Leibniz Universität Hannover

Institut für Theoretische Informatik

Entwurf einer sicheren Client-Server-Architektur für ein Notenverwaltungssystem

Autor:
Alexander Röttcher
Matrikel-Nr.: 2944160

Erstprüfer:
Prof. Dr. Heribert Vollmer

Zweitprüfer:
Dr. Arne Meier

7. August 2015

Erklärung der Selbstständigkeit

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet zu haben. Die Arbeit hat in gleicher oder ähnlicher Form noch keinem anderen Prüfungsamt vorgelegen.

Hannover, den 7. August 2015

_____

Alexander Röttcher

Inhaltsverzeichnis

1. Einleitung **7**
1.1. Aufgabenstellung und Vorgehensweise 8

2. Grundlagen **10**
2.1. Rechtliche Grundlagen 10
2.2. Kryptografie 11
2.2.1. Hashfunktionen 11
2.2.2. Symmetrische Verschlüsselung 11
2.2.3. Asymmetrische Verschlüsselung 12
2.2.4. Digitale Signaturen 13
2.2.5. PKI-Infrastrukturen 13
2.2.6. Sitzungsschlüssel 14
2.2.7. Empfehlungen vom Bundesamt für Sicherheit in der Informationstechnik . 15
2.3. Cloud-Computing 15
2.4. Was bedeutet eigentlich Sicherheit? 15
2.4.1. Facetten des Begriffs Sicherheit 16
2.4.2. Real-World-Attacken 17
2.4.3. Angriffsmöglichkeiten auf kryptografische Protokolle 20

3. Analyse ausgewählter Programme **22**
3.1. MEGA 22
3.2. TeamDrive 23
3.3. Boxcryptor 23
3.4. MobileSitter 24
3.5. WebUntis 25

4. Konzeptentwurf **27**
4.1. Der Schlüsselcontainer 30
4.2. Der Verwaltungskanal 32
4.3. Der Datenkanal 34
4.4. Datenauswahl und Kodierung 37
4.5. Das Zusammenspiel 39
4.5.1. Initialisierung 39
4.5.2. Erster Verbindungsaufbau 39
4.5.3. Dateien hochladen und teilen 40
4.6. Weitere Sicherheitsaspekte und Hinweise 41

5. Beispielimplementierung **43**
5.1. Abgrenzung zum Konzeptentwurf 44
5.2. Die Krypto-Bibliothek QCA 44
5.3. Softwareaufbau 45
5.4. Der SQLiteManager 47
5.5. Der SecurityManager 48

5.6. Der ShareManager 50
5.7. Mögliche Ansichten 51
5.7.1. Der SchoolGradeManager 52
5.7.2. Der DataManager 52

6. Fazit **54**

7. Ausblick **56**

8. Literaturverzeichnis **57**

A. Anhang **60**
A.1. Einführendes Interview 60
A.2. Reaktion auf erste Konzeptpräsentation 62
A.3. Anforderungen 65
A.3.1. Basisanforderungen 65
A.3.2. Abgeleitete Anforderungen 66
A.3.3. Spezielle Anforderungen für die Beispielimplementierung 67
A.4. Aussagen von Gruber&Petters GmbH 67
A.5. Aussagen von Secomba GmbH 69
A.6. Ergebnis der Programmanalyse 71
A.7. Allgemein verwendete Bezeichnungen 72

1. Einleitung

> Wenn ich die Dokumentation betrachte, ist das ein Aufwand, der – wenn man es vernünftig machen will – schlicht nicht zu leisten ist.
>
> Lehrkraft, siehe Anhang A.1

Im Jahre 2013 schätzte die Frankfurter Bildungsforscherin Mareike Kunter in einem Artikel von *Spiegel Online* den Arbeitsaufwand für „das Vorbereiten von Schulstunden, das Korrigieren von Klassenarbeiten, Elterngespräche, AGs und Verwaltung [auf] mehr als 40 Prozent der Arbeitszeit“ [Son13]. Auch in einem zu Beginn der Arbeit durchgeführten Interview (Anhang A.1) wurde deutlich, dass es mit den momentanen Mitteln nicht möglich ist, die Bewertung der Schülerinnen und Schüler in einem zufriedenstellenden Maß durchzuführen und gleichzeitig den Fokus auf den Unterricht nicht zu verlieren. Gesucht wird also ein System, welches die Lehrkraft in Verwaltungsaufgaben, insbesondere der Leistungskontrolle der Schülerinnen und Schüler, unterstützt.

Eine mögliche Lösung könnte der Einsatz von digitalen Hilfsmitteln sein, um dadurch die Lehrkraft langfristig von dem hohen Verwaltungsaufwand zu entlasten. Neben den Anforderungen an die Praktikabilität rückte auch die Sicherheit eines solchen Systems in den Mittelpunkt. Durch die momentan hohe Präsenz der Spionageaffären und Berichte in den Medien über massenweise Abhörung war eines der wohl wichtigsten Ziele, trotz der Nutzung von mit dem Internet verbundenen IT-Systemen den Schutz der personenbezogenen Daten der Schülerinnen und Schüler weiterhin gewährleisten zu können. Die Kenntnis der kryptografischen Mittel in der Informationstechnologie lässt sich allerdings nicht bei den Lehrkräften voraussetzen.

[**R1**] Das System muss der Lehrkraft die Möglichkeit bieten, die digital verwalteten personenbezogenen Daten der Schülerinnen und Schüler für den Zeitraum der Speicherung gegen den Zugriff Dritter (Angreifer) zu schützen.

[**R2**] Das System muss die geltenden Datenschutzgesetze einhalten.

Schließlich muss die Lehrkraft die Einschätzungen nicht nur für sich selbst verwalten. Beispielsweise werden am Ende des Halbjahres für den Druck der Zeugnisse die Gesamtnoten gesammelt. Auch zum rechtzeitigen Feststellen versetzungsgefährdeter Schülerinnen und Schüler ist ein Austausch bestimmter Informationen sinnvoll. Durch eine Digitalisierung wären die Aufzeichnungen der Lehrerinnen und Lehrer nicht mehr an einen Ort gebunden. Daraus folgt:

[**R3**] Das System muss der Lehrkraft die Möglichkeit bieten, die über das Internet mit anderen Lehrkräften zu teilenden personenbezogenen Daten der Schülerinnen und Schüler gegen den Zugriff Dritter (Angreifer) zu schützen.

Neben dem bereits bestehenden Netzwerk – dem Internet – verlocken viele kostenlose Dienste die Lehrerinnen und Lehrer dazu, digitale Hilfsmittel schon jetzt zu nutzen. In weiteren

Gesprächen mit Lehrern (siehe Anhang A.2) stellte sich u. a. heraus, dass auch aufgrund von Bequemlichkeit und der Abstraktheit möglicher Gefahren die Fragen nach der Sicherheit bzw. dem Datenschutz dabei eher hinten anstehen.

[**R4**] Das System muss eine Verletzung der Datenschutzgesetze und damit der Sicherheit der personenbezogenen Daten der Schülerinnen und Schüler verhindern.

Auf Grundlage dieser noch allgemein formulierten Anforderungen wird in dieser Arbeit ein System entworfen werden, um die erwähnten Problematiken zu lösen. Eine Liste aller erarbeiteten Anforderungen findet sich in Anhang A.3. Die in den nachfolgenden Diagrammen verwendeten Symbole für Server, Client, Schlüssel und Schloss stammen aus [CS].

1.1. Aufgabenstellung und Vorgehensweise

Eine wichtige Forderung im Interview (Anhang A.1) war die Praktikabilität des Systems, um eine notwendige Akzeptanz unter den Lehrkräften, den späteren Nutzern, zu gewährleisten. Basale Anforderungen zur Minimierung der Einarbeitungszeiten sind deshalb:

[**R5**] Das System muss fähig sein, auf vorhandener Hardware (Computer, Tablet, Smartphone) und Betriebssystemen (Windows, Linux, MacOs, Android, iOS) ausgeführt zu werden.

[**R6**] Das System muss der Lehrkraft – insbesondere der nicht mit Computern vertrauten – die Möglichkeit bieten, über gängige Eingabemasken (GUI-Standards, Standards in der handschriftlichen Verwaltung) die zu verwaltenden Daten eingeben zu können.

[**R7**] Das System muss die Interaktion mit der Lehrkraft zur Dateneingabe und Prozesssteuerung minimal halten.

Im Interview (Anhang A.1) stellte sich heraus, dass zur Notiz von Zwischeneinschätzungen am Ende des Unterrichts wenig Zeit bleibt, weshalb eine minimale Interaktion in R7 gefordert wird. Trotzdem steht für die befragte Lehrkraft die Sicherheit noch vor der Benutzerfreundlichkeit. Der Fokus wird deswegen auf einem sicheren System liegen, weshalb die Ausführung der Anforderung R6 nicht Bestandteil der Arbeit ist. Auch Anforderung R7 wird nur hinsichtlich des Sicherheitsprozesses untersucht werden:

[**R8**] Das System muss die notwendigen sicherheitstechnischen Prozesse soweit möglich im Hintergrund automatisieren.

Weiterhin ist ein zu beachtender Faktor die Eigenverantwortung der Lehrkraft. In den einführenden Gesprächen stellte sich heraus, dass vor allem beim Umstieg auf ein System die damit einhergehende Neuheit und Andersartigkeit für Ablehnung sorgt. Um die Akzeptanz zu erhöhen muss die Lehrkraft also weiterhin die Wahlmöglichkeit haben, wie sie die digitalen Medien zur Unterstützung nutzen möchte.

[**R9**] Das System muss der Lehrkraft die Möglichkeit bieten, ohne Einfluss, Kontrolle oder Vertrauen auf andere Instanzen die Verwaltung der personenbezogenen Daten der Schülerinnen und Schüler durchführen zu können.

[**R10**] Das System muss der Lehrkraft die Möglichkeit bieten, zu jedem Zeitpunkt kontrollieren zu können, wem welche Daten im Klartext zur Verfügung gestellt werden.

Schließlich ist eine Portierung der Verwaltung auf ein digitales System erst dann sinnvoll, wenn evtl. bereits bekannte Programme und Automatisierungsangebote genutzt werden können.

[R11] Das System muss fähig sein, über eine Export- und Importfunktion auf Dateiebene das sichere Speichern und Übertragen der mit anderen Programmen auf demselben Client-Betriebssystem verwalteten personenbezogenen Daten zu ermöglichen.

Der Titel fasst die gewählte Vorgehensweise zur Erfüllung der genannten Basisanforderungen bereits in vier Punkten zusammen: *Entwerfen*, *Client-Server-Architektur*, *Sicherheit* und *Notenverwaltung*. Das *Entwerfen* des Systems lässt sich dabei wiederum in drei Schritte, den Leitfaden dieser Arbeit, aufteilen.

1. Zu Beginn müssen dafür die Vorarbeiten geleistet werden. Dies umfasst das Erläutern der notwendigen Grundlagen sowie das Analysieren bestehender Systeme, die eine Lösung von Teilproblemen bieten. Auf Basis von R1 und R3 steht hier vor allem der Punkt *Sicherheit* im Mittelpunkt. Aufgrund der Abstraktheit dieses Begriffes werden deswegen eine Definition und ein Umriss der zu beachtenden Faktoren notwendig sein. Auch die Klärung der Rechtslage für die Erfüllung von R2 und R4 muss betrachtet werden.

2. Beim Entwurf kann dann auf Basis der Vorarbeiten der eigentliche Designprozess durchgeführt werden. Wesentlicher Bestandteil ist hier das Design der *Client-Server-Architektur*. Claudia Eckert beschreibt diese so: „Die Server [...] bieten Dienste an, die Clientrechner unter Verwendung von (unsicheren) Kommunikationsnetzen nutzen“ [Eck14, S. 466]. Es handelt sich in diesem Netzwerk also um zwei verschiedene Instanzen, deren Aufgaben jeweils charakterisiert werden müssen. Auch die Erfüllung von R1, R3 und R8 muss dabei beachtet werden. Neben der eigentlichen (hier sicheren) Kommunikation wird ferner der Entwurf der *Notenverwaltung* eine Rolle spielen. Dadurch wird sich zeigen, inwiefern das gesuchte System mit der gewählten *Client-Server-Architektur* realisierbar ist.

3. Zum Schluss bedarf es der beispielhaften Implementierung. Hier wird die Umsetzbarkeit des gewählten Konzepts im Vordergrund stehen. Gleichzeitig können dadurch eventuelle Problematiken bei der Umsetzung der Theorie in die Praxis festgestellt werden.

Letztendlich soll die Arbeit auch dazu dienen, interessierten Lehrerinnen und Lehrern einen Einblick in die Thematik der Kryptografie zu bieten. Dadurch soll zum Einen das Bewusstsein für einen sicheren Umgang mit schützenswerten, digitalen Daten gefördert werden. Zum Anderen wird es, vorwiegend bei der Beschreibung der *Notenverwaltung*, notwendig sein, durch gezielte Vergleiche mit der handschriftlichen Methode gewissen Vorbehalten auf den Grund zu gehen. Aufgrund des Umfangs des Themas können an einigen Stellen allerdings nur Verweise zum Vertiefen angegeben werden.

2. Grundlagen

In diesem Kapitel werden für die spätere Entwicklung der Architektur notwendige Grundlagen beschrieben. Dies schließt aufgrund der dienstlichen Tätigkeit der Lehrkraft bei der Notenverwaltung in erster Linie die rechtliche Fragestellung mit ein. In Kapitel 2.2 werden Grundbegriffe der Kryptografie vorgestellt. Es folgen eine Darstellung der Empfehlungen vom *Bundesamt für Sicherheit in der Informationstechnik* (BSI) und damit die Anforderungen an eine Implementierung. Schließlich werden aufbauend auf der Definition von Sicherheit und möglichen Angriffsszenarien die Sicherheitsanforderungen konkretisiert.

2.1. Rechtliche Grundlagen

Laut Art. 30 GG ist die Kultur- und Bildungspolitik Aufgabe der Länder. Folglich besitzt jedes Bundesland eigene Vorschriften für den Einsatz von privaten IT-Systemen durch Lehrer. Aufgrund des Bezugs zu Hannover wird hier das niedersächsische Recht als Grundlage dienen.

Zur Nutzung privater IT-Systeme durch Lehrkräfte im Dienst hat das niedersächsische Kultusministerium einen bis 2017 gültigen Runderlass [Nie] herausgegeben, der die §§8 und 16 NDSG [DN02] ausführt. Darin heißt es, dass der Einsatz „wegen der damit verbundenen datenschutzrechtlichen Risiken nur in Ausnahmefällen und nur mit Einschränkungen zugelassen werden“ [Nie, Ziffer 1.1] kann. Für eine Berechtigung bedarf es einer „schriftlichen Genehmigung der Schulleitung“ [Nie, Ziffer 2.1]. Ein solcher Ausnahmefall liegt bei den „üblicherweise zu Hause wahrgenommenen Aufgaben“ [Nie, Ziffer 1.1] vor und auch nur bei „Klassen- und Fachlehrern, Kursleitern und Tutoren“ [Nie, Ziffer 1.1]. Einzig für diejenigen Schülerinnen und Schüler, für die die Lehrkraft eine solche Funktion einnimmt, darf diese die Daten verarbeiten, „die Verarbeitung personenbezogener Daten von Erziehungsberechtigten und Lehrkräften ist auf privaten IT-Systemen nicht gestattet“ [Nie, Ziffer 1.3]. Dabei wird auch der erlaubte Datenrahmen (siehe Abb. 2.1) eingeschränkt. Aufgrund der Tatsache, dass eine solche Aufgabe weiterhin dienstliche Tätigkeit ist, bleibt die Schule als „Daten verarbeitende Stelle“ [Nie, Ziffer 1.2] für die Einhaltung des Datenschutzgesetzes in der Verantwortung. Auf dem für die Lehrkraft freigegebenen IT-System muss des Weiteren laut [Nie, Ziffer 4.1] sichergestellt werden, dass nur die jeweils befugte Lehrperson einen Zugang zu den Daten hat. Dies ist besonders bei externen Speichermedien zu beachten, bei internen ist „mindestens eine Zugriffskontrolle durch das Betriebssystem“ [Nie, Ziffer 4.1.2] einzurichten und durch technische Maßnahmen ein Online-Zugriff zu verhindern. Ein elektronisches Speichern ist zudem nur während der Funktionswahrnehmung für die jeweilige Schülerin und den jeweiligen Schüler durch die Lehrkraft rechtmäßig. In dieser Zeit muss auch bei einem Ausfall des IT-Systems die Verfügbarkeit garantiert werden. Zu guter Letzt ist ein Transport der Daten sowie die „Speicherung auf Speicherorten im Internet“ [Nie, Ziffer 4.4] nur nach einer Verschlüsselung der Daten gestattet.

Die Anforderungen R1, R3, R9 und R10 werden somit nochmals durch die Rechtslage bestätigt. Auch sind hiermit die in R2 und R4 erwähnten „geltenden Datenschutzgesetze“ konkretisiert. Ferner wird R3 weiter ausgeführt:

[R12] Das System muss die über das Internet übertragenen und auf einer Netzwerkressource gespeicherten Daten verschlüsseln.

Folgender Datenrahmen darf nicht überschritten werden:

- Namen,
- Geschlecht,
- Geburtsdatum, Geburtsort,
- Zugehörigkeit zu einer Religionsgemeinschaft,
- Klasse, Gruppe oder Kurs,
- Ausbildungsrichtung bzw. Ausbildungsberuf,
- Fächer,
- Art, Datum und Ergebnisse von Leistungskontrollen,
- Zeugnisnoten und andere Zeugniseintragungen.

Von diesen Daten dürfen nur die Daten verarbeitet werden, die für die jeweilige Aufgabenerledigung tatsächlich erforderlich sind.

Abbildung 2.1.: Erlaubter Datenrahmen auf privaten IT-Systemen, Quelle [Nie, Ziffer 3.2]

2.2. Kryptografie

„Kryptographie ist die Wissenschaft, die sich mit der Absicherung von Nachrichten beschäftigt“ [Sch06, S. 1]. Nachfolgend werden grundlegende Begriffe der Kryptografie, die in der Arbeit verwendet werden, erläutert. Dabei wird mit Alice die sendende Person, mit Bob der Empfänger bezeichnet. Mallory stellt einen Angreifer auf die Kommunikation zwischen Alice und Bob dar.

2.2.1. Hashfunktionen

Nach [Sch13a, Kapitel 14] ist eine *Hashfunktion* eine Funktion zur Berechnung einer Prüfsumme aus einer Eingabe. Die Länge der Eingabe ist dabei beliebig, die Anzahl der möglichen Prüfsummen – der Hashwerte – endlich. Unterschieden wird zwischen *kryptografischen* und *nichtkryptografischen Hashfunktionen*. An letztere wird die Anforderung gestellt, dass jeder Hashwert gleichwahrscheinlich ist und dass ähnliche Eingaben (z. B. nur ein Unterschied in einem Zeichen) nicht den selben Wert liefern. Anwendung finden diese Funktionen z. B. bei der Sicherstellung, dass eine Zeichenfolge richtig eingegeben wurde. Beinhaltet die Zeichenfolge, wie z. B. die IBAN, an einer definierten Stelle eine Prüfziffer (berechnet mit der Hashfunktion) kann damit ein Computer die Gültigkeit der restlichen Ziffern verifizieren.

Zusätzlich dazu wird bei kryptografischen Hashfunktionen die Anforderung gestellt, dass ein Angreifer nicht gezielt Kollisionen (gleiche Hashwerte bei unterschiedlicher Eingabe) herbeiführen kann. Dies ist u. a. bei digitalen Signaturen (siehe Kapitel 2.2.4) notwendig.

2.2.2. Symmetrische Verschlüsselung

Unter der symmetrischen Verschlüsselung werden nach [Sch06, S. 4f.] diejenigen Algorithmen zusammengefasst, bei welchen die Schlüssel zur Ver- und Entschlüsselung die selben sind. Die Sicherheit hängt hier von der Geheimhaltung des Schlüssels ab, da jeder, der in den Besitz des

Schlüssels kommt, Informationen chiffrieren und dechiffrieren kann. Allgemein teilt man die Algorithmen in zwei Bereiche ein: die *Stromchiffren*, die den Klartext bit- bzw. byteweise auslesen, sowie die *Blockchiffrierungen*, die den Klartext in Blöcken – meist der Größe 64 Bit – bearbeiten. Bei letzteren unterscheidet man laut [Eck14, Kapitel 7.5.3] zusätzlich in den *Betriebsmodi*, die die Verschlüsselungsmethode mehrerer Klartextblöcke festlegt. Beim *Electronic Code Book* (ECB) wird beispielsweise jeder Block einzeln mit dem Schlüssel verschlüsselt, wodurch die Entschlüsselung unabhängig der Blockfolge passieren kann. Aufgrund dieser Unabhängigkeit existieren verschiedene Sicherheitsprobleme, z. B. werden in diesem Modus gleiche Klartextblöcke auf gleiche verschlüsselte Blöcke abgebildet. Mit dem *Cipher Block Chaining* (CBC) versucht man diesem entgegenzuwirken, indem der n-te Klartextblock mit dem $(n-1)$-ten verschlüsselten Block XOR-verknüpft wird. Dadurch wird eine Verkettung der einzelnen Blöcke hervorgerufen, wodurch auch eine der Reihenfolge entsprechende Entschlüsselung notwendig wird. Weitere Details und Modi finden sich in [Eck14, Kapitel 7.5.3]. Ein bekannter Vertreter der symmetrischen Verschlüsselung ist laut [Sch13a, Kapitel 8] der *Advanced Encryption Standard* (AES).

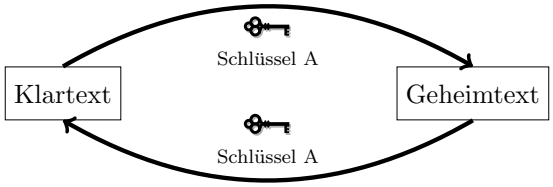

Abbildung 2.2.: Symmetrische Verschlüsselung

2.2.3. Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung liegen laut [Sch06, S. 5] zwei unterschiedliche Schlüssel vor. Man spricht auch von einem öffentlichen Schlüssel K_{pub} und einem privaten Schlüssel K_{priv}. K_{pub} wird zum Verschlüsseln, K_{priv} zum Entschlüsseln genutzt (Alternative: siehe Kapitel 2.2.4). Somit kann jeder mit dem veröffentlichen K_{pub} einen Klartext chiffrieren, wohingegen nur der Besitzer von K_{priv} den Geheimtext dechiffrieren kann. Nach [Sch13a, S. 184] wird sich dabei zu Nutze gemacht, dass die Umkehrung leicht berechenbarer mathematischer Funktionen, auch *Einwegfunktionen* genannt, nicht effizient berechenbar ist. Durch bestimmte Algorithmen, den sogenannten *Falltürfunktionen*, kann allerdings erreicht werden, dass unter Kenntnis von K_{priv} – und nur dann – die Entschlüsselung an Komplexität verliert. Ein bekannter Vertreter der asymmetrischen Verschlüsselung ist *RSA* (benannt nach den Erfindern Ron Rivest, Adi Shamir und Leonard Adleman) [Sch13a, Kapitel 11.3].

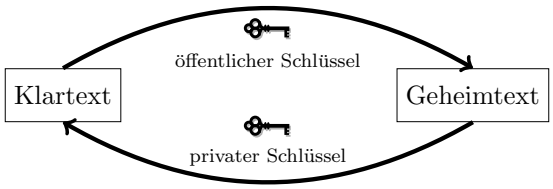

Abbildung 2.3.: Asymmetrische Verschlüsselung

2.2.4. Digitale Signaturen

Nutzt man bei der asymmetrischen Verschlüsselung K_{pub} zum Entschlüsseln und K_{priv} zum Verschlüsseln, so spricht man laut [Sch06, S. 5] von digitalen Signaturen. In [Sch06, S. 46] wird eine genauere Erläuterung gegeben und aufgezeigt, dass dies nur bei RSA der Fall ist. Bei anderen Algorithmen ist die geheime und die öffentliche Information anders implementiert. Unterschriften haben nach [Sch06, S. 41] folgende Eigenschaften:

1. Der Empfänger wird durch die Unterschrift von der willentlichen Unterzeichnung des Senders überzeugt.

2. Unterschriften sind sicher gegenüber Fälschungen.

3. Unterschriften sind nicht auf andere Dokumente übertragbar.

4. Unterschriften sind nicht zurücknehmbar.

5. Die Änderung eines unterschriebenen Dokuments nach der Unterschrift ist nicht mehr möglich.

Obwohl alle Eigenschaften in der Realität nicht vollständig zutreffen, findet die Unterschrift aufgrund eines nur schwer durchführbaren Missbrauchs ihre Anwendung. Ziel der digitalen Signatur ist es, die genannten Eigenschaften auf die digitale Welt zu übertragen. In der Praxis wird allerdings aufgrund der Ineffizienz asymmetrischer Algorithmen die in Abb. 2.4 gezeigte Vorgehensweise bei Signaturen angewendet.

1. Alice berechnet den Einweg-Hashwert eines Dokuments.

2. Alice chiffriert den Hashwert mit ihrem privaten Schlüssel, womit sie das Dokument unterzeichnet.

3. Alice sendet das Dokument und den signierten Hashwert an Bob.

4. Bob erstellt den Einweg-Hashwert des von Alice gesendeten Dokuments. Mit Alices öffentlichem Schlüssel und dem Algorithmus für elektronische Unterschriften dechiffriert er den signierten Hashwert. Stimmt dieser mit dem von ihm generierten Wert überein, ist die Unterschrift gültig.

Abbildung 2.4.: Algorithmus für Signaturen, Quelle [Sch06, S. 45f.]

Bei der genutzten Hashfunktion ist laut [Sch06, S. 46] darauf zu achten, dass diese nicht leicht umkehrbar ist. Auf diese Weise würde Alice sonst unwissentlich eine Vielzahl anderer Dokumente unterzeichnen, da Mallory ein anderes Dokument mit dem selben Hashwert erstellen könnte (siehe auch Kapitel 2.2.1).

2.2.5. PKI-Infrastrukturen

Setzt man asymmetrische Kryptografie ein, so ergeben sich laut [Sch13a, S. 506f.] vier Probleme:

1. Ein öffentlicher Schlüssel sagt nichts über die *Authentizität* der Person aus. Dadurch ist es für Bob nicht möglich, mit dem Schlüssel alleine zu prüfen, ob dieser von Alice oder Mallory kommt.

2. Stellt Alice fest, dass ihr privater Schlüssel kompromittiert wurde, so bietet ihr das asymmetrische Verfahren weiterhin nicht die Möglichkeit, jemand anderen von der *Sperrung* ihres Schlüssels zu unterrichten.

3. Es besteht die Möglichkeit des *Abstreitens*. Da sich ein Schlüssel nicht eindeutig zu einer Person zuordnen lässt, könnte Alice gegenüber Bob erfolgreich leugnen, eine Signatur ausgestellt zu haben.

4. Es fehlt die Möglichkeit, einen Schlüssel an bestimmte Bedingungen (z. B. der Gültigkeitsdauer), den sogenannten *Policies*, zu knüpfen.

In der Praxis wird deshalb laut Schmeh der öffentliche Schlüssel mit Zusatzinformationen, wie zum Beispiel dem Namen des Inhabers versehen. Eine digitale Signatur über all diese Angaben sorgt dann dafür, diesen trauen zu können. Die gesamte Datenstruktur wird *Zertifikat* genannt. Das Signieren der Zertifikate übernimmt eine sogenannte *Certification Authority* (CA), welcher alle vertrauen müssen. Zur Verifizierung braucht jeder dann nur den öffentlichen Schlüssel der CA. Hier ergibt sich wieder das Ausgangsproblem der Schlüsselverteilung, welches darüber gelöst wird, dass der eine benötigte Schlüssel eingangs über einen sicheren Kanal an alle verteilt wird. Auch dieser sollte in Form eines Zertifikats vorliegen, um nachfolgend die gleichen Möglichkeiten, z. B. das Widerrufen, wie bei den signierten Zertifikaten zu erlauben. Somit reduziert sich die Anzahl der von Hand zu prüfenden Zertifikate von der Anzahl an Kommunikationspartnern auf eines, welches von der CA kommt.

Als *Public Key Infrastructure* (PKI) wird dabei das notwendige Equipment wie eingesetzte Hard- und Software bezeichnet. In [Sch13a, Kapitel 26] und [Eck14, Kapitel 9.1.3] werden weitere diesem Prinzip zugrundeliegende Modelle und Standards beschrieben. Insgesamt folgt folgende Anforderung an das Ausgangsproblem:

[R13] Das System muss für alle eingesetzten asymmetrischen Schlüssel die Authentizität, Möglichkeit des Sperrens, Unmöglichkeit des Abstreitens und Einhalten von Policies garantieren.

2.2.6. Sitzungsschlüssel

Setzen Alice und Bob zur Kommunikation nicht direkt ihr getauschtes Geheimnis – hier *Masterschlüssel* genannt – ein, so kann man den Schaden, falls Mallory ein Schlüssel bekannt wird, laut [Sch13a, Kapitel 20.3.5] eingrenzen. Durch einen von Sitzung zu Sitzung unabhängig aus dem Masterschlüssel neu generierten Schlüssel ergibt sich die Möglichkeit, sowohl eine *Backward* als auch *Forward Security* zu garantieren. Gelangt ein Unbefugter in den Besitz des Sitzungsschlüssels, so kann er zwar alle Nachrichten der momentanen Sitzung entschlüsseln, hätte allerdings weder Zugriff auf vergangene oder zukünftige Sitzungen. Der Masterschlüssel sollte weiterhin besonders geschützt werden, da Mallory aus diesem auch die Sitzungsschlüssel berechnen kann.

[R14] Das System muss bei sitzungsbasierter Kommunikation Backward und Forward Security garantieren.

Zur Generierung des Sitzungsschlüssels mit Backward und Forward Security kann das *Diffie-Hellman-Verfahren* (siehe [Sch13a, Kapitel 11.2]) mit einem vorher ausgetauschten Schlüssel eingesetzt werden, um eine *Man-in-the-Middle-Attacke* zu verhindern. Alternativ bietet sich eine Erweiterung, das *MQV-Verfahren* (siehe [Sch13a, Kapitel 11.2.2]), an. Auch mit RSA kann man die erwähnten Vorteile erreichen – allerdings nur mit jeweils neu generierten Schlüsselpaaren, bei denen auch eine Man-in-the-Middle-Attacke verhindert werden muss (siehe [Sch13a, S. 393f.]).

2.2.7. Empfehlungen vom Bundesamt für Sicherheit in der Informationstechnik

Viele der in der Theorie beschriebenen Algorithmen lassen in ihrer Spezifikation einen gewissen Spielraum für ihren Einsatz. Vor allem die Schlüssellänge ist meist durch den Entwickler (quasi) frei wählbar. Um nicht die Sicherheit des eigentlichen Algorithmus zu gefährden und da das System in einer Behörde, der Schule, zum Einsatz kommt, muss folgende Anforderung gestellt werden:

[R15] Das System muss die Richtlinie [Inf15] des Bundesamts für Sicherheit in der Informationstechnik (BSI) bei der Verwendung kryptografischer Algorithmen einhalten.

Dabei ist darauf zu achten, dass die Empfehlungen laut [Inf15, Kapitel 1.2] aufgrund nicht vorhersehbarer Entwicklungen nur bis 2021 gelten. Als nach Ende 2015 sichere Hashfunktionen werden „SHA-256, SHA-384, SHA-512 oder SHA-512/256„[Inf15, S. 38] aufgezählt. Für Blockchiffren empfiehlt das BSI „AES-128, AES-192 und AES-256“ [Inf15, S. 22]. Als Betriebsmodi werden „Galois-Counter-Mode (GCM), Cipher-Block-Chaining (CBC) und Counter Mode (CTR)“ [Inf15, S. 23] als sicher eingestuft.

Für RSA wird geraten, nach 2016 für den Schlüssel 3000 Bits zu verwenden – 2000 Bits sei die Mindestlänge [Inf15, S. 28]. Zur Formatierung solle laut [Inf15, S. 37] EME-OAEP eingesetzt werden. RSA wird für die asymmetrische Verschlüsselung (siehe [Inf15, S. 28]) und nach [Inf15, S. 42] auch für digitale Signaturen empfohlen. In [Inf15, S. 52] wird u. a. die Verwendung des Diffie-Hellman-Verfahrens zum Aushandeln der asymmetrischen Schlüssel geraten. Um Informationen langfristig schützen zu können, werden in [Inf15, S. 31f.] weitere Punkte aufgezählt. Daraus ergeben sich:

[R16] Das System muss dem Entwickler die Möglichkeit bieten, verwendete Verfahren und Schlüssellängen ohne Änderung der darauf aufbauenden Algorithmen auszutauschen.

[R17] Das System muss bei asymmetrischer Verschlüsselung längere Schlüssel als vom BSI empfohlen verwenden.

[R18] Das System muss die Menge der über öffentliche Netzwerke übertragenen Geheimnisse minimal halten.

2.3. Cloud-Computing

In [Eck14, S. 725] heißt es, dass unter Cloud-Computing das Bereitstellen von Diensten und Ressourcen in der IT zusammengefasst wird. Darunter fällt z. B. das Ablegen von Daten bei externen Anbietern. Der Cloud-Dienst stellt dann auf Nachfrage die Ressource „Speicher“ bereit. Ähnliches gilt für Anwendungen, wo Rechenfähigkeit zur Verfügung gestellt wird. Auch wird aufgeworfen, dass mit dieser Technik neue, teilweise noch unbeantwortete Fragen im Bereich der Sicherheit auftreten. Dazu gehört z. B. die Gewährleistung, dass nur autorisierte Nutzer auf hochgeladene Daten Zugriff haben. Auch muss die Sicherheit eingesetzter Virtualisierungen garantiert werden, um Prozesse unterschiedlicher Nutzer voneinander trennen zu können. Außerdem zählt die Transparenz der Datenverarbeitung dazu.

2.4. Was bedeutet eigentlich Sicherheit?

Bruce Schneier macht in [Sch06, S. 8f.] die Aussage, dass das Maß an Sicherheit eines Algorithmus davon abhängt, wie schwer dieser zu brechen ist. Daraus schlussfolgert er, dass ein Algorithmus als (wahrscheinlich) sicher anzunehmen ist, wenn die nachfolgenden Eigenschaften erfüllt sind:

1. Der erforderliche finanzielle Aufwand zum Aufbrechen des Algorithmus ist höher als der Wert der verschlüsselten Daten (dieser nähme meist mit der Zeit ab).

2. Die hierfür erforderliche Zeitspanne ist größer als die Zeit, in der die Daten geheim bleiben müssen.

3. Das chiffrierte Datenvolumen ist kleiner als die zum Knacken des Algorithmus benötigte Datenmenge.

Seine Betonung auf „wahrscheinlich“ rührt daher, dass nach seiner Aussage immer mit neueren Entwicklungen in der Kryptoanalyse zu rechnen ist. Da diese Definition immer noch sehr abstrakt ist, sollen im Folgenden nun weitere Facetten des Begriffes Sicherheit erläutert werden, um anschließend auf mögliche Angriffe einzugehen.

2.4.1. Facetten des Begriffs Sicherheit

Bei der Definition von Sicherheit geht Schmeh in [Sch13a, Kapitel 2.2] von zwei Varianten aus, die jeweils auf den möglichen englischen Übersetzungen basieren: *Safety* und *Security*. *Safety* beschreibt nach seiner Aussage den Schutz vor zufälligen, normalerweise ungewollten Ereignissen. Dazu zählt z. B. der Schutz vor dem Ausfall von Hardwarekomponenten (aufgrund eines Defekts), voreiliges Löschen von Dateien und Überschwemmungen. *Security* stellt hingegen den Schutz vor beabsichtigten Störungen dar. Hierzu gehört z. B. der Schutz der Hardware vor Sabotage sowie der Schutz geheimer Informationen vor Angreifern und Viren. Dahingegen definiert das *NIST Computer Security Handbook* [Nis, S. 5] (Computer-)Sicherheit wie folgt:

> Computer Security: The protection afforded to an automated information system in order to attain the applicable objectives of preserving the integrity, availability and confidentiality of information system resources (includes hardware, software, firmware, information/data, and telecommunications).

Die drei dort erwähnten Schlüsselziele *Integrität*, *Verfügbarkeit* und *Vertraulichkeit* der Computersicherheit, die laut Stallings auch als *CIA triad* bekannt sind, werden von ihm in [Sta14, S. 29f.] noch ausführlicher beschrieben. Frei übersetzt definiert er diese wie folgt:

Integrität beinhaltet zwei Konzepte:

- **Datenintegrität** stellt sicher, dass Informationen und Programme nur in einer spezifizierten und erlaubten Art und Weise verändert werden.
- **Systemintegrität** stellt sicher, dass ein System seine vorgesehene Aufgabe in einer unbeeinträchtigten Art und Weise, frei von absichtlicher oder unabsichtlicher Manipulation des Systems, ausübt.

Verfügbarkeit stellt sicher, dass das System zeitnah arbeitet und autorisierten Nutzern nicht die Dienste verwehrt werden.

Vertraulichkeit beinhaltet zwei Konzepte:

- **Datenvertraulichkeit** stellt sicher, dass private oder vertrauliche Informationen unautorisierten Einzelpersonen nicht zur Verfügung gestellt oder bekannt gemacht werden.
- **Privatsphäre** stellt sicher, dass Einzelpersonen kontrollieren oder beeinflussen, welche Informationen über sie gesammelt oder gespeichert werden und durch wen und wem diese Informationen bekannt gemacht werden können.

Zusätzlich zu den CIA triad werden laut Stallings zur Vervollständigung des Sicherheitsbegriffs häufig zwei weitere Konzepte erwähnt [Sta14, S. 30] (frei übersetzt und zusammengefasst):

Authentizität ist die Eigenschaft, die eigene Identität zu beweisen und dieser zu vertrauen sowie der Gültigkeit einer Nachricht zu trauen. Dies beinhaltet die Überprüfung, dass ein Nutzer auch derjenige ist, der er behauptet zu sein, und jeder Systemeingang von einer vertrauten Quelle kommt.

Zurechenbarkeit entspricht der Nachverfolgbarkeit von ausgeführten Aktionen. Dies schließt die Unleugbarkeit, Eindringungserkennung und -verhinderung sowie das Rückgängigmachen von Aktionen mit ein. Aufgrund der Tatsache, dass wirklich sichere Systeme heutzutage ein nicht erreichbares Ziel sind, ist es notwendig, für spätere forensische Analysen Sicherheitsbrüche aufzuzeichnen, um die dafür verantwortliche Partei aufspüren zu können.

Es sei im Folgenden angenommen, dass mit Erfüllung dieser Aspekte ein System als sicher einzuschätzen ist:

[**R19**] Das System muss bei der Speicherung und Übertragung von schützenswerten Daten die Sicherheitsaspekte Security und Safety, die CIA triad Datenintegrität, Systemintegrität, Verfügbarkeit, Datenvertraulichkeit und Privatsphäre sowie die Authentizität und Zurechenbarkeit garantieren.

2.4.2. Real-World-Attacken

Lässt man die mathematischen und statistischen Aspekte von Kryptografie außer Acht, so tun sich bei der Implementierung von auf Sicherheit basierten Programmen weitere Angriffsstellen auf. Diese anschließend (ohne Anspruch auf Vollständigkeit) erwähnten Angriffe und Gegenmaßnahmen werden auch als *Real-World-Attacken* bezeichnet [Sch13a, Kapitel 17]. Alle Erläuterungen stammen, soweit nicht anders markiert, aus dieser Quelle.

Seitenangriffe

Die erste, direkt schon bei der Hardware-Implementierung zu beachtende Angriffsmethode sind die *Seitenangriffe*. Unter diesen Oberbegriff fallen Attacken, welche sich an Zusatzinformationen wie zur Laufzeit aufgenommener Messwerten bedienen, um eine bestimmte Implementierung zu knacken. Messwerte können z. B. die Zeit, der Stromverbrauch oder die auftretenden Fehler bei einer gewollten Beschädigung/ Falschbedienung des Moduls sein. Gegenmaßnahmen sind vom Hersteller durch geeignetes Manipulieren der möglichen Messwerte durchzuführen, sodass diese von außen zufällig erscheinen. Daraus lässt sich schließen, dass ein Anwender in diesem Fall nicht mehr tun kann, als die Hardware vor dem direkten Zugriff durch den Angreifer zu schützen.

Physikalische Angriffe

Eine weitere, direkt auf die Hardware durchgeführte Angriffsmethode stellen die *physikalischen Angriffe* dar. Durch Einfrieren des flüchtigen Speichers oder durch Röntgenstrahlung kann dafür gesorgt werden, dass auch nach einem Verlust der Stromversorgung der Inhalt für kurze Zeit in dem Modul erhalten bleibt. Aber auch mit flüssigem Helium und einem Elektronenmikroskop oder direktem Abgreifen der Leitungen des jeweiligen Moduls ist es möglich, Speicherbereiche und somit auch temporär genutzte geheime Schlüssel zur Laufzeit auszulesen.

Neben mehreren physikalischen Sicherheitsschichten sollte als Gegenmaßnahme auch das *sichere Löschen* in Betracht gezogen werden. So ist es notwendig, darauf zu achten, dass ein Zurückgewinnen gelöschter Daten durch einen Angreifer nicht mehr möglich ist. Dem gegenüber

steht die Problematik, dass Betriebssysteme zur Speicherverwaltung häufig Teile des Arbeitsspeichers auf die Festplatte auslagern, was ein beliebiges Programm gar nicht mitbekommt. Darüber hinaus wird aus Effizienzgründen der zu löschende Speicherbereich nur zum Überschreiben freigegeben, sodass die Daten trotzdem vorhanden bleiben. Bei magnetischen Speichern kommt die Problematik hinzu, dass erst nach mehrfachem Überschreiben mit einem bestimmten Muster ein Zurückgewinnen mit speziellen Geräten nicht mehr möglich ist. In [Sch13a, S. 332] werden mehrere Quellen mit Empfehlungen u. a. vom BSI genannt, wie das Überschreiben auszusehen hat. Somit reicht in den meisten Fällen ein einfaches Löschen nicht aus, um ein Wiederherstellen zu verhindern. Die jeweilige Art des sicheren Löschens hängt dabei stark vom Speichermanagement des Betriebssystems ab.

Malware-Angriffe

Verschafft sich eine unautorisierte Person z. B. durch Einsatz eines Virus, Computerwurms oder Trojaners Zugang zu einer Arbeitsumgebung, so spricht man von einem *Malware-Angriff*. Dieses Problem existiert allerdings nicht nur für Krypto-Module, da sich mit solcher Schadsoftware im Allgemeinen Informationen eines fremden Rechners ausspähen lassen. In den letzten Jahren häufig aufgetretene, direkt auf kryptografische Software spezialisierte Malware ist zur Ausspähung von Passwörtern oder zur Durchführung von *Darstellungsangriffen* programmiert. Letztere beschreiben häufig mit digitalen Signaturen im Zusammenhang stehende Angriffe, welche die Tatsache ausnutzen, dass ein Nutzer nicht sicherstellen kann, dass das, was er sieht, auch wirklich das ist, womit der Computer arbeitet – z. B. dass die angezeigte Datei zum Signieren nicht der Datei entspricht, die tatsächlich zum Signieren an eine Smartcard geschickt wird. Ferner können auch vom Hersteller absichtlich eingebaute Hintertüren mit in den Bereich der Malware-Angriffe zählen. Aufgrund der allgemeinen Problematik gibt es kein kryptografisches Verfahren, welches mit hundertprozentiger Sicherheit vor dieser Schadsoftware schützt. Mit einem guten Sicherheitskonzept in Unternehmen und dem Einsatz von Virenscannern durch Privatanwender, sowie die Offenlegung bzw. Evaluierung des Programmquellcodes durch den Entwickler kann das Risiko allerdings minimiert werden.

Implementierungsfehler

„Die Entwicklung von Programmcode ist von jeher eine fehleranfällige Sache“ [Sch13a, S.332]. *Implementierungsfehler* sind somit in der Praxis nicht selten und stellen gerade in der Kryptografie eine große Gefahr dar. Beispiele dafür sind die falsche Umsetzung theoretischer Algorithmen, Fehler in der Kontrolle, wer auf welche Dateien zugreifen darf, nicht durchgeführtes sicheres Löschen (siehe oben) sowie Mängel im Zusammenspiel oder der Verwendung einzelner Module. Für Gegenmaßnahmen muss in dem Gebiet der Softwareentwicklung angesetzt werden. Software ohne Sicherheitsfunktionen stellt nur den Anspruch nach korrekter Ausführung. Kryptografische Software hingegen ist „erst dann korrekt, wenn [zusätzlich] Schlüssel und Klartexte ausreichend geschützt sind“ [Sch13a, S. 335f.]. Dazu zählt laut Schmeh das sofortige und sichere Löschen ungebrauchter Schlüssel und Klartexte, das Verschlüsseln dauerhaft gespeicherter Schlüssel, ein geeigneter Selbsttest für verwendete Zufallsgeneratoren, Fehlermeldungen ohne Preisgabe geheimer Informationen sowie ein modularer Aufbau des Systems. Letzteres trägt dazu bei, einzelne Verfahren leichter analysieren und austauschen zu können. Durch eine klare Struktur wird damit auch das gesamte System übersichtlicher.

Ferner ist bei auf Kryptografie basierter Software mehr als sonst auf fehlerfreie Programmierung und ausreichende Tests zu achten. Schließlich rät Schmeh zur Quellcodeoffenlegung, der Trennung geheimer Schlüssel vom System (z. B. durch externe Speichermedien wie Smartcards) sowie einer Evaluierung der Software nach allgemeinen Kriterien (z. B. der *Common*

Criteria). Hierdurch ist es möglich, die Wahrscheinlichkeit von Implementierungsfehlern weiter einzuschränken. Anwendern empfiehlt Schmeh, lieber alt bewährte Programme als die neuste Betaversion zu nutzen und sicherheitsbetreffende Updates rechtzeitig zu installieren. Darüber hinaus sollte man wachsam sein, um bekanntwerdende Sicherheitslücken frühzeitig mitzubekommen und die Möglichkeit zum Reagieren zu haben.

Schwachstelle Anwender

Zu guter Letzt ist es der *Anwender*, der bestimmt, ob eine Implementierung als sicher angesehen werden kann. Schmeh beschreibt diese wie folgt: „Anwender sind faul, vergesslich und interessieren sich nicht für Kryptografie“ [Sch13a, S. 339]. Somit sieht er es für die Praxistauglichkeit als notwendig an, in die Systementwicklung mit einfließen zu lassen, wer das Programm am Ende nutzen wird. Eine Problematik ist die *Nicht-Nutzung* von Verschlüsselung durch den Anwender. Auch wenn dies erst einmal trivial klingt, so ist dieses Phänomen aufgrund von Bequemlichkeit oder Unwissenheit des Nutzers eines der größten Probleme im praktischen Einsatz. Dazu kommen *Bedienfehler*, welche auch aus dem fehlenden Wissen über Kryptografie resultieren. Beispielsweise könnte ein E-Mail-Programm, welches ein Versenden auch ohne vorheriger Verschlüsselung erlaubt, dazu führen, dass der Nutzer (z. B. aus Gewohnheit) die gebotene Funktion gar nicht nutzt, indem er sofort auf „Senden“ klickt.

Neben der Wahl *leicht zu erratender Passwörter* durch den Anwender ist insbesondere das sogenannte *Social Engineering* ein weiteres Problemfeld der Praxis. Unter diese Kategorie fallen Angriffe, bei der durch Täuschen des Anwenders Sicherheitsvorkehrungen umgangen werden. In Unternehmen könnte ein Angreifer z. B. als Reinigungskraft getarnt oder durch Ausnutzen eines freundlichen Mitarbeiters, der die Tür aufhält, im Gebäude nach ungesperrten PCs oder herumliegenden Dokumenten suchen. Auch das Altpapier ist eine mögliche Anlaufstelle. Viele Sicherheitsanalysen decken diese Form des Angriffs mit ab, gleichwohl ist trotz entsprechender Warnungen in den Medien ein viel größeres Problem das sogenannte *Phishing*. Durch gefälschte E-Mails oder Telefonanrufe soll hierbei erreicht werden, an geheime Informationen wie z. B. Passwörter des Nutzers zu kommen.

Wirksame Gegenmaßnahmen gegen die „Sicherheitslücke“ Anwender sind *Sensibilisierung*, *Benutzerfreundlichkeit*, *Vorschriften* und *Verhinderung*. Zum ersten Punkt gehört das Aufklären der Nutzer über mögliche Gefahren, um ihr Bewusstsein zu erhöhen. Darüber hinaus können Vorschriften in Unternehmen das Nutzen kryptografischer Software erzwingen. Um diesen letzten Ausweg zu umgehen, sollte bei der Entwicklung auf Benutzerfreundlichkeit geachtet werden. Dazu zählt, dass ein Programm durch den Einsatz von Kryptografie nicht komplizierter werden darf. Jegliche aufgrund von Verschlüsselung hervorgerufene Fehlermeldung oder Inkompatibilität sowie jeder zusätzliche Button wirkt diesem entgegen, sodass ein besonderes Augenmerk auf die Benutzeroberfläche unabdingbar ist. Am Ende hilft auch das bloße Verhindern, kryptografische Komponenten zu umgehen. Beispielsweise kann anstatt eines Passworts, welches der Nutzer preisgeben kann, ein Fingerabdruck zur Authentifizierung genutzt werden (dieser kann nicht so einfach weitergegeben werden).

Insgesamt zieht Schmeh am Ende das Fazit, dass das grundlegende Problem der Kryptografie ist, „immer mehr und immer bessere Algorithmen und Protokolle [zu entwickeln], während es nur selten gelingt, diese auf ausreichend zuverlässige Weise einzusetzen“ [Sch13a, S. 346]. Er äußert deshalb den Wunsch, neben der mathematischen Seite noch mehr die genannten Problematiken der *Real-World-Attacken* in den Mittelpunkt zu stellen.

[R20] Das System muss den Schutz vor den in Kapitel 2.4.2 genannten Real-World-Attacken garantieren.

[R21] Das System sollte, sofern sinnvoll, bereits existierende Verfahren der Kryptografie verwenden.

2.4.3. Angriffsmöglichkeiten auf kryptografische Protokolle

Auch wenn Alice und Bob ein Protokoll mit als sicher einzustufender Kryptografie verwenden, kann nicht angenommen werden, dass Mallory keine Angriffsmöglichkeit mehr besitzt. In [Sch13a, Kapitel 20.3] werden weitere Bedrohungen beschrieben, auf die im Folgenden kurz eingegangen werden soll. Dazu nimmt Schmeh an, dass Mallory das von Alice und Bob verwendete Protokoll kennt. Zusätzlich wird Mallory nicht nur versuchen, die Kommunikation abzuhören, sondern diese auch aktiv zu manipulieren.

Replay-Attacke

Fängt Mallory eine von Alice an Bob gesendete Nachricht ab, so kann er diese zwar aufgrund der verwendeten Verschlüsselung nicht lesen, aber theoretisch zu einem späteren Zeitpunkt nochmals an Bob senden. Einen solchen Angriff nennt man *Replay-Attacke*. Beinhaltet das abgegriffene Paket eine Aufforderung, dass Bob einen bestimmten Befehl ausführen soll (dies kann Mallory u. a. aufgrund seines Wissens über das Protokoll feststellen), so wird Bob dies ein weiteres Mal tun. Um eine solche Attacke zu verhindern, sollte jede Nachricht deshalb entweder einen Zeitstempel oder eine inkrementierende Nummer beinhalten. Dadurch, dass sich dann keine Nachrichten von Alice mehr ähneln, fällt die Wiederholung auf.

Spoofing-, Man-in-the-Middle- und Hijacking-Attacke

Weiterhin kann Mallory eine *Spoofing-Attacke* starten. Dabei versucht er, eine Kommunikation mit Bob aufzubauen, bei der er sich als Alice ausgibt. Zur Verhinderung sollte eine Authentifizierung im Protokoll vorgesehen sein, damit Bob die Authentizität (siehe Kapitel 2.4.1) des Gegenübers sicherstellen kann.

Startet Mallory eine *Man-in-the-Middle-Attacke* (MITM), so versucht er, beim Verbindungsaufbau unbemerkt Mittelsmann der Kommunikation zu werden. Tauschen Alice und Bob beispielsweise im Diffie-Hellman-Verfahren ihre Informationen aus, um daraus den zu verwendenden geheimen Schlüssel zu berechnen, fängt Mallory diese Nachrichten ab und tauscht die öffentlichen Schlüssel mit seinem. Alice und Bob bekommen also die falschen Informationen, um den geheimen Schlüssel zu berechnen. Tatsächlich bauen deshalb beide eine Verbindung zu Mallory auf, welcher sämtliche Nachrichten mitliest und aktiv zu seinem Vorteil verändert.

Leitet Mallory nach einem MITM-Angriff keine Nachrichten von Alice mehr weiter und übernimmt somit im Namen von Alice die Kommunikation mit Bob, so nennt man dies eine *Hijacking-Attacke*. Bei allen drei Attacken sollten Alice und Bob als Gegenmaßnahme eine Authentifizierung sowie Verschlüsselung nutzen.

Known-Key-Attacke

Wie in Kapitel 2.2.6 erwähnt, existieren auch Maßnahmen zur Schadensreduzierung, falls Mallory ein Schlüssel bekannt wird. Erlangt Mallory Kenntnis von einem geheimen Schlüssel, so kann er eine *Known-Key-Attacke* durchführen.

Verkehrsflussanalyse

Aber auch ohne Kenntnis des Inhalts kann Mallory mit einer *Verkehrsflussanalyse* an Informationen über die Kommunikation zwischen Alice und Bob gelangen. Dabei führt er eine Statistik

über den Nachrichtenaustausch, z. B. wer mit wem wann kommuniziert. Eine Verhinderung dessen ist nicht möglich, allerdings lässt sich das Aufbauen einer Statistik erschweren. Zuerst sollten sämtliche Nachrichten – nicht nur die Wichtigen – verschlüsselt werden, damit ein Trennen von interessanten und uninteressanten Daten nicht mehr möglich ist. Außer den Daten selbst sollten auch die Meta-Daten (bis auf die Zieladresse) verschlüsselt werden, um so wenig wie möglich Preis zu geben. „Dummy-Mails“ [Sch13a, S. 396] können die Analyse zusätzlich behindern.

Denial-of-Service-Attacke

Mit einer *Denial-of-Service-Attacke* kann Mallory schließlich die Kommunikation zwischen Alice und Bob unmöglich machen. Dabei kann er versuchen, Nachrichten von Alice zu blockieren oder durch Veränderungen ein erfolgreiches Entschlüsseln durch Bob zu verhindern. Auch kann er durch das Senden massenweiser falscher Nachrichten an Bob seinen Rechner überlasten oder sogar zum Absturz bringen, sodass dieser Alice nicht rechtzeitig antworten kann. Scheint es auf den ersten Blick so, dass dadurch nur die Kommunikation unterbrochen wird, könnte Mallory langfristig dadurch z. B. erreichen, dass Alice und Bob auf eine unverschlüsselte Variante zurückgreifen, da sie feststellen, dass das verschlüsselte Versenden nicht funktioniert.

Insgesamt ergibt sich aus R3 also:

[R22] Das System muss bei den eingesetzten Protokollen zum Transport schützenswerter Daten den Schutz vor den in Kapitel 2.4.3 genannten Angriffsmöglichkeiten garantieren.

3. Analyse ausgewählter Programme

In diesem Kapitel sollen nun Ideen für eine mögliche Umsetzung der aufgezählten Systemanforderungen gesammelt werden. Dazu wurden repräsentativ einige Produkte zur Analyse ausgesucht. Mit *MEGA* und *TeamDrive*, zwei Cloud-Diensten mit Ende-zu-Ende-Verschlüsselung, werden Systeme zur Erfüllung der in R3 geforderten sicheren Datenübertragung untersucht. Auch *Boxcryptor* setzt dies – allerdings unabhängig von dem gewählten Cloud-Dienst – um. Nutzt man eine der in Kapitel 2.2 vorgestellten Methoden zur Erfüllung von R1, so wird es notwendig sein, die zugehörigen Schlüssel sicher aufzubewahren. Eine Möglichkeit dafür, auch hinsichtlich R8, soll mit dem *MobileSitter* aufgezeigt werden. Schließlich ist das Hauptziel ein digitales Verwaltungssystem für die Schule. Zu diesem Zweck wird *WebUntis* analysiert werden.

Eine grobe Übersicht der Ergebnisse findet sich in Anhang A.6. Die Analysen wurden auf Grundlage der theoretischen, meist von der Firma selbst beschriebenen, Konzepte durchgeführt. Eine Differenz zu der eigentlichen Implementierung kann somit nicht ausgeschlossen werden. Diese Tatsache wird allerdings für das zu entwerfende Konzept nicht von Belang sein.

3.1. MEGA

MEGA bewirbt direkt auf der Startseite seiner Website [MEGb] die Ende-zu-Ende-Verschlüsselung seines Dienstes. Angeboten werden Tools zur Synchronisation von Dateien auf mobilen Endgeräten und Computern, ein E-Mail und Chat-Service sowie Browser-Applikationen. Ein Datei-Upload ist mit jedem Browser möglich.

Wie genau die Verschlüsselung aussieht wird in dem Entwicklerhandbuch [MEGa] für das angebotene C++ SDK beschrieben. Dateien und Ordner sind als Knoten in einer Baumstruktur, d. h. ohne Zykel, repräsentiert. Nutzeraccounts sind über die jeweilige E-Mail-Adresse referenziert. Über das SDK ist es möglich, weitere Nutzerdaten hinzuzufügen – eine AES-CBC Verschlüsselung dieser Daten wird dabei empfohlen.

Laut [MEGa, Kapitel 12.2] kommt für symmetrische Verschlüsselung AES-128 mit CBC für Datei- und Ordner-Attribute und CTR für den Dateiinhalt zum Einsatz. Für jeden Knoten wird dabei jeweils ein zufälliger 128-Bit Schlüssel generiert. Ein Dateiknoten nutzt den selben Schlüssel für Attribute und Daten sowie einen 64 Bit langen zufälligen Startwert und einen 64 Bit MAC, um die Dateiintegrität prüfen zu können. Pro Nutzeraccount wird ein Masterschlüssel generiert, der mit ECB (und symmetrischer Verschlüsselung) alle ihm bekannten Knotenschlüssel chiffriert. Dieser Masterschlüssel wird mit einem aus dem Passwort berechneten Hashwert verschlüsselt und auf dem Server von *MEGA* gespeichert. Zusätzlich besitzt jeder Nutzer einen 2048 Bit langen RSA-Schlüssel, um Knotenschlüssel zwischen Nutzern austauschen zu können. Der private Schlüssel ist dabei mit dem Masterschlüssel des Nutzers verschlüsselt gespeichert. Eine Authentifizierung zur Überprüfung, von wem geteilte Daten stammen, wird laut [MEGa, Kapitel 12.4] nicht durchgeführt. Jegliche Server unterstützen HTTPS.

Mit dem Start von *MEGA* im Jahr 2013 analysierte u. a. Jürgen Schmidt in [Sch13b] das Verschlüsselungsverfahren. Laut seiner Aussage ist die Verschlüsselung im wesentlichen dazu da, *MEGA* vor Konsequenzen aufgrund illegaler Uploads durch Nutzer zu schützen, da der Betreiber dadurch keine Kenntnis über den Inhalt hat. Praktisch ist der Cloud-Dienst allerdings in der

Lage, an die geheimen Schlüssel zu gelangen: „So könnte *MEGA* beispielsweise den vom Browser geladenen Script-Code jederzeit so ändern, dass er etwa die zur Verschlüsselung eingesetzten Schlüssel an einen Web-Server sendet“ [Sch13b]. Ferner ist es nach diesem Artikel möglich, durch eine De-Duplizierung (zur Speichereinsparung und Verkürzung der Uploadzeit), welche laut Nutzungsbedingung möglich ist, gleiche Dateiinhalte zu erkennen. Weiß man nun, dass ein Nutzer eine problematische Datei hochgeladen hat, so kann der Serverbetreiber trotz Verschlüsselung feststellen, welcher Nutzer dieselbe Datei besitzt. Weiterhin wird in einem Nachtrag ein potentielles Sicherheitsrisiko der Implementierung beschrieben. „Derartige Anfängerfehler lassen auch für den Rest der Krypto-Infrastruktur nichts Gutes ahnen“ [Sch13b]. Der Artikel [Bö13] beschreibt ebenso, wie *MEGA* unbemerkt – auch aufgrund gerichtlichem Druck – an geheime Schlüssel gelangen könnte.

Trotz längerer Recherche ließen sich nicht genauere und aktuellere Analysen zu den verwendeten Algorithmen bzw. Risiken finden. In den meisten Medien wird dahingegen häufig auf die Nutzung vom *MEGA-Cloud-Dienst* zum Tausch von illegalen Daten eingegangen. Weiterhin startete der Gründer Kim Dotcom Anfang 2013 den Aufruf [Dot13], 10.000€ demjenigen zu bieten, der *MEGAs* Sicherheitskonzept brechen könnte. Ein veröffentlichter Erfolg wurde nicht gefunden.

3.2. TeamDrive

TeamDrive von der *TeamDrive Systems GmbH* ist laut eigenen Angaben [Gmbh] ein Cloud-Dienst mit AES-256 Ende-zu-Ende-Verschlüsselung. Über ein Programm ist es möglich, Dateien sicher auf verschiedenen Geräten mit den Plattformen „Windows, Mac OS X und Linux-Desktop-Clients sowie iOS- und Android-Mobilgeräten“ [Gmbh] zu verwalten. Durch vom Nutzer generierte Links können ferner auch nicht angemeldete Personen auf die jeweils freigegebenen Daten zugreifen. Zur Synchronisation kann sowohl die *TeamDrive Cloud* (Standort ist standardmäßig Europa) als auch ein eigener Server genutzt werden.

Im Jahre 2005 wurde *TeamDrive* vom *Unabhängigen Landeszentrum für Datenschutz Schleswig-Holstein* (ULD) geprüft, mit dem Datenschutzgütesiegel ausgezeichnet und seitdem alle zwei Jahre rezertifiziert (siehe [Gmbg]). Laut dem Kurzgutachten [BHO13] ist das Produkt auch „zur Speicherung und zum Teilen von Daten und Informationen, die einem Berufsgeheimnis i.S.d. §203 StGB unterliegen, geeignet“ [BHO13, S.6].

In dem Gutachten wird weiterhin der Aufbau des Produktes aufgeführt. Zu jedem *SharedSpace*, welcher die geteilten Daten enthält, ist es möglich, Lese- und Schreibrechte für Nutzer und Gruppen zu vergeben. Mit dem Löschen des *SharedSpace* werden sowohl Daten als auch Protokolldaten automatisch entfernt. Somit kann auch mit Einsatz der Software §6 LDSG eingehalten werden. Ein Zugriff auf die Daten besteht nach der Client-Installation über ein eingebundenes virtuelles Laufwerk. Für die auf dem Client ausgeführte AES-Verschlüsselung mit 256 Bit Schlüsseln kommt die *OpenSSL Bibliothek* zum Einsatz. Aus Kompatibilitätsgründen wird in dem aktuellen Produkt sowohl Diffie-Hellman als auch RSA mit 3072 Bit zum Schlüsselaustausch genutzt. Das dafür benötigte Schlüsselpaar wird pro Gerät angelegt. Dabei ist RSA aufgrund der schnelleren Schlüsselgenerierung der Standard neuerer Clients. Mit PBPG (PrimeBase Privacy Guard) ist es zudem möglich, Nachrichten- oder Schlüsselmanipulationen zu erkennen.

3.3. Boxcryptor

Boxcryptor, ein Programm der *Secomba GmbH*, bietet laut eigener Aussage in [Gmbe] die Möglichkeit, Dateien vor dem Upload in einen Cloud-Speicher zu verschlüsseln. Dafür legt das Programm nach der Installation ein virtuelles Laufwerk an. Verschiebt der Nutzer nun eine Datei

in dieses Laufwerk, wird diese im Hintergrund verschlüsselt und dann zu dem voreingestellten Dienst übertragen. Unterstützt werden u. a. die Cloud-Dienste von Dropbox, Google Drive, SugarSync und Microsoft OneDrive sowie alle Anbieter, die das WebDAV-Protokoll anbieten.

Zur Verschlüsselung, die in [Gmbf] sehr genau beschrieben wird, wird pro Datei ein eigener Schlüssel der Länge 256 Bit eingesetzt. Die Verschlüsselung erfolgt also pro Datei und nicht in einem Container. Dabei wird AES mit CBC und PKCS7 Padding genutzt. Zur sicheren Aufbewahrung dieser symmetrischen Schlüssel generiert *Boxcryptor* pro Nutzer ein RSA-Schlüsselpaar mit einer Länge von 4096 Bit und OAEP Padding. Hat ein Nutzer eine Datei mit einem neuen Schlüssel chiffriert, so wird dieser Dateischlüssel danach mit dem öffentlichen Nutzerschlüssel gesichert. Daraus folgt, dass nur noch der Nutzer als Inhaber des privaten Nutzerschlüssels Zugriff auf den Dateischlüssel hat. Möchte man eine Datei einem anderen Nutzer freigeben, so verschlüsselt *Boxcryptor* den Dateischlüssel zusätzlich noch mit dem öffentlichen Schlüssel des jeweils anderen Nutzers, welchen er von *Boxcryptors* Schlüsselserver bekommt. Alle so chiffrierten Schlüssel werden am Ende zusammen mit der Datei im Cloud-Speicher abgelegt.

Die Generierung der Schlüssel findet auf dem Gerät des Nutzers statt. Der private Nutzerschlüssel, der den Zugriff auf alle anderen Schlüssel erlaubt, wird mit einem Passwortschlüssel (AES-Schlüssel), der aus einem Passwort des Nutzers abgeleitet wird (PBKDF2 mit HMACSHA512, 10.000 Iterationen und einem 24Byte Salt), gesichert. Das Unternehmen selbst gibt an, dass es aufgrund der Zero-Knowledge Lösung von *Boxcryptor* keinen Zugriff auf verschlüsselte Dateien hat, da zur Authentifizierung am Schlüsselserver nur ein Passworthash an den Server übertragen wird. Dies führt u. a. dazu, dass ein Vergessen des Passworts den Zugriffsverlust auf die Daten zur Folge hat. Für Firmen bietet *Boxcryptor* deshalb eine Firmenschlüssel-Option an, über welche eine Wiederherstellung durch einen vom Unternehmen eingesetzten Administrator möglich ist.

Des Weiteren bietet *Boxcryptor* an das Nutzer-Konzept angelehnte Verfahren für Gruppen und Firmen sowie, bei Aktivierung, eine Dateinamensverschlüsselung an. Auf Nachfrage (siehe Anhang A.5) wurde bestätigt, dass die gesamte Sicherheit vom Passwort des Nutzers abhängt. Durch die Trennung von Schlüsselserver (verwaltet von *Boxcryptor*) und Cloud-Speicher (verwaltet vom jeweiligen Cloud-Dienst) kommt allerdings ein erheblicher Mehraufwand auf einen Angreifer zu, um an den Klartext einer Datei zu gelangen. Aus diesem Grund ist es wahrscheinlicher, dass ein Gerät des jeweiligen Nutzers zum Ausspähen von Geheimnissen angegriffen wird. Hauptziel der Software ist es, den Dateninhalt vor dem Cloud-Dienst zu schützen.

3.4. MobileSitter

Der *MobileSitter* vom *Frauenhofer-Institut für Sichere Informationstechnologie SIT* ist nach [Gmbb] ein Passwortmanager für Smartphones. Laut [Gmbc] haben Passwortmanager einen zentralen Angriffspunkt: den verwendeten Master-Key. Sichere Verschlüsselungsalgorithmen beruhen darauf, dass alle möglichen Schlüsselkombinationen gleich wahrscheinlich sind. Dadurch, dass der Master-Key aber durch die Tastatur eingegeben werden muss und laut einer Studie von ElcomSoft „ca. 40% aller geschäftlich genutzten Passwörter in einem Wörterbruch zu finden [sind]“ [Gmbc], steigt die Wahrscheinlichkeit an den richtigen Master-Key durch Brute-Force- oder Wörterbuch-Angriffe zu gelangen.

Die Tatsache, dass solche Angriffe effektiv sind, liege an der Rückmeldung des Dienstes, ob ein Entschlüsselungsversuch erfolgreich war oder nicht. Um dies zu umgehen entschlüsselt der *MobileSitter* die Passwörter immer mit dem angegebenen Master-Key, ohne anzugeben, ob der gewählte Schlüssel der Richtige ist. Da auch die Zeichenzahl immer gleich bleibt, ist es für einen Angreifer nach [Gmbc] nicht sichtbar, ob der gewählte Master-Key richtig ist. Eine Prüfung der Korrektheit ist somit nur möglich, indem das vermeintlich entschlüsselte Passwort bei dem

jeweiligen Dienst getestet wird. Hier greifen dann die üblichen Sicherheitsmechanismen wie maximale Anzahl an Fehlversuchen, sodass ein Angreifer nicht alle Möglichkeiten durchprobieren kann. Für den Nutzer selbst bietet der *MobileSitter* eine vom Master-Key abhängige visuelle Rückmeldung in Form von unterschiedlichen Grafiken. Durch Einblendung der richtigen Grafik kann der Nutzer die korrekte Eingabe des Master-Keys erkennen.

Der Ansatz, die geringe Entropie von Passwörtern durch Erschwerung von Brute-Force-Angriffen zu kompensieren, scheint auf den ersten Blick eine gute Idee. Auch in [Sch04, S. 97] wird das Verschlüsseln auf Basis von Passwortschlüsseln aufgrund der geringen Entropie kritisiert – durch Verschleierung des Klartextes wird beim *MobileSitter* allerdings verhindert, alle möglichen Kombinationen für den Master-Key erfolgreich durchzuprobieren. Somit stellt die verringerte Anzahl der Kombinationen kein erhöhtes Risiko dar.

Allerdings hält diese Sicherheit nur für die Zeit, in der dem Angreifer keines der Passwörter bekannt ist. Der Besitz eines Passwortes ermöglicht es in diesem Fall einen Brute-Force-Angriff durchzuführen, da ein Vergleich des Passwortes mit der verschlüsselten Variante möglich wird. Auch aus diesem Grund werden laut [WFS12] nur das Passwort und keine Zusatzdaten wie Nutzername verschlüsselt (diese sind meist bekannt). Dadurch entsteht allerdings die Möglichkeit, bei einem Zugriff auf die Passwortdatei herauszufinden, bei welchen Diensten der jeweilige Nutzer registriert ist. In der FAQ [Gmbd] wird ferner erwähnt, dass eine Manipulation der Daten nicht geschützt ist, da auch darüber ein Angreifer die unerwünschte Rückmeldung erhält, ob das Passwort richtig ist. Somit sollte der Nutzer für ein regelmäßiges Backup über eine integrierte Exportfunktion sorgen. Die Idee des *MobileSitters* ist unter der Patentnummer DE102006049814B4 geschützt.

3.5. WebUntis

Mit *WebUntis* bietet die Firma *Gruber & Petters GmbH* ein Produkt an, um Daten von *Untis*, einer Software der selben Firma, mit Personen wie Lehrern, Eltern und Schülern online zu teilen [Gmba]. Die Nutzung ist sowohl über eine Software auf dem Computer, als auch über eine App auf mobilen Endgeräten mit iOS und Android möglich. Als Beispiel der geteilten Daten werden Stunden- und Vertretungspläne angeführt. Installations- und Wartungsarbeiten entfallen für den Kunden, da *WebUntis* auf den Servern der Firma läuft. Auf Nachfrage (siehe Anhang A.4) wurde erwähnt, dass *WebUntis* auch eine Notenverwaltung enthält, welche zum Eintragen von Einzelnoten und Berechnung der Gesamtnote dient. Weitere Verwaltungsaufgaben finden allerdings in der Schülerverwaltung statt.

Nach eigenen Angaben (siehe Anhang A.4) stehen die Server in Wien, zur sicheren Kommunikation kommt SSL mit 128 Bit Verschlüsselung zum Einsatz. Als Protokoll wird HTTPS genutzt, wozu der jeweilige Nutzer nur einen Standardbrowser braucht. Die Übertragung des Nutzerpasswortes findet dabei innerhalb dieser gesicherten Verbindung im Klartext statt. Es ist also nur vom Client und Server einsehbar und während des Transportes gegen Dritte sicher geschützt. Antwortet der Client nicht, so wird der Nutzer nach max. einer Stunde (Konfiguration durch die Schule möglich) automatisch abgemeldet. Außer dem sicheren Kommunikationskanal findet keine weitere Verschlüsselung statt, sodass die Daten auf dem Server im Klartext vorhanden sind. Das übertragene Passwort wird allerdings auf dem Server verschlüsselt und auch die Backups sind mit PGP gesichert (der private Schlüssel liegt dabei nur offline vor).

Da die Daten auf dem Server im Klartext vorliegen, sollte man sich als Nutzer bewusst machen, dass theoretisch ein Mitarbeiter oder Angreifer sowohl Lese- als auch Schreibzugriff auf diese Daten erlangen kann. Weiterhin sollte das Nutzerpasswort nicht bei anderen Services wie-

derverwendet werden. Zwar wird dieses auf dem Server verschlüsselt, doch könnte es bei einem MITM-Angriff auf die SSL-Verbindung oder direkt während der Authentifizierung auf dem Server abgegriffen werden. Gelingt dies, so könnte sich ein Angreifer bei den Diensten, bei welchen das Passwort auch verwendet wird, erfolgreich authentifizieren.

4. Konzeptentwurf

Wie eingangs bereits erläutert ist es das Ziel, eine *sichere Client-Server-Architektur* zu entwickeln. In Kapitel 2.4.1 wurden dazu mit R19 die Facetten des Begriffs „Sicherheit“ definiert. In diesem Fall muss dabei die Sicherheit auf dem Client, dem Server sowie bei der Nachrichtenübertragung zwischen beiden Instanzen gewährleistet werden (R1 und R3). Aufgrund der geforderten Hardwareunabhängigkeit in R5 muss der Konzeptentwurf auf einer höheren Abstraktionsebene, also einem Betriebssystem, basieren. Dies wird vor allem für die Implementierung in Kapitel 5 von Bedeutung sein. Zur Erfüllung von R12 ist die Verwendung der grundlegenden Methoden der Kryptografie (Kapitel 2.2) notwendig.

Aus R5 folgt also, dass der in Kapitel 2.4.1 erwähnte Aspekt der *Safety* durch das zugrundeliegende System als gegeben angenommen werden kann. Allein der Schutz vor voreiligem Handeln muss durch entsprechende Hinweise in der Implementierung geregelt sein (siehe R23). Unüberlegtes Handeln des Nutzers wird damit im folgenden ausgeschlossen.

[**R23**] Das System muss den Nutzer beim Versuch, Daten zu Löschen oder anderen Nutzern freizugeben, durch eine offensichtliche Meldung warnen.

Weiterhin sei die *Verfügbarkeit* aller Systeme durch die vorhandene Soft- und Hardware garantiert. Somit sind alle Systeme und Daten autorisierten Personen auf Wunsch zugänglich. Für die Datenverfügbarkeit wird dem Server eine Schlüsselrolle zugewiesen. Dieser ist als sog. Cloud-Speicher (siehe Kapitel 2.3) vorab eingerichtet, um den Dateiaustausch der einzelnen Nutzer zu jeder Zeit zu gewährleisten. Diese wiederum kommunizieren mit entsprechend konfigurierten Clients mit dem Server, um dort Daten abzulegen und mit Anderen zu teilen. Sämtliche Kommunikation findet damit über diesen Server statt. Ein grundlegendes Schema mit einem Server und zwei Nutzern, nachfolgend wie in Kapitel 2.2 Alice und Bob genannt, ist in Abb. 4.1 gezeigt.

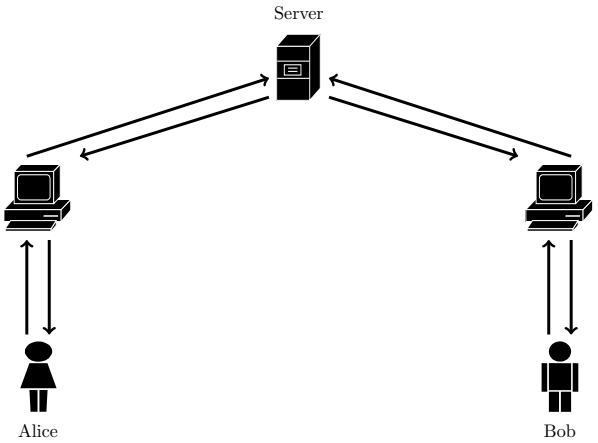

Abbildung 4.1.: Kommunikationsstruktur

Einzig ein durch Mallory, dem Angreifer (siehe Kapitel 2.2), durchgeführter Diebstahl eines Client-Systems könnte die Verfügbarkeit einschränken. Um trotzdem weiterhin die durch R2 in Kapitel 2.1 geforderte Datenverfügbarkeit einzuhalten, wird angenommen, dass der Nutzer regelmäßig eigene vom Client-System unabhängige und entsprechend gesicherte Backups der Daten anlegt oder der Server dies übernimmt. Auch könnte z. B. eine regelmäßig genutzte Methode zum Ausdrucken eine Unabhängigkeit von der Funktionsfähigkeit des IT-System ermöglichen. Ein Ausfall oder Diebstahl des Servers sei durch die in [Inf12] empfohlenen Maßnahmen nicht möglich. Dadurch ist auch die in Kapitel 2.4.3 beschriebene *Denial-of-Service-Attacke* durch massenweise versendete Anfragen oder falsche Protokollnachrichten auf dem Server ausgeschlossen.

In R9 und R10 wird die Kontrolle über die verwalteten Daten durch die Lehrkraft gefordert. Mit einer zentralen Steuerung der Zugriffsrechte über den Server wäre aufgrund eben dieser zentralen Stelle eine Manipulation durch Dritte möglich. Zusätzlich folgt aus R2, dass die jeweilige Lehrperson die Einhaltung der in Ziffer 4 [Nie] beschriebenen Datenschutzmaßnahmen garantieren muss. Dies kann aber nur passieren, wenn die Lehrkraft trotz der zentralen Datenspeicherung auf dem Server die Zugriffsrechte selbst bestimmen kann.

Es ist also ein Kompromiss zu finden zwischen der geforderten Verfügbarkeit der Daten, welche durch den Server gegeben ist, und der Kontrolle der Lehrkraft, welche durch den Server eingeschränkt wird. Dazu soll der Server zwar weiterhin eingesetzt aber das notwendige Vertrauen so gering wie möglich gehalten werden. Als Aufgaben werden dem Server deshalb nur das korrekte Verwalten der Nutzer, deren Rechte und die Datenweiterleitung zugeschrieben. Sonstige Sicherheitsanforderungen sind durch den Client zu garantieren, wodurch indirekt die Lehrkraft trotz des Servereinsatzes die Kontrolle über die Datensicherheit behält. In dem Konzept muss deshalb aber auch beachtet werden, dass Mallory versuchen wird, auf dem Server Daten mitzulesen und deren Inhalt zu ändern.

Zusätzlich wird in [Nie] angemerkt, dass die Schule weiterhin für die Einhaltung des Datenschutzes zu sorgen hat. Eine Wahl für den Standpunkt des Servers könnte somit ein gesicherter Raum in der Schule sein, sodass von der Schule garantiert werden kann, dass niemand Unbefugtes Zutritt erlangt. Bei einem externen Standort sollte die Schule diese Sicherheit über entsprechende Verträge gemäß §13 NDSG [DN02] regeln.

Durch entsprechende technische Maßnahmen vom eingesetzten System (siehe auch [Inf12]) ist anzunehmen, dass Mallory keinen Zugriff auf die Hardware der Serverseite hat. Auf der Seite des Clients könnte Mallory als Bekannter des Nutzers für kurze Zeit unbemerkt auf das System zugreifen.

[R24] Das System muss bei einem Zugriff auf das Client-Gerät durch einen Angreifer, bei welchem dieser vor dem Gerät steht und per Hand Passwörter ausprobieren oder einzelne Dateien kopieren kann, die Sicherheit der Daten garantieren.

Es ist davon auszugehen, dass ein Auseinandernehmen der eingesetzten Hardware oder längeres Ausprobieren eines Passwortes dabei nicht möglich ist, da Alice bzw. Bob sich der in Kapitel 2.1 genannten Pflichten bewusst sind und ihr System entsprechend zu schützen versuchen. Vorkehrungen gegen einen trotzdem möglichen Diebstahl sind aber weiterhin zu treffen. Die in Kapitel 2.4.2 beschriebenen *physikalischen* oder *Seitenangriffe* sind also auszuschließen – einzig nach einem Diebstahl ist Mallory dies möglich.

[R25] Das System muss bei einem Diebstahl des Client-Gerätes die Sicherheit der Daten garantieren.

Neueste Softwareupdates, entsprechende Konfigurationen, Virenfreiheit u. Ä. gelten auf allen System als vorausgesetzt. Auch hier finden sich in [Inf12] entsprechende Empfehlungen zur Umsetzung für die Serverseite. Für den Client wird insgesamt davon ausgegangen, dass die Lehrkraft die in [Infb] beschriebenen Maßnahmen einhält. Weitere für das jeweilige Betriebssystem spezifische Empfehlungen finden sich auch in [Infa].

Es kann also erwartet werden, dass die in Kapitel 2.4.1 geforderte *Systemintegrität* der IT-Systeme zu jedem Zeitpunkt gegeben ist. Ferner sind damit die in Kapitel 2.4.2 erwähnten *Malware-Angriffe* ausgeschlossen. Der Nutzer kann also darauf vertrauen, dass das WYSIWYG-Prinzip (*What-you-see-is-what-you-get*) gegeben ist. Durch die Nutzerverwaltung des jeweils eingesetzten Betriebssystems ist auch die *Zurechenbarkeit* der auf dem Client oder Server direkt ausgeführten Aktionen gegeben. Ein Prozess auf dem Server protokolliert darüber hinaus die Instanzen (über IP- bzw. MAC-Adresse zugeordnet), die einen Zugriff per Internet ausüben. Allerdings wird es für eine vollständige Zurechenbarkeit notwendig sein, in dem Verfahren die Zuordnung dieser Instanz zu einem Nutzer zu gewährleisten. Schließlich kann durch die Einhaltung der in [Infb] beschriebenen Maßnahmen davon ausgegangen werden, dass die *Anwender-Risiken* nicht existieren. Auch wenn in dem hier erwähnten Konzept die Sicherheitsfunktionen durch den Nutzer unumgänglich sein sollen, kann dadurch nicht garantiert werden, dass die Lehrkraft zum Datenaustausch kein anderes System nutzt. Diese Möglichkeit ist deshalb hier aufgrund des Pflichtbewusstseins, welches mit der Eigenverantwortung der Lehrkraft gefordert werden muss (R9), ausgeschlossen, wodurch auch die in Kapitel 2.4.3 beschriebene Folge (Umstieg auf Software ohne Kryptografie) von *Denial-of-Service-Attacken* als unmöglich angenommen wird.

Um den erwähnten Aspekt der *Privatsphäre* zu gewährleisten, wird davon ausgegangen, dass die Schülerinnen und Schüler über die Speicherung und Verteilung Ihrer personenbezogenen Daten Bescheid wissen. Auch ist die Zustimmung gegeben, dass die Kontrolle dieser Daten von der Lehrkraft übernommen wird. Insgesamt bleiben Mallory also folgende Angriffsmöglichkeiten:

1. Daten am Server mitlesen oder manipulieren
2. eine kleine Anzahl an Daten von der Festplatte (nicht RAM) mitlesen
3. das gesamte Client-System und dessen Daten durch Diebstahl langfristig entfernen
4. auf dem Transportweg Datenpakete (insbesondere die Steuerbefehle) abfangen und manipulieren

Es ist anzunehmen, dass das Lesen der Daten im Klartext durch Mallory für die Lehrkraft schlimmer ist als die Manipulation der Daten (ohne Zugriff auf den Klartext). Denn zum Einen kann eine blinde Veränderung verschlüsselter Daten mit den Backups rückgängig gemacht werden, zum Anderen fällt eine Manipulation von verschlüsselten Daten spätestens bei der Entschlüsselung auf. Somit wäre zumindest einer der Kommunikationspartner gewarnt.

Da der Lehrkraft insgesamt der komplette Sicherheitsaufwand zugeschrieben wird, wird es eine große Aufgabe des Client-Systems sein, diese dabei soweit wie möglich auf transparente Art und Weise zu unterstützen. Gleichzeitig muss davon ausgegangen werden, dass sich die Lehrkraft an die in Kapitel 2.1 erwähnte Gesetzeslage hält. Ein Überprüfen der übertragenen Daten durch den Server oder Client ist auch aufgrund des gewählten Designs nicht möglich.

Hauptaufgabe des Konzepts wird es also sein, die Sicherheit vor gewollten Angriffen, also *Security*, die Datenintegrität, die Datenvertraulichkeit sowie Authentizität der Nutzer zu garantieren. Dadurch sollen auch die in Kapitel 2.4.3 erwähnten *Spoofing-*, *MITM-*, *Hijacking-* und *Known-Key-Attacken* verhindert werden. Eine Einschränkung der Erfolgschance einer *Verkehrsanalyse* soll durch das Verschlüsseln aller möglichen Daten erreicht werden. Die *Replay-Attacke*

wird wie beschrieben durch Zeitstempel oder Nummerierung der Datenpakte verhindert.

In Kapitel 2.4.2 wird als Gegenmaßnahme zu *Implementierungsfehlern* ein modularer Softwareaufbau empfohlen. Zur Unterstützung dieser in der Informatik auch als *Divide and Conquer* bekannten Vorgehensweise erscheint es sinnvoll, diese schon vor der eigentlichen Implementierung anzuwenden. Im Ergebnis soll deshalb die Realisierung verschiedener Sicherheitsziele in einzelne Schichten, hier Kanäle genannt, aufgeteilt werden, um die Komplexität des Programms möglichst gering zu halten. Das Zusammenspiel dieser nachfolgend beschriebenen Kanäle wird dann in Kapitel 4.5 erläutert. Weiterhin wird zur Erfüllung von R16 das Konzept unabhängig von bestimmten kryptografischen Implementierungen sein. Sollte sich in Zukunft herausstellen, dass bestimmte Algorithmen unsicher sind, ist die Sicherheit des Konzeptes dadurch nicht gefährdet. Eine Liste der im Folgenden verwendeten Schlüssel mit Erklärung findet sich im Anhang A.7.

4.1. Der Schlüsselcontainer

Mit der Forderung nach Eigenverantwortung gehen wie bereits angedeutet auch eigene Pflichten bezüglich der Sicherheitsvorkehrungen mit ein. Eine nicht zentrale Verwaltung hat zur Folge, dass sämtliche Geheimnisse auf dem von der Lehrkraft eingesetzten privaten IT-System bleiben müssen. Mit den eingangs beschriebenen Voraussetzungen ist sichergestellt, dass der Client grundsätzlich als abgesichert anzunehmen ist. Trotzdem bleibt die erwähnte Problematik, dass ein Bekannter kurzfristig Zugriff auf das IT-System erlangen und z. B. von dort Daten kopieren oder dessen Inhalt lesen könnte. Den Erfolg, Klartext dadurch zu Gesicht zu bekommen, gilt es nun mit dem Schlüsselcontainer einzuschränken.

In erster Linie dürfen deshalb wie in Kapitel 2.4.2 geraten, nur aktive Schlüssel oder Klartexte im Hauptspeicher liegen. Vor allem muss eine Festplattenauslagerung vermieden werden, um zu verhindern, dass unverschlüsselte Geheimnisse durch Systemabstürze unbemerkt bestehen bleiben. Sämtliche nicht benutzte Schlüssel und Klartexte dürfen daher für den späteren Gebrauch nur sicher verschlüsselt auf der Festplatte gespeichert werden. Wie in den folgenden Kapiteln ersichtlich werden wird, ist dabei die Anzahl der zu speichernden Schlüssel von vorne herein unbekannt und kann theoretisch eine sehr große Zahl umfassen. Eine vorab koordinierte Schlüsselverwaltung ist somit essentiell, um den Überblick zu behalten. Mit dem Ziel, sicher zu stellen, dass nur die jeweils autorisierte Lehrkraft Zugriff (wie in Kapitel 2.1 geschildert) auf die Schlüssel hat, wird es des Weiteren notwendig sein, ein letztes Geheimnis beim Nutzer auszulagern. In einer Implementierung sollte dabei die Möglichkeit geboten werden, aus dem einen Geheimnis alle für die jeweiligen Klartexte benötigten Schlüssel entschlüsseln zu können. Durch die „Entschlüsselung“ der in einer Datenbank verschlüsselt abgelegten Schlüssel ist sichergestellt, dass neben der Kenntnis des Nutzergeheimnisses zusätzlich die Datenbank notwendig ist, um an die Schlüssel zu gelangen. Deshalb ist diese Methode einer „Berechnung“ der Schlüssel aus dem Nutzerpasswort (z. B. über eine binäre Addition von öffentlichem Schlüsselnamen und Nutzerpasswort) vorzuziehen, da der Besitz einer Datenbank dann nicht mehr notwendig wäre. Im Ergebnis bleibt folglich die letztendliche Kontrolle über die Schlüssel bei der Lehrkraft, ohne dass diese technische Kenntnisse von Kryptografie besitzen muss oder gezwungen ist, sich die steigende Zahl an Schlüsseln einzuprägen. Dies kommt R8 entgegen.

In [Sch13a, Kapitel 21] werden die existierenden Verfahren, sich zu authentifizieren, mit den Stichpunkten „etwas, was man weiß“ [Sch13a, S. 405], z. B. ein Passwort, „was man hat“ [Sch13a, S. 406], z. B. ein Ausweis, oder „was man ist“ [Sch13a, S. 407], z. B. ein Fingerabdruck, kurz zusammengefasst. Im Regelfall werden für die beiden letzteren Varianten zusätzliche Lesegeräte benötigt. Ein Passwort lässt sich dahingegen mit der vorhandenen Tastatur eingeben. Da durch

R5 spezielle Hardware vermieden werden soll, wird im Folgenden die Passwortmethode verwendet. Theoretisch lassen sich aber auch die anderen Verfahren nutzen.

Die Tatsache, dass mit dem Passwort A_{secret} bzw. B_{secret} mehrere Schlüssel entschlüsselt werden, führt zu der Idee, eine Art Schlüsselspeicher, siehe Abb. 4.2, anzulegen. Dieser ähnelt wegen seiner Grundfunktion einem Passwortmanager – statt Passwörter werden hier Schlüssel über ein Masterpasswort verschlüsselt abgelegt. Es liegt also nahe, die Entwicklung eines solchen Speichers von bereits bekannten Vorgehensweisen zu übernehmen.

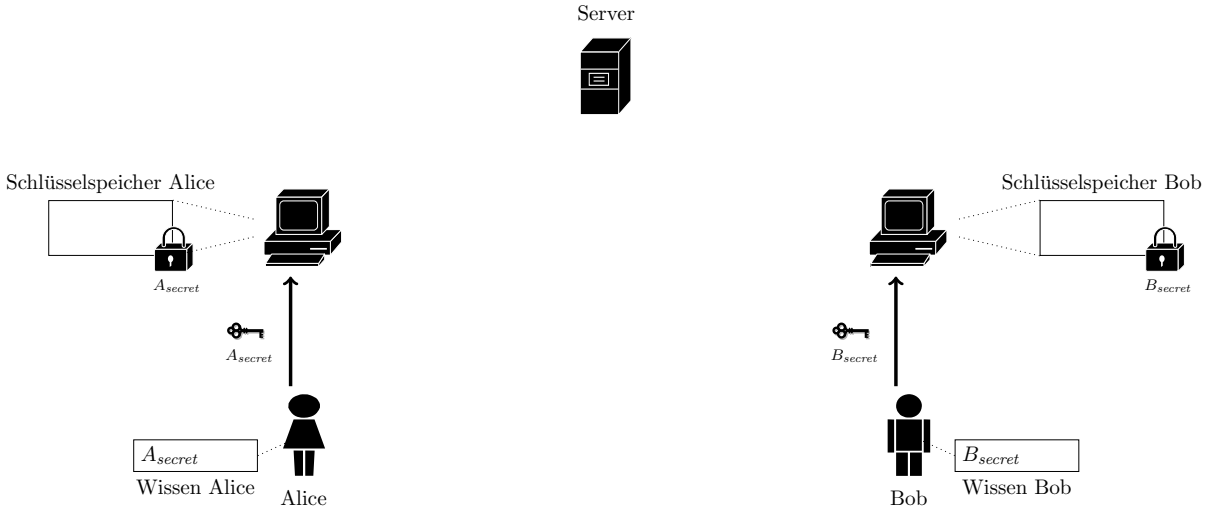

Abbildung 4.2.: Erstellen des Schlüsselcontainers

Trotz der leichten Implementierbarkeit der Passwortmethode soll eine Problematik nicht unerwähnt bleiben. In [Sch04] wird kritisiert, dass ein aus einem Passwort erzeugter Schlüssel nicht die erforderliche Entropie besitze: „Ich habe Entropieschätzungen von Standardenglisch mit weniger als 1,3 Bit pro Zeichen gesehen. [...] Wenn Sie einen 128-Bit-Schlüssel wollen, brauchen Sie ein Passwort von 98 Zeichen.“ [Sch04, S. 97]. Diese Passwortlänge ist wohl kaum einem Benutzer abzuverlangen. Allerdings gilt auch laut R24, dass Mallory nicht viel Zeit bleibt, das lokal für den Schlüsselspeicher verwendete Passwort zu knacken.

Das Nutzerpasswort ist hier also nur als weiterer, geringer Schutz zu betrachten, bevor man genutzte Schlüssel im Klartext vorliegen hat. Insgesamt wird dadurch allerdings eine Zeitverzögerung, bis Mallory an die Schlüssel kommt, erreicht. Dies kann durchaus bei dem erwähnten Bekannten für R24 aber auch bei einem Diebstahl (wie noch in Kapitel 4.4 gezeigt wird) effektiv sein.

Um trotzdem die Brute-Force-Anfälligkeit zu verringern, kann eine Methode ähnlich dem *MobileSitter* (siehe Kapitel 3.4) eingesetzt werden. Die Problematik besteht hier dadrin, dass ein Schlüssel nie an einen Server übertragen wird, sodass sich keine Möglichkeit bietet, über Eingabeeinschränkungen das Durchprobieren aller möglichen Kombinationen zu verhindern. Somit sollte der Speicher so konzipiert sein, dass unabhängig von richtiger oder falscher Passworteingabe der Schlüsselcontainer einen möglichen, gültigen Schlüssel zurückliefert und ferner das Entschlüsseln der Dateien – unabhängig vom Schlüssel – ohne Fehler abläuft. Allein am Klartext ist somit die Richtigkeit des Passwortes erkennbar sein. Deshalb ist dieses Verfahren nur sinnvoll, wenn der Klartext einer mit dem Schlüssel zu entschlüsselnden Information nicht bekannt ist. Andernfalls könnte darüber der vermeintliche Schlüssel leicht verifiziert werden. Zu beachten ist bei einer Implementierung das Schutzrecht des *MobileSitters*.

4.2. Der Verwaltungskanal

Wie in [Eck14, S. 465] angeführt, ist es für die später zu implementierende Datenintegrität und Datenvertraulichkeit zuerst notwendig, eine Identitätsprüfung, also die Sicherung der *Authentizität*, durchzuführen. Ziel ist es, den jeweiligen Gegenüber, also Client bzw. Server, von seiner eigenen Identität zu überzeugen. Auch wenn prinzipiell durch die im Datenkanal (siehe Kapitel 4.3) durchgeführte Verschlüsselung ein Datenpaket als sicher angenommen werden kann, soll dadurch verhindert werden, dass Mallory durch einfaches Vortäuschen eines Schulservers an diese Pakete kommt. Der Server wiederum braucht für sein nachfolgend beschriebenes Rechte-Management die Gewährleistung, dass der jeweilige Nutzer für den entsprechenden Zugriff autorisiert ist. Dies vervollständigt die in R19 geforderte *Zurechenbarkeit*.

Eine Kommunikation zwischen Server und Client findet nur statt, wenn der Nutzer Daten auf dem Server ändern oder lesen möchte. Es handelt sich also um eine sitzungsbasierte Kommunikation, weshalb nach der erfolgten Identitätsprüfung sowohl Server als auch Client über den selben temporären Sitzungsschlüssel verfügen sollten. Zur Einhaltung von R14 muss die Berechnung dieses Schlüssels wie in Kapitel 2.2.6 beschrieben erfolgen. Die dafür notwendigen asymmetrischen Schlüsselpaare müssen dazu zuvor zwischen Client und Server ausgetauscht worden sein.

Eine mögliche Umsetzung dessen bildet die PKI-Infrastruktur (siehe Kapitel 2.2.5). diese ist hier allerdings nicht einsetzbar. Zum Einen benötigt man eine weitere, vertrauenswürdige Instanz zum Zertifizieren der Schlüssel. Der Anwender müsste also der Sicherheit dieser Instanz im vollen Maße vertrauen, da bei einer Kompromittierung der Zertifizierungsstelle die Authentizität bei jeglicher Kommunikation nicht mehr gegeben ist. Dies widerspricht R9. Zum Anderen bedeutet solch eine Infrastruktur einen erhöhten Mehraufwand, vor allem bei hierarchischen Systemen, sowie durch zusätzliche Instanzen hervorgerufene, neue Angriffsmöglichkeiten.

Alternativ soll hier deshalb ein auf Verschlüsselung basiertes Verfahren angewendet werden, welches einem in [Sch13a, Kapitel 26.3.1] erwähnten Vertrauensmodell ähnelt. Der Ablauf zum Austausch der asymmetrischen Schlüssel mit Alice sieht damit wie folgt aus:

1. Der Serveradministrator generiert ein einmalig anzuwendendes Passwort ePw_A.
2. ePw_A wird zusammen mit dem öffentlichen Schlüssel S_{pubKey} des Servers über einen sicheren Kanal, wie z. B. ein persönliches Treffen mit Alice, übermittelt. Da der Client als frei von Malware angenommen wird, können diese Daten auch elektronisch gespeichert sein.
3. Alice kann nun S_{pubKey} und ePw_A an ihrem Client eingeben. Aufgrund der persönlichen Übergabe ist die Integrität dieser Schlüssel gesichert.
4. Der Client verschlüsselt anschließend seinen öffentlichen Schlüssel A_{pubKey} mit ePw_A und einer symmetrischen Verschlüsselungsmethode.
5. Beim ersten Verbindungsaufbau mit dem Server ist die Kommunikation noch ungesichert. Durch die Verschlüsselung der Daten mit ePw_A hat Mallory allerdings keine Möglichkeit, diese während der Übertragung zu ändern. Dazu ist es notwendig, dass die gewählte Verschlüsselungsmethode während der Übertragungszeit einer Kryptoanalyse oder Brute-Force-Attacke standhält. Sollte Mallory nach dieser Übertragung ePw_A ermitteln, so stellt dies kein Risiko dar, da nur die Integrität nicht aber die Vertraulichkeit des Inhaltes geschützt werden muss.
6. Ist das Entschlüsseln mit ePw_A beim Server erfolgreich, so weiß dieser, dass der neue Nutzer durch den Administrator gewährte Zugriffsrechte besitzt und kann ein entsprechendes Konto anlegen. Die Client-Authentizität ist also gewährleistet.

7. Bei allen weiteren Verbindungen kann nun der Sitzungsschlüssel $A_{sessionKey}$ mit den geteilten öffentlichen Schlüsseln S_{pubKey} und A_{pubKey} an den jeweiligen Instanzen nach Kapitel 2.2.6 berechnet werden. Sämtliche folgende Kommunikation findet also durch $A_{sessionKey}$ verschlüsselt statt.

8. Beim ersten Zugriff überträgt der Client dem Server nun (dank obigem Kanal verschlüsselt) zusätzliche, frei gewählte Accountinformationen wie Nutzername und Passwort (nachfolgend benötigt).

Ist $A_{sessionKey}$ bei beiden Kommunikationspartnern gleich, was sich bei einer erfolgreichen Ver- und Entschlüsselung zeigt, so ist die Integrität und Vertraulichkeit der Daten gesichert. Auch die Authentizität des Partners ist dadurch gegeben. In R13 wird weiterhin die Sperrmöglichkeit eines asymmetrischen Schlüssels gefordert. Dies muss wie bei der Schlüsselübergabe über ein persönliches Treffen stattfinden. Die unmögliche Abstreitbarkeit bzw. das Einhalten von Policies kann mit dieser Methode aufgrund der fehlenden dritten Vertrauensinstanz allerdings nicht gewährleistet werden. Für den Einsatz der Schlüssel ist dies auch nicht notwendig. Führt Bob das gleiche Verfahren, mit seinen Schlüsseln, durch, so ergeben sich die in Abb. 4.3 gezeigten sicheren Kanäle zwischen Alices bzw. Bobs Client und dem Server.

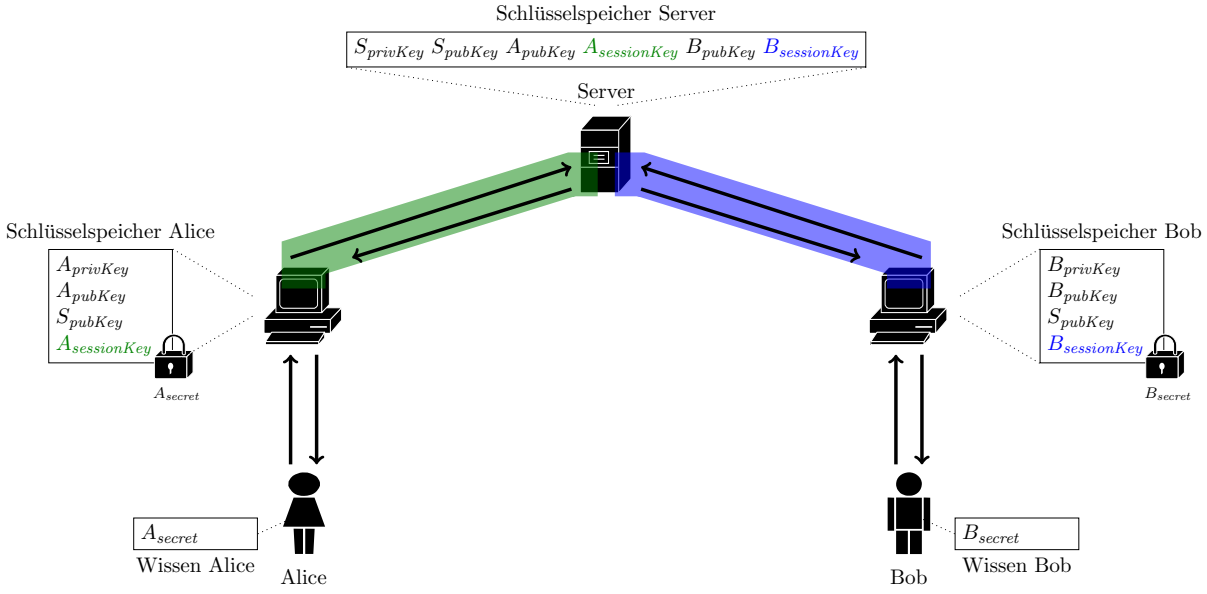

Abbildung 4.3.: Kommunikationsstruktur mit Verwaltungskanal

Um ebenfalls für die Daten auf dem Server die Integrität und Vertraulichkeit zu garantieren, wird das anfangs übermittelte Passwort benutzt. Auch dieses sollte von einem Nutzergeheimnis abhängen, damit der Server neben der Authentizität des Gerätes auch die des Nutzers überprüfen kann. Prinzipiell spricht nichts dagegen, das Passwort für den Server – z. B. eine dem Nutzer unbekannte Zufallsfolge – im oben beschriebenen Schlüsselspeicher abzulegen und damit vom dort verwendeten Nutzergeheimnis abhängig zu machen. Es sollte allerdings wie oben erwähnt nicht aus diesem Nutzergeheimnis berechenbar sein. Wird das Passwort anders ermittelt, so muss darauf geachtet werden, dass nur ein Hash des eigentlichen Wertes an den Server übertragen wird, um Probleme bei einer Wiederverwendbarkeit (siehe dazu auch Kapitel 3.5) zu vermeiden. Der bereits existierende sichere Kanal ermöglicht es nun, eine Nutzerauthentifizierung durch direktes Übertragen des Passwort(-hashes) oder über ein einfaches *Challenge-Response-Verfahren* (siehe dazu auch [Eck14, Kapitel 10.2.3]) durchzuführen.

<Sitzungsnummer, eindeutig>
<Header> z. B.: Daten ändern, löschen, anlegen Rechte ändern, löschen, anlegen Anmelden Abmelden
<Body> z. B.: Dateninhalt + Attribute Rechte + Nutzer Passwort / Passworthash

Abbildung 4.4.: Aufbau und Inhalt eines über den Verwaltungskanal geschickten Pakets

Ein über diesen Kanal geschicktes Paket könnte dann die in Abb. 4.4 gezeigte Form besitzen: über die eindeutige Sitzungsnummer können beide Parteien den Sitzungsschlüssel zuordnen und jeweils Header und Body einer Nachricht ver- bzw. entschlüsseln. Eine Manipulation bzw. ein Lesen dieser Nachrichten wäre nur dann möglich, wenn ein Angreifer den Sitzungsschlüssel oder den privaten Schlüssel einer der beiden Kommunikationspartner kennt. Die Sitzungsnummer braucht hierbei nicht verschlüsselt zu sein, da bei einer Änderung auf eine weitere gültige Nummer der Server das Paket einer anderen Identität zuordnen würde. Spätestens bei der Entschlüsselung würde dann der Server die Manipulation feststellen, da in diesem Fall bei einer gut gewählten symmetrischen Verschlüsselungsmethode aufgrund der Nutzung eines anderen Sitzungsschlüssels zur Verschlüsselung kein gültiger Klartext produziert wird. Ist die Nummer ungültig, so kann der Server das Paket sofort verwerfen bzw. entsprechende Warnungen vornehmen.

Weil der Server wie oben beschrieben auf Dateiebene arbeitet, ergeben sich zwei Rechte, die dem Nutzer entsprechend je Datei zuzuordnen sind: Schreib- und Leserechte. Weitere Informationen sollen dem Server, wie im nächsten Kapitel beschrieben, nicht zugänglich sein.

4.3. Der Datenkanal

Mit dem Verwaltungskanal wurde sichergestellt, dass kein Dritter Lesezugriff auf transportierte Pakete hat. Darauf aufbauend gilt es nun, den eigentlichen Dateninhalt vor dem Server selbst zu schützen (siehe R12). Aus diesem Grund ist ein weiterer, sich auf den Verwaltungskanal stützender *Datenkanal* notwendig.

Mit diesem Kanal wird nun eine *Ende-zu-Ende-Verschlüsselung* zwischen zwei Lehrkräften hergestellt werden. Auch in diesem Fall müssen sich die beiden Partner wieder von der jeweils anderen Identität überzeugen (Bob führt den beschriebenen Vorgang analog durch):

1. Alice legt einen *Trust-Container* an. Dieser enthält neben dem Nutzernamen (sowie eventuell weiterer Profilinformationen) den öffentlichen Schlüssel A_{encKey} zum Verschlüsseln und den öffentlichen Schlüssel $A_{verifyKey}$ zum Verifizieren von Signaturen. Die jeweils dazu gehörenden privaten Schlüssel A_{decKey} und $A_{signKey}$ verwaltet der Client lokal in seinem Schlüsselcontainer.

2. Vor der Übertragung an den Server wird der Trust-Container mit einem Einmalpasswort verschlüsselt, um eine Manipulation durch den Server zu verhindern.

3. Möchte Bob nun Alice vertrauen, so lädt er sich die jeweilige Datei vom Server herunter. Dazu muss Alice diese vorerst zum Lesen freigegeben haben.

4. Über einen sicheren Kanal, z. B. einem persönlichen Treffen, übermittelt Alice das Einmalpasswort an Bob.

5. Ist das Entschlüsseln des Trust-Containers erfolgreich, so ist Bob von der Integrität des Inhalts überzeugt.

Aufgrund des gleichen Vertrauensmodells wie beim Verwaltungskanal treffen hier die selben Eigenschaften zu. Der Server selbst „sieht“ nur zwei Dateien, deren Inhalt er nicht lesen kann. Bei zufälliger Dateinamenswahl ist es auch nicht möglich, festzustellen, dass es sich um einen verschlüsselten Trust-Container handelt – mehr dazu in Kapitel 4.6.

Allerdings sollte hier ausdrücklich auf eine mehrfache Anwendung eines Trust-Containers verzichtet werden. Verteilt Alice einen mit ein und demselben Passwort verschlüsselten Trust-Container an mehr als eine Person, so ließe sich der Inhalt unbemerkt manipulieren. Mallory, der das Einmalpasswort kennt, kann den Container herunterladen und entschlüsseln. In einem zweiten Schritt kann er nun seinen eigenen Trust-Container mit dem selben Passwort verschlüsseln und auf dem Server austauschen – gesetzt den Fall, er hat die Schreibrechte dazu. Lädt sich Bob nun den Trust-Container nach der erfolgten Manipulation herunter, so erhält er die falschen öffentlichen Schlüssel, obwohl er dank der erfolgreichen Entschlüsselung davon ausgehen muss, dass das Paket nicht kompromittiert wurde. Mallory müsste nun, um unauffällig den weiteren Verkehr abhören zu können, sämtliche von Bob an Alice geschickten Dateien abfangen, mit seinem privaten Schlüssel entschlüsseln und dem öffentlichen Schlüssel von Alice verschlüsseln. Dazu müssen natürlich Schreib- und Leserechte für Mallory auf alle von Bob an Alice geschickten Dateien vorausgesetzt sein. Im für Mallory einfachsten Fall wäre er hier der Serveradministrator selbst.

Die Kontrolle darüber, dass eingehende Dateien nicht kompromittiert werden können, hat somit jeder Nutzer selbst. Zur weiteren Sicherheit sollte die Datei nach der Benutzung sofort gelöscht werden, um ein zufälliges, zweimaliges Verteilen zu vermeiden.

Nach dem erfolgreichen Austausch der Trust-Container könnten Alice und Bob nun genauso wie mit dem Server eine Sitzung starten. Im Normalfall sind allerdings nicht beide Nutzer gleichzeitig online. Hier kommt nun der Vorteil des Servers zum Einsatz: Alice kann dank der ständigen Verfügbarkeit zu jedem beliebigen Zeitpunkt erstellte Dateien auf dem Server speichern. Unabhängig davon kann Bob diese dann auch, sollte er dazu autorisiert sein, herunterladen. Ein Aufbau einer Sitzung ist in diesem Fall also nicht von Vorteil, weshalb hier ein Ansatz ähnlich zu *MEGA*, *TeamDrive* und *Boxcryptor* gewählt wird (siehe Kapitel 3). Für jede neue Datei erstellt Alice dazu einen neuen symmetrischen Schlüssel D_{key}. Die damit verschlüsselte Datei kann nun anschließend zum Server übertragen werden. D_{key} bleibt dabei auf dem Gerät, sodass der Server keinen Zugriff auf den Dateiinhalt hat. Um nun mit Bob den Inhalt zu teilen bedarf es neben der Lese- bzw. Schreibfreigabe der Übermittlung von D_{key}:

1. Alice erstellt den in Abb. 4.5 gezeigten *Share-Key-Container* C, um Bob das Entschlüsseln und Verwalten der Datei mit der ID $Data_{ID}$ zu ermöglichen.

2. Alice verschlüsselt C mit einem neuen symmetrischen Schlüssel $K_{container}$.

3. Alice erstellt den in Abb. 4.6 gezeigten *Share-Key-Container-Wrapper* CW und lädt ihn als Datei auf den Server hoch.

Nur Bob als Besitzer des privaten Schlüssels B_{decKey} ist somit in der Lage, an den Schlüssel für C und damit den Dateischlüssel D_{key} zu gelangen. $Sender_{data}$ identifiziert Alice als Autor

Variablenname	Erklärung
D_{key}	sym. Schlüssel des Objektes
$Data_{ID}$	ID des Objektes
$Data_{author}$	Autor des Objektes
$Data_{name}$	Name des Objektes
$Sender_{data}$	Sender des Share-Key-Containers, hier: Alice

Abbildung 4.5.: Aufbau des Share-Key-Containers

Variablenname	Erklärung
$K_{container}$	symm. Schlüssel des Share-Key-Containers (mit öffentlichem Schlüssel vom Empfänger verschlüsselt, hier: B_{encKey})
C	Share-Key-Container (mit $K_{container}$ verschlüsselt)
S_c	Signatur vom unverschlüsselten C (mit privatem Signaturschlüssel des Senders erstellt, hier: $A_{signKey}$)

Abbildung 4.6.: Aufbau des Share-Key-Container-Wrappers

der Nachricht. Mit ihrem öffentlichem Schlüssel $A_{verifyKey}$ kann er deshalb sicherstellen, dass die geteilten Informationen direkt von Alice kommen und nicht manipuliert wurden. Sowohl Vertraulichkeit als auch Integrität der Daten dieses Austauschs sind somit gesichert. Dank der vorab geteilten Schlüssel lässt sich dieses Verfahren des Weiteren automatisieren – ein Eingriff das Nutzers ist hier nicht mehr gefordert, wodurch R8 entgegen gekommen wird. Schematisch ist das Teilen geheimer Informationen nach Initialisierung beider Kanäle in Abb. 4.7 gezeigt.

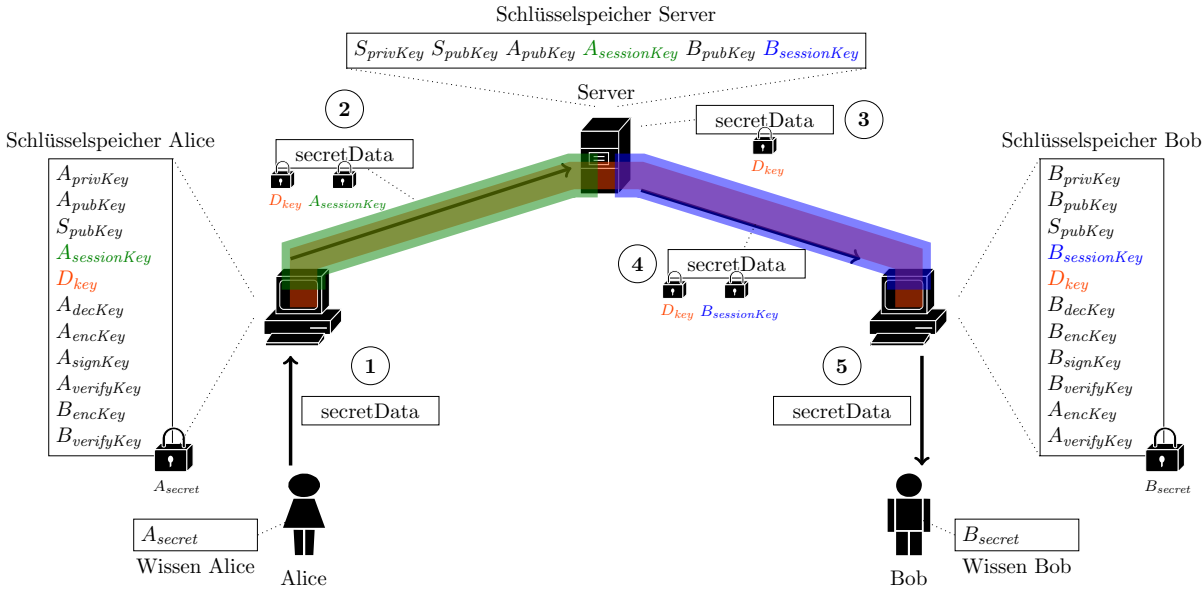

Abbildung 4.7.: Kommunikationsstruktur mit Verwaltungskanal und Datenkanal

4.4. Datenauswahl und Kodierung

In den ersten beiden Unterkapiteln wurde nun erläutert, wie Alice und Bob mit dem Server als auch unter sich sicher Informationen austauschen können. Geht man davon aus, dass jeweils ganze Dateien geteilt werden sollen, so ließe sich dies mit den bereits erwähnten Verfahren realisieren. Im Falle der Leistungsverwaltung könnte man sich nun die Datenansicht wie in Abb. 4.8 gezeigt vorstellen.

Example Window **X**

Klasse: 6B			Mathematik		Philosophie	
#	Nachname	Vorname	Notizen	Klausur 09.06.15	Notizen	Test 1
0	Mustermann	Max		1	mündlich super	3
1	Musterfrau	Gabriele	freiwilliges Referat	2		2

Abbildung 4.8.: Tabelle zur Leistungsverwaltung

Als Dateikodierung liegt es also nahe, ein aus Tabellenkalkulationsprogrammen bekanntes Format oder eine dateibasierte SQL-Datenbank zu wählen. Eine in beiden Fällen erstellte Tabelle hätte allerdings den Nachteil, dass alle Fachlehrer auch die Einträge der jeweils anderen Fächer lesen und schreiben könnten (da sowohl die Verschlüsselung als auch Rechte-Vergabe dateibasiert ist). Dies würde allerdings der in Kapitel 2.1 erwähnten Rechtslage und damit R2 und R4 widersprechen. Um das beschriebene Problem zu umgehen, könnte jeder Lehrer je Klasse eine eigene Datei anlegen und müsste dementsprechend auch jeweils sämtliche Schülernamen eingeben. Unabhängig von dem daraus resultierenden Mehraufwand entstünde auch das Problem der Inkompatibilität der einzelnen Tabellen. Beispielsweise könnte so ein automatisches Sammeln aller Endnoten zum Zeugnisdruck am Ende des Schuljahres nicht garantiert werden. Um der Forderung nach minimalem Aufwand in R7 nachzukommen, soll hier ein anderer Lösungsweg beschrieben werden. Dieser ist, passend zu Abb. 4.8, in Abb. 4.9 schematisch dargestellt.

Mit der Vorgabe, die Verschlüsselung weiter auf Dateiebene zu behalten, müssen zuerst alle personenbezogenen Daten pro Fach in einer getrennten Datei (hier 456.data) gespeichert werden. Jede weitere Datei (hier 392.data und 609.data) stellt dabei eine Spalte in der gesamten Tabelle dar. Über Unterspalten lassen sich dann verschiedene Einträge wie Klausurnoten einfügen. Um alle Fächer wieder zu einer gesamten Tabelle zusammenfügen zu können, ist es wichtig, dass die einzelnen Zeilen, also die zu den jeweiligen Schülerinnen und Schülern gehörenden Einträge, verbunden werden können. Sind alle Dateien nun als SQL-Datenbank angelegt, lässt sich je Zeile eine eindeutige ID vergeben. Diese ID muss zu jeder Zeile in jeder Datei mit abgelegt sein, um darüber einen sogenannten JOIN durchführen zu können. Zu guter Letzt definiert ein *Class-Container* (hier 478.data), welche Datei zu der jeweiligen Klasse gehört und wie diese Klasse heißt. Über entsprechende Freigaben lässt sich schließlich regeln, wer welche Spalten lesen oder schreiben darf. In Abb. 4.9 ist z. B. zu sehen, dass die Datei 720.data vom Nutzer nicht entschlüsselt werden kann. Aus diesem Grund taucht das Fach auch nicht in der Abb. 4.8 auf.

Trotz der Wahl sicherer Verschlüsselungsmethoden und dem Schutz der verwendeten Schlüssel, lässt sich nie ein erfolgreicher Angriff ausschließen. Vor allem die in Kapitel 2.4.2 aufgezeigten *Real-World-Attacken* können ein Risiko darstellen. Vergleicht man nun das beschriebene, digita-

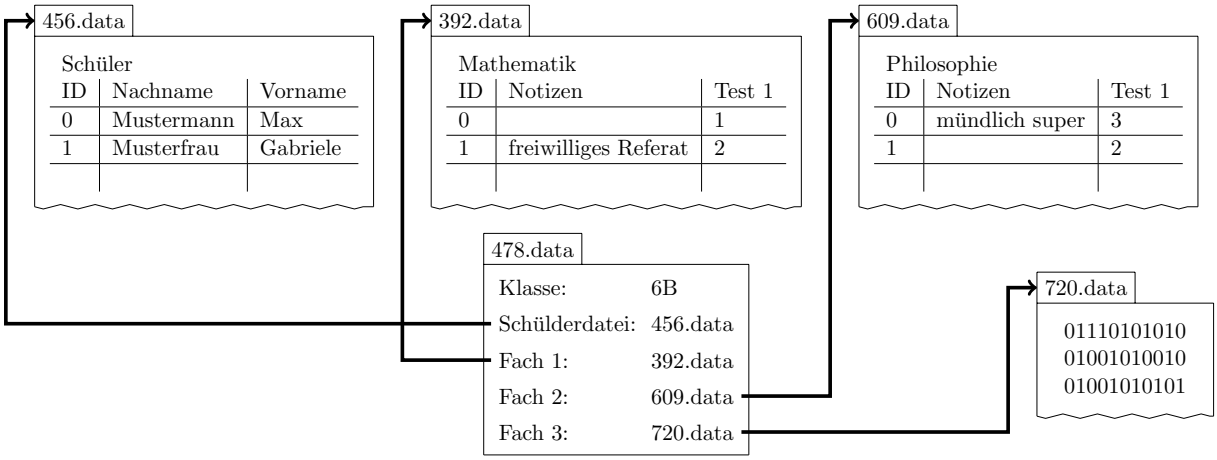

Abbildung 4.9.: Beispielschema der für Abb. 4.8 benötigten Dateitypen

le System mit der üblichen handschriftlichen Methode, so lässt sich der Zugriff eines Unbefugten auf die Dateien mit einer unautorisierten Kopie bzw. Manipulation der niedergeschriebenen Leistungen vergleichen. Beides ist in gleichem Maße unwahrscheinlich, aber trotzdem möglich. Als Kritiker könnte man nun argumentieren, dass aufgrund der Übertragung ins Internet die Angriffsmöglichkeiten steigen und sogar unbemerkt bleiben können. So ist das Risiko für den Angreifer wesentlich höher, physischen Zugriff auf die handgeschriebenen Notizen zu erlangen, als von zu Hause aus in das Serversystem einzubrechen. Unter der Annahme, Mallory kann irgendwie an den Klartext einer digitalen Datei auf dem Server gelangen, ist eine weitere Vorkehrung zu treffen.

Wie oben beschrieben enthält nur eine einzige Datei die vollen personenbezogenen Informationen der Schülerinnen und Schüler. Alle pro Fach angelegten Dateien beinhalten nur die jeweilige ID. Da diese im Idealfall zufällig und unabhängig von dem jeweiligen Namen gewählt ist, könnte Mallory mit einer entschlüsselten *Fach-Datei* wenig anfangen: er würde nur anonymisierte Leistungseinschätzungen vorfinden. Um nun die Möglichkeit auszuschließen, dass Mallory durch Kenntnis des Klartextes Einsicht auf personenbezogene Daten erhält, muss der Nutzer nur dafür sorgen, dass die Datei mit den Schülernamen nicht auf den Server hochgeladen wird. Dies erfüllt die in R18 geforderte Datenminimierung. Stattdessen könnte diese lokal auf den jeweiligen Lehrerrechnern gespeichert und über einen sicheren Kanal, z. B. per USB-Stick oder ausgedruckt, verteilt werden. Unter der Annahme, dass sich an der Klassenstruktur über das Schuljahr nichts ändert, müsste diese Maßnahme nur zu Beginn eines Schuljahres einmalig durchgeführt werden. Das jeweilige auf dem Client installierte Programm könnte dann wie gehabt die Verbindung zwischen ID und Schülernamen herstellen.

Im Vergleich zur handschriftlichen Methode ergibt sich letztendlich ein Vorteil bezüglich der Datensicherheit. Zum Einen sind bei einem Diebstahl nicht alle Informationen weg, da diese weiterhin auf dem Server bzw. Backup verfügbar bleiben. Zum Anderen kann ein Zugriff auf personenbezogene Dateien bei einem Diebstahl des Geräts erschwert werden. Zusätzlich zu dem zu hackenden Nutzergeheimnis könnte bei frühzeitiger Entdeckung des Diebstahls der Zugriff auf den Server durch Löschen des öffentlichen Schlüssels des jeweiligen Clients verhindert werden. Sind auf dem Gerät keine Dateien gespeichert, so hat Mallory auch durch Kenntnis des Nutzergeheimnisses keine Möglichkeit, durch Diebstahl an Informationen zu gelangen. Dies

erfüllt R25. Es sei denn, die oben beschriebene Erweiterung der lokalen Speicherung der Klassenliste wird angewendet. In diesem Fall könnte Mallory durch Knacken des Nutzergeheimnisses an die Schülernamen kommen – allerdings weiterhin ohne die Leistungseinschätzungen. Beide Möglichkeiten sind bei der handschriftlichen Methode so nicht gegeben.

4.5. Das Zusammenspiel

Nachdem nun die einzelnen Kanäle und Komponenten für sich vorgestellt wurden, soll im Folgenden das Zusammenspiel dieser anhand der Notenverwaltung betrachtet werden.

4.5.1. Initialisierung

Will eine Lehrkraft erstmalig ihre Daten auf dem Server verwalten, so muss sich diese die entsprechende Software von der Schule holen. Dieser Vorgang lässt sich direkt in das Genehmigungsverfahren integrieren, siehe Ziffer 2 [Nie]. Zusätzlich zu dem eigentlichen Programm erhält die Lehrkraft dabei ein Einmalpasswort, sowie den öffentlichen Schlüssel des Servers. Um einer Manipulationsmöglichkeit dieser Daten vorzubeugen, müssen das Programm (die Installationsdatei) und der Schlüssel auf einer nicht wieder beschreibbaren CD gespeichert sein. Das Einmalpasswort muss direkt in Anwesenheit der Lehrkraft am Server ausgedruckt oder angezeigt werden.

Das überlieferte Programm übernimmt dann nach dem ersten Start das Anlegen des Passwortspeichers für den Lehrer. Dazu nutzt es wie oben erwähnt ein beim Nutzer bleibendes Geheimnis. Nach dem erfolgreichen Anlegen des Speichers erstellt das Programm jeweils die später benötigten asymmetrischen Schlüsselpaare für die Serverauthentifikation, die Ent- und Verschlüsselung sowie ein Paar für die Zertifizierung und Verifizierung (siehe Abb. 4.10).

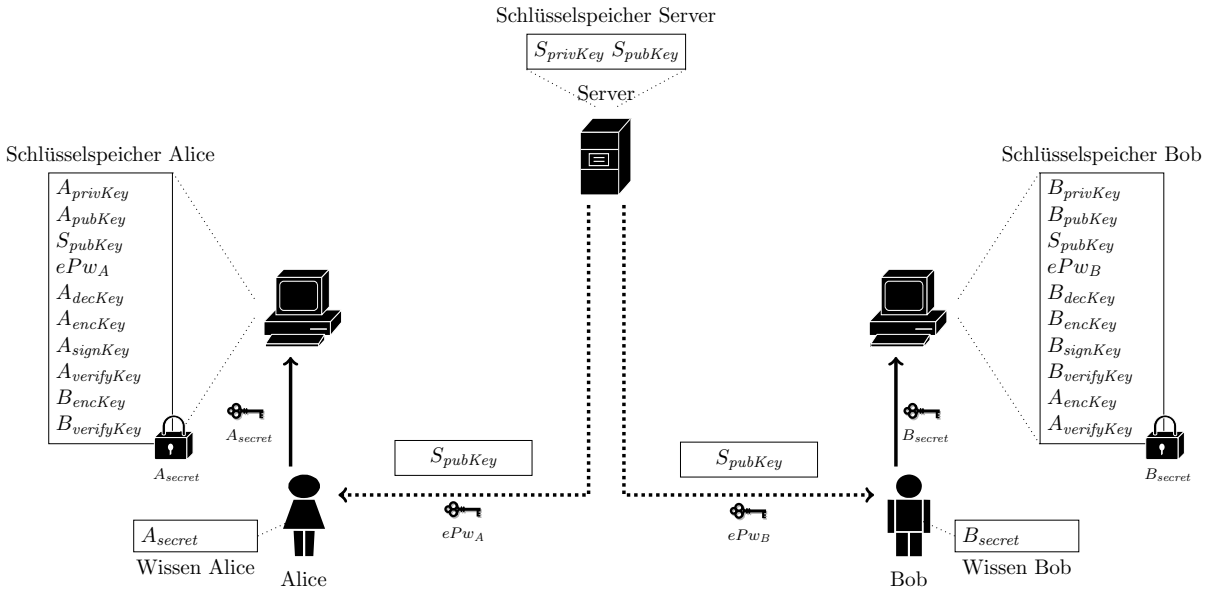

Abbildung 4.10.: Initialisierung über einen sicheren Kanal

4.5.2. Erster Verbindungsaufbau

Hat sich das Programm fertig initialisiert, kann mit dem Aufbau des Verwaltungskanals begonnen werden. Der Vorgang erfolgt dabei wie in Kapitel 4.1 beschrieben. Beide Seiten können nun

für zukünftige Verbindungen sicherstellen, mit dem richtigen Partner zu kommunizieren und gleichzeitig einen über einen Sitzungsschlüssel gesicherten Kanal aufbauen. An die Lehrkraft wird dabei die Anforderung gestellt, das S_{pubKey} besitzende Gerät zu nutzen, eine Internetverbindung eingerichtet zu haben sowie das richtige Nutzergeheimnis zu kennen.

4.5.3. Dateien hochladen und teilen

Die Klassenlehrerin Alice kann nun damit beginnen, lokal die Datei mit den Schülerdaten für ihre Klasse zu erstellen. Zusätzlich legt sie sich für ihre Fächer die einzelnen Dateien und Spalten an. Im Hintergrund kümmert sich das Programm nach dem in Kapitel 4.3 erwähnten Prinzip darum, sämtliche Dateien zu verschlüsseln und die Schlüssel sicher abzulegen (erfüllt R8). Ferner wird die Verbindung der einzelnen zusammengehörigen Dateien mit dem in Kapitel 4.4 vorgestellten *Class-Container* automatisiert übernommen (erfüllt R7). Nach Beendigung der Arbeit kann dann von allein die Übertragung auf den Server sowie das optionale, für die Diebstahlsicherheit zu empfehlende, Löschen lokaler Kopien stattfinden. Alice überträgt damit die Verantwortung für den Schutz ihrer verschlüsselten Dateien an den Server und somit an die Schule. Den Schutz der personenbezogenen Daten muss sie allerdings weiterhin indirekt durch das Schützen der Dateischlüssel garantieren (erfüllt R1 und R9). Will Alice eine Datei ändern, so kann sie diese ganz normal über das eingesetzte Client-Programm öffnen. Im Hintergrund wird dabei sichergestellt, eine aktuelle Kopie von der auf dem Server liegenden Datei lokal auf dem privaten IT-System zu erstellen.

Möchte Alice nun ihre Klassendatei mit dem Fachlehrer Bob teilen, so müssen zuerst beide ihren Trust-Container an den Server übermitteln und über einen sicheren Weg – z. B. bei einem Treffen im Lehrerzimmer – ihre Einmalpasswörter austauschen. Dabei sollte das Passwort allerdings nicht aus den genannten Gründen im Lehrerzimmer aufgehängt werden. Ein solches Vorgehen kann quasi mit dem öffentlichen Aushängen der Leistungseinschätzungen der Schülerinnen und Schüler verglichen werden, da sämtliche nachfolgende Verschlüsselung nicht mehr als sicher angenommen werden kann.

Ist dieses einmalige Vorgehen erstmal abgeschlossen, können Alice und Bob ohne zusätzlichen Aufwand Dateien und so auch den *Class-Container* austauschen (erfüllt R8). Dazu erstellt Alice den in Kapitel 4.3 erwähnten *Share-Key-Container* und gibt diesen, den *Class-Container* sowie die eigentliche Datei mit den Schülerdaten für Bob zum Lesen frei. Von Bobs Client-Programm automatisch eingelesen und synchronisiert, kann Bob nun auf Grundlage dieser Klassendatei die Spalten für seine Fächer anlegen, ohne die Fächer von Alice zu sehen (erfüllt R10). Der Server kann den Inhalt der Daten dabei nicht mitlesen (erfüllt R3).

Um Probleme durch gegenseitiges Überschreiben zu vermeiden, sollte jede geteilte Datei dem anderen Nutzer nur Leserechte gewähren. Dadurch kann sichergestellt werden, dass nur eine Person aktiv den Inhalt einzelner Dateien ändern kann. Im Falle der Datei mit den Schülerdaten kann das der Klassenlehrer sein, bei den Fächerdateien übernimmt dies der Fachlehrer. Die Gewährleistung der Lese- und Schreibrechte übernimmt wie in Kapitel 4.2 beschrieben der Server. Die Rechtevergabe führt der Nutzer selbst durch.

Am Ende eines Schuljahres bzw. zum Ende des Halbjahres könnte dann jeder Fachlehrer sein Fach mit einem Zeugnisaccount teilen. Haben die Klassenlehrer bereits die Dateien mit den Schülerdaten mit diesem Account geteilt, kann dieser dazu eingesetzt werden, die jeweils von den Lehrern vergebenen Zeugnisnoten für die Zeugniskonferenzen bzw. den Druck der Zeugnisse aufzubereiten. Auch kann dieser Sammelaccount dann schnell feststellen, welche Schüler z. B. abstiegsgefährdet sind oder zu oft gefehlt haben. Sicherlich lassen sich noch weitere Szenarien konstruieren, in welchen eine Freigabe einzelner Informationen, wie Klausurnoten in den jeweiligen Fächern, an einen zentrale Account Vorteile und Unterstützung im Schulalltag bringen kann.

4.6. Weitere Sicherheitsaspekte und Hinweise

In den vorangegangenen Kapiteln wurde eine Lösung des gesuchten Systems vorgestellt. Einige noch nicht betrachtete Aspekte sollen hier nun erwähnt werden.

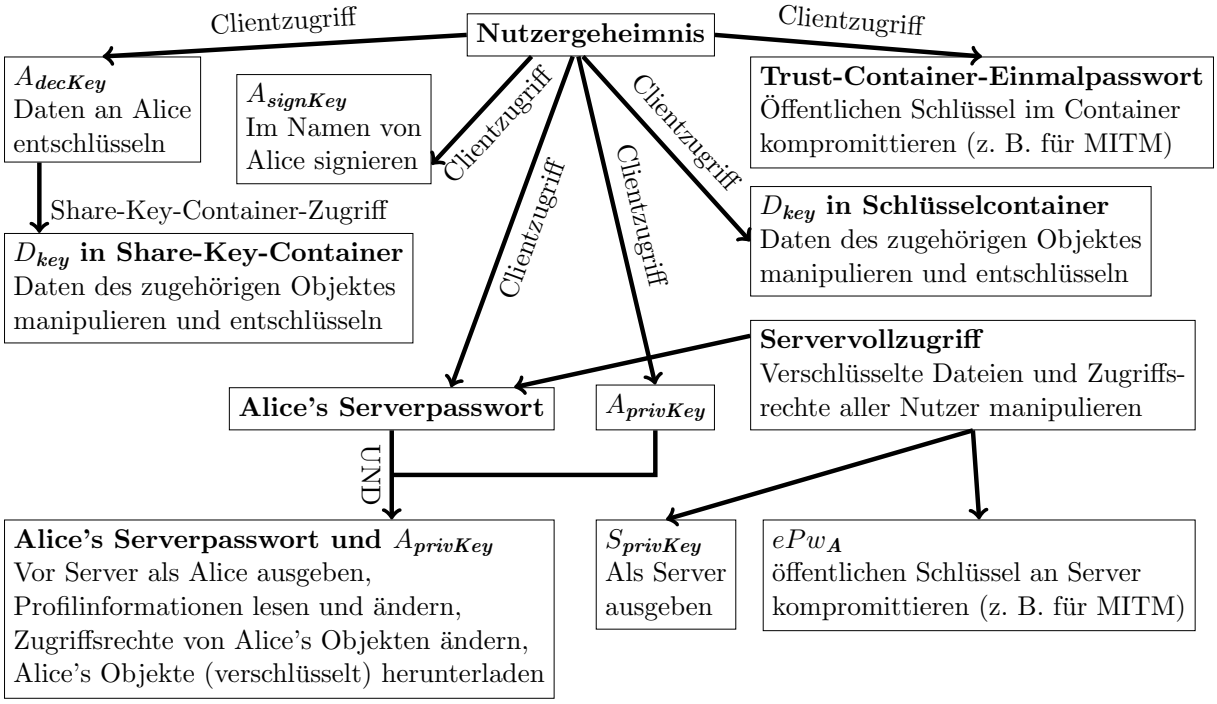

Abbildung 4.11.: Schlüsselhierarchie von Alice und Folgen der Kenntnis eines Schlüssels

Abb. 4.11 zeigt die Zusammengehörigkeit der einzelnen Schlüssel und deren Aufgaben (hier am Beispiel von Alice). D_{key} steht hier für einen beliebigen Dateischlüssel. Direkt auf dem ersten Blick ist ersichtlich, dass das Nutzergeheimnis ganz oben in der Schlüsselhierarchie steht. Dementsprechend ist dieses zu schützen und sicher zu wählen. Eine Prüfung der Passwortstärke durch den Client ist deshalb sinnvoll. Bekommt Mallory trotz Sicherheitsmaßnahmen Zugriff auf D_{key}, so lässt sich einfach eine neue Datei mit neuem Schlüssel erstellen. Allerdings waren alle bis dahin gespeicherten Informationen für Mallory einsehbar (wenn Mallory neben dem Schlüssel auch Zugriff auf die Datei hatte). Deshalb sollte man die Informationen auf möglichst viele Dateien aufteilen (wie in Kapitel 4.4 erläutert). Ferner könnte man automatisiert den Dateischlüssel regelmäßig ändern. Erlangt Mallory Zugriff auf einen der privaten Schlüssel von Alice, so ist der Schaden wie in Abb. 4.11 gezeigt deutlich größer. Wichtig ist es deswegen, dass Alice andere Teilnehmer rechtzeitig warnt, sollte sie von einer Kompromittierung erfahren.

In Kapitel 4.3 wurde angedeutet, dass durch zufällige Dateinamenswahl das Raten des möglichen Inhalts einer verschlüsselten Datei erschwert werden könnte. Führt Mallory nun eine *Verkehrsanalyse* mit Hilfe des Servers durch, so bieten sich ihm noch mehr Möglichkeiten, um auf den Inhalt bzw. Typ der Datei zu schließen. Ein *Trust-Container* kann z. B. aufgrund seiner Größe, die sich nur durch unterschiedliche Nutzerinformationen (z. B. Länge des Benutzernamens) minimal ändert, erkannt werden. Die Länge der Schlüssel bleibt immer gleich. Weiterhin soll der Trust-Container wie erwähnt sofort nach Gebrauch gelöscht werden. Eine relativ kleine Datei, die nach dem ersten Zugriff gelöscht wird, ist also mit sehr hoher Wahrscheinlichkeit ein Trust-Container. Ähnliches gilt für den Share-Key-Container-Wrapper, welcher allerdings

aufgrund seines Inhaltes etwas größer als der Trust-Container sein dürfte. Es lohnt sich also für Mallory solche Dateien eher mit Brute-Force zu knacken, da diese einen Dateischlüssel enthalten bzw. im Fall des Trust-Containers eine MITM-Attacke erlauben.

Um dem entgegenzuwirken, könnte ein Client kleine Dateien mit zufälligem Inhalt auf den Server hochladen und auch diese nach dem ersten Zugriff löschen. Andernfalls könnte in Form von zufälligem Anhang die eigentliche Größe der Container-Dateien manipuliert werden.

5. Beispielimplementierung

Nach dem erfolgten Konzeptentwurf soll nun eine Implementierung dessen vorgestellt werden. Während der Präsentation des Grundkonzeptes (siehe Anhang A.2) entstanden weitere Anforderung an die Beispielimplementierung:

[**R26**] Das System muss fähig sein, die Endnoten der täglich eingegebenen Leistungseinschätzungen der Lehrkräfte mit einem physisch getrennten Notenverwaltungssystem am Ende des Schuljahres zu synchronisieren.

[**R27**] Das System muss der Lehrkraft die Möglichkeit bieten, Fehlzeiten der Schülerinnen und Schüler mit der Option entschuldigt / nicht entschuldigt einzutragen.

[**R28**] Das System muss unabhängig von einem Schulsystem von der Lehrkraft privat nutzbar sein.

[**R29**] Das System sollte zu einer zentralen, von allen Lehrkräften genutzten Verwaltungssoftware für die Schule erweiterbar sein.

[**R30**] Das System muss der Lehrkraft die Möglichkeit bieten, Änderungen anderer Lehrkräfte an den ihnen sichtbaren Daten mit zu verfolgen und der jeweiligen Lehrkraft eindeutig zuordnen zu können.

[**R31**] Das System muss der Lehrkraft die Möglichkeit bieten, auch in der gymnasialen Oberstufe – ohne ein Klassensystem – die Leistungsverwaltung vornehmen zu können.

Anforderung R26 ist schon durch den Konzeptentwurf realisierbar, indem ein spezieller Zeugnisaccount eingerichtet wird (siehe auch Kapitel 4.5.3). R27 kann aufgrund der gesetzlichen Einschränkung des Datenrahmens (siehe Kapitel 2.1) nicht realisiert werden. Aufgrund einiger Skepsis gegenüber technischen Sicherheitsmaßnahmen unter dem Großteil der Lehrkräfte würde es sich für die Schule nicht lohnen, einen extra Server zu installieren, wodurch die Anforderungen R28 und R29 entstanden. Aus diesem Grund müssen die in Kapitel 5.1 beschriebenen Änderungen zu dem entworfenen Konzept vorgenommen werden. Bei der Implementierung wird das *Qt-Framework* [Com] eingesetzt werden. Durch die von der Programmiersprache C++ gewährten Hardwarenähe und der mit den mitgelieferten Bibliotheken größtenteils gebotenen Plattformunabhängigkeit (durch R5 gefordert) eignet sich das Framework gut für die Entwicklung sicherer Software. Zusätzlich bietet der Einsatz der Krypto-Bibliothek *QCA* (näheres in Kapitel 5.2) bestehende Implementierungen der in der Kryptografie eingesetzten Standardverfahren. Als Betriebssystem zur Entwicklung kam Ubuntu 14.04 LTS (64-Bit) zum Einsatz.

Vor dem Programmieren müssen allerdings in Kapitel 5.3 gemäß der Empfehlung in Kapitel 2.4.2 die Aufgaben auf verschiedene Module aufgeteilt werden. Danach sollen diese über wohl definierte Schnittstellen miteinander kommunizieren. Die jeweiligen Aufgaben der Module werden dann in den darauffolgenden Kapiteln beschrieben. Da der Fokus der Arbeit auf dem Sicherheitsaspekt liegt, werden notwendige Hilfsfunktionen nicht weiter ausgeführt. R30 und R31 können deswegen auch nicht beachtet werden – eine mögliche Erweiterung wird aber in Kapitel 7 beschrieben.

5.1. Abgrenzung zum Konzeptentwurf

Die wesentliche Änderung gegenüber dem vorgestellten Konzept umfasst den Verzicht auf den beschrieben Schulserver. Stattdessen soll es möglich sein, einen beliebigen, bereits im eigenen IT-System verknüpften Cloud-Server zu nutzen. Zur Vereinfachung sei angenommen, dass dieser Server direkt in das Dateisystem des genutzten Betriebssystems integriert ist, die Authentifikation gegenüber dem Server sei vor dem Programmstart erfolgt. Auch ist die zum Server vorgenommene Übertragung gegen Angriffe z. B. durch TLS gesichert. Eine Implementierung des Verwaltungskanals (siehe Kapitel 4.2) entfällt also. Dadurch wird es auch nicht möglich sein, Zugriffsrechte wie Lesen und Schreiben einer Datei zu koordinieren. Es muss also angenommen werden, dass der Nutzer in dem Cloud-Dienst diese Einstellungen selbst übernimmt. Dadurch können sowohl R28 als auch R29 (durch Nutzung des Servers vom Konzept) erfüllt werden.

Insgesamt lässt sich deshalb die grundlegende Sicherheitsfunktion mit dem Programm *Boxcryptor* (siehe Kapitel 3.3) vergleichen: die Klartextdateien werden im Hintergrund verschlüsselt gespeichert, das Schlüsselmanagement geschieht auch automatisch. Anders als bei *Boxcryptor* sollen allerdings alle Schlüssel auf dem privaten System bleiben – eine Abänderung des Datenkanals (siehe Kapitel 4.3) findet nicht statt. Zusätzlich dazu soll das Konzept der Datenkodierung und Aufteilung mit übernommen werden. Zu guter Letzt bietet eine Export- und Import-Funktion die Möglichkeit, mit Hilfe externer auf dem privaten System installierter Software, eine komplexere Informationsverwaltung als die erwähnte Tabellenstruktur vornehmen zu können (durch R11 gefordert).

5.2. Die Krypto-Bibliothek QCA

Aufgrund der Tatsache, dass sowohl in dem theoretischen Modell als auch in der Implementierung auf eine detaillierte Festlegung eines Krypto-Algorithmus verzichtet werden soll, liegt es Nahe, eine bereits vorhandene Krypto-Bibliothek zu verwenden. Weiterhin wird damit auf die Empfehlungen zur Umgehung von Implementierungsfehlern in Kapitel 2.4.2 eingegangen. Aufgrund der geforderten Unabhängigkeit zu den Verschlüsselungsalgorithmen stellt sich an eine solche Krypto-Bibliothek also die Anforderung, sowohl grundlegende, z. B. vom BSI empfohlene, Algorithmen über eine Schnittstelle bereit zu stellen, als auch intern eine gewisse Austauschbarkeit der Implementierungen zur Verfügung zu stellen (Anforderung R16). Weiterhin sollte wie in Kapitel 2.4.2 empfohlen, eine Open-Source Lösung gewählt werden, um die Fehlerwahrscheinlichkeit und Überprüfbarkeit der Kryto-Bibliothek zu gewährleisten. Auch ist eine Plattformunabhängigkeit der externen Bibliothek angesichts R5 wünschenswert.

Aufgrund dieser Anforderungen wurde die Krypto-Bibliothek *QCA* (Qt Cryptographic Architecture) Version 2.1.0 gewählt. Wie in [Pro14] nachzulesen ist, erfüllt die Bibliothek alle genannten Vorgaben. Eingesetzt werden kann sie auf allen Plattformen, die Qt unterstützt. Insbesondere Windows, Unix und MacOSX fallen darunter. Durch den in Abb. 5.1 gezeigten Plugin-Mechanismus lassen sich die einzelnen Implementierungen ohne Neukompilierung des eigentlichen Programms austauschen, was theoretisch auch eine Verschiebung der Algorithmen auf externe Hardware wie eine Smartcard erlaubt. Auch eigene Plugins sind möglich. Ferner werden sowohl die vom BSI empfohlenen Algorithmen AES und RSA als auch komplexere Aufgaben wie das Zertifikatsmanagement oder das PGP-Nachrichtensystem unterstützt. Auch eventgesteuerte Passwortabfragen, die Password-Key-Derivation-Function PBKDF2, ein sicherer Zufallsgenerator sowie verschiedene Hash-Algorithmen sind vorhanden.

In [Pro14] findet sich weiterhin eine Tabelle der von den jeweiligen Plugins unterstützten Algorithmen. Um die oben genannten Funktionalitäten, die zum Großteil im Folgenden eingesetzt werden sollen, nutzen zu können, wird das *QCA OSSL plugin version 2.0.0* genutzt. Aufbauend

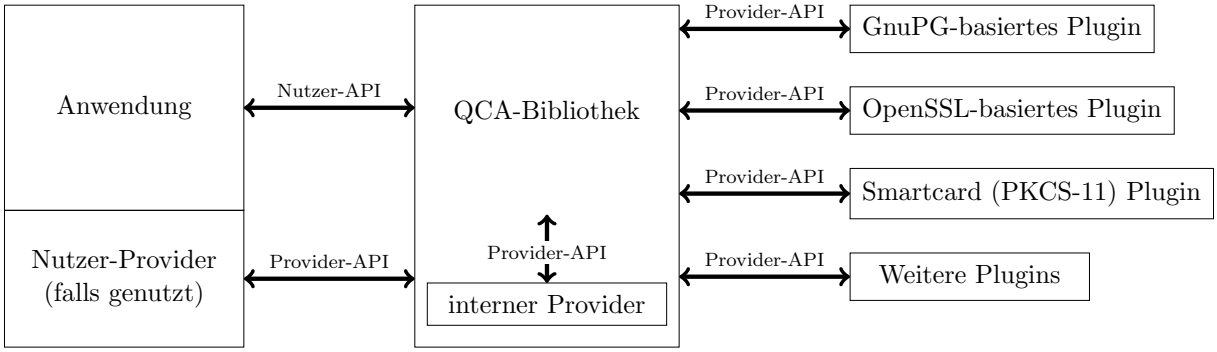

Abbildung 5.1.: QCA Architektur, Quelle [Pro15]

auf OpenSSL, einer Open-Source-Bibliothek für u. a. Krypto-Algorithmen [SE], muss diese auf dem jeweils eingesetzten System vorhanden sein. Weitere Informationen sowie eine detaillierte Dokumentation der API finden sich unter [Pro15]. Lizenziert ist die Bibliothek unter der LGPL.

5.3. Softwareaufbau

Im Folgenden wird grob die Struktur der Implementierung vorgestellt werden. Unter Anwendung des *MVC-Pattern* wurden die verschiedenen Klassen gemäß ihrer Aufgaben zusammengefasst. Mit Hilfe von Multithreading soll sichergestellt werden, dass die GUI jederzeit auf Eingaben des Nutzers reagieren bzw. Fortschritte anzeigen kann. Auch wird dadurch der Einsatz der Events der QCA-Bibliothek ermöglicht. Schematisch ist dies in Abb. 5.2 dargestellt.

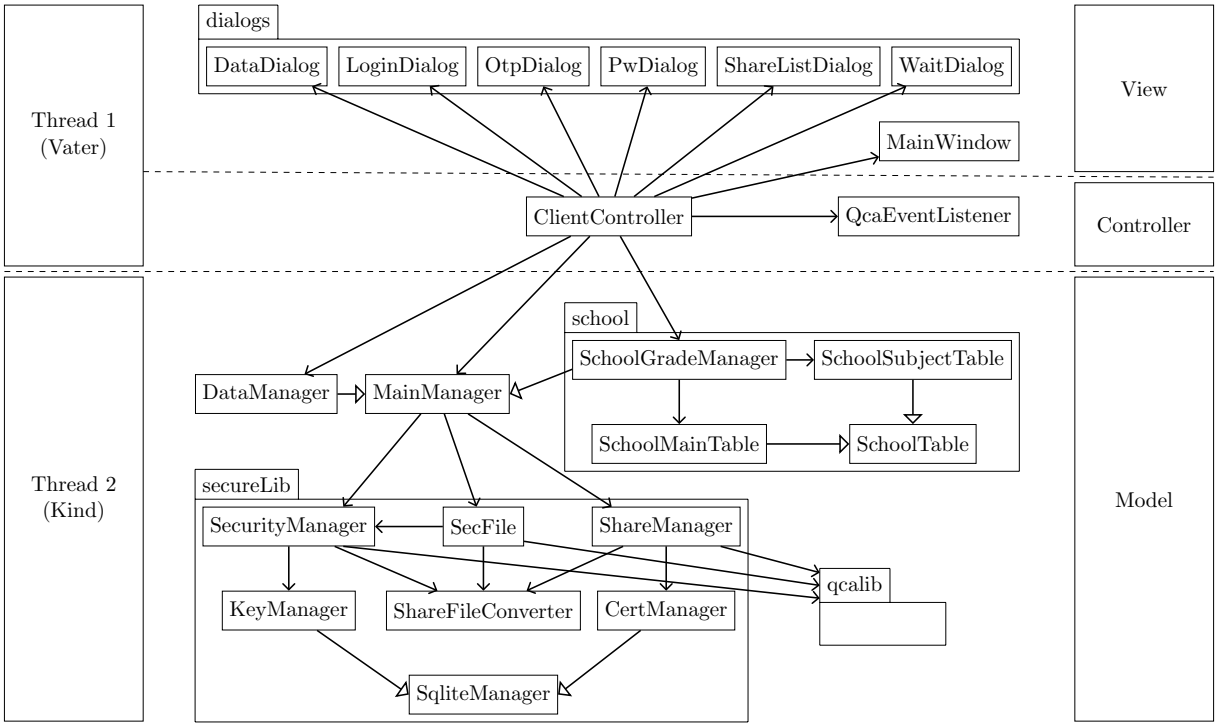

Abbildung 5.2.: UML-Diagramm mit Threads und MVC-Pattern

Im Mittelpunkt aller Klassen steht der *ClientController*. Mit seiner Initialisierung in der *main()*-Methode werden das *MainWindow* und der *QcaEventListener* übergeben. Der Constructor kümmert sich dann um das richtige Verbinden und Initialisieren der weiteren Module. Unterstützt wird dieser Vorgang durch den *Signal-Slot-Mechanismus* von Qt (eine Art *Oberserver-Pattern*). Infolge dessen besteht die Aufgabe des Controllers darin, die Signale der Module entsprechend zu modifizieren und an den jeweiligen Slot eines anderen Moduls weiterzuleiten.

Abbildung 5.3.: Ansicht des MainWindow mit Tabelle aus Abb. 4.8

Die Hauptansicht des *MainWindow* ist in Abb. 5.3 gezeigt. Neben den Optionen, um ein Objekt zu erstellen, auszuwählen und zu speichern finden sich in der rechten oberen Ecke drei Buttons, die die Hauptfunktionen des Konzepts implementieren: seinen *Trust-Container* exportieren, den einer anderen Instanz importieren und Daten teilen. Weiterhin landet man beim Logout wieder im *LoginDialog*, der auch zu Beginn zu sehen ist. Im Hintergrund werden dabei die temporär entschlüsselten Daten bzw. das Nutzerpasswort gelöscht, sodass ein Zugriff auf Nutzerdaten erst wieder mit der Eingabe des Passwortes möglich ist. Im mittleren Bereich wird bei Auswahl eines Objektes dessen Inhalt dargestellt – in Abb. 5.3 ist dies die zur Abb. 4.8 passende Ansicht des *SchoolGradeManagers*. Zur Nutzerinteraktion existieren folgende Dialoge:

DataDialog: Aktionsauswahl für Export-/Importansicht (siehe Kapitel 5.7.2)

LoginDialog: Nutzerauswahl, Login und Erstellen eines neuen Nutzers

OtpDialog: Ein- und Ausgabe des Einmalpasswortes

PwDialog: Eingabe des Nutzerpasswortes

ShareListDialog: Auswahl der zu teilenden Daten und Nutzer

WaitDialog: Eine „Warte-Anzeige“

Die Steuerung der Dialoganzeige bzw. das Anstoßen der dazugehörigen Methoden im Model wird durch die Verknüpfungen im *ClientController* sichergestellt. Den Mittelpunkt der *Model-Klassen* stellt das *secureLib*-Package dar. In ihm sind sämtliche Klassen enthalten, welche die im Konzept beschriebenen Funktionen des Clients implementieren. Auf den *SqliteManager* (siehe

Kapitel 5.4), den *SecurityManager* (siehe Kapitel 5.5) und den *ShareManager* (siehe Kapitel 5.6) wird noch weiter unten eingegangen werden. Auch die Aufgaben des *KeyManagers* und *CertManagers* werden dort erläutert.

Die *SecFile*-Klasse umfasst grundlegende Funktionen, um mit auf der Festplatte verschlüsselten Objekten arbeiten zu können. Soll auf den Inhalt einer solchen Datei zugegriffen werden, so kann initial ein *SecFile*-Objekt durch Übergabe der Daten-ID und dem *SecurityManager*-Objekt erstellt werden. Danach kann dieses Objekt mit allen weiteren Klassen (auch außerhalb des Packages) geteilt werden. Über Lese- und Schreibmethoden ist dann eine Interaktion mit dem Objekt möglich, ohne von der Verschlüsselung wissen zu müssen (Stichwort: *Information Hiding*).

Schließlich bietet der *ShareFileConverter* statische Methoden, um die einzelnen Variablen der im Konzept beschriebenen Container in einen Bytestream und wieder zurück konvertieren zu können. Dabei wird auch durch spezielle Start- und Endsequenzen sowie Prüffunktionen sichergestellt, dass ein eingelesenes Objekt nicht manipuliert wurde. Sämtliche kryptografische Methoden werden wie bereits erwähnt durch die *QCA*-Bibliothek bereitgestellt.

Mit der *MainManager*-Klasse wird das Zusammenspiel der einzelnen Klassen der *secureLib* ermöglicht. So stellt diese grundlegende Implementierungen wie Login und Logout sowie die in der GUI bereitgestellten Aktionen dem *ClientController* zur Verfügung. Über Signal und Slots wird die notwendige *Inter-Thread-Communication* ermöglicht. Da der *MainManager* selbst nur die einfache Texteingabe für Objekte bereitstellt, existieren der *SchoolGradeManager* (siehe Kapitel 5.7.1) sowie der *DataManager* (siehe Kapitel 5.7.2), um das Funktionsangebot zu erhöhen. Durch weitere Vererbung kann an dieser Stelle angesetzt werden, um zusätzliche Aufgaben mit dem Programm erledigen zu können, ohne den Hintergrund der Ver- und Entschlüsselung kontrollieren zu müssen.

5.4. Der SQLiteManager

Pro Nutzer müssen zwei Datenbanken angelegt werden. Der *KeyManager* verwaltet dabei sämtliche Schlüssel, die der *SecurityManager* (siehe Kapitel 5.5) benötigt, um Objekte ver- und entschlüsseln zu können. Zusätzlich werden dort im Fall von Alice die privaten Schlüssel A_{decKey} und $A_{signKey}$ verwaltet. Das Speichern der zugehörigen öffentlichen Schlüssel ist nicht notwendig, da *QCA* mit der Methode *toPublicKey()* die eindeutige Berechnung dieser aus dem privaten Schlüssel ermöglicht. Ein beispielhafter Inhalt einer solchen Datenbank ist u. a. in Abb. 5.4 gezeigt (mehr dazu in Kapitel 5.5). Mit dem *CertManager* werden alle Schlüssel verwaltet, die der *ShareManager* braucht, um Inhalte mit anderen Nutzern zu teilen. Damit jeweils ein schneller Zugriff auf die benötigten Schlüssel gewährleistet werden kann, wurde je Manager als Basis eine SQL-Datenbank gewählt – genauer SQLite, um die Datenbank in einer Datei abspeichern zu können. Diese Tatsache wird allerdings vor dem *Security-* bzw. *ShareManager*, welche über definierte Funktionen an die bestimmten Schlüssel gelangen, versteckt.

Der Zugriff funktioniert über in den Datenbanken pro Zeile eindeutig generierten IDs. Intern wird mit *structs* die eindeutige Zuordnung zwischen Information und ID-Spalte sichergestellt. Während die IDs beim *CertManager* einfach nur inkrementiert werden, ist dies beim *KeyManager* komplexer. Zuerst wird dort zwischen den Daten für ein Objekt und speziellen Daten unterschieden (näheres siehe Kapitel 5.5) – für beide wird jeweils eine eigene Tabelle angelegt. Bei ersteren Daten bildet dann der Dateiname die ID. So kann sichergestellt werden, dass bei der Übermittlung eines Schlüssels an einen anderen Nutzer über die ID ein Objekt eindeutig identifiziert werden kann. Weiterhin ist sichergestellt, dass jede ID nur einmal existiert, da in

einem Ordner ein Dateiname nur einmal vorkommen darf. Bei den speziellen Daten stellt der Typ die eindeutige Referenzierung auf eine Zeile sicher.

Da sowohl die Verwaltung im *CertManager* als auch im *KeyManager* einen ähnlichen Aufbau hat, sind grundlegende Funktionen im *SQLiteManager* definiert, von dem beide erben. Ferner sollte erwähnt werden, dass beide Manager keine Form der Ver- und Entschlüsselung erlauben. Sie speichern sämtliche überlieferte Einträge unabhängig ihres Inhalts ab – den Schutz der Daten übernimmt der anschließend erwähnte *SecurityManager*.

5.5. Der SecurityManager

Der SecurityManager ist der Grundbaustein zur Ver- und Entschlüsselung von Informationen. Damit verwaltet er auch das für den Schlüsselcontainer notwendige Nutzergeheimnis (siehe Kapitel 4.1). Aufgrund der erwähnten Patentrechte war es bei der Implementierung nicht möglich, die Idee des *MobileSitters* (Kapitel 3.4) zur Erschwerung eines Brute-Force-Angriffes zu implementieren. Stattdessen wird als „spezielles Datum“ ein Hash (SHA-512 mit 100.000 Iterationen) des zu Beginn gewählten Passwortes im *KeyManager* abgelegt. Durch Überprüfen des eingegebenen Passwortes mit diesem Wert kann auf Anfrage die Gültigkeit zurückgeliefert werden. Zur Berechnung des Passwortschlüssels aus dem eingegeben Passwort wird die in QCA implementierte und in [Sch13a, S. 457f] beschriebene PBKDF2-Methode verwendet, um aus einer beliebigen Passwortlänge einen Schlüssel der Länge 256-Bit zu generieren. Schematisch ist der Funktionsumfang auch unter Darstellung des Zusammenspiels mit dem *KeyManager*, welcher hier das Bereitstellen der Informationen aus der Datei user1_key.sqlite übernimmt, und der *SecFile*-Klasse in Abb. 5.4 gezeigt. Das Ziel ist es dort, den Inhalt sowie die Attribute der Datei 123.data zur späteren Nutzung zu entschlüsseln und in einem *SecFile*-Objekt für den späteren Gebrauch bereitzustellen.

Bei dem Großteil der Methoden kommt aufgrund der Empfehlungen des BSI (siehe Kapitel 2.2.7) und der durch QCA gebotenen Methoden AES-256 mit CBC für die symmetrische Verschlüsselung zum Einsatz. Bei asymmetrischer Verschlüsselung wird deshalb RSA (PKCS#1, Version 2.0, EME-OAEP) mit einer Schlüssellänge von 4096 Bit eingesetzt. Aufgrund der Struktur von AES ist es notwendig im Schlüsselcontainer neben dem Schlüssel auch den Initialisierungsvektor abzulegen. Über eine zentrale Methode wird für jedes zu verschlüsselnde Datum die zufällige Generierung und Validierung des benötigten AES-Schlüssels sowie des Initialisierungsvektors vorgenommen. Genutzt wird dazu der Zufallsgenerator der QCA-Bibliothek.

Eine vom *SecurityManager* verwaltete Datei wird Objekt genannt. Jedes Objekt wird wie im Konzept erwähnt mit einem symmetrischen Schlüssel verschlüsselt auf dem Datenträger abgelegt. Beim Anlegen eines neuen Objektes werden zusätzlich die in Abb. 5.4 oben gezeigten Metadaten verschlüsselt dem *KeyManager* zum Speichern übergeben. Dazu ist der dort gezeigte Vorgang des Lesens umzukehren. Initialisierungsvektoren (IV), Datenschlüssel und Objekt-ID werden jeweils mit QCA zufällig generiert. Einzig die ID, um einzelne Zeilen direkt auszuwählen, und der Typ, für eine schnelle Filterung, liegen im Klartext vor. Da die Objekt-ID, wie in Kapitel 5.4 erwähnt, dem Dateinamen entspricht, ist ein Zugriff direkt mit Verbindung des Arbeitsverzeichnisses möglich. Dieses wurde beim Anlegen des Nutzers als „spezielles Datum“ gespeichert. Damit Daten lokal verarbeiten werden können, kann mit einer öffentlichen Methode eine Liste aller bekannten Objekte geholt werden (optional ist dabei eine Filterung nach Typ möglich). Mit einem *DataObject* der Liste, welches den Objektnamen und die ID enthält, lässt sich dann wie in Abb. 5.4 links gezeigt das zugehörige *SecFile*-Objekt über eine zweite Methode abrufen.

Wie im Konzept (siehe Kapitel 5.7.2) beschrieben, muss, wenn man ein Objekt mit einem an-

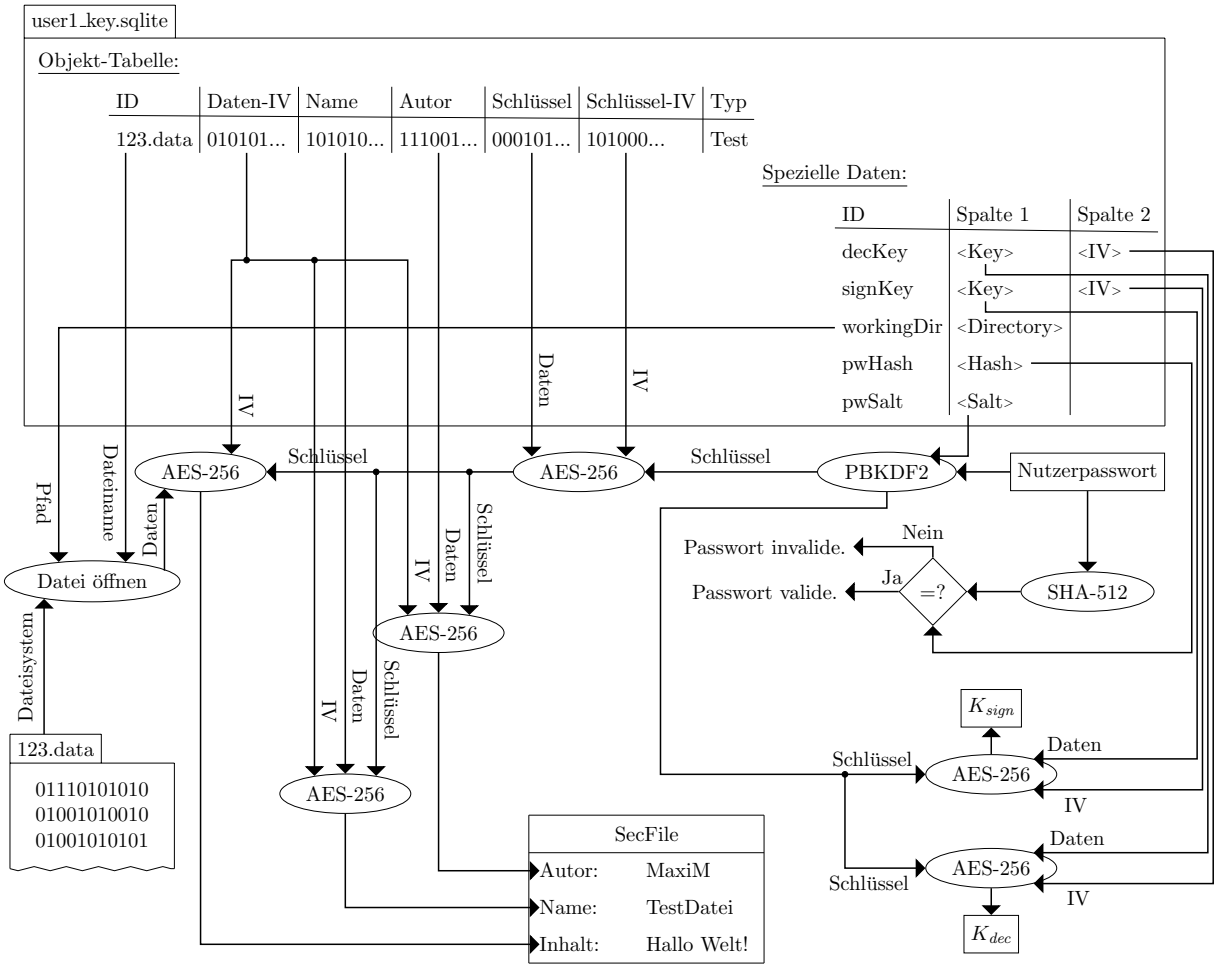

Abbildung 5.4.: Funktionsübersicht des SecurityManagers

deren Nutzer teilen möchte, der Objekt-Schlüssel in einem *Share-Key-Container* gespeichert werden. Dazu bietet der *SecurityManager* die Möglichkeit für eine übergebene Objekt-ID und dem öffentlichen Schlüssel des anderen Nutzers das Byte-Array des *Share-KeyContainer-Wrappers* zu erstellen. Dazu greift der *SecurityManager* auf den zu Beginn erstellten Signierschlüssel K_{sign} zu (siehe Abb. 5.4 rechts). Genutzt wird diese Methode vom *ShareManager*.

Über eine weitere Methode kann ein neues Objekt, für welches schon eine Datei sowie Schlüssel, Initialisierungsvektor und Metadaten existieren, in die Verwaltung des *KeyManagers* übernommen werden. Dies geschieht allerdings nur, wenn die übergebene ID noch nicht bekannt ist. Genutzt wird auch diese Methode vom *ShareManager*, um ein Objekt eines anderen Nutzers importieren zu können. Benötigt wird dazu wie im Konzept beschrieben der private Schlüssel K_{enc} zum Entschlüsseln des Schlüssels für den *Share-Key-Container*.

Schließlich bietet der *SecurityManager* eine Reihe an öffentlichen Methoden, um übergebene Daten zu ver- und entschlüsseln. Das dazu notwendige Nutzerpasswort bzw. der daraus generierte Schlüssel verlassen dazu nie den *SecurityManager* und müssen somit den anderen Klassen nicht bekannt sein.

5.6. Der ShareManager

Die zum Teilen der Objekte benötigten Methoden sind im *ShareManager* zusammengefasst. Mit dem intern genutzten *CertManager* werden auch die dazu notwendigen Informationen verwaltet. Um die Ver- und Entschlüsselung einzelner Informationen vornehmen zu können, muss dem *ShareManager* der genutzte *SecurityManager* bekannt sein. Dadurch ist sichergestellt, dass sämtliche Kryptografie weiterhin im *SecurityManager* stattfindet.

Im Konzept wurde die Nutzung eines Einmalpasswortes festgelegt, um die Authentizität des zu vertrauenden Nutzers überprüfen zu können. Da es sich hier um nichts anderes als ein normales Passwort handelt, muss auch an dieser Stelle die in [Sch04, S. 97] erwähnte Problematik der Schlüsselentropie beachtet werden. In diesem Fall bieten sich allerdings zwei Vorteile gegenüber dem Nutzerpasswort: zum Einen kann das Passwort vom Programm selbst erzeugt und damit die Gleichwahrscheinlichkeit aller Symbole sichergestellt werden, zum Anderen wird dieses Passwort, wie der Name schon sagt, nur einmalig angewendet und kann somit auch länger als ein übliches Passwort sein.

Mit *QCA::Random* existiert ein Zufallsgenerator, von dem hier angenommen wird, dass alle Symbole mit der gleichen Wahrscheinlichkeit ausgegeben werden. Mit Hilfe der Methode *QCA::Random::randomChar()* lassen sich also $2^8 = 256$ gleich wahrscheinliche Zufallszahlen generieren. Da nun ein Nutzer das generierte Passwort im System eingeben muss, dürfen nur vorhandene Zeichen der Tastatur verwendet werden. Nimmt man das Standardalphabet mit 26 Groß- und 26 Kleinbuchstaben, die Zahlen 0 bis 9, die Sonderzeichen !“§$%&()=?{}[]\/*+-_#><|;,:.’~@€ und das Leerzeichen, ergeben sich mind. 95 verschiedene Symbole, die auf der in Deutschland eingesetzten QWERTZ-Tastatur vorhanden sind. Sowohl der Zufallsgenerator als auch der zu generierende Schlüssel arbeiten auf Bit-Ebene, weshalb als Symbolanzahl pro Zeichen die nächste unter 95 liegende Zweierpotenz gewählt wird: $2^6 = 64$. Es lassen sich somit 64 druckbare Symbole für ein Zeichen einsetzen. Will man nun einen 128-Bit langen Schlüssel generieren, so braucht man 22 dieser Zeichen: 21 mit jeweils 6 Bit und eines mit 2 Bit. Für einen 256-Bit langen Schlüssel bräuchte man dementsprechend 42 6 Bit Symbole und 1 mit 4 Bit. Laut dem BSI gilt auch AES-128 als sicher, weshalb hier zugunsten der Benutzerfreundlichkeit der kürzere Schlüssel gewählt wird.

Insgesamt ergibt sich damit zum Generieren des Schlüssels (PW), des Initialisierungsvektors (IV) sowie des dazugehörigen lesbaren Einmalpasswortes (OTP) der Algorithmus 1. Die Ausdrücke DRUCKBARES_ZEICHEN_6BIT und DRUCKBARES_ZEICHEN_2BIT sind dabei jeweils Arrays, welche jedem möglichen Binärwert eindeutig eines der oben genannten druckbaren Zeichen zuordnet. Die angewendete Modulo-Operation, welche die 256 Zufallszahlen auf 64 reduziert, behält hierbei die Gleichwahrscheinlichkeit der 64 Symbole bei (jeweils vier verschiedene und gleich wahrscheinliche Zufallszahlen werden auf ein Symbol gemapped).

Die Methoden zum Erstellen des *Trust-Containers* und *Share-Key-Container-Wrappers* (siehe Kapitel 4.3) fallen genauso wie beim Konzept aus. In beiden Fällen werden die jeweils notwendigen Informationen gesammelt, um diese anschließend mit Hilfe der Verschlüsselungsmethoden des *SecurityManagers* zu sichern. Das Einmalpasswort für den *Trust-Container* wird dazu wie oben beschrieben generiert und dem Nutzer einmalig mit dem *OtpDialog* angezeigt. Zusätzlich zum Konzept ist ein *Trust-Container-Wrapper* notwendig. Dieser enthält neben dem verschlüsselten *Trust-Container* den unverschlüsselten Initialisierungsvektor IV des verwendeten Schlüssels PW. Beim *Share-Key-Container-Wrapper* wird zusätzlich zu dem Ergebnis der Export-Methode des *SecurityManagers* ein eindeutiger Header an den Dateibeginn geschrieben, damit das Client-Programm des Empfängers die geteilten Daten automatisch als solche erkennt. Da AES einen Initialisierungsvektor fordert, müssen diese zusätzlich zu dem im Konzept erwähnten

Algorithmus 1 Generierung des Einmalpasswortes

1: Fülle IV mit einer 128-Bit-Zufallszahl.
2: leere PW.
3: leere OTP.
4: Setze i auf 0.
5: **while** $i < 21$ **do**
6: Setze C auf eine 8-Bit-Zufallszahl. ▷ QCA::Random::randomChar()
7: Setze C auf C modulo 64. ▷ Bekomme 6-Bit-Zufallszahl
8: Hänge Binärwert von C an PWP an.
9: Hänge DRUCKBARES_ZEICHEN_6BIT[C] an OTP an.
10: Inkrementiere i um 1.
11: **end while**
12: Setze C auf eine 8-Bit-Zufallszahl. ▷ QCA::Random::randomChar()
13: Setze C auf C modulo 4. ▷ Bekomme 2-Bit-Zufallszahl
14: Hänge Binärwert von C an PWP an.
15: Hänge DRUCKBARES_ZEICHEN_2BIT[C] an OTP an.

symmetrischen Schlüsseln geteilt werden.

Beim Empfänger fragt das Client-Programm nach Betätigen des Buttons zum Import der Vertrauensinformationen nach dem Einmalpasswort des *Trust-Containers*. Daraus wird anschließend mit den zur Generierung (siehe Algorithmus 1) genutzten Arrays der zugehörige binäre Schlüssel ermittelt. Mit dem übertragen Initialisierungsvektor können anschließend die übertragenen Daten entschlüsselt und in der Datenbank des *CertManagers* mit dem Nutzerpasswort verschlüsselt abgelegt werden.

Der *MainManager* ruft im Anmeldeprozess nach der erfolgreichen Initialisierung des *SecurityManagers* und der Passwortvalidierung eine öffentliche Methode des *ShareManagers* zum Suchen nach einer Datei mit einem *Share-Key-Container-Wrapper* auf. In dieser wird in einem übergebenen Verzeichnis (beim *MainManager* das Arbeitsverzeichnis) nach einer Datei mit dem oben erwähnten speziellen Header gesucht. Wird eine solche gefunden und kann der *SecurityManager* die Informationen erfolgreich mit dem privaten Schlüssel K_{enc} entschlüsselt, so wird der Dateischlüssel automatisch zum Login importiert. Der Nutzer kann somit sofort auf das neu hinzugefügte Objekt zugreifen. R8 wird also erfüllt.

5.7. Mögliche Ansichten

Aufbauend auf den implementierten Sicherheitsfunktionen soll im Folgenden gezeigt werden, welche Möglichkeiten sich zur Notenverwaltung ergeben. Der *SchoolGradeManager* versucht dabei exemplarisch die Umsetzbarkeit der in Kapitel 4.4 beschriebenen Datenrepräsentation zu zeigen. Mit dem *DataManager* soll die Möglichkeit des Exports und Imports von verschlüsselt geteilten Daten demonstriert werden (gefordert in R11). Dadurch soll es möglich sein, bereits existierende Software durch den bis hierhin entwickelten Mechanismus zum sicheren Verteilen von Dateien erweitern zu können. Schließlich entstand während der Implementierungsphase auch eine Variante, in der es dem Nutzer ermöglicht wird, einfache Texteingaben in den Objekten zu Speichern (Standardfunktion des *MainManager*, siehe Abb. 5.2). Aufgrund des geringen Funktionsumfangs (Text eingeben und ändern) soll auf dieses Modul nicht weiter eingegangen werden. Alle drei Module sind in der aktuellen Fassung voneinander durch Präprozessor-Anweisungen getrennt, sodass vor dem Kompilieren entschieden werden muss, welches Modul zum Einsatz kommen soll. Eine weitere Abstraktionsebene über welche der Wechsel der beschriebenen Mo-

dule stattfinden könnte, ist aktuell nicht vorhanden, wäre aber als Erweiterung denkbar.

5.7.1. Der SchoolGradeManager

In Kapitel 4.4 wird beschrieben, wie die Sicherheit der zu teilenden Noten durch die gewählte Formatierung noch sicherer gemacht werden kann. *Qt* bietet bereits mit den Klassen *QTableView* und *QAbstractItemModel* fertige Methoden zur Darstellung von Tabellen unter Einhaltung des MVC-Pattern. Mit *SQLite* bietet sich ein dazu passendes Konzept zur Verwaltung von Tabellen. Aus diesem Grund werden in der Implementierung beide Verfahren angewendet.

Die Klasse *SchoolTable* verwaltet ein Objekt von *Qt's QSqlTableModel*, einer von *QAbstractItemModel* erbenden und speziell für SQL-Datenbanken entwickelten Klasse, und bietet über die Methode *SchoolTable::insertRows(int)* (protected) das gezielte Einfügen von neuen Zeilen. Von dieser Oberklasse erben *SchoolMainTable*, welche die Schülerliste verwaltet, und *SchoolSubjectTable*, welche jeweils ein zur *SchoolMainTable* gehörendes Fach darstellt.

Zusätzlich zu dem Funktionsumfang der *SchoolTable*, sendet die *SchoolMainTable* ein Signal aus, sobald über den öffentlichen Slot *sloAddRow()* eine neue Schülerin bzw. ein neuer Schüler hinzugefügt wurde. Über dieses Signal werden alle zugehörigen *SchoolSubjectTables*, welche über ihren Konstruktor von der zugehörigen *SchoolMainTable* wissen, über die Änderung informiert und können dementsprechend ihre Zeilenanzahl aktualisieren. Um zu verhindern, dass eine *SchoolSubjectTable* von dieser Änderung nichts erfährt, da z. B. der Nutzer, der die Änderung an der *SchoolMainTable* durchführt keine Lese- bzw. Schreibrechte für die *SchoolSubjectTable* hat, wird bei jeder Initialisierung der *SchoolSubjectTable* auf Änderungen geprüft. Die jeder Schülerin und jedem Schüler eindeutig zugewiesenen ID stellt dabei sicher, dass sämtliche Einträge der Einzeltabellen einander zugeordnet werden können (mehr in Kapitel 4.4). Über einen weiteren Slot in der *SchoolSubjectTable* können ferner neue Spalten angelegt werden, um z. B. Klausurnoten oder andere Leistungskontrollen pro Schüler zu speichern.

Da SQLite auf Dateibasis arbeitet und für den vollen Funktionsumfang sämtliche Einträge entschlüsselt sein müssen, werden während der Bearbeitung einer Klasse alle zugehörigen Datenbanken temporär im Klartext auf der Festplatte abgelegt. Zwar existiert auch ein Plugin [SQL], mit welchem SQLite Anfragen an eine entsprechend verschlüsselte Datei stellen kann, doch ist dieses nicht standardmäßig in Qt enthalten und über eine spezielle Lizenz geschützt. Weil der temporäre Ordner nicht dem Cloud-Ordner entspricht und somit nur auf dem lokalen System existiert und weiterhin die Daten zur Anzeige sowieso im Klartext vorliegen müssen, soll hier aufgrund der nicht erhöhten Sicherheitsrisiken auf den Mehraufwand verzichtet werden. Eine *CleanUp*-Methode des *SchoolGradeManager*, welche außer bei einem Programmabsturz immer aufgerufen wird, sobald eine Schulklasse oder das Programm geschlossen wird, stellt zusätzlich sicher, dass temporär entschlüsselte Dateien so schnell wie möglich von der Festplatte gelöscht werden. Das koordinierte Laden und Speichern der verschiedenen Objekte übernimmt schließlich der *SchoolGradeManager* als direkte Kindklasse des *MainManagers*.

5.7.2. Der DataManager

Der *DataManager* ist im Gegensatz zum *SchoolGradeManager* vergleichsweise einfach aufgebaut. Auch er erbt vom *MainManager*. Objekte sind hier als eine Art Container für verschlüsselte Dateien zu betrachten. In einem Objekt bzw. Container kann sich jeweils eine Datei befinden. Über eine Objekt-Liste, in welcher alle Objekte stehen, die im Typ als extern gekennzeichnet wurden, hat der Nutzer die Möglichkeit, ein zu bearbeitendes Objekt auszuwählen. Ist dies geschehen, so kann er im *DataDialog* entweder das Objekt teilen (siehe Kapitel 5.6), den Inhalt aus einer Klartext-Datei importieren oder in eine Datei entschlüsselt exportieren. In dem Container wird

nur der Inhalt, nicht aber der Dateiname gespeichert. Um den Dateityp (also die Dateiendung) beim Exportieren zu kennen, ist es sinnvoll, diese im Objektnamen beim Anlegen des Objektes kenntlich zu machen. Ist in dem System für die eingegebene Endung des ausgewählten Dateinamens ein Default-Programm hinterlegt, so wird dieses automatisch nach dem Export mit der jeweiligen Datei gestartet.

Es empfiehlt sich aus Sicherheitsgründen, die exportierten Dateien im temporären Verzeichnis abzulegen, damit diese genau wie beim *SchoolGradeManager* nach Beenden des Programms automatisch entfernt werden. Dabei darf allerdings nicht vergessen werden, alle Änderungen nach dem Speichern in den Container zurück zu importieren und damit sicher aufzubewahren. Dem Nutzer bleibt es jedoch im aktuellen Programm überlassen, den Speicherort zu wählen.

6. Fazit

Das niedersächsische Kultusministerium legt in dem Runderlass [Nie] den Rahmen für eine digitale Notenverwaltung fest. Dort heißt es u. a., dass stets zu gewährleisten ist, dass nur die jeweilige Lehrkraft Zugriff auf verwaltete personenbezogene Daten hat. Auch durch R1 wurde eine solche Sicherheitsmaßnahme gefordert. Weiterhin sollte durch R3 ein gezieltes Teilen bestimmter Daten zwischen Lehrkräften über das Internet ermöglicht werden. Hier schreibt das niedersächsische Recht eine Verschlüsselung sowohl beim Transport der Daten als auch bei der Speicherung auf Netzwerkressourcen vor.

In Kapitel 4 wurde dazu ein mögliches Konzept vorgestellt. Über eine Einteilung der Maßnahmen in die drei Ebenen „verschlüsselte Kommunikation mit einem entfernten Server“ (Verwaltungskanal), „Verschlüsselung der verwalteten Daten“ (Datenkanal) und „Notenverwaltung auf Basis verschlüsselter Daten“ (Datenkodierung) konnten die verschiedenen Anforderungen auf getrennte Schichten aufgeteilt werden. Zur Beachtung der Hardwareunabhängigkeit in R5 wurde davon ausgegangen, dass grundlegende Sicherheitsmerkmale wie der physische Schutz oder die Systemintegrität durch die Hard- bzw. Software des zugrundeliegenden Systems bereits vorhanden sind. Im Mittelpunkt des Konzeptes und der Implementierung stand es deshalb, die Authentizität einer anderen Instanz (Server oder Lehrkraft), die Datenintegrität und die Vertraulichkeit der Daten sicherzustellen. Durch R9 konnte dabei kein Vertrauensmodell mit dritter Vertrauensinstanz gewählt werden. Stattdessen wurde ein anderes Vorgehen mit der Möglichkeit entwickelt, die Authentizität des jeweiligen Kommunikationspartners zu prüfen und die Sperrung der dazu verwendeten asymmetrischen Schlüssel bei deren Kompromittierung zu veranlassen. Auf weitere durch eine vollständige PKI (siehe Kapitel 2.2.5) gebotenen Eigenschaften musste dabei verzichtet werden.

Durch den Verwaltungs- und den Datenkanal wurde schließlich die Integrität und Vertraulichkeit der Serverkommunikation zur Realisierung von R3 und die Sicherung der Dateninhalte (R1) erreicht. Die analysierten Produkte *MEGA*, *TeamDrive* und *Boxcryptor* zeigen jeweils ein relativ ähnliches Basiskonzept auf: jedes verschlüsselte Objekt, meist die Datei, bekommt einen eigenen, symmetrischen Schlüssel. Durch Anwendung dieses Prinzips konnte schließlich R1 erfüllt werden. Weiterhin bot sich dadurch die Grundlage, durch asymmetrische Verfahren, gezielt die Schlüssel und damit den Zugriff auf den Klartext an andere Lehrkräfte zu verteilen. Dies erfüllt R3. Durch die lokal verwalteten Schlüssel, für welche sich der Nutzer nur ein Nutzergeheimnis merken muss, damit das Client-Programm die Sicherheit der Daten gewährleisten kann (erfüllt R8 und R7 hinsichtlich der Sicherheit), bleibt die Kontrolle über den Klartextzugriff in der Hand der Lehrkraft (erfüllt R10). Allerdings wird R10 durch die Tatsache, dass eine geteilte Datei unbemerkt durch den Autor an Dritte weitergegeben werden kann, eingeschränkt. Die Lehrkraft muss also dem anderen Nutzer vertrauen, dass dieser dies nur auf Rücksprache tut. Zu Beachten bleibt ferner, dass das Nutzergeheimnis besonders zu schützen ist (siehe Abb. 4.11).

Mit der Beispielimplementierung und dem dafür notwendigen Verzicht auf einen speziell konfigurierten Server (siehe 5.1) konnte schließlich auch praktisch gezeigt werden, dass das gewählte Sicherheitskonzept trotz eines geforderten Verteilungsmechanismus' (R3) nicht von einem bestimmten Server abhängt. Durch das *Qt*-Framework und der *QCA*-Bibliothek für kryptografische Funktionen wurde auch hier zum Großteil eine Unabhängigkeit zur Hardware (R5) erreicht. Mit *QCA* war ferner eine Umsetzung der Empfehlungen vom Bundesamt für Sicherheit in der Informationstechnik (BSI) möglich. Die Wahl eines Nutzerpasswortes als Geheimnis, um auf

alle benötigten Nutzerschlüssel zugreifen zu können, ermöglicht es, die Implementierung ohne Zusatzhardware wie Kartenlesegeräte nutzen zu können. Schließlich bietet die Klasse *DataManager* eine Methode, die verwalteten Daten als verschlüsselten Container zu nutzen, um Dateien anderer Programme importieren zu können. Dadurch wird die Erfüllbarkeit von R11 beispielhaft gezeigt.

Durch den Fokus auf die Sicherheitsaspekte kann das entworfene System somit die laut Rechtslage notwendige Zugriffskontrolle auf den Klartext und die Verschlüsselung für Transport und Speicherung im Internet bieten. Die in R2 geforderte Einhaltung der Datenschutzgesetze ist folglich gegeben. Gleichwohl kann das System nicht sämtliche Punkte des Runderlasses [Nie] erfüllen. Insbesondere bei der Einhaltung des Datenrahmens sowie der Vorgabe, über welche Schülerinnen und Schüler personenbezogene Daten auf dem privaten IT-System gespeichert und eingesehen werden dürfen, ist durch die gewählte Datenkodierung in Kapitel 4.4 nur eine Unterstützung gegeben. Eine Verhinderung (R4) kann deshalb nicht realisiert werden – auch weil die letztendliche Entscheidung zur Nutzung eines solchen Systems bei der Lehrkraft bzw. der Vorgabe durch die Schule liegt.

Insgesamt konnte also eine sichere Grundlage zur Notenverwaltung entwickelt und damit das gesuchte System zur Unterstützung der Lehrkraft gefunden werden. Auf die Forderung nach einer bekannten Eingabemaske (R6) wird anschließend im Ausblick eingegangen.

7. Ausblick

Durch den Sicherheitsfokus musste in der Arbeit der für den Nutzer sichtbare Teil des Programms, also die GUI, vernachlässigt werden. Für die Marktreife des Verwaltungssystems wird es somit notwendig sein, über Nutzerstudien zu prüfen, wie eine minimale Interaktion mit dem Programm (R7) und eine bekannte Eingabemaske (R6) erreicht werden kann. Weiterhin muss das Funktionsangebot ausgebaut werden. So wurde mit R30 die Forderung nach einer Nachverfolgbarkeit gestellt, wem welche Änderungen zuzuordnen sind. Auch wenn in dem Konzept davon ausgegangen wurde, dass nur der jeweilige Inhaber einer Datei Änderungen durchführen darf, könnte langfristig ein gemeinsames Verwalten von Dateien von Vorteil sein. Zur Realisierung dessen müsste also z. B. zu jeder Datei ein Änderungsprotokoll mit signierten Eintragungen und Hashes über den Dateizustand gespeichert werden. Stimmt der vom Nutzer berechnete Hashwert nicht mit dem letzten Hashwert überein, kann dadurch eine nicht protokollierte Änderung festgestellt werden. Weiterhin fordert R31, die Notenverwaltung für die gymnasiale Oberstufe erweitern zu können. Insgesamt kann durch diese und ähnliche Zusatzfunktionen die Praktikabilität und damit die Akzeptanz eines solchen digitalen Systems langfristig erhöht werden. Wichtig ist dabei der Bezug zu den Nutzern, also den Lehrkräften. Auch für die in Kapitel 2.4.2 angesprochen Problematik der *Nicht-Nutzung* von Kryptografie könnte dies hilfreich sein.

Zu der Optimierung der Benutzerfreundlichkeit gehört ferner die Handhabung großer Dateien. Momentan wird in der Beispielimplementierung zur Ver- und Entschlüsselung jeweils die gesamte Datei geschrieben bzw. gelesen, was zu Lasten des Arbeitsspeichers geht. Durch ein geeignetes Dateikonzept und die weitere Nutzung der Blockkodierung müsste diese Problematik noch behoben werden. Auch eine Reduzierung der Ergebnisse einer *Verkehrsflussanalyse* scheint sinnvoll (siehe Kapitel 4.6). Weiterhin sei darauf hingewiesen, dass ein Vergessen des Passwortes den Verlust sämtlicher Daten zur Folge hat. Hier sollte mit Hinblick auf die Praktikabilität eine Notlösung entwickelt werden.

Sowohl im Konzept als auch in der Implementierung wurden weder das Löschen von Daten noch Ändern von Passwörtern oder Schlüssel beachtet. Durch entsprechende Funktionen sollten solche Optionen dem Nutzer geboten werden – vor allem bei der Sperrung von asymmetrischen Schlüsselpaaren ist dies notwendig. Eventuell könnte eine regelmäßige, automatische Änderung symmetrischer Schlüssel oder Aufforderungen zur Änderung des Nutzergeheimnisses zusätzliche, langfristige Sicherheit bieten.

Dank des modularen Aufbaus sowohl im Konzept als auch in der Implementierung ist jegliche Erweiterung, welche nicht das grundlegende Sicherheitskonzept ändert, möglich. Ferner muss geprüft werden, ob nicht für den langfristigen Einsatz eine PKI mit dritter Vertrauensinstanz sinnvoll ist, um die vollen Sicherheitseigenschaften von asymmetrischen Schlüsseln gewährleisten zu können. Auch könnte dadurch der administrative Aufwand (Austausch der öffentlichen Schlüssel per Hand) minimiert und sogar automatisiert werden. Dazu ist allerdings eine Einschränkung von R9 und R10 notwendig.

Die Implementierung ist ohne Prüfung und Betrachtung lizenz- oder exportrechtlicher Belange durchgeführt worden. Bei einer weiteren Nutzung oder Umsetzung ist dies nachzuholen. Hierzu sei auf die Seiten des Bundesamt für Wirtschaft und Ausfuhrkontrolle[Aus] verwiesen.

8. Literaturverzeichnis

[Aus] Bundesamt für Wirtschaft und Ausfuhrkontrolle. *Ausfuhrkontrolle*. URL: http://www.ausfuhrkontrolle.info/ausfuhrkontrolle/de/ (besucht am 07. 08. 2015).

[BHO13] Andreas Bethke und Stephan Hansen-Oest. *Technisches und rechtliches Rezertifizierungs-Gutachten – Kurzgutachten –. Einhaltung datenschutzrechtlicher Anforderungen durch das Produkt "TeamDrive 3"*. Okt. 2013. URL: http://archiv.teamdrive.net/static/pdf/de/TeamDrive-Rezertifizierung-Kurzgutachten-2013.pdf (besucht am 29. 06. 2015).

[Bö13] Hanno Böck. *Eine Sicherheitslücke – oder auch nicht*. 2013. URL: http://www.golem.de/news/mega-eine-sicherheitsluecke-oder-auch-nicht-1309-101379.html (besucht am 29. 06. 2015).

[Com] Qt Company. *Qt – Home*. URL: http://www.qt.io/ (besucht am 11. 07. 2015).

[CS] Inc Cisco Systems. *Network Topology Icons*. URL: http://www.cisco.com/web/about/ac50/ac47/2.html (besucht am 11. 07. 2015).

[DN02] Der Landesbeauftragte für den Datenschutz Niedersachsen. *Niedersächsiches Datenschutzgesetz (NDSG)*. Jan. 2002. URL: http://www.lfd.niedersachsen.de/download/32372 (besucht am 05. 07. 2015).

[Dot13] Kim Dotcom. *Tweet: #Mega's open source encryption remains unbroken! We'll offer 10,000 EURO to anyone who can break it. Expect a blog post today.* Jan. 2013. URL: https://twitter.com/kimdotcom/status/297173196166295554 (besucht am 19. 07. 2015).

[Eck14] Claudia Eckert. *IT-Sicherheit: Konzepte – Verfahren – Protokolle*. 9. Auflage. Oldenbourg Wissenschaftsverlag, 2014.

[Gmba] Gruber & Petters GmbH. *WebUntis*. URL: http://www.grupet.at/HTML/produkteUntis.php?p=webuntis (besucht am 29. 06. 2015).

[Gmbb] MobileSitter GmbH. *MobileSitter. Sicheres Passwortmanagement für Smartphones*. URL: http://www.mobilesitter.de/index_de.php (besucht am 29. 06. 2015).

[Gmbc] MobileSitter GmbH. *Sicherheit von MobileSitter*. URL: http://www.mobilesitter.de/android/security_de.php (besucht am 29. 06. 2015).

[Gmbd] MobileSitter GmbH. *Sicherheit von MobileSitter*. URL: http://www.mobilesitter.de/android/faq_de.php (besucht am 29. 06. 2015).

[Gmbe] Secomba GmbH. *Über Boxcryptor*. URL: https://www.boxcryptor.com/de/boxcryptor (besucht am 29. 06. 2015).

[Gmbf] Secomba GmbH. *Die wichtigsten technischen Fragen zu Boxcryptor*. URL: https://www.boxcryptor.com/de/technischer-%C3%BCberblick (besucht am 29. 06. 2015).

[Gmbg] Teamdrive Systems GmbH. *Datensicherheit und Datenschutz haben für TeamDrive oberste Priorität*. URL: http://teamdrive.com/de/zertifizierung/ (besucht am 29. 06. 2015).

[Gmbh] Teamdrive Systems GmbH. *TeamDrive FAQ*. URL: http://teamdrive.com/de/faq-2/ (besucht am 29. 06. 2015).

[Infa] Bundesamt für Sicherheit in der Informationstechnik. *Basisschutz für den Computer*. URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/MeinPC/BasisschutzComputer/basisschutzComputer_node.html (besucht am 06. 07. 2015).

[Infb] Bundesamt für Sicherheit in der Informationstechnik. *Zwölf Maßnahmen zur Absicherung gegen Angriffe aus dem Internet*. URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/Wissenswertes_Hilfreiches/Service/Checklisten/Massnahmen_gegen_Internetangriffe.html (besucht am 06. 07. 2015).

[Inf12] Bundesamt für Sicherheit in der Informationstechnik. *Sicherheitsempfehlung für Cloud Computing Anbieter: Mindestanforderungen in der Informationssicherheit*. BSI-Bro12/314. Feb. 2012. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Mindestanforderungen/Eckpunktepapier-Sicherheitsempfehlungen-CloudComputing-Anbieter.pdf?__blob=publicationFile (besucht am 29. 06. 2015).

[Inf15] Bundesamt für Sicherheit in der Informationstechnik. *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Version 2015-01. BSI TR-02102-1. Feb. 2015. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile (besucht am 29. 06. 2015).

[MEGa] MEGA. *Developers – Documentation*. URL: https://mega.nz/#doc (besucht am 29. 06. 2015).

[MEGb] MEGA. *MEGA. THE PRIVACY COMPANY*. URL: https://mega.nz/#info (besucht am 29. 06. 2015).

[Nie] Kultusministerium Niedersachsen. *Verarbeitung personenbezogener Daten auf privaten Informationstechnischen Systemen (IT-Systemen) von Lehrkräften*. URL: http://www.voris.de/jportal/portal/t/5jf/page/bsvorisprod.psml?doc.hl=1&doc.id=VVND-VVND000031214&documentnumber=1&numberofresults=4&showdoccase=1&doc.part=F&paramfromHL=true#focuspoint (besucht am 29. 06. 2015).

[Nis] *An introduction to Computer Security: The NIST Handbook*. National Institute of Standards und Technology, 1995.

[Pro14] Delta Project. *Qt Cryptographic Architecture (QCA)*. Nov. 2014. URL: http://delta.affinix.com/qca/ (besucht am 03. 07. 2015).

[Pro15] Delta Project. *Qt Cryptographic Architecture*. Juni 2015. URL: http://delta.affinix.com/docs/qca/index.html.

[Sch04] Bruce Schneier. *Secrets & Lies: IT-Sicherheit in einer vernetzten Welt*. dpunkt.verlag GmbH, 2004.

[Sch06] Bruce Schneier. *Angwandte Kryptographie: Der Klassiker*. Pearson Studium, 2006.

[Sch13a] Klaus Schmeh. *Kryptografie: Verfahren, Protokolle, Infrastrukturen*. 5. Auflage. dpunkt.verlag GmbH, 2013.

[Sch13b] Jürgen Schmidt. *Mega-Facts*. Jan. 2013. URL: http://heise.de/-1789018 (besucht am 29. 06. 2015).

[SE] OpenSSL SE. *About the OpenSSL Project*. URL: https://www.openssl.org/about/ (besucht am 03. 07. 2015).

[Son13] Heike Sonnberger. *Schul-Klischees im Faktencheck: Lehrer haben es leicht – oder doch nicht?* März 2013. URL: http://www.spiegel.de/schulspiegel/wissen/faktencheck-wie-viel-arbeiten-lehrer-und-wie-viel-freizeit-haben-sie-a-874089.html (besucht am 17. 07. 2015).

[SQL] SQLite. *SQLite Encryption Extension*. URL: https://www.sqlite.org/see/doc/trunk/www/readme.wiki (besucht am 13. 07. 2015).

[Sta14] William Stallings. *Cryptography and network security: principles and practice*. 6. Auflage. Pearson Education Limited, 2014.

[WFS12] Ruben Wolf, Frederik Franke und Markus Schneider. *iMobileSitter: Sicheres Passwortmanagement in Zeiten von digitalen Schlüsseldiensten und Advanced Per-*

sistent Threats. Juni 2012. URL: http://www.mobilesitter.de/downloads/mobilesitter-kes-special-mobile-security.pdf (besucht am 29. 06. 2015).

A. Anhang

A.1. Einführendes Interview

Am 8. April 2015 wurde ein Interview mit einer Lehrkraft durchgeführt, um eine Einschätzung der Probleme bei der momentanen Durchführung der Verwaltungsaufgaben zu bekommen. Aus Datenschutzgründen wird der Name der beteiligten Person nicht genannt. Stattdessen werden die Antworten mit dem Buchstaben L versehen, der Interviewer war Alexander Röttcher (A. R.). Nachfolgend werden sowohl die Fragen als auch die dazu getroffenen Antwort der interviewten Person sinngemäß zusammengefasst wiedergegeben.

L: Im Schulalltag ist man mit der Dokumentation des Leistungsstands der Schülerinnen und Schüler ganz schön im Stress. Mit der Art und Weise, wie ich das selber mache, bin ich nicht wirklich glücklich. Ich bin als Quereinsteiger Lehrer geworden und seit mittlerweile acht Jahren dabei. Auch habe ich das Referendariat durchlaufen. Also bin ich auf dem Stand, dass ich die Ausbildung habe und vernünftig arbeiten kann. Es existieren allerdings – wenn ich schaue wie ich selber arbeite oder auch die Kollegen arbeiten – immer noch einige Problempunkte. Wenn ich die Dokumentation betrachte, ist das ein Aufwand, der – wenn man es vernünftig machen will – schlicht nicht zu leisten ist. Ich habe, wenn ich Physik unterrichte, Versuche auf und ab zu bauen, die Tafel zu wischen und vieles mehr. Und dazu dann noch überlegen, welche Note jeder Schüler bekommt? Das funktioniert so nicht.
Für mich stellt sich also die Frage, wie man diese Problematik angehen kann. Was ganz praktisch ist, ist wenn man einen Kalender hat, in welchem die Namen der Schüler stehen. Dort kann man dann z. B. die jeweilige Tendenznote notieren. Also + oder - oder Kreise – je nachdem, wie die momentane Einschätzung ist. Das ist aber noch vergleichsweise aufwändig. Was sich als vorteilhaft für mich erwiesen hat ist, wenn man die Extreme festhält. Wenn man also aufschreibt, dass ein Schüler eine besonders gute Leistung erbringt oder eben eine besonders schlechte Leistung. Alles dazwischen ist im Bereich des Neutralen. Das ist praktikabel und würde schon reichen: aus den Tendenzen ergibt sich dann die Endnote, je nachdem ob noch besondere Leistungen in Form von Referat oder Hausaufgabenausarbeitung oder Tafelvortrag oder Anderem dazu kommt.
Aber sich für jeden Schüler genau zu überlegen (am Besten noch ein paar Sätze notieren), wie der Leistungsstand ist, funktioniert in der Realität nicht. Ich bin dann mit der Dokumentation auch so gefangen, dass ich gar nicht mit den Gedanken bei dem Lernweg der Schüler bin. Und das kann nicht Sinn der Sache sein.
Jetzt stellte sich mir die Frage, wie man so etwas in ein System bekommt, welches überschaubar und praktikabel ist. Aufgrund der inzwischen weiten Verbreitung von Computern und Tablets dachte ich an ein digitales System. Ich bin ein Lehrer mit einer gewissen Affinität für Technik – das ist aber durchaus nicht so verbreitet. Die Akzeptanz eines neuen Systems steht und fällt damit, dass man es hinkriegt, es den Lehrerinnen und Lehrern so schmackhaft zu machen, indem es an die gewohnten Vorgehensweisen anknüpft und vielleicht auch etwas verbessert. Man muss ich das wie folgt vorstellen: Es gibt bereits bestehende Strukturen und so unhandlich die auch sind, die Leute haben sich daran gewöhnt. Wenn ich nun den Vorschlag mache, die gesamte Notenverwaltung auf dem Rechner durchzuführen, und das funktioniert einmal nicht, muss ich dafür gerade stehen. Auch wenn das ganz normal ist und allen Beteiligten bekannt ist, wie nach-

teilig und fehlerhaft das alte System ist. Alle sind eben an dieses System gewöhnt und deswegen ist es schwierig, etwas neues einzuflechten. Eine gewisse Durststrecke muss da eingeplant werden.

A. R.: Stehen Sie denn persönlich einer digitalen Hilfestellung bei den genannten Problemen ohne Vorbehalte gegenüber? Wie sehen das Ihre Kollegen?

L: Es gibt ja schon kommerzielle Software in diese Richtung. Trotzdem ist es etwas, was im Kollegium und bei mir selbst gewisse Vorbehalte hat, weil es nicht durchschaubar ist, was im Hintergrund abläuft. Beispielsweise das Löschen: wenn Sie ein RAID-System haben, auf welchem die Daten gespiegelt werden, genügt es nicht, wenn man das Original löscht. Dazu kommt, dass auch die Backups dann natürlich von diesen Servern gelöscht werden müssten. Eine Kontrolle dessen ist als Nutzer nur schwer bzw. gar nicht möglich.
Bezüglich der Akzeptanz müsste man auch Überzeugungsarbeit leisten: was wäre z. B., wenn ich die Notizen auf dem Smartphone habe und ich verliere das? Was passiert dann? Oder der Akku ist alle? Momentan habe ich meine schriftlichen Notizen und wenn die Weg sind, habe ich natürlich auch ein Problem. Mit dem Kalender ist das also auch nicht das Beste. Wenn ich dann dank einer Software Backups habe und das Geräte geht verloren, dann kann ich mir ein neues Gerät anschaffen und mit den alten Daten weiterarbeiten. Sollten die Daten zusätzlich mit einem Verschlüsselungssystem sicher verschlüsselt sein, dann kann auch niemand anderes damit etwas anfangen. Das beruhigt natürlich.

A. R.: Stellen Sie sich vor, Sie und Ihre Kollegen würden ihre Notizen und Einschätzungen zu den Schülerinnen und Schülern digital führen. Meistens gibt es dann einen Administrator, der die Überwachung und Koordinierung der technischen Infrastruktur übernimmt. Können Sie sich z. B. vorstellen, dass ein solcher Administrator auch die Steuerung der Bearbeitungs- und Zugriffsrechte auf Dateien durchführt? Oder sollte dies lieber dezentral durch jeden Nutzer selbst gemacht werden?

L: Das ist eine sehr gute Frage. Ehrlich gesagt bin ich da noch nicht festgelegt. Ich würde auch sagen, dass man sich wirklich gut überlegen muss, was eine praktikable und sinnvolle Lösung ist. Also z. B. beim Klassenlehrer: dass der Einblick in alle Noten hat, macht absolut Sinn. Der unterrichtet aber nicht nur in der Klasse sondern auch in anderen Klassen. Dies muss irgendwo einstellbar sein. Mit einer geeigneten Oberfläche, sodass man auch nicht den Überblick verliert.

A. R.: Die Idee hinter der Frage war, dass ein Administrator wahrscheinlich auch zur Wahrnehmung seiner Aufgaben Einsicht in die Daten hätte oder bekommen könnte. Dies hätte zur Folge, dass ein Nutzer das Gefühl haben könnte, im Gegensatz zur schriftlichen Lösung beobachtbarer zu sein. Durch die Verteilung der Steuerung könnte ein Lehrer die Kontrolle über seine Daten behalten.

L: Finde ich absolut logisch. Es bleibt nur die Frage, ob die Lehrer eine Abneigung gegenüber der daraus resultierenden Verwaltung – wer Zugriff auf welche Daten hat – haben. Ich denke aber, dass das kein Problem ist, solange es handhabbar und sicher bleibt. Eigentlich muss man doch nur einen Haken setzen: Zugriff erlauben oder nicht erlauben. Das ist im Grunde überschaubar, denke ich. Man muss allerdings mit Widerstand einiger Kollegen rechnen.

A. R.: Für diese könnte das ja im Zweifelsfall ein Administrator übernehmen...

L: Das wäre eine Möglichkeit. Nochmal zu der notwendigen Überzeugungsarbeit: es ist

wichtig, wenn man eine Änderung anstrebt, dass man irgendwas in der Hand hat, womit man den Lehrkräften sofort Vorteile aufzeigen kann. Zum Beispiel die Datensicherheit: wenn ich mein Gerät verloren oder nicht dabei habe, kann ich trotzdem ganz normal arbeiten. Das ist, wie ich finde, ein sehr starkes Argument. Man sollte dazu auch nicht aus den Augen verlieren, dass die Organisation in der Oberstufe durchaus sehr verschieden von der Mittelstufe ist. Da haben wir keine Klassen mehr, dort gibt es Kurse und außerdem noch den Tutor, der über seine Tutanden mehr oder weniger einen Überblick hat. Aber das Konzept Klassenlehrer gibt es dann nicht mehr. Alternativ könnte man in diesem Fall eine zentrale Verwaltung dem Oberstufenkoordinator nahe legen.

A. R.: Wir sprechen hier zum Einen vom Begriff Sicherheit, zum Anderen von der Benutzerfreundlichkeit. Wenn Sie jetzt beide Faktoren abwägen müssten, was würde überwiegen?

L: Meine persönliche Meinung ist ganz klar: Sicherheit auf jeden Fall. Ich fürchte aber um die Akzeptanz. Bei uns hat jetzt an der Schule allerdings ein erheblicher Wechsel stattgefunden. Also es sind dieses Schulhalbjahr 40% neue Lehrer dazu gekommen. Damit sind sehr viele junge Kollegen da, die beweglicher sind und bestimmt nicht so festgelegt. Dadurch erscheint eine Änderung des Systems immer machbarer.

A. R.: Ich könnte mir vorstellen, innerhalb meiner Arbeit ein Konzept zu entwerfen, wie man möglichst sicher personenbezogene Daten der Schülerinnen und Schüler auf dem privaten IT-System verwalten kann. Allerdings wird es vom Umfang her wohl zeitlich nicht realisierbar sein, Ihnen am Ende ein fertiges Programm präsentieren zu können, welches alle beschriebenen Probleme löst. Eine Prinzipimplementierung könnte aber denkbar sein. Hätten Sie noch einen letzten Wunsch bezüglich des Inhalts meiner Arbeit?

L: Von dem Ergebnis insgesamt würde ich mir für mein Gefühl mehr versprechen, wenn man an etwas Bestehendes anknüpft. Da kann man wahrscheinlich größere Schritte machen, als wenn man von Grund auf loslegen muss. Aber ich kann das wirklich nicht einschätzen. Ich kann mir auch vorstellen, dass man quasi für bestehende Programme einen sicheren Unterbau liefert, der dafür sorgt, dass die digitale Verarbeitung datenschutztechnisch in Ordnung ist. Die meisten Lehrer kennen sich mit Excel oder Excelderivaten aus.
Aber wissen Sie, wenn ich mir das heutzutage so angucke: Ich habe vor acht Jahren in der Industrie mit CRM-Systemen (Customer-Relationship-Management) gearbeitet und was da möglich ist und gemacht wurde. Und an der Schule macht dann jeder Lehrer mit so einer Exceltabelle sein eigenes Süppchen. Also mir fällt da gar nichts mehr zu ein. Vielleicht wäre ja auch ein CRM-System für eine komplette Schule, mit allen Verwaltungsaufgaben ideal. Das wäre dann aber wohl ein Projekt für die "Schule der Zukunft". Insgesamt benötigt es wohl viel mehr Druck von Oben, bis sich überhaupt etwas bewegen wird.

A. R.: Vielen Dank für das Interview.

A.2. Reaktion auf erste Konzeptpräsentation

Während der Präsentation am 29.04.2015 eines ersten Konzeptentwurfs, aus dem letztendlich das Konzept in Kapitel 4 entstand, wurden einige Bewertungen und Wünsche für eine darauf basierende Beispielimplementierung genannt. Anwesend waren vier Lehrkräfte, deren sinngemäße Aussagen aufgrund des Datenschutzes im Folgenden den Buchstaben A bis D zugeordnet wurden.

Zu Beginn des Vortrags wurden die Anwesenden kurz nach den Wünschen und Vorstellungen bezüglich einer digitalen Notenverwaltung gefragt:

A: Uns stellt sich immer die Frage, wie wir untereinander zusammen arbeiten. Da geht es gar nicht so sehr darum, wie der Einzelne seine Noten erhebt, sondern, dass wir untereinander zusammenarbeiten müssen, um z. B. Versetzungsgefährdungen frühzeitig aussprechen zu können. Da müssen wir die Noten der anderen Kollegen wissen und das funktioniert jetzt hier zum Teil schon über Clouds: die Noten werden werden bei einem externen Anbieter gespeichert werden, die Kollegen melden sich an und tragen die Leistungseinschätzungen der jeweiligen Schülerinnen und Schüler ein. Ich bin da sehr skeptisch, ob man bei solch einem Anbieter inoffiziell die Schülerdaten speichern sollte. Aber so weit ich weiß ist das mit unserem Schulserver, den wir für die Notenverwaltung einsetzen, nicht möglich, dass man von außerhalb darauf zugreifen kann. Das ist ein abgeschlossenes System.
B: VPN ist auch möglich. Wir haben nur keinen Zugriff, die Administratoren können das. Die technische Möglichkeit ist also da.
A: Das hat ja seinen Grund, dass es nicht nach Außen geöffnet wird für normale Lehrkräfte. Eine externe, inoffizielle Verarbeitung kann allerdings auch nicht die Lösung sein.
B: Vor allem handeln wir damit am Rande der Legalität.
A: Was man häufig hört von Kollegen, ist dann: wer will denn das wissen? Das interessiert doch gar keinen. Die Gefahren sind einfach zu abstrakt. Zu weit weg. Das Problem zwischen Verordnungen und Verboten und der Realität ist, dass sie in der Schule sehr weit auseinander gehen. Wahrscheinlich weiter auseinander als in anderen Betrieben. Es ist einfach so. Lehrer sind kleine Götter in ihrem Bereich und machen einfach erstmal.

A. R.: Es macht also Sinn, ein sicheres System anzubieten, allein um genau so etwas zu verhindern?

B: Also was wir bräuchten wäre im Grunde genommen eine Plattform, die direkt mit unserem Server im Haus kommuniziert und auf die ich mobil zugreifen kann. **C:** Und der Administrator [eine Lehrkraft] sagt dazu ganz klar: Nein, no way. Er möchte auf jeden Fall Noten vom Lehrer (also die täglichen) getrennt halten von den Zeugnisnoten. Auch möglichst durch verschiedene Hardware, sodass das wirklich verschiedene Welten sind.
B: Das ist ja auch machbar. Es wäre aber auch super, wenn wir ein Verwaltungssystem für den Schulalltag hätten auf dem wir einstellen könnten, dass alle Lehrer, die den jeweiligen Schüler unterrichten, auch Zugriff auf seine Noten haben. Wenn ich dann irgendwelche Noten ändere, dann sollten auch alle die Änderungen live mitverfolgen können – vielleicht sogar zurückverfolgen, wer die Änderungen gemacht hat.
C: Super wäre auch, wenn man die Fehlzeiten mal irgendwie verwalten würde. Dann sieht man automatisch, dann und dann war der nicht da. Da ist er entschuldigt, da ist er nicht entschuldigt gewesen. Also da könnte man sich schon Unterstützung erhoffen. Allgemein muss man aber natürlich auch sehen, ab wann das Überwachungspotential zu groß wird.
B: Stimmt, dann guckt man ja: „Mhm, Kollege so und so hat seine Noten immer noch nicht eingetragen."

Nach der Erläuterung des entwickelten Systems und der Klärung von Verständnisfragen kamen folgende Bedenken zum Ausdruck:

D: Der Aufwand ist zu groß für eine Umstellung an der Schule. Hier sind ja Strukturen – von der ganzen Schule – angelegt, die abgebildet sind in den bestehenden Systemen. Auch

wenn das alles andere als optimal läuft, wie ich finde. Wobei die Möglichkeit des Überschreibens der Daten von Kollegen hier ein Problem sein könnte. Wenn ich wollte, könnte ich innerhalb von einer halben Stunde die ganze Zeugniskonferenz durcheinander bringen, weil ich immer schnell irgendwo ein paar Noten ändere. Da ist überhaupt keine Hürde. Und ich glaube, dass dadurch auch einige Fehler bei uns passieren. Das hält sich allerdings noch im Rahmen. Und dann die Kollegenwelt hier, die sich mit Mühe und Not an das jetzige System gewöhnt hat. Das Ergebnis müsste also genügend Verbesserungen vorweisen, um überhaupt zur Debatte stehen zu können.

A.R.: Besteht die Skepsis also eher aufgrund einer vermuteten Mehrarbeit durch die Umstellung als wegen einer generellen Skepsis gegenüber Computern?

D: Ich denke, da ist beides drin. Die Mehrarbeit, sich auf etwas neues einzustellen, ist sowieso ein Problem. Unterstützend könnte es dabei sein, wenn ich mit meiner privaten Notenverwaltung, mit welcher ich die Leistungseinschätzungen bereits notiere, eine Anbindung an das Schulsystem hätte. Das ist bei uns eine problematische Sache, wie ich finde. Ob das mit dem Zeugniskalender verwaltet wird oder auch mit einer privaten, digitalen Verwaltung ist ja momentan egal. Wir müssen es am Ende per Hand in die schulinterne Notenverwaltung eingeben. Das nervt höllisch und ist auch wirklich ein Punkt, der nicht mehr zeitgemäß ist. Aber so läuft es momentan. Auch wenn dies viele Kollegen stört, so ist der Aufwand anscheinend noch nicht groß genug, dass die überwiegende Mehrheit eine Änderung fordert.
A: Der jetzige Zustand ist ja, dass wahrscheinlich viele Kollegen am Erlass [Nie] vorbei ihre Daten speichern und verwalten. Genau das wird dann auch noch bagatellisiert: „wer will denn diese Daten sehen?“. Grund dafür ist wohl Bequemlichkeit. Aber wenn man sie dafür sensibilisiert, dass es sich, so wie es momentan läuft, am Rand der Legalität befindet, dann muss man ja im Prinzip was machen. Diese Technikskepsis, die ich auch feststelle, stirbt mit der Zeit langsam aus. Seit ich an dieser Schule bin hat sich das Kollegium fast zur Hälfte verjüngt.
C: Vielleicht bräuchte es auch einfach den Druck von Oben.
A: Einfach erstmal etwas anbieten für die Kollegen, die das nutzen wollen. Nicht gleich verpflichtend machen. Wie gesagt, ich habe von einigen Leuten in diesem Halbjahr die Aufforderung bekommen, meine Noten in einem in die Cloud hochgeladenen Dokument einzutragen. Wenn man dann aber ein sicheres System als Angebot hätte, dann würde es auch genutzt werden. Natürlich nur, wenn es auch entsprechend kommuniziert wurde: „das ist sicher, das könnt ihr dafür nehmen“. Man muss ja nicht immer gleich alles vorschreiben. Man kann ja erstmal ein Angebot machen, das ist da und dann wird sich das von alleine schon durchsetzen.
C: Ganz große Anforderung bei nicht technikaffinen Lehrern ist der Bedienkomfort. Man muss bei einer Neueinführung sehr stark auf der Hut sein, dass sich das langfristig trägt. Das System muss gewährleisten, dass man sich um die ganzen Schlüssel im Hintergrund als Nutzer nicht kümmern muss. Einfach nur ein paar Passwörter oder Fragen gestellt bekommt und dann durchgeführt wird. Eine Verschlüsselung sollte out-of-the-box- laufen.
D: Ich finde, das bringt noch gar nichts, wenn am Ende kein fertiges Produkt entsteht. Solange brauchen wir sowieso nichts den Kollegen vorstellen. Alles andere wird als Zeitverschwendung und „jetzt kommen die da wieder mit ihren fixen Ideen an“ abgetan. Bis zur Umsetzung eines neuen Produktes gehen Jahre ins Land. Das ist eigentlich immer so. Wenn man mit so einer Idee kommt, dann sollte man sich am Besten jetzt eine Gruppe zusammen suchen, die das interessant findet. Diese kann man dann vergrößern und dann, wenn das Produkt marktreif ist, die Arbeit leisten, das in die Schule zu tragen.

A.R.: Es macht also durchaus Sinn, auch wenn es nur eine Beispielimplementierung ist,

darauf zu achten, dass so ein Programm einzeln nutzbar ist? Quasi als optionale Hilfe oder sogar Ersatz zu der momentanen schriftlichen Methode. Und gleichzeitig eine mögliche Erweiterbarkeit im Auge zu behalten, damit, sollte die Idee einen Erfolg darstellen, eine flächendeckende Installation möglich ist.

C: Da würden Sie wahrscheinlich mehr Akzeptanz finden.

A.3. Anforderungen

A.3.1. Basisanforderungen

Folgende grundlegende Anforderungen sind aus dem Interview (Anhang A.1) und der Präsentation (Anhang A.2) abgeleitet worden:

[**R1**] Das System muss der Lehrkraft die Möglichkeit bieten, die digital verwalteten personenbezogenen Daten der Schülerinnen und Schüler für den Zeitraum der Speicherung gegen den Zugriff Dritter (Angreifer) zu schützen.

[**R2**] Das System muss die geltenden Datenschutzgesetze einhalten.

[**R3**] Das System muss der Lehrkraft die Möglichkeit bieten, die über das Internet mit anderen Lehrkräften zu teilenden personenbezogenen Daten der Schülerinnen und Schüler gegen den Zugriff Dritter (Angreifer) zu schützen.

[**R4**] Das System muss eine Verletzung der Datenschutzgesetze und damit der Sicherheit der personenbezogenen Daten der Schülerinnen und Schüler verhindern.

[**R5**] Das System muss fähig sein, auf vorhandener Hardware (Computer, Tablet, Smartphone) und Betriebssystemen (Windows, Linux, MacOs, Android, iOS) ausgeführt zu werden.

[**R6**] Das System muss der Lehrkraft – insbesondere der nicht mit Computern vertrauten – die Möglichkeit bieten, über gängige Eingabemasken (GUI-Standards, Standards in der handschriftlichen Verwaltung) die zu verwaltenden Daten eingeben zu können.

[**R7**] Das System muss die Interaktion mit der Lehrkraft zur Dateneingabe und Prozesssteuerung minimal halten.

[**R8**] Das System muss die notwendigen sicherheitstechnischen Prozesse soweit möglich im Hintergrund automatisieren.

[**R9**] Das System muss der Lehrkraft die Möglichkeit bieten, ohne Einfluss, Kontrolle oder Vertrauen auf andere Instanzen die Verwaltung der personenbezogenen Daten der Schülerinnen und Schüler durchführen zu können.

[**R10**] Das System muss der Lehrkraft die Möglichkeit bieten, zu jedem Zeitpunkt kontrollieren zu können, wem welche Daten im Klartext zur Verfügung gestellt werden.

[**R11**] Das System muss fähig sein, über eine Export- und Importfunktion auf Dateiebene das sichere Speichern und Übertragen der mit anderen Programmen auf demselben Client-Betriebssystem verwalteten personenbezogenen Daten zu ermöglichen.

A.3.2. Abgeleitete Anforderungen

Folgende Anforderungen sind aus den Basisanforderungen, den Sicherheitsgrundlagen und der Rechtslage abgeleitet worden:

[**R12**] Das System muss die über das Internet übertragenen und auf einer Netzwerkressource gespeicherten Daten verschlüsseln.

[**R13**] Das System muss für alle eingesetzten asymmetrischen Schlüssel die Authentizität, Möglichkeit des Sperrens, Unmöglichkeit des Abstreitens und Einhalten von Policies garantieren.

[**R14**] Das System muss bei sitzungsbasierter Kommunikation Backward und Forward Security garantieren.

[**R15**] Das System muss die Richtlinie [Inf15] des Bundesamts für Sicherheit in der Informationstechnik (BSI) bei der Verwendung kryptografischer Algorithmen einhalten.

[**R16**] Das System muss dem Entwickler die Möglichkeit bieten, verwendete Verfahren und Schlüssellängen ohne Änderung der darauf aufbauenden Algorithmen auszutauschen.

[**R17**] Das System muss bei asymmetrischer Verschlüsselung längere Schlüssel als vom BSI empfohlen verwenden.

[**R18**] Das System muss die Menge der über öffentliche Netzwerke übertragenen Geheimnisse minimal halten.

[**R19**] Das System muss bei der Speicherung und Übertragung von schützenswerten Daten die Sicherheitsaspekte Security und Safety, die CIA triad Datenintegrität, Systemintegrität, Verfügbarkeit, Datenvertraulichkeit und Privatsphäre sowie die Authentizität und Zurechenbarkeit garantieren.

[**R20**] Das System muss den Schutz vor den in Kapitel 2.4.2 genannten Real-World-Attacken garantieren.

[**R21**] Das System sollte, sofern sinnvoll, bereits existierende Verfahren der Kryptografie verwenden.

[**R22**] Das System muss bei den eingesetzten Protokollen zum Transport schützenswerter Daten den Schutz vor den in Kapitel 2.4.3 genannten Angriffsmöglichkeiten garantieren.

[**R23**] Das System muss den Nutzer beim Versuch, Daten zu Löschen oder anderen Nutzern freizugeben, durch eine offensichtliche Meldung warnen.

[**R24**] Das System muss bei einem Zugriff auf das Client-Gerät durch einen Angreifer, bei welchem dieser vor dem Gerät steht und per Hand Passwörter ausprobieren oder einzelne Dateien kopieren kann, die Sicherheit der Daten garantieren.

[**R25**] Das System muss bei einem Diebstahl des Client-Gerätes die Sicherheit der Daten garantieren.

A.3.3. Spezielle Anforderungen für die Beispielimplementierung

Folgende Anforderungen sind speziell für die Beispielimplementierung aus dem Interview (Anhang A.1) und der Präsentation (Anhang A.2) abgeleitet worden:

[**R26**] Das System muss fähig sein, die Endnoten der täglich eingegebenen Leistungseinschätzungen der Lehrkräfte mit einem physisch getrennten Notenverwaltungssystem am Ende des Schuljahres zu synchronisieren.

[**R27**] Das System muss der Lehrkraft die Möglichkeit bieten, Fehlzeiten der Schülerinnen und Schüler mit der Option entschuldigt / nicht entschuldigt einzutragen.

[**R28**] Das System muss unabhängig von einem Schulsystem von der Lehrkraft privat nutzbar sein.

[**R29**] Das System sollte zu einer zentralen, von allen Lehrkräften genutzten Verwaltungssoftware für die Schule erweiterbar sein.

[**R30**] Das System muss der Lehrkraft die Möglichkeit bieten, Änderungen anderer Lehrkräfte an den ihnen sichtbaren Daten mit zu verfolgen und der jeweiligen Lehrkraft eindeutig zuordnen zu können.

[**R31**] Das System muss der Lehrkraft die Möglichkeit bieten, auch in der gymnasialen Oberstufe – ohne ein Klassensystem – die Leistungsverwaltung vornehmen zu können.

A.4. Aussagen von Gruber&Petters GmbH

Die folgenden Fragen bezüglich des Produktes *WebUntis* wurden in zwei E-Mails vom 11.05.2015 und 28.05.2015 von Gruber&Petters beantwortet.

Frage 1: Wo kann ich auf Ihrer Website eine technische Spezifikation bezüglich der Sicherheit Ihres Systems finden? Mich würde dabei interessieren, mit welchen Verfahren bzw. Algorithmen Sie die zwischen den einzelnen Geräten geteilten Informationen (z. B. des digitalen Klassenbuches) gegen unbefugte Zugriffe absichern.
Antwort: Grundsätzlich haben wir eine Webapplikation und alle Daten einer Schule liegen zentral auf unseren Servern. Browser und/oder Apps sind im wesentlichen nur Clients, die die Daten anzeigen bzw. Eingaben entgegennehmen. Die Server sind nach dem Stand der Technik gegen unbefugte Zugriffe abgesichert. Rechtmäßige Benutzer identifizieren sich über Passwörter und die Befugnisse werden aus vergebenen Rechten und dem Stundenplan selbst abgeleitet. Wir haben das Konzept der "eigenen" Daten. Die Defintion von ëigene"hängt vom Kontext ab. Für einen Fachlehrer sind eigene Schüler die Schüler seines Unterrichtes, für einen Klassenlehrer die Schüler seiner Klasse. Bei Noten darf als etwa ein Fachlehrer die Noten seiner Unterrichtsschüler sehen, ein Klassenlehrer die Noten aller Schüler seiner Klasse.

Frage 2: Haben Sie auch eine Notenverwaltung in Ihrem System integriert? Laut Forum soll ein solches Modul vorhanden sein. Leider kann ich auf Ihrere Homepage allerdings keine detaillierteren Informationen bekommen.
Antwort: Ja, wir haben eine Notenverwaltung in WebUntis. Es geht hier primär um das Erfassen von Einzelnoten und die Ableitung einer Gesamtnote. Diese wird an Schülerverwaltungen übergeben, die dann Zeugnisse erstellt und die Gesamtnoten längerfristig verwaltet. Leider gibt es dazu noch nicht sehr viel Dokumentation. Im Handbuch zu WebUntis, das Sie auf unserer

Homepage herunterladen können, sollten Sie aber zumindest ein bißchen finden.

Frage 3: Ihre Antwort zu *Frage 1* geht schon in die richtige Richtung. Ich leite daraus folgende Vorgehensweise ab:

a) der Nutzer-PC baut eine sichere Verbindung zu Ihrem Server (Standort(e)?) auf. Nutzen Sie hierbei SSL oder ein anderes (z. B. eigenes) Verfahren? Mit welchen Parametern (Verschlüsselungs- und Hash-Algorithmen sowie Schlüssellängen)?

b) der Nutzer gibt seinen Nutzernamen und das Passwort ein. Die Daten werden über mden (z. B. durch SSL) geschützten Kanal zum Server übertragen. Wird das Passwort gehasht oder im Klartext übertragen?

c) der Server vergleicht das eingegebene Passwort mit dem in einer Datenbank zum Nutzernamen gehörenden Passwort. Stimmen diese überein, so generiert der Server eine sitzungsbasierte ID (SID – über diese werden die Nutzerrechte geprüft)

d) der Server überträgt die SID an den Nutzter-PC. Der Nutzer kann nun Eingaben tätigen. Bei jeder Anfrage an den Server wird die SID mitgesendet, sodass der Server prüfen kann, ob der Nutzer die nötigen Rechte hat.

e) meldet sich der Nutzer ab oder antwortet der Nutzer-PC eine gewisse Zeit lang nicht (wie lange?) wird die SID auf dem Server gelöscht und der Nutzer muss sich beim nächsten Mal erneut anmelden

Antwort:

a) Standort Wien, Interxion Rechenzentrum
SSL, 128 bit, SHA-1 With RSA Encryption

b) im Klartext, hashen ist nicht möglich, weil verschiedene Authentifizerungsmethoden möglich sind, z. B. LDAP und wir da das Passwort im Klartext brauchen. Außerdem ist es ohnehin durch SSL bereits verschlüsselt.

c) ja, entweder der WU interne Mechanismus oder ein externes LDAP oder SSO-System

d) ja

e) von der Schule konfigurierbar, max. 1 Stunde

Frage 4: Da Sie von einer Webapplikation sprechen, nehme ich an, dass die Kommunikation auf dem HTTP/HTTPS-Protokoll basiert? Die Clients sind also nichts anderes als (modifizierte) Browser?
Antwort: HTTPS, die Clients sind Standardbrowser

Frage 5: Werden alle übertragenen Eingaben nur durch den (z. B. durch SSL) geschützten Übertragungskanal geschützt? D.h. sämtliche Eingaben sind auf Ihrem Server im Klartext einsehbar? Oder werden die Daten vor dem Übertragen zusätzlich verschlüsselt, sodass Sie keineMöglichkeit haben, auf dem Server die Daten im Klartext einzusehen? Wie würde dann eine solche Datenverschlüsselung aussehen, wo werden die zugehörigen Schlüssel verwaltet?
Antwort: Neben der Übertragungsverschlüsselung erfolgt keine zusätzliche Verschlüsselung.
Frage 6: Werden die Daten auf Ihrem Server im Klartext oder verschlüsselt gespeichert?

Wie sind die zugehörigen Schlüssel gesichert? Angenommen, ein Fremder hat Zugriff auf sämtliche Dateien auf Ihren Servern. Könnte er dann sämtliche Daten aller Kunden einlesen und bearbeiten? Oder ist ein zusätzlicher Aufwand, wie z. B. das herausfinden bestimmter Schlüssel notwendig? Gilt dies auch für sämtliche Backups?
Antwort: Passwörter werden aufwändig verschlüsselt (Einwegverfahren) gespeichert, weil damit ein Angreifer in der Regel am meisten anfangen könnte (Stichwort Passwortwiederverwendung)
Zum Service gehören tägliche Backups, die mit PGP verschlüsselt gespeichert werden. Auf dem Server liegt nur der Public Key für die Verschlüsselung, der Private Key wird offline sicher verwahrt. Andere Daten werden derzeit auf dem Server nicht verschlüsselt gespeichert.

Frage 7: Wie wird mit der SID (falls Sie diese so wie oben beschrieben implementieren) festgestellt, dass die jeweilige Anfrage wirklich von dem Nutzer kommt und nicht durch einen Dritten, der die SID abgefangen hat? Sollten Sie ein anderes Verfahren (also keine SID) nutzen, wie sieht dieses dann aus? Wird z. B. bei jeder Anfrage der Nutzername und das Passwort übertragen?
Antwort: Die SID ist im wesentlichen durch SSL geschützt und könnte nur durch Man-in-the-middle oder CSRF herausgefunden werden. Wir versuchen unsere Applikation gegen CSRF gut zu schützen.

Frage 8: Arbeiten Sie mit verteilten Systemen? Sind z. B. Daten durch unterschiedliche Server räumlich voneinander getrennt? Gibt es eine Trennung zwischen Schlüssel-/Authentifizierungs-Server und Daten-Server?
Antwort: Wir haben ein verteiltes System: Firewall – Lastverteilung mit SSL Terminierung – Webserver – Tomcat Application Server – Datenbankserver

Frage 9: Gäbe es auch eine Möglichkeit, Ihr System auf einem Schulserver laufen zu lassen und somit von Ihren Servern unabhängig zu sein?
Antwort: Manche Schulen betreiben WebUntis noch selbst, aber wir bieten WebUntis nur mehr als Service an. Ausnahmen gibt es nur mehr für Großkunden, wie etwa die Stadt Hamburg.

A.5. Aussagen von Secomba GmbH

Die folgenden Fragen bezüglich des Produktes *Boxcryptor* wurden in einer E-Mail vom 11.05.2015 von Secomba GmbH beantwortet.

Frage 1: In Ihrem technischen Überblick [Gmbf] schreiben Sie, dass der jeweilige Dateischlüssel mit dem öffentlichen Schlüssel des Nutzers verschlüsselt wird. Desweiteren schreiben Sie, dass ein AES-Mantelschlüssel existiert, mit dem alle anderen AES-Schlüssel verschlüsselt werden. Sind damit nur die Dateinamensschlüssel und Gruppenschlüssel gemeint? Oder auch die Dateischlüssel? Dies würde dann meiner Meinung nach im Widerspruch zur vorherigen Aussage stehen.
Antwort: Damit sind lediglich die AES-Schlüssel gemeint, die an unseren Server übertragen werden – also den Dateinamen- und die Gruppenschlüssel. Der Dateischlüssel wird weiterhin ausschließlich mit dem öffentlichen Schlüssel des Nutzers / der Gruppe verschlüsselt.

Frage 2: Ist es richtig, dass ich, sobald ich das Nutzer-Passwort weiß, sämtliche Schlüssel (Nutzerschlüssel, Dateischlüssel, Gruppenschlüssel, etc.) entschlüsseln kann?

Antwort: Das ist korrekt. Aus dem Nutzer-Passwort wird ein Schlüssel abgeleitet, der (über Zwischenstationen) jeden weiteren Schlüssel im Zugriffsbereich des Nutzers entschlüsseln kann.

Frage 3: Möchte Alice eine Datei mit Bob teilen, so schreiben Sie, dass Alice den Dateischlüssel mit Bobs öffentlichem Schlüssel verschlüsselt und zusammen mit der verschlüsselten Datei abspeichert. Wie kommt Alice in diesem Fall an Bobs Schlüssel? Wird dieser über Ihren Schlüssel-Server verteilt? Angenommen dieser würde gehackt werden, so könnte ein Angreifer Bobs öffentlichen Schlüssel durch seinen öffentlichen Schlüssel tauschen und somit eine Man-In-The-Middle Attacke starten (indem er nach dem Erhalt des Dateischlüssels von Alice diesen auch an Bob weiterleitet und im Folgenden die Kommunikation "belauscht" oder sogar manipuliert). Oder sehen Sie für diesen Fall eine weitere Sicherheit vor?
Antwort: Auch das ist korrekt. Um tatsächlich eine Gefahr für den Nutzer darstellen zu können, müsste der Angreifer allerdings nicht nur unseren Server hacken, sondern zudem Zugriff auf die Dateien bekommen - sprich den Cloud-Provider zusätzlich hacken (von dem Fall, dass der PC des Nutzers direkt angegriffen wird, einmal abgesehen). Der Aufwand, zwei bestimmte Firmen erfolgreich zu infiltrieren ist jedoch so hoch, dass dieses Angriffsszenario i.d.R. nur für Angreifer mit sehr hohen Ressourcen (z. B. Regierungen) theoretisch durchführbar wäre. Unser Fokus liegt in erster Linie im Schutz der Daten vor dem Cloud-Storage-Providern selbst.
Ein weiterer Gedanke zum Angriffsszenario: Ein Angreifer würde bei einem solch gezielten Angriff mit sehr hoher Wahrscheinlichkeit einfach das Gerät des betroffenen Nutzers selbst angreifen (z. B. über Trojaner, Social Engineering, etc.), anstatt zwei Firmen zu hacken, da der Aufwand dazu einfach wesentlich geringer ist.

Frage 4: Erstellt ein Nutzer ein Passwort, so wird dieses nachher in gehashter Form an den Schlüsselserver übertragen und dort erneut gehasht gespeichert. Das gleiche Verfahren geschieht bei jeder Anmeldung. Wenn ich das richtig verstehe ist in dem übertragenen Passwort-Hash kein Salt oder Zeitstempel enthalten, da dieser erst auf dem Server hinzugefügt wird. Ein Angreifer, der diesen Hash abgreift könnte also eine Reply-Attacke starten und sich als Alice authentifizieren.
Antwort: Um eine Replay-Attacke zu starten, müsste ein Angreifer zunächst an den Login-Hash, der übertragen wird, herankommen. Da alle Daten per SSL übertragen werden, kann der Angreifer diesen nicht auf den Übertragungsweg abgreifen. Die Sicherheit entspricht in dieser Hinsicht also dem aktuellen Stand der Technik, wie er auch von Banken etc. beim Login verwendet wird. Aber selbst wenn der Angreifer den Hash abgreifen würde – er kann diesen nur dazu verwenden, die Nutzereinstellungen auf unserer Webseite zu ändern, oder die verschlüsselten Schlüssel des Nutzers abzugreifen. Diese könnte er mit dem Login-Hash jedoch nicht entschlüsseln – dazu ist ein eigener, in anderer Weise aus dem Passwort erzeugter Schlüssel notwendig, der nie über das Netzwerk geschickt wird. Dateien des Nutzers könnte er daher selbst mit dem Besitz des Login-Hashes und der Dateien nicht entschlüsseln, da durch den Passwort-Hash kein Rückschluss auf das Passwort möglich ist.

A.6. Ergebnis der Programmanalyse

In Tabelle A.1 und A.2 sind die wesentlichen Ergebnisse der Programmanalyse aus Kapitel 3 dargestellt.

	MEGA	**TeamDrive**	**Boxcryptor**
Funktion	Cloud-Dienst	Cloud-Dienst	Datei-verschlüsselung
Ende-zu-Ende-Verschlüsselung	ja	ja	ja
Datenzugriff	Browser	virtuelles Laufwerk	virtuelles Laufwerk
symm. Verschlüsselung	AES-128	AES-256	AES-256
Einsatz/Generierung	pro Datei/Ordner	pro *SharedSpace*	pro Datei
Schlüsselschutz	Masterschlüssel	Nutzername und Passwort	asym. Schlüssel
Speicherort	Server	Client	Server
asymm. Verschlüsselung	RSA-2048	Diffie-Hellman, RSA-3072	RSA-4096
Einsatz	Tausch sym. Schlüssel	Tausch sym. Schlüssel	Tausch und Schutz sym. Schlüssel
Generierung	pro Nutzer	pro Gerät	pro Nutzer
Schutz privater Schlüssel	Masterschlüssel	Nutzername und Passwort	Nutzerschlüssel (PBKDF2)
Sicherheit Serverkommunikation	SSL	unbekannt	SSL
Serverstandort	unbekannt	Europa	je nach Cloud-Dienst
Bemerkung	Passwortschutz des Masterschlüssels	Datenschutz-gütesiegel	Dateiverlust bei Vergessen des Passwortes

Tabelle A.1.: Übersicht MEGA, TeamDrive und Boxcryptor

	MobileSitter	**WebUntis**
Funktion	Passwortmanager	Schulverwaltung
Ende-zu-Ende-Verschlüsselung	-	nein
Datenzugriff	mobile App	Browser
symm. Verschlüsselung	AES	-
Einsatz/Generierung	ein Schlüssel für alle Geheimnisse, Gruppierung möglich	-
Schlüsselschutz	Nutzerpasswort	-
Speicherort	Nutzergerät	-
Sicherheit Serverkommunikation	-	SSL mit 128 Bit
Serverstandort	-	Wien
Bemerkung	keine Rückmeldung über Richtigkeit des Nutzerpasswortes, durch Patent geschützt	Daten im Klartext auf dem Server

Tabelle A.2.: Übersicht MobileSitter und WebUntis

A.7. Allgemein verwendete Bezeichnungen

$S_{privKey}$ öffentlicher Schlüssel des Servers zum Aufbau des Verwaltungskanals

S_{pubKey} privater Schlüssel des Servers zum Aufbau des Verwaltungskanals

A_{secret} Nutzergeheimnis, um den Schlüsselcontainer von Alice zu entschlüsseln

ePw_A Einmalpasswort, mit welchem Alice A_{pubKey} sicher an den Server übertragen kann

A_{pubKey} öffentlicher Schlüssel von Alice zum Aufbau des Verwaltungskanals

$A_{privKey}$ privater Schlüssel von Alice zum Aufbau des Verwaltungskanals

$A_{sessionKey}$ symmetrischer Sitzungsschlüssel von Alice und dem Server

A_{encKey} öffentlicher Schlüssel von Alice, um Daten an sie verschlüsselt zu übertragen

A_{decKey} privater Schlüssel von Alice, um an sie gesendete Daten zu entschlüsseln

$A_{verifyKey}$ öffentlicher Schlüssel von Alice, um Signaturen von ihr zu prüfen

$A_{signKey}$ privater Schlüssel von Alice, um Signaturen zu erstellen

B_{secret} Nutzergeheimnis, um den Schlüsselcontainer von Bob zu entschlüsseln

ePw_B Einmalpasswort, mit welchem Bob B_{pubKey} sicher an den Server übertragen kann

$B_{privKey}$	öffentlicher Schlüssel von Bob zum Aufbau des Verwaltungskanals
B_{pubKey}	privater Schlüssel von Bob zum Aufbau des Verwaltungskanals
$B_{sessionKey}$	symmetrischer Sitzungsschlüssel von Bob und dem Server
B_{encKey}	öffentlicher Schlüssel von Bob, um Daten an ihn verschlüsselt zu übertragen
B_{decKey}	privater Schlüssel von Bob, um an ihn gesendete Daten zu entschlüsseln
$B_{verifyKey}$	öffentlicher Schlüssel von Bob, um Signaturen von ihm zu prüfen
$B_{signKey}$	privater Schlüssel von Bob, um Signaturen zu erstellen
D_{key}	ein pro Datei neu generierter symmetrischer Schlüssel zur Verschlüsselung dieser