitel 2.1 genannten Aufgaben übernimmt die entfernte und zentrale Datenhaltung wie der Name schon andeuten soll *primär* die Aufgabe der Aufbewahrung von Daten und Abrechnung für die Bereitstellung dieses Dienstes. Dabei spielt die Art der Daten keine Rolle. So sollen beispielsweise auch die vom Türsteher für den Authentifizierungsvorgang benötigten Daten in der Datenhaltung abgelegt werden.
Der Benutzer wird sich gegenüber der Datenhaltung (indirekt über den Türsteher) nur wegen Abrechnungszwecken authentifizieren müssen. Die Verbindung zur Datenhaltung wird dabei mittels *TLS* [tlsc] abgesichert. Dies ist nötig, da die Datenhaltung das Konzept der *offenen Datenhaltung* verfolgt:
Daten werden im Gegensatz zu anderen Ansätzen, die oft Dateisystemestrukturen verwenden, in Form sogenannter Datenpakete abgelegt, also einfachen Bytesequenzen. Jedem Datenpaket wird eine eindeutige Identifikation (*ID*) zugewiesen, das heißt, es gibt keine zwei Datenpakete mit gleicher ID in einer Datenhaltung. Eine ID eines Datenpakets darf keine Rückschlüsse auf den Inhalt des Datenpakets zulassen. Anhand dieser ID kann *jeder Türsteher* bei der Datenhaltung ein Datenpaket erfragen, lesen, schreiben oder löschen. Diese vier Operationen charakterisieren eine schlichte Datenschnittstelle, die von der Datenhaltung angeboten wird. Diese anspruchlose Schnittstelle ermöglicht einen einfachen Einsatz der Sicherheitsmechanismen (Kapitel 3.3) aufgrund überschaubarerer, abzusichernder Angriffsmöglichkeiten und erleichtert somit folglich die anschließende Sicherheitsanalyse (Kapitel 4).
Darüberhinaus bietet eine anspruchlose Schnittstelle die Möglichkeit verschiedener Umsetzungsmöglichkeiten der Datenhaltung (Http/Php/MySql, Nativer Client, Peer-To-Peer-Netz, ...). Zu beachten ist, daß der Inhalt aller Datenpakete in der Datenhaltung (durch den Türsteher) symmetrisch verschlüsselt ist, getreu dem Leitsatz: „Sobald etwas persistent ist (den Arbeitsspeicher verläßt), ist es auch verschlüsselt!“.
Für die Datenhaltung erscheinen also alle Datenpakete als mit einer ID versehene,

zufällige Bytesequenzen, da diese für sie nicht entschlüsselbar sind. Durch die Tatsache, daß jeder Türsteher Zugriff auf alle Datenpakete bei Kenntnis der entsprechenden IDs hat, kann auch jeder Datenverarbeiter über den Türsteher mutwillig oder unbeabsichtigt Datenpakete durch Löschen oder Überschreiben zerstören, aber aufgrund der Verschlüsselung nur mit Kenntnis des Schlüssels im Klartext auslesen. Da die ID in Abhängigkeit der Beschreibung des Datenpaketinhaltes gewählt wird, kann nur derjenige Datenpakete überschreiben oder löschen, der auch weiß, wie deren IDs lauten. Zu vergleichen ist dies mit einem Telefonbuch, in dem man mit Kenntnis eines Namens und der zugehörigen Anschrift (entspricht der ID) eine Telefonnummer (entspricht dem Datenpaket) nachschlagen kann. Ein entscheidender Unterschied besteht hingegen darin, daß nur der Datenhaltung eine Auflistung aller IDs bekannt ist, d.h. jeder kann im Rahmen einer Anfrage an die Datenhaltung *nur eine* konkrete ID ansprechen. Nun wird auch deutlich, warum die Kommunikation zwischen Türsteher und Datenhalter abgesichert werden muss: Denn ein Angreifer könnte durch Abhören Kenntnis von IDs erlangen und die zugehörigen Datenpakete dann gezielt überschreiben oder löschen.
Durch einen hinreichend großen Wertebereich, aus dem IDs möglichst gleichverteilt mit einer surjektiven, aber nicht injektiven Abbildung gewählt werden sollen, ergibt sich gemäß des Geburtstagsparadoxons eine hinreichend kleine Kollisionswahrscheinlichkeit für IDs. Damit besteht ein entscheidender qualitativer Unterschied zu einer 100%igen Abgrenzung von IDs untereinander, der als Nachteil des Konzept der offenen Datenhaltung festgehalten werden muss. Denn mit einer gewissen Wahrscheinlichkeit können zwei Datenpakete mit verschiedenen Beschreibungen die gleiche ID erhalten, womit Inkonsistenzen durch unbeabsichtigtes Überschreiben möglich sind. Da die Wahrscheinlichkeit für diesen Fall exponentiell in Bezug auf die Maximallänge der IDs sinkt, kann mit ausreichend langer, aber dennoch praktikabler ID-Länge die Wahrscheinlichkeit kleiner als beispielsweise die eines Hardwareversagens der beteiligten Komponenten gehalten werden. Die Kollisionswahrscheinlichkeit läßt sich immer kleiner wählen als eine beliebige, vorgegebene Wahrscheinlichkeit für ein anderes Versagen. (siehe Tabelle 3.1) Zusammenfassend läßt sich der Nachteil der offenen Datenhaltung auf das Problem der Kenntnis von IDs fokussieren.

3.1.3 Datenverarbeitung

Die Datenverarbeitung ist ein lokal ausgeführtes Programm, das z.B. von einem Serviceanbieter gestellt wird und dessen Kommunikationsmöglichkeit auf die mit dem Türsteher eingeschränkt sind. Der Türsteher startet den Datenverarbeiter, so daß dieser in einer vom Türsteher kontrollierten Umgebung läuft (vgl. Kapitel 3.3.3). Da eine Kommunikation auch über das Persistieren auf lokalen Datenträgern erfolgen könnte, ist die einzige Möglichkeit der Persistenz über den Türsteher gegeben. Weiterhin sind insbesondere Netzwerkzugriffe der Datenverbeitung wegen Unterbindung direkter Kommunikation mit Dritten ebenfalls nicht gestattet. Für das datenverarbeitenden Programm beschränkt sich die Kenntnis von Informationen über den beim Türsteher authentifizierten Benutzer auf den Benutzernamen, damit dieser personenbezogen arbeiten kann. Während einer Datenverarbeitungssitzung ist es dem Datenverarbeiter möglich, die vom Türsteher angebotene Datenschnittstel-

le zu nutzen. Erweiterungen des Türstehers können einige dieser Einschränkungen aufheben, vgl. Kapitel 6.

3.1.4 Bewertung

Das oben beschriebene Konzept der *Trennung nach Zuständigkeiten*, also insbesondere die Separation von Datenverarbeitung und Datenhaltung durch einen Türsteher bietet „Security by Design“ im Vergleich zu „Security by Policy“ (vgl. Kapitel 2.5). Denn alle sicherheitsrelevanten Funktionen werden im Türsteher vereint und auf diese samt der nötigen Schnittstellen sogar beschränkt.
Somit wird bezüglich des erwähnten Vertrauensproblems (Kapitel 2.2) das Vertrauen vom Anbieter eines datenverarbeitenden Softwarepaketes auf die Funktionalität des Türstehers verlagert, genauer auf die Funktionalität der aktuellen Version des Türsteherprogramms. Dieses ist quellcodeoffen und sollte möglichst wenigen Änderungen unterliegen, da bei jeder Änderung neues Vertrauen in die neu entstandene Version geschaffen werden muss. Hier zeigt sich auch ein entscheidener Vorteil zu bestehenden Ansätzen, die jene Trennung nicht vornehmen: Das datenverarbeitende Programm kann Closed Source sein, um geistiges Eigentum der Anwendungsentwickler zu schützen, was ohne Trennung mit gleichzeitiger Beweispflicht für sicherheitsrelevante Funktionalitäten nicht gewährleistet wäre, da diese eine Offenlegung des Quelltextess erfordert.
Das Trennen nach Zuständigkeiten bietet außerdem den Vorteil, daß Softwareaktualisierungen nur pro Zuständigkeitsbereich durchzuführen sind und nicht für alle Komponenten gemeinsam. D.h. Änderungen an einem Bereich betreffen die anderen Bereiche nicht; so kann beispielsweise aus Performanzgründen die Datenhaltung ihre Datenbank wechseln ohne den Datenverarbeiter oder den Türsteher zu beeinflussen. Meist werden in der Praxis jedoch Änderungen bei der Datenverarbeitung auftreten, die aber insbesondere keinerlei Auswirkungen auf Vertrauen in die aktuelle Version des Türsteherprogramms haben.
Durch das Trennen nach Zuständigkeiten und einheitlichen Schnittstellen bietet sich außerdem der Vorteil der Geräteunabhängigkeit. So ist es denkbar, daß der Türsteher und Datenverarbeiter auch auf mobilen Endgeräten laufen können.
Als Nachteil ist zu erwähnen, daß aufgrund der lokalen Ausführung des Datenverarbeiters die Rechenkapazität im Gegensatz zu Ansätzen, die beispielsweise Cloud Computing verfolgen, auf die lokale Machine begrenzt ist. Nachteilig aus Sicht des Anwendungsentwicklers setzt das Trennen nach Zuständigkeiten und der daraus resultierenden Schnittstellen zwischen den Akteuren außerdem eine Anpassung der datenverarbeitenden Software voraus. Da alle Vorgänge, die die Persistierung von Daten betreffen, über die Datenschnittstelle des Türstehers ablaufen und jeglicher Netzwerkverkehr unterbunden wird, entstehen offensichtlich große konzeptionelle Einschränkungen für eine Datenverarbeitungssoftware, die auf Netzwerkverbindungen angewiesen ist. Jeweilige Erweiterungen des Türstehers können einige dieser Einschränkungen aufheben, vgl. Kapitel 6.

3.2 Umsetzung des konkreten Lösungansatzes

Dieses Kapitel beschreibt, wie die erwähnten Konzepte aus Kapitel 3.1 umgesetzt werden. Beginnend mit einem Überblick über den strukturellen Zusammenhang der Einzelkomponenten, werden die Schnittstellen in Bezug auf Datenfluß genauer betrachtet. Das Konzept der Offenen Datenhaltung wird rechnerisch erfaßt und anhand des Starts einer Datenhaltungssitzung die Kopplung der Datenverarbeiter an den Türsteher beschrieben.

3.2.1 Architektur

Die bisher erwähnten Komponenten Datenverarbeiter, Türsteher und Datenhaltung hängen folgendermaßen zusammen: Pro Maschine (und dies kann beispielsweise auch ein mobiles Endgerät sein) läuft genau dann eine Türsteherinstanz in einem eigenständigen Prozess, wenn mindestens eine Datenverarbeitungssitzung aktiv ist. Dies bedeutet insbesondere, daß mehrere Datenverarbeiter pro Maschine aktiv sein können. Diese laufen jeweils in einem abgeschotten Prozess, der vom Türsteher kontrolliert wird, so daß eine Kommunikation nur mit dem Türsteher auf der gleichen Maschine möglich ist. Die Entscheidung für einzelne Prozesse ist durch die bessere Möglichkeit der Kontrolle der Datenverarbeiter durch den Türsteher motiviert. Die zentrale Datenhaltung läuft auf einer dedizierten Maschine, die von überall über Netzwerk erreichbar sein sollte. Die Kommunikation der Komponenten läuft einheitlich über die Netzwerkschnittstelle. Auf der Anwendungsebene wird die Datenschnittstelle (vgl. Kapitel 3.2.3) verwendet. Die Datenverarbeiter auf einer Maschine kommunizieren mit dem zugehörigen Türsteher und dieser mit der zentralen Datenhaltung. Abbildung 3.1 verdeutlicht den beschriebenen Zusammenhang der Komponenten graphisch im Überblick.

3.2.2 Datenpaket, ID und Offene Datenhaltung

Wie bereits erwähnt, werden Daten in Form von Datenpaketen abgelegt. Ein Datenpaket ist eine Bytesequenz. Um dieses benennbar und damit auffindbar zu machen, wird jedem Datenpaket ein eindeutiger Name, eine ID zugewiesen. Eindeutig bedeutet dabei: es gibt keine zwei verschiedene Datenpakete mit der gleichen ID in der Datenhaltung.
Die ID eines Datenpaketes kommt bei den oben beschriebenen Stellen an der Datenschnittstelle zum Einsatz. Der Datenverarbeiter legt die ID für ein bestimmtes Datenpaket anhand seiner Anwendungslogik fest, das heißt anhand einer beliebig langen, aber eindeutigen textuellen Beschreibung des Datenpaketinhalts, damit dieses auffindbar bleibt. Da ein Beschreibungswert aber oftmals nicht ein Element aus der Menge der zugelassenen IDs ist, muss auf Seiten des Datenverarbeiters die Beschreibung in die Menge der möglichen IDs abgebildet werden (nicht zwingend unumkehrbar), um die Datenschnittstelle des Türstehers nutzen zu können. Weil die Datenhaltung aus der ID keine Rückschlüsse auf den Inhalt (bzw. dessen Beschreibung) des zugehörigen Datenpakets ziehen können soll, kommt beim Türsteher eine Hashfunktion zum Einsatz, die die ID der Datenverarbeitungsinstanz wieder in die

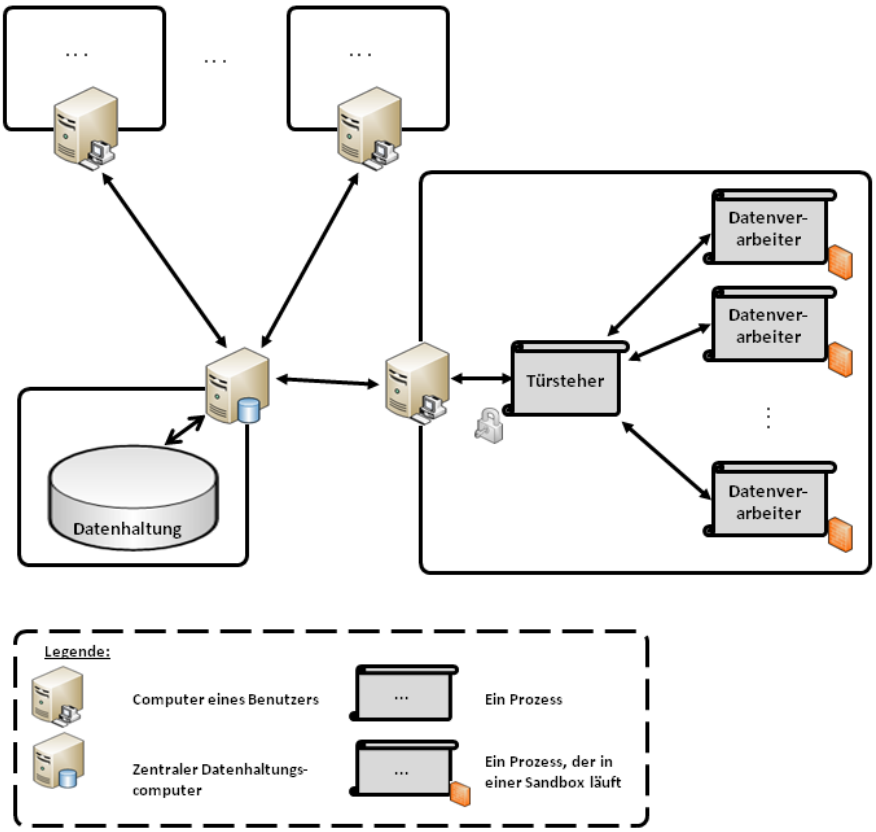


Abbildung 3.1: Architektonischer Zusammenhang der Komponenten Datenverarbeiter, Türsteher und Datenhaltung.

Menge der möglichen IDs abbildet, bevor sie für die Datenschnittstelle der Datenhaltung genutzt wird. Diese Abbildungen werden in Kapitel 3.3 genauer erläutert.
Die Länge einer jeden ID und damit die Größe des Wertebereichs aus dem die IDs ausgewählt werden, ist aus zwei Gründen entscheidend für das Konzept der Offenen Datenhaltung:
Erstens muß das grundsätzliche Problem der Offenen Datenhaltung minimiert werden, nämlich daß bei der surjektiven, aber nicht injektiven (da die Beschreibung beliebig lang ist) Abbildung durch eine Hashfunktion zweier verschiedener Datenpaketbeschreibungen aufgrund des Schubfachprinzips die entstandenen IDs der beiden Beschreibungen identisch sein können. Dieses Phänomen nennt man Kollision. Je größer die Maximallänge der IDs, desto unwahrscheinlicher wird eine Kollision, welche unbeabsichtigtes Überschreiben und damit auftretende Konsistenzprobleme zur Folge hätte. Klar ist, daß bei gleichen Datenpaketbeschreibungen beim Datenverarbeiter deterministisch auf gleiche IDs abgebildet wird. Ebenso entstehen beim Türsteher gleiche IDs bezüglich der Datenhaltung für gleiche IDs vom Datenverarbeiter genau dann, wenn der angemeldete Benutzer für beide IDs identisch ist. D.h. IDs werden in der Datenhaltung bis auf die grundsätzliche Kollisionsmöglichkeit nach Benutzern separiert. (vgl. Kapitel 3.3.5)
Anhand der Formel für das Geburtstagsparadoxon läßt sich die Wahrscheinlichkeit

P für mindestens eine Kollision bei k zufällig erzeugten IDs in Abhängigkeit der Maximalbitlänge b der IDs folgendermaßen angeben:

$$P = 1 - e^{-\frac{k \cdot (k-1)}{2N}} \text{ mit } N = 2^b$$

Tabelle 3.1 gibt eine Übersicht über ausgewählte Kollisionswahrscheinlichkeiten.

b	k	P
128	10^9	$1,469 \cdot 10^{-21}$
160	10^6	$3,421 \cdot 10^{-37}$
160	10^9	$3,421 \cdot 10^{-31}$
160	10^{12}	$3,421 \cdot 10^{-25}$
256	10^9	$4,318 \cdot 10^{-60}$
256	10^{12}	$4,318 \cdot 10^{-54}$

Tabelle 3.1: Kollisionswahrscheinlichkeit in der Offenen Datenhaltung

Eine lineare Vergrößerung der ID-Länge bringt also exponentielles Schrumpfen der Kollisionswahrscheinlichkeit mit sich. Dieser Zusammenhang wird in Abbildung 3.2 aufgezeigt.

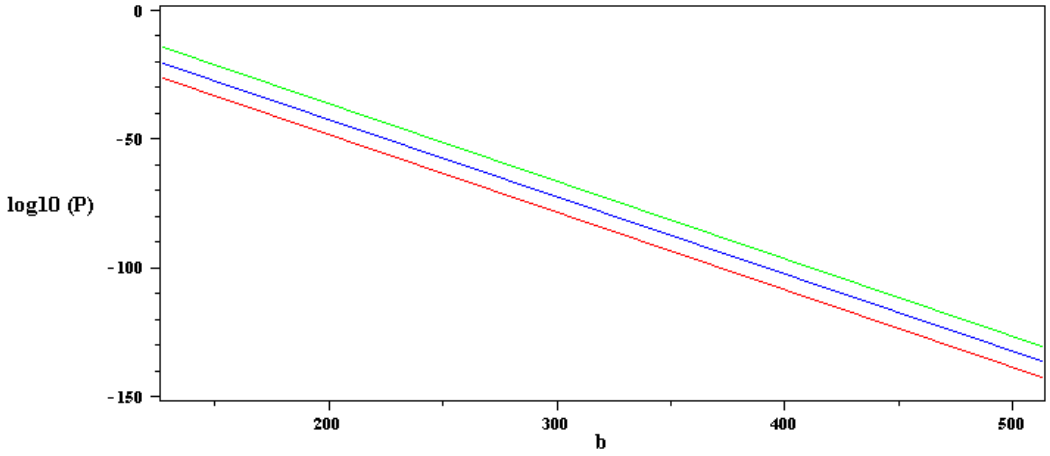


Abbildung 3.2: Zusammenhang zwischen Maximalbitlänge b der IDs und der Kollisionswahrscheinlichkeit P bei $k = 10^6$ (rot), $k = 10^9$ (blau) und $k = 10^{12}$ (grün) zufälligen IDs.

Zweitens wirken sich zu lange IDs negativ auf die Performanz der Datenhaltung aus. Denn diese muß alle Datenpakete anhand ihrer ID indizieren und somit hängt die Laufzeit der Anfragenbeantwortung bei üblicher Wahl eines Baums für den Index linear mit der Länge der ID zusammen.
Diese zwei gegenläufigen Aspekte erfordern einen Kompromiss für die Länge der ID, der sich durch alle Bereich der Implementierung zieht, da sowohl der Datenverarbeiter, Türsteher, als auch die Datenhaltung wegen gemeinsamer Schnittstellen davon betroffen sind. Typischerweise bewegt sich die ID-Länge im Bereich von einigen hundert Bits, um Konsistenzprobleme sehr unwahrscheinlich zu machen, da dies Vorrang vor Performanzüberlegungen der Datenhaltung hat.

3.2.3 Datenschnittstelle

Die Datenschnittstelle wird über Netzwerkschnittstellen an zwei Stellen genutzt. Zum einen bietet der Türsteher diese dem Datenverarbeiter an und zum anderen wird diese dem Türsteher von der Datenhaltung angeboten.
Die in Kapitel 3.1.2 eingeführten vier Operationen der Datenschnittstelle sind folgendermaßen definiert:

Operation	Parameter	Rückgabewert
Erfragen	ID	Boolean
Lesen	ID	Data / Null
Speichern	ID, Data	Boolean
Löschen	ID	Boolean

Tabelle 3.2: Signatur der Operationen der Datenschnittstelle

- Erfragen:
  - Parameter: ID
  - Rückgabewert: Wahrheitswert, ob unter der angegebenen ID ein Datenpaket vorhanden ist.
- Lesen:
  - Parameter: ID
  - Rückgabewert: Unter der ID abgelegtes Datenpaket. Wenn keins vorhanden ist, dann spezieller, ausgezeichneter Rückgabewert, der angibt, daß keine Datenpaket vorhanden ist.
- Speichern / Ablegen / Schreiben:
  - Parameter: ID, Datenpaket
  - Rückgabewert: Wahrheitswert, ob vor dem Ablegen des Datenpakets unter der angegebenen ID schon ein (anderes) Datenpaket vorhanden war.
- Löschen:
  - Parameter: ID
  - Rückgabewert: Wahrheitswert, ob vor dem Löschen des Datenpakets unter der angegebenen ID ein Datenpaket vorhanden war, dieses also gelöscht worden ist.

3.2.4 Start einer Datenverarbeitungssitzung

Anhand des Ablaufs eines Datenverarbeitungssitzungstarts soll beispielhaft das Zusammenspiel zwischen Türsteher und Datenverarbeiter auf einer Maschine insbesondere in Hinblick auf die Sicherheitsmechanismen aufgezeigt werden:
Der Datenverarbeiter wird nicht wie bei bisherigen Ansätzen direkt gestartet, son-

dern stattdesssen wird wie folgt vorgegangen: Der Türsteher wird gestartet, falls er zu diesem Zeitpunkt nicht bereits ausgeführt wird. Daher läuft höchstens eine Türsteherinstanz gleichzeitig. Dabei muss beim Starten die Signatur der Version des startenden Türstehers durch Interaktion mit dem Anwender überprüft werden, d.h. die Implementierung muss dies überprüfbar unterstützen. Daraufhin wird dem Türsteher eine Datenverarbeitungssitzungsanfrage mit einem Parameter übergeben, der angibt, welches Datenverarbeitungsprogramm gestartet werden soll, da es verschiedene Datenverarbeitungsprogramme von verschiedenen Softwareanbietern gibt. Anschließend wird die Authentifikation des Benutzers beim Türsteher beispielsweise durch Eingabe von Benutzername und Passwort durchgeführt. Darüberhinaus sind auch andere Authentifizierungsmöglichkeiten denkbar. Nach Einrichtung des abgesicherten Netzwerkkanals zum Datenhalter wird eine einmalige zufällige Sitzungidentifikation, die ausreichend lang ist, damit diese nicht erraten werden kann oder einer bisherigen Sitzungidentifikation entspricht, als Parameter beim der nun zu startenden Datenverarbeitungsinstanz übertragen. Daraufhin kontaktiert der Datenverarbeiter ebenfalls über die Netzwerkschnittstelle den Türsteher. Damit werden für die Dauer der Sitzung die vorherigen Authentifizierungsdaten beim Türsteher dem Netzwerkkanal zwischen Türsteher und Datenverarbeitungsinstanz zugeordnet. Diese Zuordnung bleibt bis zur Beendung der Sitzung bestehen. Der Türsteher steht mit potentiell mehreren Datenverarbeitungsinstanzen über je einen Netzwerkkanal in Verbindung. Hervorzuheben ist, daß erst nach erfolgreicher Authentifizierung eines Benutzers der Türsteher die angeforderte Datenverarbeiterinstanz in einem eigenen abgeschotten Prozess startet.

3.3 Sicherheitsmechanismen

Das folgende Kapitel beschreibt die verwendeten Sicherheitsmechanismen. Nach Erläuterung einiger Grundvoraussetzungen wird zunächst auf strukturelle Mechanismen wie das Signieren der Türsteherprogrammversion, die kontrollierte Ausführungsumgebung (Sandbox) und die Absicherung der Kommunikation zwischen Türsteher und Datenhalter eingegangen. Diese werden anhand der von der Java Plattform, die bei der in Kapitel 5 beschriebenen Implementierung eingesetzt wird, bereitgestellten Sicherheitsmechanismen dargestellt. Im Anschluß werden die Authentifizierung, die Registrierung und die damit verbundenen Schlüssel(-ableitungen) beschrieben. Daraufhin wird die Absicherung der Kommunikation zwischen Türsteher und Datenhalter mittels *TLS-PSK* aufgezeigt. Das Kapitel endet mit den Anpassungen eines Datenpaketes und dessen ID, um einen direkten Informationsfluß vom Datenverarbeiter zur Datenhaltung zu verhindern.

3.3.1 Lokales Vertrauen

Für alle folgenden Sicherheitsaspekte wird grundsätzlich davon ausgegangen, daß der Benutzer der verwendeten Maschine vertraut, also dort keine Spy- oder Maleware läuft. Diese könnten beispielsweise den Arbeitsspeicher zur Laufzeit des Türstehers auslesen und somit die Schlüssel, die zur Ver- und Entschlüsselung genutzt werden,

erhalten. Ein weiteres Problem sind sogenannte Keylogger, die alle Tastaturanschläge aufzeichnen können und somit eine auschließlich passwortbasierte Authentifizierung umgehen können. Außerdem wird davon ausgegangen, daß der Plattform der Programmiersprache, hier Java, und insbesondere dem Compiler vertraut wird, da ansonsten Angriffe nach Thompson [Tho07] möglich wären.

3.3.2 Signieren der Türsteherprogrammversion

Ein entscheidenes Sicherheitsmerkmal ist die Sicherstellung der Ausführung einer vertrauensvollen Version des Türsteherprogramms, denn im Türsteher sind alle sicherheitsrelevanten Funktionalitäten vereint. Ausgehend vom Vertrauensproblem aus Kapitel 2.2 muss aus Benutzersicht vor Ausführung der aktuellen Programmversion des Türstehers Vertrauen in die Funktionsweise derselben geschaffen werden. Dies stellt zwei Anforderungen an die Plattform der Programmiersprache, in der der Türsteher implementiert wird:
Erstens muß der Binärcode des Türstehers an den zugehörigen Quellcode gebunden werden können, da die Funktionsweise von Binärcode nicht direkt überprüfbar ist. Quellcode und Binärcode werden im Folgenden als Bündel aufgefaßt. Bei der verwendeten Java Plattform kann der Binärcode (*class*-Dateien) zusammen mit dem Quellcode (*java*-Dateien) in ein Archive (*jar*-Datei) als Bündel verpackt werden.
Zweitens muß das Vertrauen des Benutzers in den Quellcode auf Vertrauen des Benutzers in eine vertrauensvolle Instanz verlagert werden können, weil von einem Benutzer nicht erwartet werden kann, Quellcode auf korrekte Funktionsweise zu überprüfen. Dieser Mechanismus wird von der Java Plattform folgendermaßen bereitgestellt:
Der Türsteher wird als Bündel (*jar*-Datei) verpackt und darauf hin mit dem Programm *jarsigner* [jar] signiert. Das Signieren wird von einer vertrauensvollen Instanz (z.B. [tuv]) nach einem Audit mit dem privaten Schlüssel derselben durchgeführt. Um die Signatur überprüfen zu können, also verifizieren zu können, daß das Archiv tatsächlich von der vertrauensvollen Instanz signiert worden ist, wird der zugehörige öffentliche Schlüssel benötigt, der dem Archiv in Form eines Zertifikates hinzugefügt wird (vgl. [jar], Abschnitt „The Signed JAR File“). Dabei kommt es insbesondere auf die Authentizität des öffentlichen Schlüssels an. Diese wird durch eine Kette von Zertifikaten sichergestellt (vgl. [Cer]). Die Verifizierung wird von der Java Plattform anhand einer Kette bis hin zu den in der Plattform fest installierten Wurzelzertifikaten übernommen. Realisiert wird dies durch das bei der Implementierung verwendete Java Webstart [weba], indem vor der Ausführung des Türstehers die Signatur des Türsteher-Archiv verifiziert wird. Eine erfolgreiche Verifizierung bedeutet, daß das Archiv von der vertrauensvollen Instanz, die dabei namentlich angezeigt wird, signiert worden ist. Die vertrauensvolle Instanz drückt durch ihre Signatur aus, daß sie den an den Binärcode gebundenen Quellcode durchgesehen hat und dessen sachgemäßer Funktionalität vertraut. Dabei hat der Benutzer die Möglichkeit, den Ausgang der Verifizierung, also ob eine Kette bis zu einem Wurzelzertifikat existiert oder nicht, transparent einzusehen und anhand des Verifizierungsergebnisses, der Ausführung des Türstehers zuzustimmen oder diese abzulehnen.
Außerdem ist zu beachten, daß zusammen mit der Beschreibung über den Programmort des Datenverarbeiters auch der des Türstehers vom Anwendungsentwickler des

Daternverabeiters festgehalten wird. Dies geschieht im Rahmen von Java Webstart über ein sogenanntes Java Network Launching Protocol (*JNLP*) in Form einer XML-Datei. Darin werden unter anderem die URLs der *jar*-Archive des Türstehers und des Datenverarbeiters festgelegt.
Zusammenfassend kann der Benutzer vor Ausführung des Türstehers einsehen, welche Instanz den Quellcode durchgesehen hat, dessen Funktionsweise also vertraut, und *entscheiden*, ob er dieser Instanz *vertraut*, d.h. bereit ist, den an den Quellcode gekoppelten Türsteherbinärcode auszuführen. Abbildung 3.3 verdeutlicht die Interaktion mit dem Benutzer. Nachteilig ist, daß jede Änderung der Funktionsweise des Türstehers ein neues Signieren durch eine Instanz erfordert, denen die Benutzer vertrauen. Daher sollten Änderungen am Türsteher so selten wie möglich vorgenommen werden.

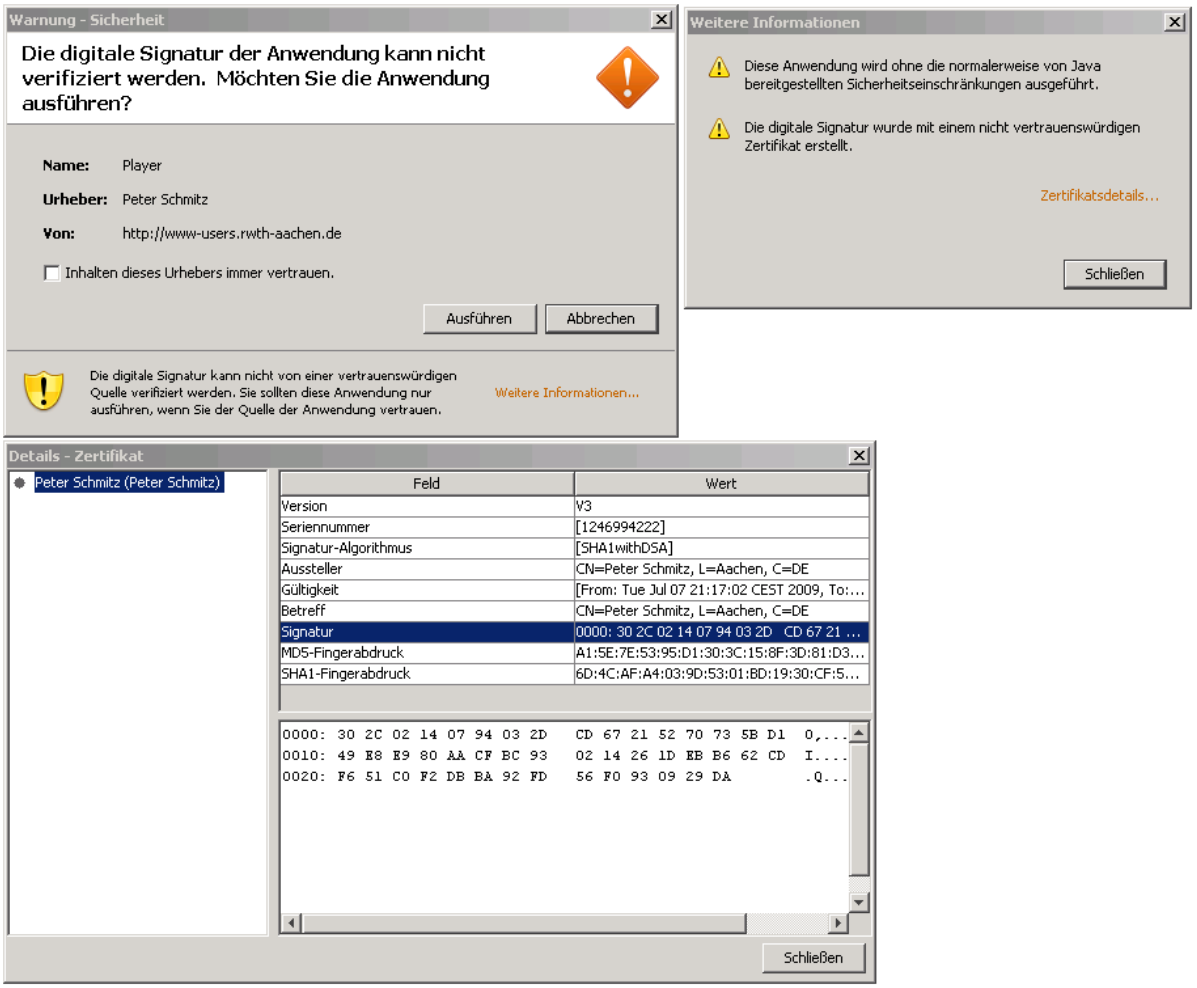


Abbildung 3.3: Beispielhafte Nutzerinteraktion mit Java Webstart: Zertifikatskette und Entscheidung über Türsteherprogrammausführung

3.3.3 Sandbox für einen Datenverarbeiter

Ein weiteres schon in Kapitel 2.4 und Kapitel 3.1.1 erwähntes entscheidendes Sicherheitsmerkmal ist die Kontrolle des Datenverarbeiters durch den Türsteher, um Seitenkanäle zu unterbinden. Bei Java wird dies durch Ausführen der konkreten Datenverarbeitungsprogramminstanz in einer sogenannten *Sandbox* erreicht. Dieser Mechanismus wird von der Java Plattform bereitgestellt (siehe [san]). Der sogenannte *Security Manager* prüft bei jedem Zugriff des Datenverarbeiters auf sicherheitsrelevante Ressourcen, ob dieser erlaubt ist und gewährt den Zugriff nur bei entsprechender Berechtigung. Anhand von *policies* werden Berechtigungen vor dem Start des Datenverarbeiters festgelegt, sind aber für jeden Datenverarbeiter gleich. Der Türsteher startet jede Datenverarbeitungsinstanz in einem neuen Prozess, um dabei die Berechtigungen für die gesamte Prozesslaufzeit fest vorzugegeben, indem eine Datei mit allen Berechtigungen (*Policy File*) als Startparameter übergeben wird:

```
java -Djava.security.manager -Djava.security.policy=policyfile SomeApp
```

Die Berechtigungen eines Datenverarbeiters beschränken sich auf das Aufbauen einer Netzwerkverbindung zum Türsteher, der ebenfalls auf der lokalen Maschine läuft und auf einem bestimmten Port Verbindungen entgegennimmt, um wie in Kapitel 3.2.4 beschrieben einen Netzwerkkanal aufzubauen. Daher ergibt sich folgende Berechtigungsdatei (wobei der Port noch ersetzt werden muss), die kurz vor dem Start eines Datenverarbeiters in einem temporären Verzeichnis angelegt bzw. auf korrektes Bestehen überprüft wird:

```
grant {
   permission java.io.SocketPermission "localhost:port", "connect, resolve";
};
```

Eventuell sind noch weitere Berechtigungen nötig, so daß es nicht bei der einen Berechtigung bleibt. Diese sollten aber nicht sicherheitsrelevant sein, wie zum Beispiel:

```
grant {
  permission java.lang.reflect.ReflectPermission "suppressAccessChecks";
};
```

3.3.4 TLS-SRP zwischen Türsteher und Datenhaltung

In Bezug auf Benutzer, Türsteher und Datenhalter liegt eine besondere Situation vor: Stellvertretend tritt der Türsteher für den gerade angemeldete Benutzer als Client gegenüber dem verwendeten Datenhalter (Server) auf. Dabei verwendet der Benutzer einen Benutzernamen (Loginnamen) und ein Passwort.
Als sicherheitsrelevante Zielsetzung in diesem Kapitel soll einerseits die Verbindung zwischen Türsteher und Datenhalter abhörsicher gemacht werden, um keine Datenpaket-IDs preiszugeben und andererseits ist die Authentifizierung des Benutzers beim Datenhaltern nötig, damit letzterer Abrechnungszwecke verfolgen kann

und nicht willkürlich von Fremden mit Daten überhäuft wird. Obwohl alle Datenpakete verschlüsselt beim Datenhalter abgelegt sind, ist eine zusätzliche Authentifizierung des Datenhalters sinnvoll, denn diese gibt die Gewissheit, daß Konsistenzprobleme oder vorgetäuschter Datenverlust durch einen böswilligen Datenhaltern vermieden wird. Außerdem soll *Perfect Forward Secrecy* (PFS) als weitere Anforderung erfüllt sein, um auch alte Sitzungen gegen Attacken schützen zu können.
Diese drei Anforderungen werden mittels *Transport Layer Security* (TLS) [TLSa] umgesetzt. TLS bietet eine bidirektionale, verschlüsselte, integritätsgeschütze Verbindung für die Anwendungsebene mit optionaler Authentifizierung. Man unterscheidet dabei verschiedene Möglichkeiten einen gemeinsamen Schlüssel für die abhörsichere Verbindung zu erzeugen, als auch verschiedene Authentifizierungsmöglichkeiten, wobei (wie oben erwähnt) die gegenseitige Authentifizierung verwendet wird. Der Benutzer, durch den Türsteher vertreten, besitzt kein (selbst kein selbstsigniertes) Zertifikat, da der Türsteher zum Zeitpunkt des Anmeldevorgangs nur Kenntnis vom Benutzernamen und Passwort hat. Insbesondere hat der Türsteher keinen Zugriff auf ein sogenanntes Key-/Certificate-Store und es steht ihm auch kein externes Medium mit dem asymmetrischen Schlüsselpaar des Benutzers zur Verfügung. Daher wird auf Zertifikate verzichtet. In diesem Falle kann bei (nicht-erweitertem) TLS nur die anonyme Diffie-Hellman Variante angewendet werden. Zwar bietet diese Variante PFS, aber leidet unter dem Man-in-the-Middle Problem und ist somit anfällig für aktive Angriffe.
Die Kenntnis von Benutzernamen und Passwort legen Erweiterungen von TLS als Lösung nahe, bei denen die gegenseitige Authentifizierung mittels Kenntnis eines *Pre Shared Secrets* durchgeführt wird. Im Folgenden werden die zwei TLS-Erweiterungen TLS-PSK (*Pre Shared Key*) [TLSd] und TLS-SRP (*Secure Remote Password*) [TLSb] voneinander abgegrenzt und eine Entscheidung für TLS-SRP begründet:
Die Erweiterung TLS-PSK setzt als gemeinsames, im Vorfeld ausgetauschtes Geheimnis einen *Pre Shared Key* (PSK) ein, wohingegen TLS-SRP einen *Password-Verifier* verwendet, um Türsteher und Datenhalter gegenseitig zu authentifizieren. Von den drei TLS-PSK Varianten kommt die RSA-Variante wegen fehlender Zertifikate und ebenso wie die Standard TLS-PSK Variante wegen fehlendem PFS nicht in Frage. Daher bleibt nur die Diffie-Hellman Variante, die zwar PFS bietet, aber keinen Schutz vor aktiven Angreifern ([TLSd] Section 7.2). TLS-SRP bietet hingegen zusätzlich auch Schutz gegen aktive Angreifer und dank des Diffie-Hellman ähnlichen Vorgehens auch PFS. Außerdem benötigt es in der Standardvariante keine Zertifikate. Die gegenseitige Authentifizierung wird nur mittels Kenntnis des *Password Verifier*s erreicht ([TLSb] Section 2.7). Es sei zu erwähnen, daß eine geringe Entropie im Benutzernamen und Passwort im Vergleich zu PSK dennoch eine hohe Entropie im erzeugten gemeinsamen Schlüssel zur Folge hat und somit online und offline Brute-Force Angriffe schwierig bleiben. ([SRPb] Section 7.2)
TLS-SRP erfüllt alle gestellten Anforderungen und wird somit für die Kommunikation und Authentifikation zwischen Türsteher und Datenhalter eingesetzt. Die einzelnen Schritte des Verbindungsaufbaus, Schlüsselaustausches und der Authentifizierung werden in [TLSb] beschrieben.

3.3.5 Registrierung, Schlüssel und Authentifizierung

Im Folgenden wird auf die Benutzerregistrierung beim Türsteher, also die Einrichtung eines Benutzer, die verschiedenen abgeleiteten Schlüssel, sowie auf den Authentifizierungsvorgang eines Benutzers beim Türsteher eingegangen.
Dabei wird eine Schlüsselableitungsfunktion *KDF* mit beliebig vielen Parametern verwendet, die aus ihrer Eingabe eindeutig einen symmetrischen Schlüssel erzeugt. Als vergleichbares Vorgehen läßt sich [PBK] heranziehen. Da eine kleine Änderung einer der Eingabeparameter eine große Änderung des Schlüssels bewirken soll, bietet sich die kryptographische Hashfunktion *SHA* über die Konkatenation der Parameter als *KDF* an.
Desweiteren sei angemerkt, daß der Türsteher bezüglich Schlüssel, Registrierungs- oder Authentifizierungsinformationen nur im Arbeitsspeicher zustandvoll ist. Das heißt, diese Informationen sind nur zur Türsteherlaufzeit im Arbeitsspeicher vorhanden und werden ausschließlich in der Datenhaltung persistiert, was im Folgenden beschrieben wird.

Benutzerregistrierung

Die Registrierung eines Benutzers verläuft zweistufig: Im ersten Schritt registriert sich der Benutzer beim Türsteher, also einer gerade ausgeführten Türsteherinstanz. Der Benutzer wählt einen Benutzer-/Loginnamen *Loginname-TS* und Passwort *Pwd* seiner Wahl und gibt diese beim Türsteher ein. Außerdem gibt er im Falle einer Infrastruktur, die mehrere Datenhaltungen unterstützt, eine eindeutige Identifikation *DH-ID* eines Datenhalters an, bei dem die Nutzerdaten abgelegt werden sollen. Je nach Szenario, wenn beispielsweise nur ein Datenhalter angedacht ist, kann diese *DH-ID* auch fest im Türsteher vorgegeben sein. Der Benutzername und das Passwort werden daraufhin auf eine bestimmte Länge aufgefüllt bzw. beschnitten. Aus diesen Eingaben wird deterministisch ein symmetrischer Schlüssel *UserPwdKey* abgeleitet:

$$UserPwdKey = KDF(Loginname\text{-}TS,\ Pwd,\ \text{„UserPwdKey“})$$

Im zweiten Schritt wird aus dem Loginnamen bezüglich des Türstehers ein Loginname für den Datenhalter abgeleitet:

$$Loginname\text{-}DH\text{-}ID = SHA(Loginname\text{-}TS,\ DH\text{-}ID,\ \text{„Loginname-DH“})$$

Der Türsteher berechnet analog dem Vorgehen in [SRPb] Section 3 den *Password-Verifier* aus einem zufälligen Salt *Salt* und dem *Pwd*. Der *Password-Verifier*, zugehöriges *Salt* und *Loginname-DH-ID* werden angezeigt und der Benutzer aufgefordert sich damit bei dem Datenhalter mit der *DH-ID* zu registrieren. Dies geschieht beispielsweise über eine Webseite des Datenhalters, auf der der *Loginname-DH-ID*, *Passwort-Verifier* und zugehöriges *Salt* eingegeben werden (beispielsweise als konkatenierte, base64kodierte Zeichenkette, da alle Elemente feste Längen haben und somit wieder trennbar sind). Der Datenhalter prüft dabei, ob unter *Loginname-DH-ID* bereits ein Benutzer registriert ist und verweigert in diesem Falle eine Registrierung, um Überschreiben von Datenpaketen durch den neuen Benutzer zu verhindern.

Desweitern liegt es im Ermessen des Datenhalters, noch weitere Informationen über den Benutzer abzufragen, die für ein Abrechnungsmodell für die Nutzung des Datenhalters nötig sind. Zu beachten ist, daß die Verbindung zum Server der Webseite auch abgesichert werden muss, da der *Password-Verifier* nicht in falsche Hände fallen darf, weil sonst ein Angreifer sich als Datenhalter mit *DH-ID* ausgeben kann. Dabei kann man aber auf bewährte Techniken wie *HTTPS* zurückgreifen, die eine serverseitige Authentifizierung gewährleisten und vom Browser unterstützt werden. Der Datenhalter speichert die Zuordnung von *Loginname-DH-ID* zu *Salt* und *Password-Verifier* je registriertem Benutzer und eventuell zusätzlicher Angaben zu Abrechnungszwecken. Nach erfolgreicher Registrierung bei einem Datenhalter kehrt der Benutzer zurück zur immer noch aktiven Türsteherinstanz, die die Registrierung wie folgt abschließt:
Ein dauerhafter, persönlicher und symmetrischer Schlüssel *UserMasterKey* wird erzeugt. Dieser wird zufällig mit Hilfe eines Pseudo Random Number Generators (PRNG) generiert (Seed ist die aktuelle Systemzeit mit möglichst Nanosekundenpräzision) und ist damit sehr wahrscheinlich einmalig, vgl. Geburtstagsparadoxon. Eine erfolgreiche Registrierung beim verwendeten Datenhalter vorausgegangen, wird nun eine Verbindung zum selbigen wie in Kapitel 3.3.4 beschrieben aufgebaut. Der *UserMasterKey* wird mit dem *UserPwdKey* symmetrisch verschlüsselt und in einem Datenpaket namens *UserMasterKey-Token* in der Datenhaltung abgelegt. Die zugehörige ID dieses Datenpakets wird aus *UserPwdKey* abgeleitet:

$$ID_{UserMasterKey-Token} = KDF(UserPwdKey,\ \text{„UserMasterKey-Token“})$$

Damit kann man nur mittels des richtigen Loginnamens und des richtigen Passwortes den zugehörigen *UserMasterKey* auslesen oder gar löschen.

Abgeleitete Benutzerschlüssel

Aus dem *UserMasterKey* werden zwei weitere Schlüssel deterministisch abgeleitet, die im Kapitel 3.3.6 Verwendung finden:

- *UserEncryptionKey* : Dieser wird verwendet, um Inhaltsdaten zu ver- und entschlüsseln, also alle Datenpakete die zwischen Datenverarbeitung und Datenhaltung über den Türsteher übertragen werden.

  $$UserEncryptionKey = KDF(UserMasterKey,\ \text{„UserEncryptionKey“})$$

- *UserSaltingKey* : Dieser wird genutzt, um IDs verschiedener Benutzer voneinander abzugrenzen.

  $$UserSaltingKey = KDF(UserMasterKey,\ \text{„UserSaltingKey“})$$

Authentifizierung eines Benutzers

Die Authentifizierung eines Benutzers gegenüber dem Türsteher geschieht durch Eingabe von Loginnamen und Passwort beim Türsteher. Anschließend überprüft

der Türsteher die Eingabe folgendermaßen:
Wie in Kaptiel 3.3.4 beschrieben wird laut dem Vorgehen in [SRPb] Section 3 mittels TLS-SRP eine Verbindung zu einem Datenhalter aufgebaut, eine gegenseitige Authentifizierung durchgeführt und die Verbindung abgesichert. Falls der Verbindungsaufbau durch den Datenhalter verweigert wird, ist der Benutzer nicht beim Datenhalter registriert und ein Authentifizierung scheitert. Im Erfolgsfall ist es anschließend möglich, das *UserMasterKey-Token* auszulesen. Dabei ergibt sich die ID des *UserMasterKey-Token*s wie bei der Registrierung beschrieben und es kann das zugehörige Datenpaket angefordert werden. Falls das Datenpaket nicht existiert, ist diese Kombination von Loginname und Passwort wahrscheinlich nicht abschließend über den Türsteher registiert worden. Falls es vorhanden ist, wird der Inhalt mit *UserPwdKey* symmetrisch entschlüsselt und bei Erfolg erhält man den *UserMasterKey* des Benutzers. Bei unwahrscheinlichem Mißerfolg der Entschlüsselung ist das Datenpaket zerstört oder überschrieben worden, da *UserPwdKey* sowohl für die ID als auch für die Verschlüsselung benutzt worden ist. Dieser *UserMasterKey* wird wie in Kapitel 3.2.4 beschrieben dem Netzwerkkanal zugeordnet. Beim Beenden der Datenverarbeitungssitzung wird diese Zuordnung aufgehoben und alle Schlüssel für diese Sitzung im Arbeitsspeicher verworfen.
Wie aus dem Vorgang zu erkennen ist, gibt es bei Verlust des Benutzernamens und/oder des Passwortes keine Möglichkeit, den *UserMasterKey* auszulesen, wodurch alle Daten dieses Benutzers in der Datenhaltung unerreichbar werden.
Außerdem wird nicht der *UserPwdKey* als *UserMasterKey* verwendet, damit ein Benutzer auch sein Passwort ändern kann, ohne das alle von *UserMasterKey* abhängenden Verschlüsselungen neu verschlüsselt werden müssen.

3.3.6 Sicherheitsrelevante Anpassungen eines Datenpakets

Es existieren verschiedene Möglichkeiten der Kommunikation, also der Übermittlung von Informationen von der Datenverarbeitung über den Türsteher zur Datenhaltung mittels der beschriebenen Datenpakete. Die offensichtlichsten Möglichkeiten finden über den Inhalt des Datenpakets statt. Somit müssen Maßnahmen getroffen werden, diese Kommunikationsmöglichkeiten zu unterbinden. Auf diese wird nun eingangen:

Datenpaket-ID

Der Datenverarbeiter wählt eine beliebige textuelle Beschreibung eines Datenpakets für die Adressierung der Datenpakete und bildet diese auf den Wertebereich für IDs ab, um die Datenschnittstelle nutzen zu können. Der Wertebereich richtet sich nach dem Wertebereich der Abbildung der IDs beim Türsteher bezüglich der Datenhaltung. Im Gegensatz zur Datenschnittstelle zwischen Datenverarbeiter und Türsteher dürfen dort keine Informationen mittels der ID übertragen werden. Als Abbildung wird die *kryptographische Hashfunktion SHA* benutzt. Somit kann die Datenhaltung mit einer ID keine Rückschlüsse auf die zugehörige ID der Datenverarbeitung schließen, da die Hashfunktion eine Einwegfunktion ist. Auch Informationsübertragung über verschiedene ID-Längen ist aufgrund der festen Länge der Hashwerte ausgeschlossen. Es bietet sich an, diese Hashfunktion auch seitens des Datenverarbeiters

einzusetzen, um die textuelle Beschreibung in den gleichen Wertebereich abzubilden, aber dies liegt im Zuständigkeitsbereich des Datenverarbeiterentwicklers. So könnte dieser auch trivialerweise die Datenpaketbeschreibung auf die Länge des Hashwertes beschränken und keine Abbildung verwenden. Festzuhalten ist, daß der Wertebereich für IDs aufgrund der Wiederverwendung der Datenschnittstelle immer gleich ist.
Außerdem müssen die IDs verschiedener Benutzer noch voneinander abgegrenzt werden, denn zwei Benutzer könnten die gleich Datenpaketbeschreibung als Ausgangspunkt für die ID nutzen, damit den gleichen Hashwert als ID erzeugen und so das Datenpaket des anderen Benutzers überschreiben. Diese Abgrenzung geschieht, indem man in die Hashfunktion beim Türsteher den benutzerspezifischen Schlüssel *UserSaltingKey* einfließen läßt. Ein weiterer Grund für den Einfluß dieses Schlüssels ist, daß so von einem Angreifer keine Tabellen mit Paaren von ungehashten und gehashten IDs (vgl. [Oec03] bzgl. Rainbow Tables) sinnvoll eingesetzt werden können, mit denen eine Kommunikation möglich wäre.

Abbildung 3.4 skizziert schematisch die Abbildungen bezüglich der IDs und die Anpassung mit dem *UserSaltingKey*.

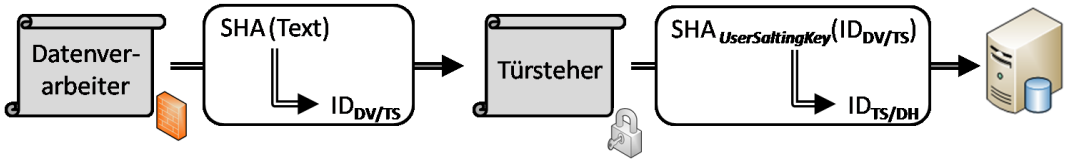


Abbildung 3.4: Anpassungen der Datenpaket-ID von Datenverarbeitung(DH) über Türsteher(TS) zur Datenhaltung(DH)

Datenpaket-Inhalt

Die offensichtlichsten Möglichkeiten der Informationsübermittlung, also der Kommunikation zwischen Datenverarbeiter und Datenhaltung, sind durch den Inhalt eines Datenpakets gegeben. Der Türsteher unterbindet die Informationsübermittlung durch die folgenden Anpassungen.

Ver- und Entschlüsselung : Bei einer Anfrage bzgl. der Datenhaltungsschnittstelle eines Datenverarbeiters an den Türsteher über die Netzwerkschnittstelle wird der diesem Netzwerkkanal zugeordnete *UserEncryptionKey* verwendet, um den symmetrischen Schlüssel *TokenEncryptionKey* speziell für das aktuelle Datenpaket abzuleiten:

$$TokenEncryptionKey = KDF(UserEncryptionKey, \text{ID})$$

Die ID ist die ID der Anfrage nach oben beschriebener Anpassung. Der *TokenEncryptionKey* wird verwendet, um den Inhalt des Datenpaketes erstens vom Datenverarbeiter zur Datenhaltung symmetrisch zu verschlüsseln und zweitens von der Datenhaltung zum Datenverarbeiter symmetrisch zu entschlüsseln. Das bedeutet, daß der Inhalt jedes Datenpakets insbesondere auch in Abhängigkeit der zugehörigen ID verschlüsselt wird. Somit werden zwei Datenpakete mit gleichem Inhalt

aber verschiedenen Ausgangs-IDs (vom Datenverarbeiter bestimmt) unterschiedlich verschlüsselt.

Längenanpassung : Im Gegensatz zu festen Längen der IDs sind die Inhalte der Datenpakete unterschiedlich lang, weil sie vom Datenverarbeiter bestimmt werden. Da auch über eine Folge von verschiedenen Längen der Inhalte der aufeinanderfolgenden Datenpakete kommuniziert werden kann, können Längenanpassungen vorgenommen werden. Mögliche Denkansätze werden als Erweiterungen im Kapitel 6 beschrieben.

4

Sicherheitsanalyse

Dieses Kapitel beleuchtet verschiedene Angriffsmöglichkeiten auf den vorgestellten Lösungsansatzes aus Kapitel 3. Dabei sei als Ausgangspunkt das Vertrauen in die lokale Machine (vgl. Kapitel 3.3.1) und die Java Plattform festzuhalten. Im folgenden wird zwischen passiven oder aktiven und internen oder externen Angriffen unterschieden:

- Einem *passiv*en Angreifer ist es nur möglich, einen Kommunikatiosnkanal abzuhören und bedroht somit nur die Vertraulichkeit der übermittelten Daten.
- Ein *aktiv*er Angreifer hingegen kann darüber hinaus Daten löschen, hinzufügen oder in irgendeiner anderen Art und Weise die übermittelten Daten des Kommunikationskanals ändern. Dadruch ist Authentifikation, Datenintegrität und Vertraulichkeit der Daten gefährdet.
- Ein *intern*er Angriff meint eine Angriffsmöglichkeit durch einen der Akteure Datenverarbeiter, Türsteher oder Datenhalter.
- Bei einem *extern*er Angriff hingegen ist ein außenstehender Angreifer gemeint.

In Form einer Auflistung wird jeweils eine Bedrohung dargestellt, die eine Angriffsmöglichkeit bieten könnte, daraufhin eine Einschätzung gegeben, also welche Konsequenzen diese Bedrohung mit sich bringt; anschließend werden jeweils mögliche Angriffe aufgezeigt und zuletzt beschrieben, inwiefern der vorgestellte Lösungsansatz dagegen schützt. Diese *nie vollständige* Auflistung wird nach passiven oder aktiven und internen oder externen Angriffen unterteilt:

4.1 Angriffsmöglichkeiten

4.1.1 Intern, Aktiv

- **ID-Wahl (aktiv):**
  - *Bedrohung:* Jeder Benutzer hat freie ID-Wahl.

  - *Einschätzung:* Zwar wird diese ID benutzerspezifisch (*UserSaltingKey*) gehasht, aber Kollisionen können dennoch benutzerübergreifend auftreten.
  - *Mögliche Angriffe:* Löschen bzw. Überschreiben von Daten anderer Benutzer in der Datenhaltung durch *aktives* Ausprobieren vieler zufälliger IDs.
  - *Wie schützt der Lösungsansatz:* Aufgrund der Länge der entstehenden Hashes wird eine Kollision nicht unmöglich, aber sehr unwahrscheinlich und damit annehmbar vernachlässigbar (vgl. Kapitel 3.2.2). Beim Versuch Datenpakete mit vielen zufälligen IDs zu löschen, ist es sehr unwahrscheinlich überhaupt ein vorhandenes Datenpaket zu treffen.

- **Konspiration**
  - *Bedrohung:* Konspirierende Datenhaltung und Datenverarbeiter.
  - *Einschätzung:* Kommunikationsversuche der beiden: Schon ein unidirektionaler Kommunikationskanal von Datenverarbeiter zur Datenhaltung umginge eine Verschlüsselung der Datenpakete und würde persönliche Informationen der nicht vertrauenswürdigen Datenhaltung offenbaren.
  - *Mögliche Angriffe:*
    1. Datenverarbeiter bestimmt Datenpaketinhalt.
    2. Datenverarbeiter bestimmt Datenpaketinhaltslänge und durch eine Folge von Längen kann ein Kommunikationskanal aufgebaut werden.
    3. Datenverarbeiter bestimmt Datenpaket-ID.
    4. Wiederholung gleicher Inhalte als Datenübertragungsmittel.
    5. Zeitattacken: durch die Folge von Pausen zwischen zwei aufeinanderfolgenden Datenpaktanfragen kann ein Kommunikationskanal aufgebaut werden.
  - *Wie schützt der Lösungsansatz:*
    1. Die symmetrische Verschlüsselung mit nur dem Türsteher bekannten Schlüssel verhindert eine direkte Kommunikation.
    2. Ansätze zur Obfuskierung der Längen wird als Erweiterung in Kapitel 6 angedacht.
    3. Hashing der ID mit Hilfe von *UserSaltingKey* bildet die Ausgangs-ID nicht nur auf einem dem Datenverarbeiter unbekannte ID ab, sondern verhindert auch die Kollision bei gleichen Ausgangs-IDs zwei verschiedener Benutzer.
    4. Zwei Datenpakete werden trotz gleichem Inhalt aber mit verschiedenen IDs werden auch mit verschiedenen Schlüsseln (*TokenEncryptionKey*) verschlüsselt.
    5. Zeitattacken werden nur erschwert, aber nicht verhindert, indem die Datenpaketanfragen aller laufenden Datenverarbeiter vermischt wer-

den. (FIFO) Insbesondere nur, wenn auch mehrere Datenverarbeiter auf der aktuellen Maschine laufen.

- **Datenintegrität**
  - *Bedrohung:* Der Datenhalter hat Zugriff auf alle Datenpaketinhalte.
  - *Einschätzung:* Obwohl jeder Datenpaketinhalt in der Datenhaltung verschlüsselt ist, könnte dieser nach Belieben vom Datenhalter geändert werden.
  - *Mögliche Angriffe:*
    1. Zufälliges Ändern eines Datenpaketinhaltes durch den Datenhalter.
    2. Der Datenhalter vertauscht den Inhalt zweier Datenpakete (die vom gleichen Benutzter stammen) vertauschen.
  - *Wie schützt der Lösungsansatz:*
    1. Eine bei der Verschlüsselung optional verwendete Integritätsprüfung verhinderte ein zufälliges Ändern.
    2. Obwohl die Integrität der vertauschten Datenpakete jeweils erhalten bleibt, wird ein Vertauschen dennoch erkannt, da die ID in die Ableitung des *TokenEncryptionKey*s mit einfließt.
- **Doppelauthentifikation des Benutzers**
  - *Bedrohung:* Insbesondere bei nicht versierten Benutztern lassen sich ähnlich dem *Phishing* erneut die Authentifizierungsdaten durch einen böswilligen Datenverarbeiter nach Abfrage beim Türsteher erfragen.
  - *Einschätzung:* Damit hat der Datenverarbeiter genau wie der Türsteher Kenntnis vom *UserMasterKey* des aktuellen Benutzers.
  - *Mögliche Angriffe:* Die Kenntnis dieses Hauptschlüssels bietet keinen Vorteil für Daten des aktuellen Datenverarbeiters. Aber es könnte versucht werden, Daten des gleichen Benutzers aber anderer Datenverarbeitungsprogramme zu entschlüsseln.
  - *Wie schützt der Lösungsansatz:* Die Sandbox, in der jeder Datenverarbeiter läuft, verhindert einen direkten Zugriff auf die Datenschnittstelle der Datenhaltung, über die die Datenpakete anderer Datenverarbeitungsprogramme zugreifbar sind. Der Zugriff eines jeden Datenverarbeiters ist auf die Datenschnittstelle des Türstehers eingeschränkt und dort sind alle IDs bereits benutzerspezifisch angepasst und Datenpakete bereits entschlüsselt, bzw. werden dort erst verschlüsselt.

4.1.2 Intern, Passiv

- **ID-Wahl (passiv):**
  - *Bedrohung:* Jeder Benutzer hat freie ID-Wahl.

  - *Einschätzung:* Zwar wird diese ID benutzerspezifisch (*UserSaltingKey*) gehasht, aber Kollisionen können dennoch benutzerübergreifend auftreten.
  - *Mögliche Angriffe:* Löschen bzw. Überschreiben von Daten anderer Benutzer in der Datenhaltung durch *unabsichtliche* Kollisionen.
  - *Wie schützt der Lösungsansatz:* Aufgrund der Länge der entstehenden Hashes wird eine Kollision nicht unmöglich, aber sehr unwahrscheinlich und damit annehmbar vernachlässigbar (vgl. Kapitel 3.2.2).

4.1.3 Extern, Aktiv

- **ID-Kenntnis**
  - *Bedrohung:* Kenntnis von IDs (anderer Benutzer)
  - *Einschätzung:* Der einzige Weg, Kenntnis von IDs zu erlangen, ist die Kommunikation zwischen den Türstehern und der Datenhaltung abzuhören (da von Vertrauen in die lokale Maschine ausgegangen wird).
  - *Mögliche Angriffe:* Mit der Kenntnis von IDs können die zugehörigen Datenpakete auch verändert (gelöscht) werden.
  - *Wie schützt der Lösungsansatz:* Da die Kommunikation zwischen einem Türsteher und Datenhaltung mittels *TLS-SRP* abgesichert wird, können Angreifer dazwischen keine IDs auslesen.
- **Sandbox**
  - *Bedrohung:* Die Berechtigungen für die Sandbox des Datenverarbeiters werden in einer Datei (*Policy File*) auf dem lokalen Rechner abgelegt (vgl. Kapitel 3.3.3).
  - *Einschätzung:* Das Anlegen der Berechtigungsdatei (*Policy File*) / deren Überprüfung und der Start des Datenverarbeiungsprozesses (bei dem diese Datei zu Beginn vom Security Manager der virtuellen Maschine ausgelesen wird) werden nicht atomar ausgeführt. Daher besteht keine Garantie, daß die Berechtigungsdatei beim Start die gleiche wie beim Anlegen ist.
  - *Mögliche Angriffe:* Ein Angreifer in Form eines lokal laufenden Programms könnte zwischen Anlegen und Start die Berechtigungsdatei austauschen.
  - *Wie schützt der Lösungsansatz:* Der vorgeschlagene Ansatz geht von Vertrauen in die lokale Maschine aus, also inbesondere, daß keine Programme bösartiger Natur laufen.
- **Türsteherversion**
  - *Bedrohung:* Von wesentlicher Bedeutung ist die korrekte Version des ausgeführten Türstehers.

- *Einschätzung:* Ein unbemerkter Austausch der Türsteherprogrammversion würde alle Sicherheitsmerkmale umgehen können.
- *Mögliche Angriffe:* Das Türsteherprogramm könnte von einem Angreifer gegen ein bösartiges ausgetauscht werden, indem die URL des Türstehers in der *JNLP*-Datei ausgetauscht wird, beispielsweise entweder vom Anwendungsentwickler oder kurz vor dem Start derselben beispielsweise beim Herunterladen.
- *Wie schützt der Lösungsansatz:* Durch Interaktion mit dem Benutzer vor dem Start des Türstehers soll der Benutzer dazu angehalten werden, zu überprüfen, ob eine vertrauensvolle Version des Türstehers vorliegt. Anhand der Zertifikatskette kann nachvollzogen werden, wer diese Version zertifiziert hat (vgl. Kapitel 3.3.2). Dieses Vorgehen wird von der Java Plattform bereitgestellt, der wiederum vertraut wird.

4.1.4 Extern, Passiv

- **Identität des Benutzers**
  - *Bedrohung:* Während des Aufbaus einer TLS-SRP Verbindung zwischen Türsteher und Datenhalter wird der Loginname im Klartext übertragen.
  - *Einschätzung:* Durch mitlesen des Loginnamens beim Verbingungsaufbau, läßt sich der entstehnde Kommunikationskanal einem Benutzer zuordnen.
  - *Mögliche Angriffe:* Verkehrsaufkommen und Statistiken zum Nutzerverhalten sind erstellbar.
  - *Wie schützt der Lösungsansatz:* Der vorgestellte Lösungsansatz schützt nicht dagegen, da dies eine Eigenschaft des verwendeten TLS-SRP Protokolls ist. Dennoch sei darauf hingewiesen, daß es sich dabei nicht um den selbstgewählten Benutzernamen handelt, da stattdessen der Loginname-DH-ID verwendet wird. Außerdem erschwert ein wechselnder Benutzerstandort die Erstellung jener Statistiken.

5

Beispielimplementierung anhander der Programmiersprachen Java und Scala

5.1 Grundlagen

Dieses Kapitel beschreibt die für die Beispielimplementierung verwendeten Grundlagen. Als Basisprogrammierplattform kommt die Java Platform, Standard Edition (Java SE) zum Einsatz. Insbesondere wird Java Webstart hervorgehoben, welches die Auslieferung von Javaprogrammen sehr vereinfacht und gleichzeitig nötige Sicherheitsmechnismen zur Verfügung stellt. Darauf aufsetzend werden die Vorzüge der Programmiersprache Scala beschrieben und speziell auf das massiv zum Einsatz kommende Aktorenrahmenwerk eingegangen. Anschließend wird das eigenentwickelte Hotswapper-Rahmenwerk beschrieben, das das Scala Aktorenrahmenwerk sinnvoll erweitert. Das Kapitel schließt mit einer kurzen Beschreibung der verwendeten externen Bibliotheken ab.

5.1.1 Java Plattform

Die Java Platform, Standard Edition (Java SE) von Sun Microsystems (im Folgenden in Version 1.6.0_17) umfasst unter anderem die Java Virtual Machine (JVM), also die Laufzeitumgebung für Zwischencode den sogenannten Bytecode, der aus Java-Quelltext kompiliert wird. Außerdem enthält sie verschiedene Bibliotheken als Schnittstellen in Form eines sogenannten Application Programming Interface (*API*), die Standardfunktionen wie beispielsweise Datei- oder Netzwerkzugriffe bereitstellen. Die Kombination einer Laufzeitumgebung für Zwischencode und betriebssystemunabhängiger APIs machen Java-Programme nur abhängig von einer implementierten Laufzeitumgebung für eine spezielle Kombination aus Computerplattform und Betriebssystem, die für viele Systeme darunter auch mobile Plattformen derzeit existiert. Java-Programme werden in der Java Programmiersprache verfaßt. Die Java Programmiersprache (spezifiziert in [GJSB05]) ist eine universelle, nebenläufige, klassenbasierte, objektorientierte Programmiersprache. Sie ist angelehnt an C und ist statisch typisiert.

Die Entscheidung für Java im Rahmen der Beispielimplementierung ist insbesondere an persönlichen Präferenzen, aber auch der Plattformunabhängigkeit von Java orientiert. So ließe sich diese Implementierung mit vergleichsweise wenig Aufwand auf beispielsweise mobile Endgeräte portieren.

Webstart

Als eine hervorzuhebende Technik der Java Plattform ist *Java Webstart* [webb] zu nennen. Diese ermöglicht es, Java-Anwendungen über das Internet unabhängig vom Browser zu starten und in der lokalen Java Virtual Machine auszuführen. Dabei wird die Anwendung in Form einer JAR-Datei auf einem Server zur Verfügung gestellt, diese bei Bedarf auf den Benutzerrechner heruntergeladen und dort in einem Zwischenspeicher für die nächste Ausführung bereitgehalten. Vor jeder Ausführung wird ein Abgleich der lokalen Version mit der auf dem Server durchgeführt, so daß immer mit der aktuellen Version gearbeitet wird. Die Beschreibung aller nötigen Angaben für die Ausführung einer Webstartanwendung wird durch das *Java Network Launching Protocol (JNLP)* geregelt. JNLP verwendet das XML-Format für JNLP-Dateien, die Informationen wie den Ablageort der JAR-Dateien, die Hauptklasse oder zusätzliche Laufzeiteinstellungen, als auch Anwendungsparameter enthalten. Außerdem kann in der JNLP-Datei auch eine spezielle Version der benötigten Java Laufzeitumgebung angegeben werden, die vor Ausführung von Java Webstart bei Bedarf auf dem aktuellen System eingerichtet wird. Somit braucht ein Anwendungsentwickler insbesondere nicht auf verschiedene Javaversionen achten, sondern kann die bei der Implementierung Verwendete benutzen. Als Beispiel sei die JNLP-Datei für den Türsteher angegeben, die per Webstart gestartet wird.

```
<?xml version="1.0" encoding="utf-8"?>
<jnlp spec="1.0+" codebase="http://localhost/" href="tuersteher.jnlp">
  <information>
    <title>Tuersteher</title>
    <vendor>Peter Schmitz</vendor>
    <description>Tuersteher</description>
  </information>
  <security>
    <all-permissions />
  </security>
  <resources>
    <jar href="tuersteher.jar" />
    <j2se version="1.6+" href="http://java.sun.com/products/autodl/j2se"
                         initial-heap-size="64m" max-heap-size="320m"/>
  </resources>
  <application-desc>
    <argument>8888</argument>
    <argument>java-datenverarbeiter</argument>
    <argument>http://localhost/datenverarbeiter.jar</argument>
    <argument>localhost</argument>
    <argument>4444</argument>
```

```
    <argument>arg1</argument>
    <argument>arg2</argument>
  </application-desc>
</jnlp>
```

5.1.2 Scala

Scala [Sca] ist eine universelle Programmiersprache, die entwickelt wurde um gebräuchliche Programmiermuster in einer eleganten, präzisen und typsicheren Weise auszudrücken. Sie integriert reibungslos Merkmale von objektorientierten und funktionalen Sprachen. Dabei integriert sie sich in die Java Plattform derart, daß Scala-Quellcode Java-Code verwenden (aufrufen) kann (umgekehert auch via Reflections). Außerdem wird Scala-Quellcode zu Bytecode für die JVM kompiliert. Im Folgenden kommt Scala Version 2.7.7 final zum Einsatz. Hervorzuheben sind Paradigmen wie automatische Typinferenz bei statischer Typsierung, Funktionen als First Class Objects und Funktionen höherer Ordnung mit Currying, hierarchische Organisation von Klassen, Funktionen und Variablen, Case Classes und Pattern Matching, Mehrfachvererbung via linearisierter Traits, Closures, implizite Typkonvertierung und die in Version 2.8 zu erwartetenden Continuations. Auch durch viele kleine andere Verbesserungen wie z.B. nahtlose Integration von XML in Scala-Quellcode wird die Größe des Quellcodes typischerweise um Faktor zwei bis drei im Vergleich zu funktionsgleichem Java-Code reduziert und damit der Quellcode präziser, einfacher zu lesen und leichter wartbar. Viele Unternehmen, die Java für kritische Anwendungen einsetzen, wenden sich Scala zu, um die Produktivität der Entwicklung, die Skalierbarkeit und Verlässlichkeit der Anwendnung zu steigern. So zum Beispiel das soziale Netzwerk Twitter, das die kritische Infrastruktur des Backends von Ruby zu Scala wegen nötiger Skalierbarkeit portiert hat.

Aktoren Rahmenwerk

Als Basisrahmenwerk kommt das Aktorenrahmenwerk (siehe [HO06] und [HO08]) von Scala zum Einsatz. Die Grundidee von Aktoren ist die parallele Ausführung von Funktionseinheiten, den sogenannten Aktoren, die mit Hilfe von Nachrichten kommunizieren. Nachrichten können eines beliebigen Typs sein. Jeder Aktor hat eine Art Postfach für eingehende Nachrichten anderer Aktoren und kann Nachrichten in Postfächer anderer Aktoren (oder auch in sein eigenes) ablegen. Man spezifiziert je Aktortyp (Klasse), wie auf eine eintreffende Nachricht reagiert wird. Aktoren können dabei auf einzelne Threads je Aktor (auch mit Hilfe eines Threadpools) oder auf nur einen Thread abgebildet werden, so daß selbst einige tausende gleichzeitig aktive Aktoren effizient realisierbar sind.
Der große Vorteil von Aktoren ist die Nähe zur Anwendungslogik, insbesondere bei nebenläufigen Vorgängen; so inbesondere bei Anwendungen die auf verschiedene Prozesse und/oder Maschinen verteilt sind, was Nebenläufigkeit impliziert.
Die Scala API ergänzt die Aktoren, die alle einem Prozess laufen, um das Konzept der Remote-Aktoren. Das heißt, ein solcher Aktor kann an einen lokalen Netzwerkport gebunden werden und so eingehende Nachrichten von Aktoren, die auf anderen Ma-

schinen (oder Prozessen) laufen, empfangen. Diese verbinden sich mit einem Aktor unter einer bestimmten IP-Adresse und Port. Dabei werden insbesondere die Serialisierung und die Netzwerkübertragung von Nachrichten von dem Remote Aktoren Paket übernommen. Zusätzlich wird noch nach einer Identifikationen unterschieden, so daß verschiedene Aktoren unter der gleichen Adresse und Port erreichbar sind. Somit lassen sich verteilte Anwendungen auf Aktorenbasis realisieren.

5.1.3 Hotswapper-Rahmenwerk

Aufbauend auf dem Aktorenrahmenwerk, das von Scala bereitgestellt wird, kommt das eigenentwickelte Hotswapper-Rahmenwerk zum Einsatz. Denn es stellte sich heraus, daß ein entscheidender Nachteil der Aktoren ihre Zustandslosigkeit ist. Zwar lassen sich Zustände über Objektvariablen realisieren, aber insbesondere verschiedene Reaktionen auf eingehende Nachrichten je Zustand lassen sich nur unübersichtlich über Verzweigungskontrollstrukturen mit Objektvariablen realisieren. Daher kommt das Hotswapper-Rahmenwerk als Aufsatz auf das Aktorenrahmenwerk zum Einsatz, das diskrete Zustände für Aktoren bereitstellt, zu denen ein Aktor wechseln kann. Ein Aktor befindet sich immer genau in einem Zustand. Je Zustand wird definiert, wie auf eingehende Nachrichten reagiert wird.
Im Detail wird diese Funktionalität über sogennante Verhaltensweisen (Behaviour) gelöst. Eine solche Verhaltensweise bestimmt den Umgang mit bestimmten eingehenden Nachrichten. Dann lassen sich die Verhaltensweisen kombinieren und für jeden möglichen Aktorenzustand (State) eine Verhaltensweise als sogennantes Zustandsverhaltensweise (StateBehaviour) festlegen. Der Name Hotswap leitet sich von der Möglichkeit des Austausch des Verhaltens eines Aktors zur Laufzeit ab. Damit ist es sogar möglich, in einer ankommenden Nachricht eine neue Verhaltensweise zu kapseln und diese dann vom empfangenden Aktor als neue Zustandsverhaltensweise zu verwenden. Somit lassen sich Anwendungen ohne Neustart also zur Laufzeit unterbrechungsfrei mit neuem Programmcode konfigurieren.

5.1.4 Bibliotheken

Folgenden externe Bibliotheken sind zusätzlich zur Laufzeitumgebung von Java benutzt worden :

- SRP Protokoll Implementierung von Jordan Zimmerman vom 18.06.09. Siehe http://www.jordanzimmerman.com/index.php?n=3
- JavaBase64 Version 1.3.1: Eine Konvertierungsbibliothek, um ByteArrays base64 zu kodieren. Siehe www.sauronsoftware.it/projects/javabase64/
- Postgres JDBC4 Version 8.4-701: Die Java Database Connectivity(JDBC) Treiber für die Anbindung der Datenbank an den Java-Datenhalter. Beziehbar unter http://jdbc.postgresql.org/

5.2 Architektur

Im folgenden Kapitel wird der Aufbau und das Zusammenspiel der einzelnen Komponenten, sprich Datenverarbeiter, Türsteher und Datenhalter, beschrieben. Dabei wird besonders die bei der Aktorenfunktionsweise nötige Behandlung von Nachrichten betrachtet. Zunächst wird ein Überblick gegeben und anschließend auf die einzelnen Komponenten eingegangen.

5.2.1 Überblick

Grundsätzlich ist die Beispielimplementierung weitgehend in Scala umgesetzt. Bei grundlegenden Operationen, insbesondere bei den sicherheitsrelevanten Funktionen, wird auf die von Java bereitgestellten zurückgegriffen. Wie bei den Grundlagen erwähnt kommt das Aktorenrahmenwerk und dessen Aufsatz (Hotswapper) massiv zum Einsatz. Denn jede Komponente ist als Aktor, genauer als Remote-Aktor mit Hotswap-Funktionalität realisiert. Die Abbildung 5.1 zeigt einen Überblick aus Aktorensichtweise mit einer Unterscheidung nach Maschine, Prozessen und Aktoren. Beispielhaft werden die aktiven Aktoren im Türsteher und deren möglichen Zustände hervorgehoben. Eine genauere Betrachtung jeder Komponente folgt in den Unterkapiteln im Anschluß.

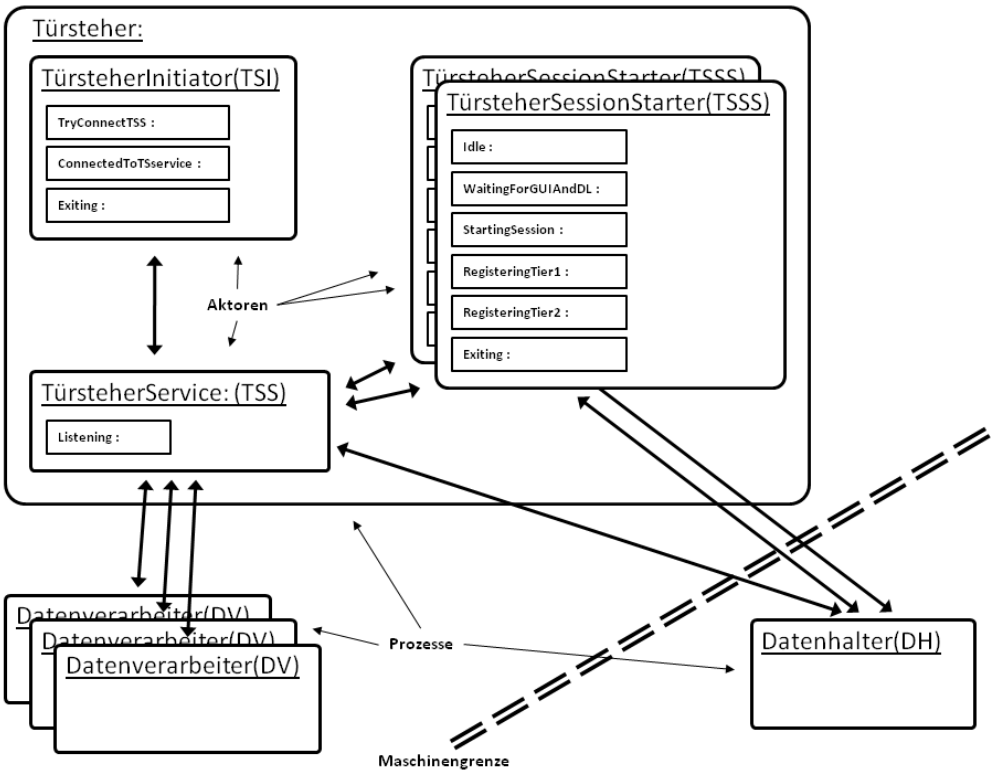


Abbildung 5.1: Prozess- & Aktorenüberblick auf die verwendeten Komponenten, speziell Türsteher

5.2.2 Türsteher

Der Türsteher ist ein per Webstart gestartetes Programm auf dem Anwendungsrechner. Es läuft mit vollen Berechtigungen in einem eigenen Prozess und besteht aus drei Aktoren, die im Folgenden vorgestellt werden:

Türsteher Initiator (TSI)

Dieser Aktor dient als Einstiegspunkt von Webstart und wird als erstes mit Parametern zur gewünschten Datenverarbeitungssitzung gestartet. Im Startzustand „TryConnectTSS“ überprüft dieser, ob der Türsteher Service(TSS) bereits auf der aktuellen Maschine auf einem angegebenen Port erreichbar ist. In diesem Falle, wird eine Verbindung zum (Remote-)Aktor TSS hergestellt, in den Zustand „ConnectedToTSservice“ gewechselt und die Aufforderung zum Start einer Datenverabreitungssitzung übertragen. Für den Fall, daß der TSS nicht läuft, wird dieser durch Türsteher Initiator gestartet und anschließend wieder in den Zustand „TryConnectTSS“ gewechselt und wie oben eine Verbindung hergestellt und die Datenverabreitungssitzung übertragen. Somit wird pro Maschine der Türsteher nur einmal ausgeführt. Damit endet der Lebenszyklus des Türsteher Initiators.

Türsteher Service (TSS)

Vom Türsteher Initiator(TSI) als Aktor gestartet übernimmt der Türsteher Service(TSS) den TSI-Prozess. Die Hauptaufgabe des TSS ist die Bearbeitung von Sitzungsnachrichten (*SESSIONMSG*) aller Datenverabrbeitungssitzungen. Dies setzt eine Verwaltung von Datenverabeitungssitzungen voraus, die anhand einer Sitzungsidentifikation (SessionID) umgesetzt wird, die bei allen eingehenden Nachrichten eines jeden Datenverabeiters übermittelt wird. Diese SessionID ist eine zufällige 160-Bitfolge, um Eindeutigkeit und Abgrenzung zu anderen Datenverarbeitern zu erreichen. Diese Sitzungsidentifikation ist nur dem Türsteher und dem zugehörigen Datenverarbeiter bekannt. Zu einer Sitzung gehört neben der SessionID der Benutzername und insbesondere die abgesicherte Socketverbindung zur Datenhaltung und die bereits ausgelesen und abgeleiteten Schlüssel *UserEncryptionKey* und *UserSaltingKey*.
Der Türsteher Service reagiert in seinem einzigen Zustand „Listening“ auf folgende Anfragen in Form von Nachrichten:

- Antwort auf Anfragen des TSI, ob der TSS erreichbar ist (*HELLOTSS*):
  Es wird dem Türsteher Initiator mit einem *HELLOTSI* als eine Art Lebenszeichen geantwortet.
- Aufforderung zum Start einer neuen Datenverarbeitungssitzung (*DVSTARTINFO*):
  Diese Aufforderung startet einen neuen Türsteher Session Starter(TSSS) Aktor, an den die Startinformation weitergeleitet wird. Dabei enthält *DVSTARTINFO* unter anderem die URL des zu startenden Archivs des Datenverarbeiters, die IP-Adresse und Port des Datenhalters und optionale Parameter

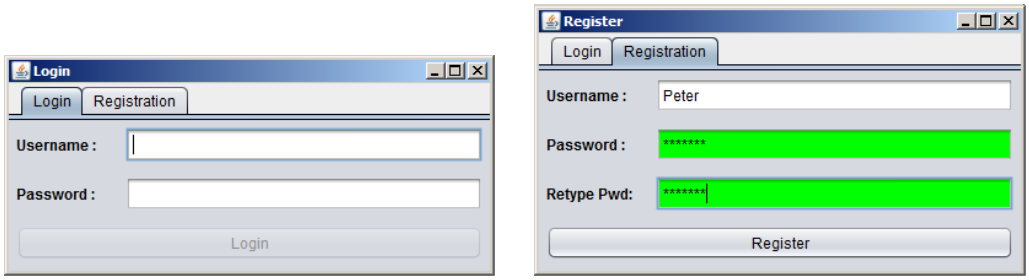
für den Datenverarbeiter. Daraufhin erwartet der TSS entweder auf ein *CANCELED* für einen Abbruch oder ein *SERVICESESSION* als erfolgreich vorbereiteter Sitzung, die dann zu der Menge der aktuellen Sitzungen hinzugefügt wird. Im letzteren Fall ist TSS bereit für den Empfang einer *SESSIONMSG* der vorbereiteten Datenverarbeitungssitzung und wird mit einem *ACK* antworten, damit der TSSS den Datenverarbeiter starten kann.

- Datenschnittstellenoperationen einer Datenverarbeitungssitzung (*SESSIONMSG*): Bei allen Sitzungsnachrichten wird die SessionID von der Datenverabreitungsinstanz mitübertragen, um eindeutig die Sitzungen voneinander abzugrenzen. Es werden folgende Sitzungsnachrichten unterschieden, wobei die ersten beiden der Sitzungsverwaltung dienen und die letzten vier die Datenschnittstelle umsetzen:
  - *SESSIONMSG(SessionID, HELLO)*: Als erster Kontakt wird gegenseitig ein *HELLO* übermittelt, um eine Verbindung zu überprüfen.
  - *SESSIONMSG(SessionID, GOODBYE)*: Diese Nachricht beendet eine Sitzung beim TSS und entfernt die Sitzung aus der Menge der aktiven Sitzungen. Nach dieser Nachricht werden alle weiteren Nachrichten mit der gleichen SessionID, also von der gleichen Datenverabreitungssitzung verworfen.
  - *SESSIONMSG(SessionID, CONTAINS(ID))*: Zunächst wird die ID mithilfe des *UserSaltingKey* der zugehörigen Sitzung angepaßt und die Anfrage über die Socketverbindung an den Datenhalter per *SOCKETMSG* weitergeleitet. Mit dem Ergebnis des Datenhalters wird dem Datenverarbeiter geantwortet.
  - *SESSIONMSG(SessionID, READ(ID))*: Es wird genauso wie bei *CONTAINS* verfahren mit dem Unterschied, daß die vom Datenhalter kommenden Daten mit dem *UserEncryptionKey* symmetrisch entschlüsselt werden bevor sie an den Datenverarbeiter weitergeleitet werden.
  - *SESSIONMSG(SessionID, WRITE(ID, data))*: Analog zu *CONTAINS* wird ID angepaßt und geantwortet, aber *data* vor Übertragung an den Datenhalter mit *UserEncryptionKey* symmetrisch verschlüsselt.
  - *SESSIONMSG(SessionID, DELETE(ID))*: Entspricht dem Vorgehen bei *CONTAINS*.

Türsteher Session Starter (TSSS)

Die Hauptaufgabe des Aktors Türsteher Session Starter(TSSS) ist es, eine Datenverarbeitungssitzung nach erfolgreicher Benutzeranmeldung vorzubereiten. Außerdem kann eine Registrierung eines neuen Benutzers durchgeführt werden.
Der Türsteher Session Starter kann sechs verschiedene Zustände annehmen:

- **Idle**: In diesem Zustand wird der TSSS-Aktor vom TSS gestartet und ist bereit zum Empfang eines *SESSIONREQUEST*s vom TSS, der die nötigen Informa-



(a) LoginGUI (b) RegisterGUI

Abbildung 5.2: Login- und RegisterGUI

tionen *DVSTARTINFO* zum Start eines Datenverarbeiters enthält. Nun wird das GUI gestartet, das eine Anmelde- oder Registrierungsmöglichkeit bietet. Abbildung 5.2 illustriert die zugehörigen Benutzeroberflächen. Parallel dazu wird im Hintergrund das JAR-Archiv des Datenverarbeiters in einen temporären Ordner auf die Festplatte heruntergeladen, falls eine neue Version existiert. Nach Ingangsetzung der GUI und des Herunterladens in separaten Threads wechselt der TSSS in den Zustand „WaitingForGUIAndDL“.

- **WaitingForGUIAndDL**: In diesem Zustand verharrt der TSSS bis er von beiden obigen Threads per *JARDOWNLOADCOMPLETE* und *LOGINREQUEST* bzw. *REGISTERREQUEST* benachrichtigt wird. Dann liegen Eingaben bei dem GUI vor und das Archiv ist heruntergeladen worden. Abhängig vom Eingabetyp, also ob ein Anmeldevorgang (*LOGINREQUEST*) oder eine Registrierung (*REGISTERREQUEST*) vorliegt, wird in „StartingSession“ oder „RegisteringTier1“ gewechselt und die GUI-Information zum nächsten Zustand durch Senden an sich selbst weitergereicht.
- **StartingSession**: Der übermittelte *LOGINREQUEST* vom vorherigen Zustand enthält den Benutzernamen und das Passwort, die beide zuvor aufgefüllt und beschnitten worden sind. Nun wird eine Sitzung für den TSS vorbereitet, indem zunächst der *Loginname-DH-ID* bestimmt wird, daraufhin eine TLS/SRP-Verbindung zum Datenhalter etabliert wird. Bei Erfolg wird der *UserMasterKey* ausgelesen, mit dem abgeleiteten *UserPwdKey* entschlüsselt und der *UserEncryptionKey* und *UserSaltingKey* abgeleitet. Zusammen mit dem Benutzernamen, der Socket-TLS-Verbindung, einer zufälligen 160-Bit langen SessionID und den beiden zuletzt genannten Schlüsseln wird ein Sitzungobjekt *SERVICESESSION* generiert und an den TSS gesendet. Bei Mißerfolg, entweder bedingt durch einen Fehler beim TLS/SRP-Verbindungsaufbau, oder fehlenden oder nicht entschlüsselbarem *UserMasterKey* wird ein *CANCELED* und ein *DVSTARTINFO* für eine Wiederholung an den TSS gesendet und in den Zustand „Exiting“ gewechselt. Bei Erfolg wird auf eine Bestätigung von *SERVICESESSION* gewartet und beim Eintreffen derselben der Datenverarbeiter in einem eigenen Prozess in einer Sandbox gestartet. Als Parameter werden der Port des TSS, die SessionID, der Benutzername und die optionalen Argumente aus der JNLP-Datei übergeben. Daraufhin wird in den Zustand „Exiting“ gewechselt.

- **RegisteringTier1**: Im Falle einer Registrierungsaufforderung *REGISTER-REQUEST*, die Benutzernamen und Passwort enthält, beginnt die zweistufige Registrierung mit „RegisteringTier1“: Zunächst wird der *Loginname-DH-ID* bestimmt und der SRP-Verifier aus dem Passwort berechnet. Die byteweise Konkatenation wird base64-kodiert in einem neuen Fenster (siehe Abbildung 5.3) angezeigt. Diese soll in die Zwischenablage kopiert werden und beim Datenhalter zur Registrierung verwendet werden. Nach dem Schließen des Fenster durch den Benutzer wechselt der TSSS in den Zustand „RegisteringTier2“. Dabei wird *REGISTERREQUEST* an sich selbst gesendet, um die nötigen Informationen für den Abschluss der Registrierung bereitzustellen.

Abbildung 5.3: Registrierungsinformation für den Datenhalter

- **RegisteringTier2**: Die zweite Stufe der Registrierung geht davon aus, daß beim Datenhalter der SRP-Verifier hinterlegt worden ist, d.h. das Fenster mit den Registrierungsinformationen sollte erst geschlossen werden, wenn die Registrierung beim Datenhalter abgeschlossen ist. Darauf wird zunächst versucht mit Benutzername und Passwort eine TLS/SRP-Verbindung aufzubauen. Bei Erfolg wird ein zufälliger symmetrischer *UserMasterKey* erzeugt und mit dem aus Benutzernamen und Passwort abgeleiteten *UserPwdKey* verschlüsselt und über die Datenschnittstelle des Datenhalters, die über TLS/SRP-Verbindung erreichbar ist, in der Datenhaltung abgelegt. Sowohl bei Erfolg als auch bei Mißerfolg wird ein *CANCELED* und ein *DVSTARTINFO* für eine Wiederholung an den TSS gesendet und in den Zustand „Exiting“ gewechselt.
- Exiting: In diesem Zustand beendet sich der TSSS selbst.

5.2.3 Datenverarbeiter

Die Datenverarbeitungsinstanz läuft in einer Sandbox als eigener Prozess. Der benötigete Programmcode liegt in einem JAR-Archiv auf einem Server des Anwendungsentwicklers bereit und wird zuvor vom Türsteher heruntergeladen. Es wird in Form eines Rahmenwerks die Datenschnittstelle für Anwendungsentwickler bereitgestellt. Zunächst werden die beim Start übertragenen Parameter (Port des TSS, SessionID, Benutzername, optionale Argumente) ausgewertet. Daraufhin wird der TürsteherService(TSS) auf der lokalen Maschine kontaktiert, indem eine *SESSIONMSG(SessionID, HELLO)* gesendet wird und auf ein *HELLO* als Antwort gewartet wird. Die dem Anwendungsentwickler gebotene Datenschnittstelle aus Ka-

pitel 3.2.3 wird über foldende Nachrichten abgebildet, die an den TSS gesendet werden:

1. *SESSIONMSG(SessionID, CONTAINS(ID))*
2. *SESSIONMSG(SessionID, READ(ID))*
3. *SESSIONMSG(SessionID, WRITE(ID, data))*
4. *SESSIONMSG(SessionID, DELETE(ID))*

Es wird jeweils auf eine Antwort gewartet, bevor eine weitere Nachricht gesendet werden kann. Eine *SESSIONMSG(SessionID, GOODBYE)* wird beim Beenden der Datenverarbeitungsinstanz an den TSS gesendet, um die Sitzung auch beim Türsteher zu beenden, also inbesondere symmetrische Schlüssel aus dem Arbeitsspeicher zu entfernen und die Verbindung zum Datenhalter zu beenden.

5.2.4 Datenhalter

Der Datenhalter besteht aus drei Serveranwendungen: Es läuft in dieser Beispielimplementierung eine Postgres-Datenbank [pos], die so konfiguriert ist, daß sie nur lokal, also von Anwendungen auf dem Server, erreichbar ist. Diese enthält eine Datenbank mit zwei Tabelle, eine für die eigentlichen Nutzdaten und eine weitere für Authentifizierungsdaten.
Die Tabelle für Nutzdaten organisiert die für jedes Datenpaket typische 160Bit langen ID durch Aufsplitten in fünf 32Bit Integer, um auf primitive Datentypen zurückzugreifen. Dadurch läßt sich inbesondere sukzessiv der Index je Tabellenfüllungsgrad erweitern, indem zunächst nur über die erste Integerspalte ein Index erstellt wird. Bei hohem Füllungsgrad werden Kollisionen wahrscheinlicher und es kann eine weiter Integerspalte zum Index hinzugefügt werden. Dadurch bleibt der Index insbesonder in Hinblick auf Index-Caching im Arbeitsspeicher klein. Folgendes Datentabellenschema wird benutzt:

```
CREATE TABLE data
(
  id1 integer NOT NULL,
  id2 integer NOT NULL,
  id3 integer NOT NULL,
  id4 integer NOT NULL,
  id5 integer NOT NULL,
  data bytea NOT NULL
)
CREATE INDEX data_id1_idx
  ON data
  USING btree
  (id1);
```

Für die Authentifizierungsdaten, also die Abbildung von *Loginname-DH-ID* als „identifier“ auf den java-serialisierten SRP-Verifier (Passwort Verifier), wird folgendes Ta-

bellenschema benutzt, indem beide Bytesequenzen base64-kodiert als Text abgelegt werden. Hier bestehen offensichtlich noch Optimierungsmöglichkeiten.

```
CREATE TABLE auth
(
  identifier text NOT NULL,
  srpverifier text NOT NULL,
  CONSTRAINT identifierconstraint PRIMARY KEY (identifier)
)
```

Eine weitere Tabelle wäre für die gewünschte Erfassung von Abrechnungsinformationen denkbar. Dies liegt in der Hand eines konkreten Datenhaltungsanbieters und soll nicht im Rahmen dieser Beispielimplementierung erfasst werden.
Weiterhin läuft ein Webserver, der eine Website (*HTTPS*) bereitstellt, über die die beim Türsteher angezeigte Registrierungsinformation und zusätzliche für den Datenhalter zu Abrechnungszwecken nötige Informationen eingegeben werden. Diese werden dann über ein PHP-Skript mit einer Datenbankverbindung in der Authentifizierungstabelle abgelegt.
Außerdem lauscht das eigentliche Datenhalterprogramm auf einem Port des Serverrechners auf eingehende Verbindungen von Türstehern. Dieses Datenhalterprogramm unterhält per Java Database Connectivity(JDBC) eine Datenbankverbindung zur Postgresdatenbank. Im Falle einer eingehenden Verbindung wird eine abgesicherte TLS/SRP-Socketverbindung mit Hilfe der Authentifizierungsdaten aus der Authentifizierungstabelle etabliert. Bei Mißerfolg wird die eingehende Verbindung geschlossen. Bei Erfolg wird ein neuer Aktor gestartet, der über eine neue Verbindung zur Datenbank *SOCKETMSG*es des verbundenen Türstehers bearbeitet, die über die obige Socketverbindung eingehen. Pro verbundenem Türsteher gibt es eine Verbindung zur Datenbank und einen Aktor. Das Skalierungsproblem bei vielen verbundenen Türstehern kann durch die eingangs beschriebene Möglichkeit des Abbildens der Scala Aktoren auf einen Threadpool oder nur einen einzigen Thread umgangen werden.

5.3 Umsetzung der Sicherheitsmechnismen

Im folgenden Kapitel soll die Umsetzung der angewandten Sicherheitsmechanismen separat beleuchtet werden. Bei der Implementierung werden alle sicherheitsrelevanten Grundfunktionen, wie z.b. eine kryptographisch sichere Hashfunktion oder die Verschlüsselung, in das Paket „common.security“ ausgelagert, um diese an einer zentralen Stelle zu vereinen und damit austauschbar zu machen oder noch unbekannte Sicherheitslücken zu schließen.

5.3.1 Plattformsicherheit

Zu den eingebauten Sprachsicherheitsmerkmalen des Java Compilers und der virtuellen Maschine gehören (vgl. [GJSB05] und [JVM]) insbesondere :

- Starke Datentypisierung
- Automatisches Speichermanagement
- Bytecode Verifikation
- Sicheres Laden von Klassen

Darüberhinaus schützt die Zugriffskonrollarchitektur der Java Plattform vor Zugriff auf kritische Ressourcen, z.B. Dateien auf der lokalen Festplatte, oder heiklen Anwendungscode, z.B. Methoden in einer Klasse. Alle Entscheidungen über Zugriffskontrollen werden von einem sogenannten *Security Manager* ausgehandelt. Java Applets und Java Web Start Anwendungen laufen im Gegensatz zu Anwendungen, die ohne explizite Angabe eines Security Managers über das Java-Kommando gestartet werden, automatisch mit installiertem Security Manager. Über sogennante Policy Files können Berechtigungen spezifiziert werden.
Java organisiert sicherheitsrelevante Algorithmen in einer flexiblen, erweiterbaren Vorgehensweise durch die *Java Cryptography Architecture*, vgl. [JCA]. Designziele sind sowohl Implementierungsunabhängigkeit und -interoperabilität, also auch Algorithmenunabhängigkeit und -erweiterbarkeit. Dabei können dynamisch sogenannte „Cryptographic Service Providers“ hinzugefügt werden, die Algorithmen, die dynamisch per Name angesprochen werden können, implementieren. In der verwendeten Laufzeitumgebung hat SUN eigene Provider, so z.B. mittels der Java Cryptography Extension(JCE), standardgemäß hinzugefügt, was vor Version 1.4 separat eingefügt werden musste. Abbildung 5.4 verdeutlicht das dynamische Vorgehen, bei der Suche nach einem per Namen und optionalem Provider angesprochenem Algorithmus.

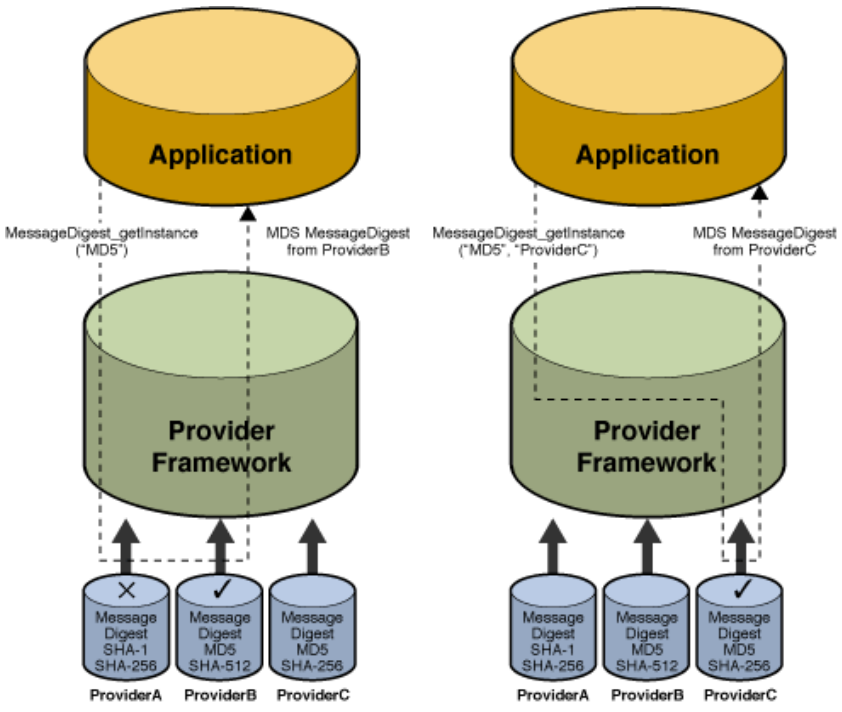


Abbildung 5.4: Java Cryptography Architecture

5.3.2 Webstart/JAR-Signierung

Wie im voherigen Kapitel erwähnt bedingt Java Webstart einen installierten Security Manager. Die nötigen Policy-Einträge werden nicht über eine lokale Datei, sondern über die JNLP-Datei angegeben, die von einem Server heruntergelanden wird. Die Türsteher JNLP-Datei gibt alle Berechtigungen an den Türsteher durch folgende Zeilen (vgl. Kapitel 5.1.1) :

```
<security>
  <all-permissions />
</security>
```

Ein weiteres wichtiges Merkmal von Webstartanwendungen ist, daß alle eingebundenen JAR-Dateien signiert sein *müssen*. Wie in Kapitel 3.3.2 genauer beschrieben ist vor Ausführung der Anwendung über eine interaktive Oberfläche (siehe Abbildung 3.3) nachvollziehbar, wer die Anwendung „unterschrieben“ hat. Das Türsteherarchiv „tssigned.jar“ wird beispielhaft folgendermaßen signiert:

```
jarsigner -keystore mykeystore.keystore -storepass pwd
          -keypass pwd tssigned.jar petomat
```

Dabei wird Gebrauch von einem sogenannten Keystore gemacht, das wie folgt beispielhaft erstellt wurde:

```
keytool -genkeypair -dname "cn=Peter Schmitz, L=Aachen, c=DE"
        -alias petomat -keypass pwd -keystore mykeystore.keystore
        -storepass pwd -validity 360
```

Diese direkte Signierung mit einem selbsterstellten Schlüsselpaar dient nur zu Testzwecken im Rahmen dieser Beispielimplementierung. Im vertrauensvollen Einsatz hingegen wird eine Zertifikatskette bis zu einer Certificate Authority vorausgesetzt.

5.3.3 Sandbox (Policyfile)

Nachdem das JAR-Archiv „dv.jar“ des Datenverarbeiters in ein temporäres Verzeichnis auf der lokalen Festplatte heruntergeladen worden ist, wird eine Policy Datei „sessionID.policy“ mit folgendem Inhalt ebenso in diesem Verzeichnis generiert:

```
grant {
  permission java.net.SocketPermission "localhost:port", "connect, resolve";
};
```

Daraufhin wird der Datenverarbeiter wird vom Türsteher über folgenden Befehl in einem eigenen, neuen Prozess gestartet:

```
java -Djava.security.manager -Djava.security.policy=tmpdir/sessionID.policy
     -jar tmpdir/dv.jar port sessionID username arg1 arg2 ... argN
```

Wie in Kapitel 5.3.1 erwähnt wird bei Aufruf über das Java-Kommando standardgemäß kein Security Manager installiert, daher wird explizit ein Security Manager angeben, der über die Datei „tmpdir/sessionID.policy“ die Berechtigungen des Datenverarbeiters erhält. Außerdem werden noch der Port des Türstehers, zu dem sich der Datenverarbeiter lokal verbinden wird, der Benutzername und beliebige weitere Argumente „arg1, arg2, ... argN“, die aus der JNLP-Datei übernommen werden, an den Datenverarbeiter als Parameter übergeben.

5.3.4 Secure Hash Algorithm und Pseudo Random Number Generator

Als Pseudo Random Number Generator(PRNG) kommt der kryptographisch sichere Zufallszahlengenerator in Form der Klasse „ java.security.SecureRandom“ mittles des in der verwendeten Laufzeitumgebung integrierten Providers „SunJCE version 1.6“ zum Einsatz. Laut Java API erfüllt dieser die statistischen Zufallsgeneratortests definiert in FIPS 140-2, Security Requirements for Cryptographic Modules, section 4.9.1, siehe [FIPb]. Weiterhin liefert dieser eine nichtdeterministische Ausgabe, die kryptographisch stark im Sinne von RFC 1750: Randomness Recommendations for Security ([Ran]) ist.
Diese Beispielimplementierung stützt sich auf die Implementierung des Secure Hash Algorithms(SHA) des Providers „SunJCE version 1.6“. Umgesetzt wird dies mittels der Klasse „MessageDigest“ im Paket „ java.security“. Dynamisch wird der Algorithmus bei Aufruf der Methode „ getInstance(„SHA“)“ angegeben und obiger Standardprovider gewählt.

5.3.5 AES-Verschlüsselung

Zur symmetrischen Ver- und Entschlüsselung der Datenpakete in in der Datenhaltung wird nach dem Advanced Encryption Standard(AES) verfahren. Als Betriebsart für die Blockchiffre wird der Cipher Block Chaining(CBC) Mode verwendet und nach dem Public Key Cryptography Standards #5 (PKCS5) gepadded. Wie in den obigen Kapitel wird dabei wieder auf den Sun-Provider zurückgegriffen und beispielsweise ein Cipherobjekt zum Verschlüsseln wie folgt instanziert:

```
val encCipher = Cipher.getInstance("AES/CBC/PKCS5Padding")
```

Die AES-Schlüssellänge beträgt 128 Bit, da diese bei Standardinstallationen aufgrund von Einfuhrbestimmungen in manchen Ländern auf die sogenannte „Strong Strength Cryptography“ im Gegensatz zu „Unlimited Strength Cryptography“ mit Schlüssellängen von 192 Bit und 256 Bit eingeschränkt ist. Für die Betriebsart CBC wird ein Initializierungsvektor IV für die erste XOR Operation mit den zu verschlüsselten Daten benötigt. Dieser muss bei Aufruf der Ver- oder Entschlüsselungsmethoden als Parameter angegeben werden und entspricht der ID des Datenpakets. Daher bedarf es keinem *TokenEncryptionKey* wie in Kapitel 3.3.6 erwähnt. So wird zum Beispiel, der *UserMasterKey* mit dem *UserPwdKey* unter Einfluß der *UserMasterKeyTokenID* wie folgt verschlüsselt:

```
val encryptedUserMasterKey = symEncrypt(userMasterKey.keyBytes,
                                        userPwdKey,
                                        userMasterKeyTokenID.idBytes)
```

5.3.6 Schlüsselableitungsfunktion(KDF)

Als Schlüsselableitungsfunktion mit beliebig vielen Argumenten wird auf SHA zurückgegriffen. Zunächst werden alle Argumente byteweise konkateniert und anschließend gehasht. Der entstehende Schlüssel muss die gleiche Bitlänge haben, wie die bei der Verschlüsselung verwendete, daher wird das Ergebnis auf die nötige Schlüssellänge gekürzt. In der Implementierung wird dabei darauf geachtet, den Zweck des Schlüssels als ein Argument anzugeben, um diesen eindeutig von anderen Schlüssel abzugrenzen. So zum Beispiel die Ableitung für den UserPasswordKey, bei der außerdem der Benutzername und Password aufgefüllt und beschnitten wurden:

```
val userPwdKey = KDF(username, password, "UserPwdKey")
```

5.3.7 TLS/SRP

Die Implementierung setzt nicht TLS gepaart mit einer Erweiterung per SRP um, sondern bediente sich nur des SRP Protokolls mithilfe der SRP-Bibliothek von Jordan Zimmerman. Dabei wird der SRP-Verifier beim Türsteher während des ersten Registrierungsschrittes aus dem Passwort des Benutzer berechnet und später beim Datenhalter unter dem *Loginname-DH-ID* abgelegt. Das zugehörige zufälliges Salt wird in dem SRP-Verifier mit gekapselt. Die SRP-Implementierung verlangt nach Gruppenparametern N (Modulo) und g (Generator) für die arithmetischen Operationen des SRP-Protokolls. Dabei werden diese fest kodiert im Programmcode verankert und richten sich nach den Vorschlägen für sichere Gruppenparameter im Anhang von [TLSb]. Bereitgehalten werden die Bitgrößen 1024, 1536, 2048, 3072, 4096, 6144 und 8192 für N. Client und Server verwenden beide eine im Programmcode vorgegeben Standardbitgröße von 1024. Spätere Implementierungen könnten Mechanismen zur anfänglichen Aushandlung einer jener Bitgrößen umsetzen. Zu beachten ist, daß die Authentifikation schon bei 1024 Bit auf moderner Hardware im Sekundenbereich liegt.

5.3.8 Datenpaketanpassungen

Beim Türsteher werden die zwei Anpassungen aus Kapitel 3.3.6 eines Datenpaketes vorgenommen:
Zum einen wird die Datenpaket-ID folgendermaßen angepaßt:

```
def mapID(id, userSaltingKey) = ID(hash(userSaltingKey.keyBytes, id.idBytes))
```

Zum anderen wird jedes Datenpaket beim Weiterreichen vom Datenverarbeiter zum Datenhalter oder umgekehrt zuvor ver- bzw. entschlüsselt, wobei die angepaßte ID

als Initzialisierungsvektor für AES/CBC verwendet wird. Beispielsweise sei die Verschlüsselung beim Ankunft einer *SESSIONMSG(sessionID, WRITE(id, data))* aufgezeigt:

```
val session = serviceSessions(sessionID)
val mappedID = mapID(id, session.userSaltingKey)
val enc = Crypt.symEncrypt(data, session.userEncryptionKey, mappedID.idBytes)
val dhAnswer = processSessionMessage(sessionID, WRITE(mappedID, enc))
reply(dhAnswer.asInstanceOf[Boolean])
```

5.4 Evaluierung

Einen wichtigen Aspekt stellt die Evaluierung des vorhandenen Systems dar, denn diese Beispielimplementierung soll als Grundstein für praktische Anwendungen dienen. Zu erwähnen ist, daß es sich bei der Implementierung um eine nicht optimierte Version handelt. Aufgrund der Ergebnisse hier kann man ähnlich einem Profiling von Prozessor- oder Speicherauslastung eines Computerprogramms Anhaltspunkte für eine Optimierung beispielsweise in Bezug auf Datendurchsatz finden.

5.4.1 Aufbaubeschreibung

Alle einer Evaluierung unterzogenen Szenarien stützen sich auf globale Parameter, die hier wie folgt festgelegt werden:
Die ID-Länge ist aufgrund der verwendeten Hashfunktion SHA auf 160Bit festgelegt. Die Anzahl gleichzeitiger Datenverarbeiter ist sowohl global(aus Datenhaltersicht), als auch lokal(aus Türstehersicht) auf den einen Datenverarbeiter während der Messung begrenzt. Die Datentabelle der Postgresdatenbank ist zu Beginn jeder Messung leer. D.h. es wird zu Beginn jeder Messung ein Benutzer registriert und mit diesem die Anmeldung durchgeführt. Bei allen Szenarien werden die Zugriffsbeschränkungen (Sandbox) des Datenverarbeiters außer Kraft gesetzt, damit dieser Ergebnisse auf der lokalen Festplatte festhalten kann. Es werden folgenden Maschinen und deren Konfigurationen Verwendung finden:

- Maschine U :
  - Hardware : Intel Xeon E5410 QuadCore 4x2.33GHz, 4GB RAM
  - Betriebssystem : Ubuntu 64Bit, Linux version 2.6.28-15-generic
  - Java : Version „1.6.0_14“ Java(TM) SE Runtime Environment (build 1.6.0_14-b08)
    Java HotSpot(TM) 64-Bit Server VM (build 14.0-b16, mixed mode)
  - DB : Postgres, Version 8.4
  - WAN : Direkte Anbindung ans RWTH-Netz
- Maschine M :
  - Hardware : AMD Athlon X2 2x3.00GHz

  - Betriebssystem : Windows XP 32Bit
  - Java : Version „1.6.0_17“ Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
    Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)
  - DB : keine
  - WAN : Per WLAN zu DSL Deutsche Telekom AG (15044kBit/s DL ; 1156kBit/s UL)
- Maschine P :
  - Hardware : Intel Core2 Duo E6300, 2GB DDR2, WD740GD Raptor 10000U/min
  - Betriebssystem : Vista 32Bit, Windows Version 6.0.6002
  - Java : Version „1.6.0_17“ Java(TM) SE Runtime Environment (build 1.6.0_17-b04)
    Java HotSpot(TM) Client VM (build 14.3-b01, mixed mode, sharing)
  - DB : Postgres, Version 8.4
  - WAN : Per WLAN zu DSL2000 Deutsche Telekom AG (2304kBit/s DL ; 224kBit/s UL)

5.4.2 Szenarien

Da aufgrund vieler Variationsmöglichkeiten für Parameter der Szenarien und vorallem einiger nicht erfaßbarere Einflußfaktoren, wie z.B. Verkehr anderer Netzwerkteilnehmer, werden die folgenden Szenarien entweder aussagekräftig oder realitätsnah ausgewählt.

Szenario 1

Der Datenhalter (und damit die Postgresdatenbank) und ebenso der Webstartclient, der den Türsteher und dieser dann den Datenverarbeiter startet, sind auf Machine P in Betrieb. Dadurch ist eine direkte Kommunikation über die Netzwerkschnittstelle der lokalen Machine P möglich, um einen maximalen Durchsatz zu erzielen, der ansonsten durch die Netzwerkverbindung ausgebremst wird. Der Türsteher läuft mit aktivierter AES-Verschlüsselung des Datenpaketinhaltes.
Getestet werden zunächst Schreiboperationen beim Datenverarbeiter, indem immer größer werdende Datenpakete über den Türsteher zum Datenhalter geschrieben werden. Es werden Paketgrößen von 0kB bis 32kB in 128Bit Schritten, darüber in 1Kb Schritten mit zufälligem Inhalt erzeugt. Anschließend werden die gleichen Datenpakete wieder ausgelesen. Es findet also eine Variation der Datenpaketgröße statt. Bei beiden Operationen wird der Durchsatz in kB/Sek je Datenpaketgröße gemessen und in Abbildung 5.5 graphisch aufgetragen.

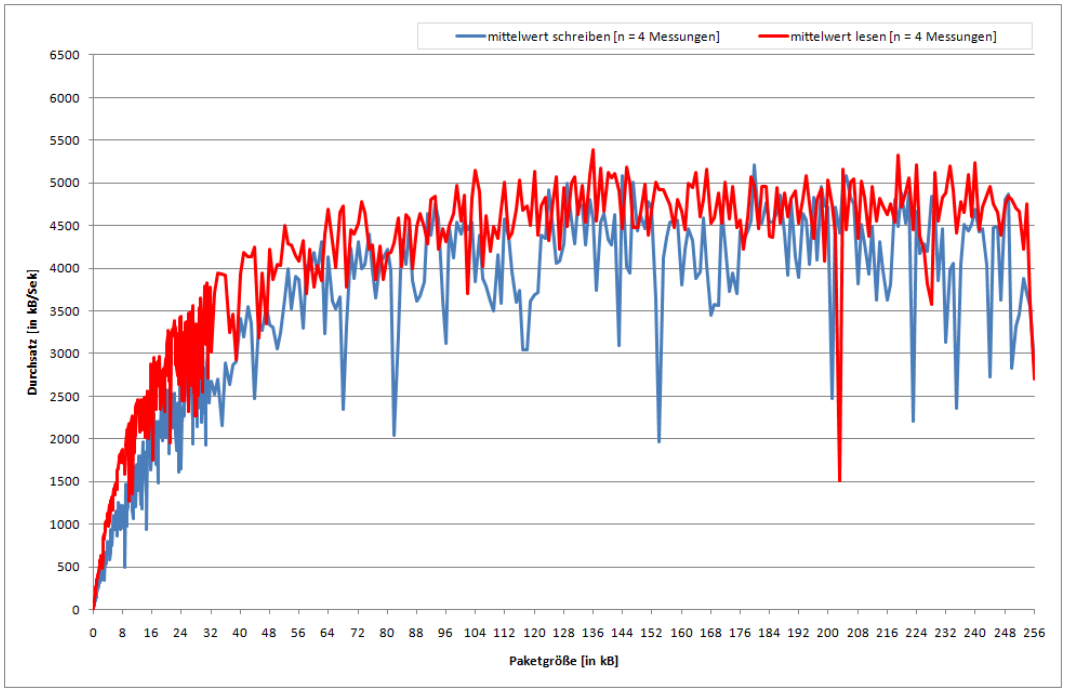


Abbildung 5.5: Evaluierungsergebnis Szenario 1

Szenario 2

Dieses Szenario entspricht dem Szenario 1 bis auf die deaktivierte AES-Verschlüsselung beim Türsteher. Wieder wird der Durchsatz in kB/Sek je Datenpaketgröße bei Schreib- und Leseoperationen gemessen und in Abbildung 5.6 dargestellt.

Szenario 3

Diesmal wird ein typisches Szenario angenommen, das ebenfalls an den Aufbau des vorherigen Szenarien 1 angelehnt ist: Der Datenhalter befindet sich auf einem dedizierten Server, hier Maschine U. Der Client hingegen auf einem anderen Rechner (Maschine B) mit schneller DSL-Internetanbindung. Bezüglich Schreib- und Leseoperationen wird erneut der Durchsatz in kB/Sek je Datenpaketgröße gemessen. Abbildung 5.7 zeigt die Ergebnisse der Versuchsdurchführungen.

Szenario 4

Dieses gleichfalls typische Szenario baut auf Szenario 3 auf, indem der Client auf einer Machine mit langsamerem DSL-Anschluss operiert. Bezüglich Schreib- und Leseoperationen wird erneut der Durchsatz in kB/Sek je Datenpaketgröße gemessen. Abbildung 5.8 legt die Ergebnisse dar.

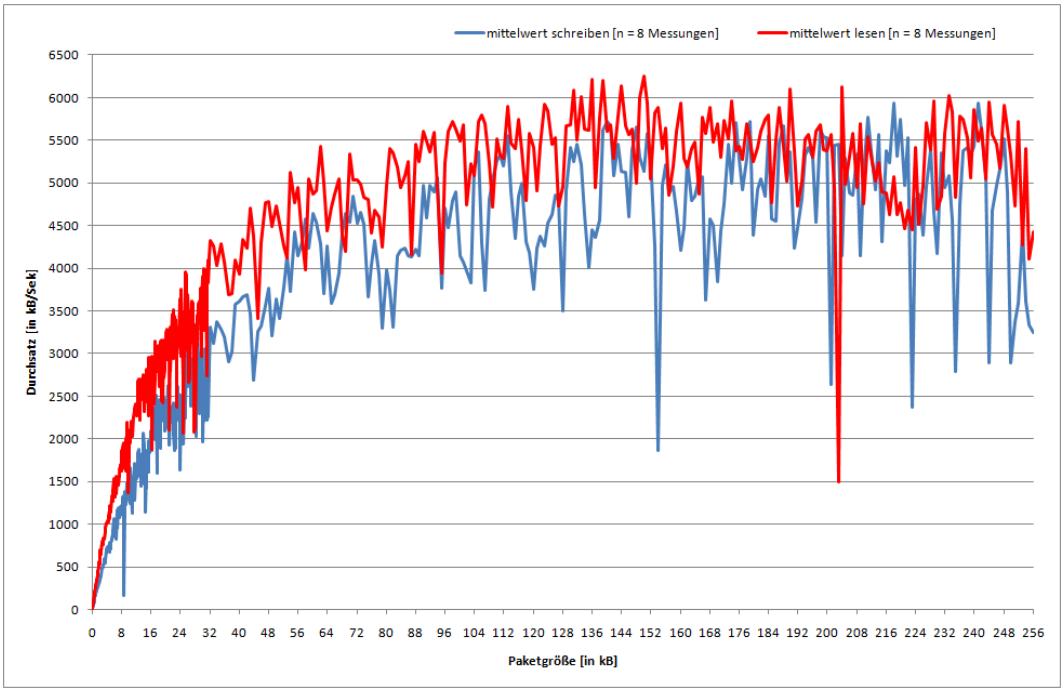


Abbildung 5.6: Evaluierungsergebnis Szenario 2

5.4.3 Diskussion

Beobachtungen

In Szenario 3 und 4 konnte sowohl auf Server-, als auch auf Clientseite keine nennenswerte Prozessorlast beobachtet werden. Selbst in den ersten beiden Szenarien wurden die beiden Prozessorkerne nicht voll ausgelastet. Ebenso stellte auch die Festplattenaktivität subjektiv keinen bedeutetenden Engpass dar.
Festzustellen ist, daß der Durchsatz mit größer werdenden Datenpakten zunimmt. Bei den beiden letzten Szenarien mit verteilter Rechnerstruktur sind folgende untersuchenswerte Effekte zu beobachten:

- Es scheint ein auf den ersten Blick logarithmischer Zusammenhang zwischen Durchsatz und Datenpaketgröße vorzuliegen, was aber einer genauer Untersuchung bedarf.
- Außerdem entstehen in unregelmäßigen Abständen maschinenunabhängige Ausreißer, die den Druchsatz kurzzeitig stark mindern.
- Desweiteren fluktuiert der Schreibdurchsatz weniger als der Lesedurchsatz.
- Je geringer die maximale Übertragungskapazität der verwendeten Leitung ist, desto kleiner sind die Paketgrößen, ab denen ein guter (z.B. 80% des maximal möglichen Durchsatzes) Durchsatz möglich ist.

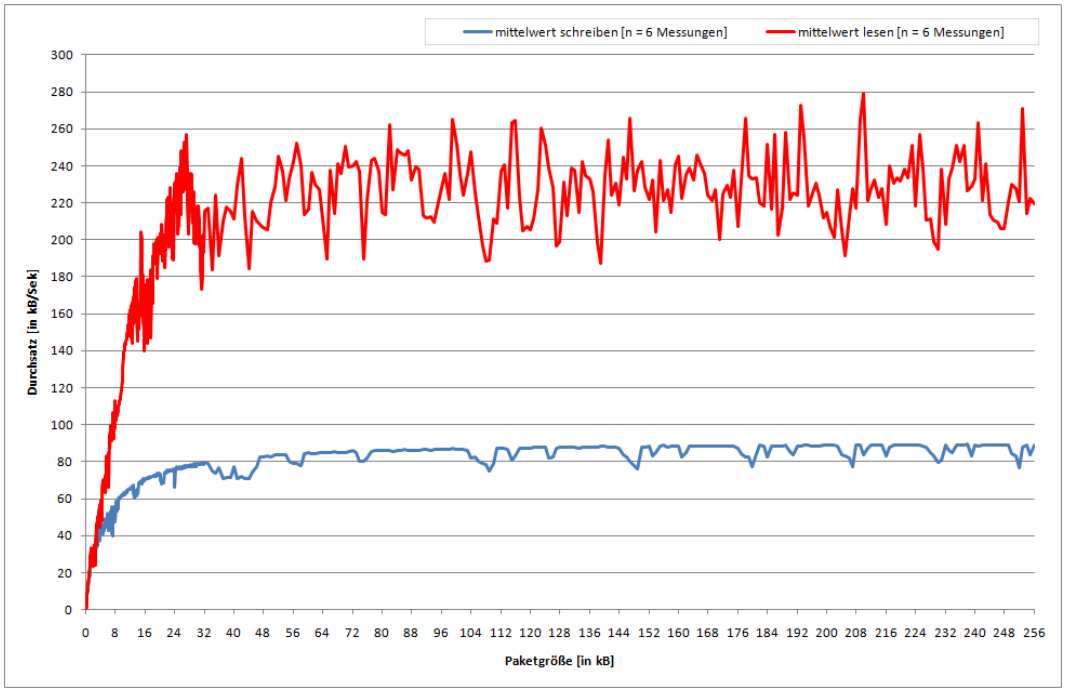


Abbildung 5.7: Evaluierungsergebnis Szenario 3

Zielsetzung

Zweck dieser Evaluierung ist es, einen Ausgangspunkt vorzubereiten, Anhaltspunkte für eine optimale Datenpaketgröße bezüglich Datendurchsatz zu finden. Dabei waren verschiedene Einflußgrößen maßgeblich, so z.B. die DSL-Übertragungsgeschwindigkeit oder ob es sich um Schreib- oder Leseoperationen handelt. Bei asymmetrisch Übertragungen wie beispielsweise DSL ergeben sich andere Bereich für gute (z.B. 80% des maximal möglichen Durchsatzes) Paketgrößen bei Schreib- und Leseoperationen. Insbesondere in Hinblick auf mögliches Aufteilen von Datenpaketen auf mehrere kleine aber gleich lange durch den Türsteher (vgl. Kapitel 6.3) sind offensichtlich Durchsatzeinbußen zu erwarten. In diesem Falle ist die Datenpaketgröße für alle Operationen gleich und daher wird die Entscheidung für die Auswahl einer Paketgröße schwerer, die beiden Operationen im Bezug auf Maximierung des Druchsatzes genügt.
Außerdem ist interessant zu untersuchen, welcher Durchsatz sich auf mobilen Endgeräten ergäbe, da insbesondere dort auch der Prozessor einen Engpass darstellen wird.

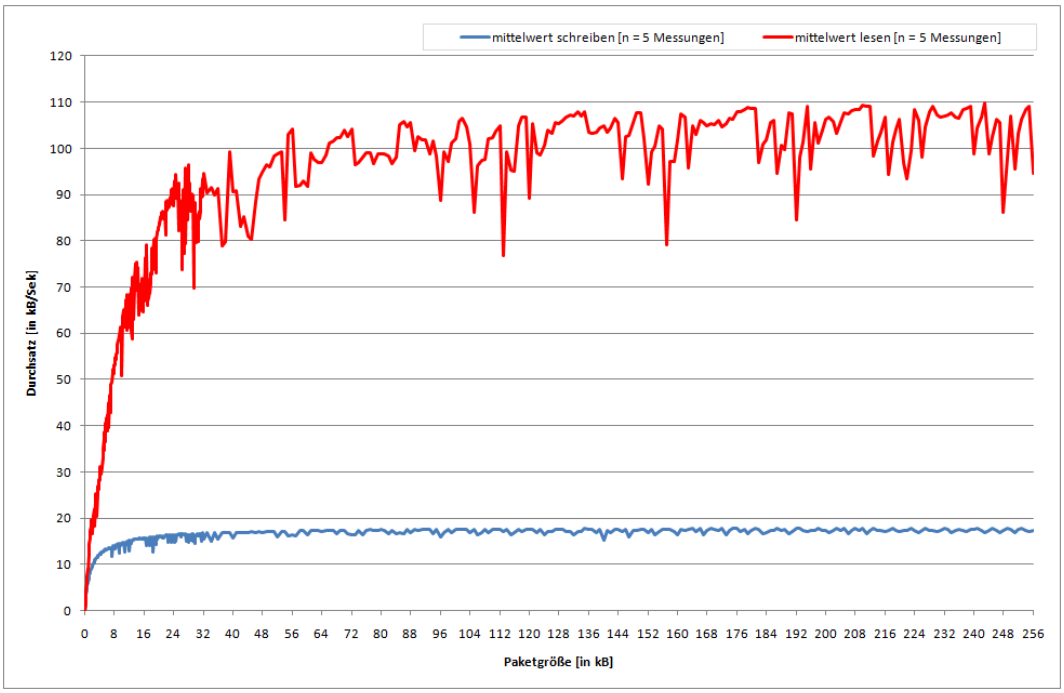


Abbildung 5.8: Evaluierungsergebnis Szenario 4

6 Erweiterungen und Zusammenfassung

In der vorgestellen Arbeit lassen sich Ansatzpunkte für Erweiterungen bzw. Verbesserungen finden. Denkrichtungen dabei zielen auf Effizienz (im Sinne von Performanz), Flexibilität und Sicherheit ab.

6.1 Konzeptionelle Erweiterungen

Der in Kapitel 3 vorgestellte Lösungsansatz läßt sich konzeptionell in Hinblick auf Performanz und Flexibilität durch folgendes erweitern:

- **Kommunikation von Türstehern**: Die einzige Kommunikationsmöglichkeit von Datenverarbeitern untereinander besteht im gemeinsamen Ablegen und Auslesen von Datenpaketen bei der Datenhaltung, was sich als sehr ineffizient herausstellen dürfte. So könnte eine Erweiterung darin bestehen, Türstehern und damit Datenverarbeitern, die auf verschiedenen Geräten laufen, untereinander Kommunikation zu gestatten. Dadaruch sind kollaborative Programme möglich, wie z.B. im einfach Fall ein Chat-Programm. Dabei entstehen aber neue Herausforderungen der Absicherung der Kommunikation.
- **Zusätzliche lokale Datenhaltung**: Jedem Datenverarbeiter wird nur die Datenschnittstelle des Türstehers angeboten, die wiederum die Datenschnittstelle der entfernten, zentralen Datenhaltung nutzt. Denn da auch der direkte Zugriff auf lokale Datenträger einen Kommunikationskanal darstellen kann, wird dem Datenverarbeiter auch diese Möglichkeit genommen. Dies gilt insbesondere auch für temporärer Dateien oder ähnliches. Da für viele Anwendungen diese aber unverzichtbar sind, liegt eine Erweiterung der Türsteherfunktionalität um Zugriff auf lokale Datenträger nahe. Diese Erweiterung ähnelt sehr der bisherigen Datenschnittstelle des Türstehers für die Datenverarbeitung in Richtung Datenhaltung, indem zusätzlich vom Türsteher eine zweite Datenschnittstelle der gleichen Form bereit gestellt wird, die ebenfalls die gleichen Sicherheitsanpassungen einsetzt, die Datenpakete hingegen nur lokal ablegt.

- **Dezentrale Datenhaltung**: Dank der schlichten Datenschnittstelle läßt sich die zentrale Datenhaltung auch dezentral organisieren. Denkbar wäre auf einem bestehenden Peer2Peer-Netz aufzusetzen, welches dann die Operationen der Datenschnittstelle zur Verfügung stellt. Dadurch würde sowohl die Netzwerklast, als auch der Speicherplatzbedarf von einer zentralen Stelle auf ein Netzwerk von Knoten verteilt werden. Wuala [wua] ist ein gutes Beispiel dafür.

6.2 Implementierungsverbesserungen

Da es sich bei der Beispielimplementierung nur um ein Grundgerüst für zukünftige Implementierungen handelt, sind Überarbeitungen angebracht:

- **Optimierungen**: Im Vordergrund der Designziele der Beispielimplmentierung stand die Klarheit der Konzeptumsetzung und es wurde nicht auf Performanz optimiert. Mithilfe der Evaluierungsmessungen sind Optimierungsansatzpunkte aufdeckbar.
- **Ergonomie**: An vielen Stellen lassen sich Interaktionen mit dem Benutzer klarer gestalten und Rückmeldungen über den Programmstatus anzeigen. Außerdem besteht Nachholbedarf bei der Fehlerbehandlung für Fehler jeglicher Art. Insbesondere da es sich um eine verteilte Anwendung handelt sind viele Fehlerquellen vorhanden, die alle abgefangen werden müssen. Desweiteren sind zum Beispiel Passwortänderungen noch nicht möglich und könnten angegangen werden.
- **Mobile Endgeräte**: Von Interesse wäre eine Anpassung der Beispielimplementierung an die Laufzeitumgebung auf mobilen Endgeräten. Denn dadurch würde sich nicht nur ein größeres Gerätefeld erschließen, sondern auch neue Evaluierungsmöglichkeiten und Performanzverbesserungen angehen.
- **TLS/SRP**: Die Beispielimplementierung verwendete zwar SRP, um eine gegenseitige Authentifizierung vom Benutzer über den Türsteher und dem Datenhalter durchzuführen, deckte aber nicht die Absicherung der Verbindung von Türsteher und Datenhalter mittels TLS ab. Hier könnte noch eine geeignete Bibliothek angepaßt Verwendung finden.

6.3 Sicherheitsrelevante Erweiterungen

Ebenso läßt sich über Erweiterungen nachdenken, die sicherheitsrelevant sind:

- **Zugriff auf Daten anderer Benutzer**: Eine offensichtliche Einschränkung des vorgeschlagenen Lösungsansatzes ist die Einschränkung, daß ein Benutzer nur auf seine eigenen Daten Zugriff hat. Für einen Zugriff auf Daten anderer Benutzer bieten sich Konzepte wie Access Control List(ACL) oder Broadcast Encryption ähnliche Vorgehen an. Außerdem läuft ein gegenseitiger Datenzugriff auf asymmetrische Verschlüsselung mit dem Umstand der Verteilung und Bereitstellung von öffentlichen Schlüssel (Public Key Infrastructur) hinaus.

- **Längenanpassung**: In Kapitel 3.3.6 wurde das Problem der Kommunikation von Datenverarbeiter und Datenhalter mittels einer Folge verschieden langer Datenpakete angesprochen. Jede Länge würde einem Buchstaben in einem Kommunikationsalphabet entsprechen und eine bidirektionale Verbindung ermöglichen. Dies stellt zwar keine effiziente Methode der Kommunikation dar, aber eine denkbare. Der offensichtlichste Ansatz, dieses Problem in den Griff zu bekommen, ist das Auffüllen (Padding) auf ein Vielfaches einer festgelegten Paketlänge und anschließendes Zerlegen in Datenpakete dieser Länge beim Türsteher. So würden alle Buchstaben in Buchstabengruppen zusammengefaßt, deren entsprechenden Datenpaket eine Länge zwischen zwei Vielfachen der festgelegten Länge haben. Je größer jene festgelegte Länge ist, desto weniger Buchstabengruppen gibt es und desto größer werden diese, und somit eine Kommunikation erschwert.
  An dieser Stelle besteht noch großes Potential, abzuwägen, welche Kommunikationsmöglichkeiten wie effizient durchfürbar sind und wie schwerwiegend diese in Bezug auf Datensicherheit sind.

6.4 Zusammenfassung

Die vorliegende Arbeit zeigte einen Ansatz zur Erhöhung des Schutzes von Inhaltsdaten bei entfernter Datenhaltung. Der Schutz bezieht sich auf die Verarbeitung der Inhaltsdaten, sowie den Transfer zu und Aufbewahrung bei der entfernten Datenhaltung. Realisiert wurde dies durch die Trennung nach Zuständigkeitsbereichen: Datenverarbeiter, Türsteher und Datenhalter. Der Türsteher ist Vermittler zwischen Datenhalter und Datenverarbeiter und kontrolliert insbesondere die Kommunikationsmöglichkeiten des Datenverarbeiters. Potentiell laufen mehrere Datenverarbeiter zusammen mit einem Türsteher auf der lokalen Maschine des Anwenders und der Datenhalter läuft auf einer entfernten zentralen Stelle. Alle sicherheitsrelevanten Funktionen sind im Türsteher vereint, so beispielsweise die benutzerspezifische Verschlüsselung aller Datenflüsse vom Datenverarbeiter zur Datenhaltung. Die Authentifizierung eines Benutzers findet gegenüber dem Türsteher statt und bzgl. Abrechnungszwecken gegenüber dem Datenhalter. Daten werden in Form von Datenpaketen organisiert, die aus einer ID (benutzerspezifischer Hashwert) und einer Bytesequenz, die die Nutzdaten repräsentiert, bestehen. Im Gegensatz zum Cloud-Computing, findet die Verarbeitung der Daten auf der lokalen Maschine des Benutzers statt. Obwohl dort die Rechenkapazität eingeschränkt ist, wird dafür die Verarbeitung nicht auf der oftmals nicht vertrauten entfernten Umgebungen eines Serviceanbieters durchgeführt. Dieses Vertrauensproblem in den Serviceanbieter war maßgeblich für die Ausrichtung der vorliegenden Arbeit. Das Vertrauensproblem wurde durch die Einführung eines quellcode-offenen Türstehers verlagert. Anhand einer Beispielimplementierung mit Java und Scala wurde die Machbarkeit und ein Grundgerüst für zukünftige Implementierungen für den Produktiveinsatz überprüft. Die Evaluierung zeigte passablen Datendurchsatz für eine nicht optimierte Beispielimplementierung. Des Weiteren stellte sich heraus, dass der Durchsatz stark abhängig von der gewählten Datenpaketgröße ist.

Literaturverzeichnis

[BDJ04] Brinkman, R., J. Doumen und W. Jonker: *Using secret sharing for searching in encrypted data.* Lecture notes in computer science, Seiten 18–27, 2004.

[BFD+04] Brinkman, R., L. Feng, J. Doumen, PH Hartle und W. Jonker: *Efficient tree search in encrypted data.* Information Systems Security, 13(3):14–21, 2004.

[BGN05] Boneh, D., E.J. Goh und K. Nissim: *Evaluating 2-DNF formulas on ciphertexts.* In: *TCC*, Band 3378, Seiten 325–341. Springer, 2005.

[Bla93] Blaze, M.: *A cryptographic file system for UNIX.* In: *Proceedings of the 1st ACM conference on Computer and communications security*, Seiten 9–16. ACM New York, NY, USA, 1993.

[BW07] Boneh, D. und B. Waters: *Conjunctive, subset, and range queries on encrypted data.* Lecture Notes in Computer Science, 4392:535, 2007.

[CCDSP97] Cattaneo, G., L. Catuogno, A. Del Sorbo und P. Persiano: *The design and implementation of a transparent cryptographic file system for Unix.* Technischer Bericht, Citeseer, 1997.

[Cer] `http://java.sun.com/javase/6/docs/technotes/guides/security/certpath/CertPathProgGuide.html#Introduction`.

[fipa] `http://www.itl.nist.gov/fipspubs/fip180-1.htm`.

[FIPb] `http://csrc.nist.gov/publications/fips/fips140-2/fips1402.pdf`.

[FKK06] Fu, K., S. Kamara und T. Kohno: *Key regression: Enabling efficient key distribution for secure distributed storage.* In: *Proceedings of the Symposium on Network and Distributed Systems Security*, 2006.

[GJSB05] Gosling, James, Bill Joy, Guy Steele und Gilad Bracha: *The Java Language Specification, Third Edition.* Addison-Wesley Longman,

Amsterdam, 3 Auflage, June 2005.

[gma] http://mail.google.com.

[GMSW06] Grolimund, D., L. Meisser, S. Schmid und R. Wattenhofer: *Cryptree: A Folder Tree Structure for Cryptographic File Systems.* In: *25th IEEE Symposium on Reliable Distributed Systems, 2006. SRDS'06*, Seiten 189–198, 2006. http://dcg.ethz.ch/publications/srds06.pdf.

[gmx] http://www.gmx.net/.

[Goh03] Goh, E.J.: *Secure indexes.* An early version of this paper first appeared on the Cryptology ePrint Archive on October 7th, 2003.

[gooa] http://docs.google.com/support/bin/answer.py?hl=en&answer=87149.

[goob] http://www.google.com/intl/en/privacypolicy.html.

[HO06] Haller, Philipp und Martin Odersky: *Event-Based Programming without Inversion of Control.* In: *Proc. Joint Modular Languages Conference*, Springer LNCS, 2006.

[HO08] Haller, Philipp und Martin Odersky: *Scala actors: Unifying thread-based and event-based programming.* Theoretical Computer Science, 2008.

[hot] http://www.hotmail.com.

[jar] http://java.sun.com/javase/6/docs/technotes/tools/solaris/jarsigner.html.

[JCA] http://java.sun.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html.

[JVM] http://java.sun.com/docs/books/jvms/.

[KRS+] Kallahalla, M., E. Riedel, R. Swaminathan, Q. Wang und K. Fu: *Plutus: Scalable secure file sharing on untrusted storage.*

[MvOV01] Menezes, Alfred J., Paul C. van Oorschot und Scott A. Vanstone: *Handbook of Applied Cryptography.* CRC Press, 2001.

[Oec03] Oechslin, P.: *Making a faster cryptanalytic time-memory trade-off.* In: *Advances in Cryptology–CRYPTO*, Band 3, Seiten 617–630. Springer, 2003.

[PBK] http://tools.ietf.org/html/rfc2898#section-5.2.

[pos] http://www.postgresql.org/.

[Ran] http://tools.ietf.org/html/rfc1750.

[Sah08] Sahai, Amit: *Computing on Encrypted Data.* In: *ICISS '08: Proceedings of the 4th International Conference on Information Systems Security*, Seiten 148–153, Berlin, Heidelberg, 2008. Springer-Verlag.

[san] `http://java.sun.com/javase/6/docs/technotes/guides/security/spec/security-specTOC.fm.html`.

[Sca] `http://www.scala-lang.org/`.

[Sch05] Schneier, Bruce: *Angewandte Kryptographie - Der Klassiker. Protokolle, Algorithmen und Sourcecode in C*. Pearson Studium, München, 2005.

[srpa] `http://srp.stanford.edu/`.

[SRPb] `http://tools.ietf.org/html/rfc2945`.

[SWP00] Song, D.X., D. Wagner und A. Perrig: *Practical techniques for searches on encrypted data*. In: *2000 IEEE Symposium on Security and Privacy, 2000. S&P 2000. Proceedings*, Seiten 44–55, 2000.

[Tho07] Thompson, K.: *Reflections on trusting trust*. 2007.

[TLSa] `http://tools.ietf.org/html/rfc5246`.

[TLSb] `http://tools.ietf.org/html/rfc5054`.

[tlsc] `http://www.ietf.org/dyn/wg/charter/tls-charter.html`.

[TLSd] `http://tools.ietf.org/html/rfc4279`.

[tuv] `http://www.tuv.com/de/zertifizierung_gutachten_zu_it_sicherheit.html`.

[Uca] Ucal, M.: *Searching on Encrypted Data*.

[unc] `https://uncommons-maths.dev.java.net/`.

[weba] `http://java.sun.com/javase/technologies/desktop/javawebstart/index.jsp`.

[webb] `http://java.sun.com/javase/6/docs/technotes/guides/javaws/developersguide/contents.html`.

[wua] `http://www.wuala.com`.