Bachelor Thesis

Authentifikation an einer webbasierten Applikation mittels Mifare DESFire und Public-Key-Verfahren

Zur Erlangung des akademischen Grades eines

B. Sc.

Fachhochschule Schmalkalden
Fakultät Informatik
Referent: Prof. Dr. Dietmar Beyer
Korreferent: Dipl.-Math. Michael Otto

eingereicht von:
Kai Trott
kaitrott@gmail.com

Volker Zeihs
volker.zeihs@gmail.com

Schmalkalden, den 15. März 2013

Danksagung

Zu Beginn danken wir all denen, die diese Arbeit ermöglicht haben.

Wir danken Prof. Dr. Dietmar Beyer für sein Engagement für den Bereich der IT-Sicherheit an der Fachhochschule Schmalkalden. Dieser Einsatz ermöglichte es uns diese Bachelorarbeit entstehen zu lassen. Außerdem bedanken wir uns für seine durchweg sehr gute Betreuung bei der Erstellung dieser Arbeit.

Dipl.-Math. Michael Otto danken wir, dass er als Korreferent zur Verfügung stand.

Besonderer Dank gilt unseren beiden Korrektur-Lesern, Dipl.-Wirtschaftsinf. (FH) Beatrice Florat und Dipl.-Wirtschaftsinf. (FH) Jonas Groß. Sowie B. Sc. Christian Linde, dessen Gespräche und Unterhaltungen für uns generell sehr anregend und konstruktiv waren.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Abkürzungsverzeichnis

3DES Triple-DES, wobei Schlüssel 1 und 2 unabhängig sind und Schlüssel 1 gleich dem Schlüssel 3 ist

3KDES Triple-DES, wobei alle Schlüssel unabhängig von einander sind

AES Advanced Encryption Standard

AID Application Identifier

API Application Programming Interface

BSI Bundesamt für Sicherheit in der Informationstechnik

CA Certification Authority

DBMS Datenbankmanagementsystem

DES Data Encryption Standard

DESFire DES „Fast, Innovative, Reliable and Enhanced“

DLIES Discrete Logarithm Integrated Encryption Scheme

DMA Direct Memory Access

ECIES Elliptic Curve Integrated Encryption Scheme

eID eID-Funktion (Online-Ausweisfunktion)

FeliCa Felicity Card

GPL GNU General Public License

GTK GIMP Toolkit

HTTPS Hypertext Transfer Protocol Secure

KiB Kibibyte

KT Kaskade-Tag

MAD Mifare Application Directory

NFC Near Field Communication

nPa der neue Personalausweis

NXP Semiconductors Next eXPerience Semiconductors

ODBC Open Database Connectivity

PKI Public Key Infrastruktur

RFID Radio-Frequency Identification

RSA Rivest, Shamir und Adleman

SSL Secure Sockets Layer, die alte Bezeichnung für Transport Layer Security

UID Unique Identifier

Kapitel 1

Einleitung

Am 1. November 2010 wird der neue Personalausweis (nPa) in Scheckkartengröße eingeführt. Eines der Neuerungen des nPa ist ein eingebauter 13,56 MHz RFID-Transponder[1], in dem unter anderem das Lichtbild und zwei Fingerabdrücke hinterlegt sind. Der nPa ermöglicht es, sich auch im Internet eindeutig ausweisen zu können. Doch nach über zwei Jahren gibt es kaum Anwendungen für diese Funktion.

Wir müssen uns tagtäglich an bis zu 20 verschiedene Passwörter[2] und Pincodes erinnern, was uns mehr oder weniger gut gelingt. Da wäre ein System, bei dem man sich durch einen Gegenstand, den man besitzt, in diesem Fall der Ausweis, authentifiziert, wünschenswert.

Die Ursachen für die Zurückhaltung im geschäftlichen und öffentlichen Bereich lassen sich hierbei nur abschätzen. Zum einen ist es ein Problem, dass nicht alle Bürger der Bundesrepublik bereit sind, sich auf allen mehr oder weniger seriösen Webseiten mit ihrem Personalausweis auszuweisen. Zum anderen muss ein Unternehmen bzw. ein Webseitenbetreiber erst ein Berechtigungszertifikat vom Bundesverwaltungsamt erwerben und die damit verbundenen Kosten tragen. Darüber hinaus wird eine Nutzungsgebühr des eID-Servers fällig. Diese Kostenfaktoren können gerade in kleineren oder durch Spenden finanzierten Bereichen nicht getragen werden.

[1] RFID steht für Radio-Frequency Identification.
[2] Vergleiche [Sch07].

1.1 Problemstellung

Einige ähnliche Systeme haben es geschafft, sich im geschäftlichen Bereich durchzusetzen, wie zum Beispiel die *Secure-ID* von der Firma RSA-Security. Sogar im privaten Bereich gibt es Systeme wie *Persona* von Mozilla oder *YubiKey* von Yubico. Diese Systeme sollen das Anmelden an Webseiten sicherer und einfacher machen. Doch konnten diese Systeme nicht die breite Masse der Benutzer erreichen. Der Standard für die Authentifikation im Internet ist immer noch die Eingabe eines Benutzernamens und Passwortes. Dass das Verwenden von Benutzernamen und Passwörtern im *World Wide Web* nicht nur umständlich, sondern vor allem nicht sicher ist, wird im folgenden Abschnitt beispielhaft aufgezeigt.

Verfolgt man die Nachrichten, scheint ein Hacker-Angriff dem nächsten zu folgen. Oft werden dabei die Kundendatenbanken mit den Benutzernamen und Passwörtern entwendet, so wie Mitte 2012. Zu diesem Zeitpunkt veröffentlichte die Hackergruppe *D33D* 450.000 Nutzerdaten von Yahoo. Unter den entwendeteten Daten befinden sich unter anderem auch Benutzernamen und Passwörter.[3]
Ein etwas aktuellerer Fall ereignete sich in der letzten Januarwoche des Jahres 2013. Hierbei gelang es Angreifern ca. 250.000 Benutzerdatensätze von Twitter zu stehlen. Darunter befanden sich unter anderem die Benutzernamen, die E-Mail-Adressen, die Session-Token und die Passwörter.[4]

Mit Hilfe solcher Informationen können Benutzerkonten übernommen und missbraucht werden. Handelt es sich, wie in diesem Fall, um E-Mail-Konten, besteht die Gefahr, dass über das E-Mail-Konto eines Benutzers noch weitere Konten übernommen werden können. Viele Anbieter stellen zum Beispiel eine *Passwort vergessen*-Funktion bereit, bei der ein neues Passwort per E-Mail versendet wird.

Google hat dieses Problem erkannt und experimentiert mit einem neuen Loginverfahren, welches es ermöglichen soll, mit einen USB-Dongle an den Produkten von Google zu authentifizieren. Auch über die Verwendung eines RFID-Transponder wird nachgedacht.[5] Dafür soll ein Browser-Plugin verwendet werden, welches die Einsatzmöglichkeiten voraussichtlich enorm einschränken wird.

[3] Vergleiche [yah12].
[4] Vergleiche [twi13] und [Lor13].
[5] Vergleiche [Mey13].

1.2 Ziel der Arbeit

Ziel dieser Arbeit ist eine prototypische Implementation eines Systems, welches die folgende Kriterien erfüllen. Es soll

- modular,
- preisgünstig,
- sicher,
- einfach in der Handhabung und
- quelloffen sein.

Diese Kriterien werden nachfolgend näher erläutert.

Das System sollte möglichst modular sein, damit es mit einem breiten Spektrum von Anwendungen zusammenarbeiten kann. Dies ist zum Beispiel durch standardisierte Schnittstellen möglich, über die die einzelnen Komponenten des Systems miteinander kommunizieren. Außerdem sollte jedes dieser Komponenten so wenig wie möglich Abhängigkeiten zu anderen Programmen oder Ressourcen haben.

Natürlich spielt der Preis eine wesentliche Rolle. Ein weitverbreitetes Authentifikationssystem für die Verwendung in großen Computernetzen ist das *Secure-ID*-System von RSA Security. Daher soll sich der Preis an den eines *Secure-ID*-Keys in Höhe von 40 Euro orientieren.

Die Sicherheit des Systems ist von entscheidender Bedeutung. Zum einen soll es nicht möglich sein die Sicherheitshürden des Prototypen zu umgehen, wenn Daten vom Webserver entwendet werden. Zum anderen soll die Sicherheit des Systems durch Abhören der Verbindung nicht zu kompromittieren sein.

Die Bedienung des Systems soll möglichst einfach und intuitiv sein, da Fehler in der Bedienung häufig die Sicherheit eines Systems verringern.

Diese Arbeit und alles, was im Rahmen dieser entwickelt, geschrieben oder dokumentiert wird, wird der Allgemeinheit frei zur Verfügung gestellt.

1.3 Abgrenzung und Aufbau

Das in dieser Arbeit entwickelte Authentifikationsystem unterscheidet sich vor allem durch die Verwendung eines RFID-Transponders und seiner Quelloffenheit gegenüber schon bestehenden Systemen wie *Secure-ID* oder *Persona*. Der *YubiKey* hingegen bietet auch eine RFID-Lösung an. Doch verwendet dieser den sehr unsicheren Mifare Classic RFID-Chip.[6] Somit ist dieses System, unter Verwendung der RFID-Schnittstelle, nicht für die Authentifikation geignet.

Der Aufbau des in dieser Arbeit beschriebenen Systems ermöglicht es, im Gegensatz zu herkömmlichen Systemen, frei verfügbare Hardwaresysteme zu verwenden. Zwar beschränkt sich diese Arbeit auf die Verwendung von Mifare Desfire und kompatiblen Lesegeräten, doch ist es durch die freie Verfügbarkeit des Quelltextes und dieser Arbeit möglich, dieses System auch auf andere Hardwaresysteme zu migrieren.

Bei einigen Systemen entstehen bei der Verwendung nicht unerhebliche Kosten. Bei *Secure-ID* oder dem neuen Personalausweis ist es erforderlich Serviceverträge mit den Betreibern abzuschließen. Diese können je nach Lizenz und Laufzeit mehrere Tausend Euro im Jahr betragen. Dies ist bei der Verwendung von quelloffener Software nicht notwendig.

Der Aufbau dieser Arbeit orientiert sich an den Zyklen der Softwareentwicklung. Diese umfassen die Analyse, den Entwurf, die Implementierung und die Verifikation.

Das Kapitel 2 *Analyse* beschreibt den zu entwickelnden Prototypen und dessen Umfeld. Es werden Probleme und Lösungsansätze beschrieben. Zu diesen zählt insbesondere die Problematik der Authentifizierung sowie die der Public Key Infrastruktur (PKI). Die verschiedenen Hardwaresysteme werden beschrieben und gegeneinander abgewogen.

Im Kapitel 3 *Entwurf* werden Strukturentscheidungen getroffen und näher erläutert. Darunter zählen unter anderem das Authentifikationsprotokoll und dessen Komponenten.

Die eigentliche Implementierung befindet sich im Kapitel 4 *Implementierung*. Dort werden die wichtigsten Programmteile aus Sicht der Implementierung beschrieben.

[6] Für weiterfürende Informationen vergleiche [cla12] und [BFG+12].

Im letzten Kapitel 5 *Fazit* wird die Arbeit zusammengefasst und ein kleiner Ausblick in die Zukunft gegeben. Daneben wird die Implementierung einer Prüfung auf mögliche Schwachstellen des Systems untersucht und solche aufgezeigt.

Kapitel 2

Analyse

Im diesem Kapitel werden die Anwendungsfälle des Prototypen im Abschnitt 2.1 beschrieben. Die Abschnitte 2.2 bis 2.4 befassen sich mit den Grundlagen der Authentifikation und den Public-Key-Infrastrukturen sowie den kryptographischen Verfahren. Im letzten Abschnitt 2.5 wird ein Überblick über die verfügbare, verwendete Hardware gegeben.

2.1 Anwendungsfälle

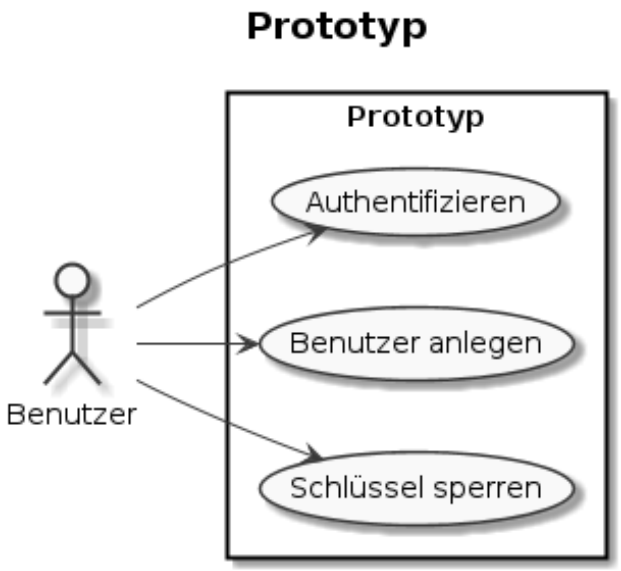


Abbildung 2.1: Anwendungsfall-Diagramm, Quelle: Eigene Darstellung

Dieses Kapitel dient zur Erfassung der Ziele des in dieser Arbeit entstandenen Prototypen. Es werden die Anforderungen für das in dieser Arbeit beschriebene Softwaresystem definiert. Darunter zählt das externe Systemverhalten sowie die Leistung, die vom System erbracht werden muss, um den gegebenen Zielen gerecht zu werden. Ein Authentifikationssystem umfasst zumeist drei elementare Funktionen: das Authentifizieren, das Hinzufügen eines Benutzers und das Löschen eines Benutzers.

Die Abbildung 2.1 zeigt das Anwendungsfall-Diagramm mit den drei Anwendungsfällen *Authentifikation*, *Benutzer anlegen* und *Schlüssel sperren*. Diese werden nachfolgend näher betrachtet.

2.1.1 Authentifikation an der Webapplikation

Name Authentifizieren

Kurzbeschreibung Ein Benutzer kann sich mit Hilfe eines privaten Schlüssels, der auf einem RFID-Transponder gespeichert ist, an einer Webapplikation authentifizieren.

Ziel Benutzer an einer Webapplikation authentifizieren

Kategorie primär

Vorbedingung Benutzer ist dem System bekannt

Nachbedingung Erfolg Benutzer ist authentifiziert

Nachbedingung Fehlschlag Benutzer ist nicht authentifiziert

Akteur Benutzer

Auslösendes Ereignis Benutzer will sich authentifizieren

Standardablauf

1. Benutzer lädt die Webapplikation
2. Benutzer klickt auf „Log-In“
3. System startet den Authentifikationsvorgang
4. System authentifiziert Benutzer

Erweiterungen 4.a System informiert Benutzer über den Erfolg oder Misserfolg der Authentifikation

Alternative 4.b System kann den Benutzer nicht authentifizieren

2.1.2 Benutzer am System registrieren

Name Benutzer anlegen

Kurzbeschreibung Damit sich ein Benutzer am System authentifizieren kann, muss er Information an der Webapplikation hinterlegen.

Ziel einen neuen Benutzer im System hinzufügen

Kategorie sekundär

Nachbedingung Erfolg Benutzer wird im System angelegt

Nachbedingung Fehlschlag Benutzer wird nicht im System angelegt

Akteur Benutzer

Auslösendes Ereignis Benutzer will sich an der Webapplikation registrieren

Standardablauf

1. Benutzer lädt die Webapplikation
2. Benutzer klickt auf „Registrieren“
3. System startet den Registrierungsvorgang
4. Benutzer sendet seinen öffentlichen Schlüssel an die Webapplikation
5. Benutzer wird im Systen registriert

Erweiterungen 5.a System informiert Benutzer über den Erfolg oder Misserfolg der Registrierung

Alternative 5.b System kann den Benutzer nicht registrieren

2.1.3 Sperren eines Schlüssels

Name Schlüssel sperren

Kurzbeschreibung Es besteht die Möglichkeit, dass ein privater Schlüssel gestohlen oder verloren wird. Trifft eines der beiden Ereignisse zu, ist es erforderlich den öffentlichen Schlüssel zu sperren bzw. löschen.

Ziel einen öffentlichen Schlüssel aus dem System entfernen

Kategorie sekundär

Nachbedingung Erfolg der öffentlicher Schlüssel wurde aus dem System entfernt, das Schlüsselpaar kann nicht mehr verwendet werden

Nachbedingung Fehlschlag Schlüssel wird nicht aus dem System entfernt

Akteur Benutzer

Auslösendes Ereignis Benutzer möchte seinen Schlüssel sperren lassen

Standardablauf
1. Benutzer lädt die Webapplikation
2. Benutzer meldet sich mittels Benutzernamen und Passwort am System an
3. Benutzer klickt auf „Bestätigen“
4. System entfernt den Schlüssel aus dem System

Erweiterungen 4.a System informiert den Benutzer über den Erfolg oder Misserfolg der Sperrung

Alternative 2.a Benutzername oder Passwort sind falsch, Benutzer wird abgewiesen
4.b Schlüssel wird nicht entfernt

2.2 Authentifikation und deren Protokolle

In diesem Kapitel werden die verschiedenen Authentifikations-Verfahren und ihre Protokolle beschrieben und auf ihre Eignung in großen Computernetzen wie zum Beispiel das „World Wide Web“ betrachtet. Bei der Erstellung dieses Kapitels wurde sich hauptsächlich an dem Buch „Kryptografie – Verfahren, Protokolle, Infastrukturen“ von Klaus Schmeh[7] orientiert.

Authentifikation begegnet uns nicht nur im Internet, sondern auch bei ganz alltäglichen Dingen, wie zum Beispiel beim Bezahlen mit der EC-Karte. Dort authentifizieren wir uns gegenüber dem Bankterminal.
Beim Reisen ins Ausland müssen wir uns ebenso authentifizieren bzw. ausweisen. Dies geschieht gegenüber den Behörden mit unserem Reisepass.
Will man in die Vereinigten Staaten von Amerika reisen, reicht es den dortigen Behörden nicht, dass man einen gültigen Reisepass besitzt. Es werden zusätzlich unverwechselbare Eigenschaften einer Person, in diesem Fall der Fingerabdruck und das Aussehen, über einen NFC-Chip[8] im Pass ausgelesen und mit der zu überprüfenden Person verglichen.

[7] [Sch07]
[8] NFC steht für Near Field Communication.

Diese Beispiele zeigen die drei Möglichkeiten der Authentifikation. Im ersten Beispiel wird eine bestimmte Information abgefragt und diese mit der hinterlegten verglichen. Diese Form der Authentifikation bezeichnet man als *Authentifikation durch Wissen*. Hierbei finden nicht nur Passwörter oder Pincodes Verwendung. Es könnte auch eine Reihenfolge oder Platzierung von verschiedenen Personen oder Gegenständen auf einer Matrix sein.[9]

Im zweiten Beispiel authentifiziert sich eine Person durch einen Pass, also durch etwas in seinem Besitz. Dies nennt man *Authentifikation durch Besitz*. Überträgt man dieses Konzept in die Welt der Computernetze, so benötigt man eine Art elektronischen Ausweis. Dieser muss nicht den Namen oder ein Bild der Person enthalten. Es reicht, wenn ein solcher elektronischer Ausweis eine geheime Information besitzt. Hierzu zählt auch das im Rahmen dieser Arbeit entstandene System.

Im letzten Beispiel-Szenario werden unverwechselbare persönliche Merkmale überprüft. Hierfür können der Fingerabdruck, das Aussehen, die Unterschrift, die Stimme oder die Iris einer Person verwendet werden, um nur einige zu nennen. Der Fachbegriff für diese Variante der Authentifikation ist die *Authentifikation durch Eigenschaften*.

Die *Authentifikationen durch Besitz und Eigenschaften* sind in der Regel komfortabler zu benutzen als die *Authentifikation durch Wissen*. Die komfortable Bedienung ist ein wichtiger Punkt, wenn es um die Sicherheit eines kryptografischen Systems geht. Denn die Sicherheit eines Systems hängt immer von seiner Benutzung ab. Vergisst ein Benutzer häufig seine Passwörter, ist er geneigt sich diese aufzuschreiben oder für mehrere Konten oder Systeme das gleiche Passwort zu verwenden. Dadurch kann die Sicherheit eines kryptografischen Systems kompromittiert werden.

Die *Authentifikation durch Eigenschaften* hat im Internet einige weitere Nachteile. Man authentifiziert sich mit unveränderlichen persönlichen Eigenschaften gegenüber anderen. Werden diese Information mit anderen persönlichen Daten verknüpft, bieten sie ein hohes Risiko der widerrechtlichen Nutzung. Zwar gibt es die erste Versuche, wie die ISO-Norm 24745, die Speicherung und Verarbeitung dieser Daten einzuschränken. Diese Entwicklung steckt jedoch zur Zeit noch in den Kinderschuhen.

[9] Vergleiche [Sch07].

Diese Gründe haben die Autoren dazu bewegt, eine *Authentifikation durch Besitz* als Grundlage für das entwickelte System zu wählen. Bei der Umsetzung eines Verfahrens mit *Authentifikation durch Besitz* gibt es grundsätzlich mehrere Möglichkeiten der Umsetzung. Im Rahmen dieser Arbeit wird jedoch nur das *Challenge Response*-Verfahren näher betrachtet.

Alle *Challenge Response*-Verfahren haben eins gemein: bei der Authentifikation wird das Geheimnis, zum Beispiel das Passwort oder die Pin, nicht über das Netz übertragen. Dies ist ein großer Vorteil von *Challenge Response*-Verfahren. Vielmehr wird eine nicht-geheime Information, von der Authentifizierungsstelle an den zu Authentifizierenden gesendet (die so genannte *Challenge*). Aus der Challenge und einer geheimen Information, zum Beispiel ein Passwort oder eine PIN, wird eine Antwort (die so genannte *Response*) gebildet und an die Authentifikationsstelle zurückgesendet. Diese Stelle kann nun die Richtigkeit der geheimen Information überprüfen.

Bei einer speziellen Form des *Challenge Response*-Verfahrens wird die asymmetrische Verschlüsselung eingesetzt, um Challenge und Response zu berechnen. Bei diesem Verfahren ist es nicht erforderlich die geheime Information, in diesem Fall der private Schlüssel, an die Authentifikationsstelle zu übermitteln oder dort zu speichern, da diese nur den zugehörigen öffentlichen Schlüssel benötigt. Das ist ein Vorteil gegenüber Verfahren mit symmetrischer Verschlüsselung. Daher eignet sich das *Challenge Response* mit asymmetrischer Verschlüsselung besonders für die Verwendung in Computernetzen.

2.3 Public Key Infrastruktur

In diesem Abschnitt sollen verschiedenen Verfahren zur Verwendung von asynchroner Verschlüsselung in Computernetzen unter dem Gesichtspunkt der Tauglichkeit für das in dieser Arbeit beschriebene System betrachtet werden. Es wird sich bei den Kapiteln 2.3 bis 2.3.3 an den dem Buch „Kryptografie und Public-Key-Infrastrukturen im Internet“ von Klaus Schmeh[10] orientiert.

[10][Sch01]

Die Verwendung von asynchronen Verschlüsselungsverfahren ohne Einsatz einer geeigneten Infrastruktur kann einige Probleme verursachen. Dazu zählen:

- die Authentizität der Schlüssel,
- die Sperrung von Schlüsseln,
- die Verbindlichkeit und
- das Durchsetzen einer Policy.

Das Problem der Authentizität der Schlüssel besteht in der Möglichkeit des Fälschens des öffentlichen Schlüssels. Gelingt es einem Angreifer seinen öffentlichen Schlüssel als den von einer anderen Person zu verbreiten, kann auch nur er die verschlüsselten Nachrichten lesen.

Die Sperrung von Schlüsseln wird dann wichtig, wenn der private Schlüssel in die falschen Hände gerät. Wird dies bemerkt, sollte kein Kommunikationspartner den zugehörigen öffentlichen Schlüssel nutzen. Die Frage ist aber: Wie werden alle Kommunikationspartner über den Diebstahl informiert?

Die Verbindlichkeit in einer PKI garantiert die Beweisbarkeit wem welcher privater Schlüssel gehört. Das spielt zum Beispiel bei der Signatur eine entscheidende Rolle.

Oft ist es erforderlich einige Regeln für das Generieren, das Verteilen und das Arbeiten mit öffentlichen und privaten Schlüsseln aufzustellen. Diese Regeln fasst man unter dem Begriff *Policy* zusammen. Das Problem hierbei besteht in der Umsetzung dieser.

Diese vier Probleme lassen sich gut durch eine Public Key Infrastruktur lösen. Nun gibt es mehrere verschiedene Ansätze eine solche PKI aufzubauen. Alle diese Ansätze sind mehr oder weniger für das in dieser Arbeit beschriebene System geeignet und sollen im Folgenden auf ihre Tauglichkeit untersucht werden.

2.3.1 Direct Trust

Direct Trust ist die einfachste Form einer PKI. Hierbei geht man davon aus, dass sich beide Parteien vertrauen. Das Problem der *Authentizität des Schlüssels* lässt sich im Direct Trust trivial lösen. Es genügt beispielsweise, wenn sich beide Parteien treffen, um ihre öffentlichen Schlüssel zu tauschen. Sie könnten sich ihre Schlüssel auch per E-Mail zusenden. Eine Manipulation kann ausgeschlossen werden, durch Vergleichen eines kryptografischen Hashwertes über einen sicheren Kanal, wie zum Beispiel das Telefon.

Die Sperrung eines Schlüssels lässt sich ebenso über Direct Trust realisieren. Möchte man, dass die jeweils andere Partei einen Schlüssel nicht weiter verwendet, teilt man es ihr einfach mit.

Die Verbindlichkeit jedoch lässt sich nicht oder nur sehr schwer mit Direct Trust realisieren. Jede der beiden Parteien kann abstreiten einen bestimmten Schlüssel zu besitzen.

Das Durchsetzen einer Policy gestaltet sich mit Direct Trust schwierig. Stellt man sich ein größeres Computernetz vor, so müssten sich alle Teilnehmer auf eine Policy verständigen. Da es aber immer unterschiedliche Meinungen und Interessen gibt, wird dies nur schwer gelingen.

2.3.2 Web of Trust

Das Web of Trust ist eine einfache Form einer PKI. Die Idee hierbei ist es die Echtheit eines öffentlichen Schlüssels eines Unbekannten über ein Netz von gegenseitigem Vertrauen und Direct Trust zu sichern.

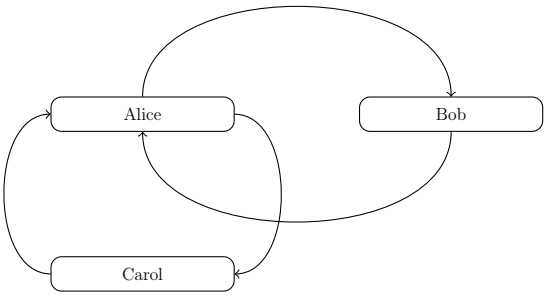
Im diesem Absatz wird ein Szenario beschrieben, welches den Schlüsselaustausch mit Hilfe von Web of Trust beschreibt. Zusätzlich wird dieses Szenario in der Abbildung 2.2 dargestellt.

Möchte ein Benutzer Alice einen öffentlichen Schlüssel von Benutzer Bob, so können Alice und Bob diesen mit Hilfe von Direct Trust austauschen. Möchte jetzt Benutzer Carol auch den Schlüssel von Bob und kennt schon den Schlüssel von Alice, so kann Alice den Schlüssel mit dem Namen von Bob versehen, signieren und Carol

zusenden.

Einen solches Signieren von öffentlichen Schlüsseln, der auch den Namen des Benutzers enthält, nennt man *Zertifikat*. Carol kann nun sicher sein, dass der Schlüssel korrekt ist. Im Web of Trust entstehen so Vertrauensketten. Stellt man sich nun ein Netz mit sehr vielen über die ganze Welt verteilten Benutzern vor, ist es sehr schwer solche Ketten des Vertrauens zu finden. Ist nur ein Glied der Kette nicht vertrauenswürdig, kann die Echtheit des Schlüssels nicht sicher gestellt werden.

***Phase 1:* Schlüsselaustausch durch Direct Trust**



***Phase 2:* Schlüsselaustausch durch Web of Trust**

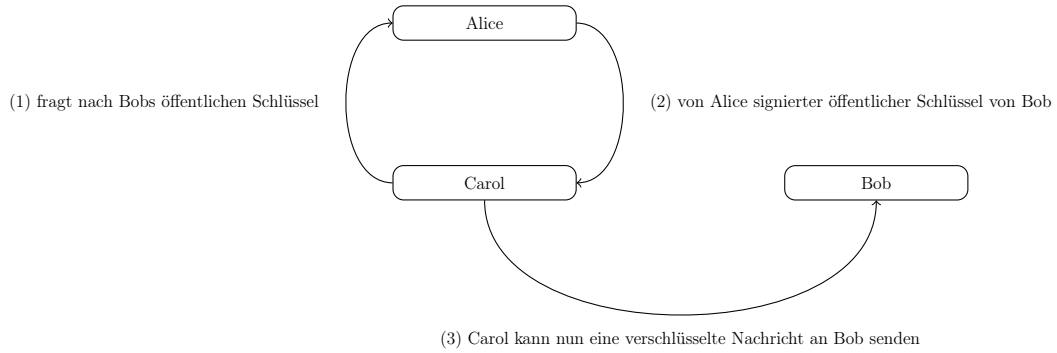


Abbildung 2.2: Schlüsselaustausch mit Web of Trust, Quelle: Eigene Darstellung

Bei der Sperrung des Schlüssels verhält es sich ähnlich wie beim Direct Trust. Möchte ein Benutzer nicht mehr, dass sein öffentlicher Schlüssel verwendet wird, muss er allen, die seinen öffentlichen Schlüssel haben könnten, dies mitteilen.

Die Verbindlichkeit wird im Vergleich zum Direct Trust durch das Web of Trust schon recht gut gelöst, da es durch die erstellten Zertifikate Beweise gibt.

Aber das Durchsetzen einer Policy ist im Web of Trust sehr schwer, dies wird umso deutlicher je größer das Computernetz wird.

2.3.3 Hierarchical Trust

Bei der PKI Hierarchical Trust gibt es eine unabhängige Instanz, die alle öffentlichen Schlüssel signiert (siehe Abbildung 2.3). Diese Instanz wird meist Certification Authority (CA) genannt. Hierbei muss jeder Benutzer den öffentlichen Schlüssel der CA haben und sich zum Beispiel über Direct Trust vergewissern, dass dieser echt ist. Des Weiteren muss sich jeder Benutzer an der CA mit seinem öffentlichen Schlüssel registrieren. Natürlich bringt all dies nichts, wenn die CA nicht von einer vertrauenswürdigen Stelle betrieben wird.

Möchte ein Benutzer einen neuen öffentlichen Schlüssel, fragt er diesen an der CA an. Die CA signiert den gewünschten Schlüssel und sendet ihn dem Benutzer zu. Da alle Benutzer den öffentlichen Schlüssel der CA haben, können sie diese Signatur überprüfen und damit auch die Authentizität des Schlüssels.

Für die Sperrung eines Zertifikates muss dies nur der CA mitgeteilt werden. Sie kann dann sofort diese Information an alle weiterleiten.

Das Problem der Verbindlichkeit lässt sich durch das Erzeugen und das Verteilen der Schlüsselpaare durch die CA lösen.

Die Umsetzung und die Überwachung einer Policy kann die CA übernehmen.

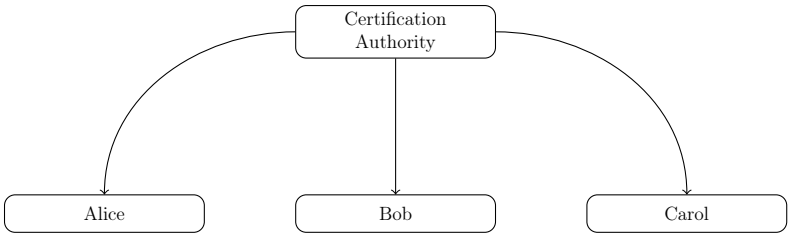


Abbildung 2.3: Schlüsselaustausch im Hierarchical Trust, Quelle: Eigene Darstellung

2.3.4 Wahl der PKI

Die PKIs *Web of Trust* und *Hierarchical Trust* sind nur bedingt für dieses System einsetzbar. Sie betrachten die Verbreitung von Schlüsseln im gesamten Netz. Für diese Arbeit wird jedoch nur ein Verfahren benötigt, welches die Probleme zwischen Nutzer und Webapplikation löst.

Hierfür bietet sich das Konzept des *Direct Trust* an. Überträgt man dieses Konzept auf das in dieser Arbeit beschriebene Authentifikations-System, könnte man das Problem der Authentizität eines Schlüssels durch das Übertragen eines kryptografischen Hashwertes über SSL ermöglichen.

Hierfür wird ein Hash-Funktion auf dem vom Benutzer erzeugten öffentlichen Schlüssel ausgeführt. Die Webapplikation führt bei sich ebenso eine Hash-Funktion auf dem empfangenen öffentlichen Schlüssel des Benutzers aus. Dieser wird dann über HTTPS[11] bereitgestellt. Der Benutzer muss jetzt diese beiden Hash-Werte vergleichen, um eine Manipulation des öffentlichen Schlüssels, während der Übertragung zur Webapplikation auszuschließen.

Das Sperren eines Schlüssels erfolgt dann über das einfache Mitteilen der Sperrung an der Webapplikation. Die Probleme der Verbindlichkeit und der Durchsetzung einer Policy spielen hierbei keine Rolle.

Ein *Hierarchical Trust*-Ansatz wäre denkbar. Dabei wäre die Webapplikation die CA. Diese hätte die Aufgabe die Schlüsselpaare zu generieren und diese über den RFID-Transponder zu verteilen. Hier müsste eine Sperrung nur der Webapplikation mitgeteilt werden. Dadurch wären die zwei Probleme der Authentizität und der Sperrung eines Schlüssels gelöst.

Der eigentliche Unterschied der beiden beschriebenen Problemlösungen ist die Stelle an der das Schlüsselpaar erzeugt wird. Entweder bei dem Benutzer und eigentlichem Eigentümer oder an der Webapplikation. Gerade aus diesem Grund wurde sich in der Implementierung für die Erzeugung beim Nutzer entschieden. Dadurch entfällt die gesicherte Übertragung des erzeugten Schlüssels vom Webserver zum RFID-Transponder.

Es ist durchaus denkbar das System auf andere Arten einer PKI abzuändern.

[11] HTTPS steht für Hypertext Transfer Protocol Secure.

2.4 Wahl des kryptographischen Verfahres

Bei den betrachteten asymmetrischen kryptografischen Verfahren wurde sich auf die vom Bundesamt für Sicherheit in der Informationstechnik (BSI)[12] empfohlenen beschränkt. Diese sind der Discrete Logarithm Integrated Encryption Scheme (DLIES), der Elliptic Curve Integrated Encryption Scheme (ECIES) und der RSA-Algorithmus[13].

Das ECIES-Verfahren hat gegenüber den beiden anderen einen großen Vorteil. Ein ECIES-Schlüssel ist wesentlich kleiner als ein vergleichbar sicherer Schlüssel des DLIES- oder RSA-Verfahrens.

Die beiden Verfahren ECIES und DLIES benötigen zusätzliche Komponenten, wie ein symmetrisches Verschlüsselungsverfahren sowie eine Schlüsselableitfunktion, welche eine Hashfunktion sein kann. RSA benötigt hingegen keine weiteren Komponenten.

Für die Implementierung des Prototypen wurde sich für das RSA-Verfahren entschieden, da es für dieses Verfahren genügend Bibliotheken in den meisten Programmiersprachen gibt. Dies ist leider nicht der Fall für das ECIES- oder das DLIES-Verfahren. Die RSA-Implementierungen in Bibliotheken, wie zum Beispiel *OpenSSL*, sind über Jahre hinweg gepflegt und verbessert worden, sodass man davon ausgehen kann, dass diese keine größeren Implementationsfehler aufweisen.

Die Implementierung des ECIES- oder DLIES-Verfahrens und die Sicherstellung der Funktionalität sowie die Richtigkeit der Implementierung würde den geplanten Umfang dieser Arbeit wesentlich überschreiten.

[12] Vergleiche [bsi13].

[13] RSA steht für die Anfangsbuchstaben der Nachnamen der Erfinder des RSA-Algorithmus: Rivest, Shamir und Adleman.

2.5 Selektion der Hardware

Die Auswahl, der zu verwendenden Hardware wurde durch die nachfolgenden Anforderungskriterien ermittelt. Vorausgesetzt wurde, den Zielen der Arbeit in Kapitel 1.2 entsprechend, die Verwendung quelloffener Software und sicherer kryptografischer Verfahren.

2.5.1 RFID-Transponder

Zur Auswahl stehen mehrere verschiedene RFID-Transponder-Typen. Die Transponder werden auf einzelne Exemplare der Firmen NXP Semiconductors und Felicity Card (FeliCa) beschränkt. Gerade RFID-Transponder von NXP Semiconductors sind weltweit im Einsatz und werden durch quelloffene Software vollständig unterstützt.

Kriterien für die Transponder-Wahl

Die Kriterien für die RFID-Transponder-Wahl sind:

- die Verschlüsselungsmethode,
- die Kosten,
- die Erweiterbarkeit,
- die Speicherkapazität und
- die Schnittstellen.

Nachfolgend werden die einzelnen Punkte näher erläutert.

Der Verschlüsselungs-Algorithmus muss nach aktuellem Standard als sicher gelten und möglichst keine bekannten Angriffsmöglichkeiten bieten.

Die Kosten sollten für den Endanwender tragbar sein. Dieses Kriterium ist gering gewichtet, da es ein relatives Verhältnis ist. Der Preis sollte sich möglichst nahe am

Durchschnitt aller RFID-Transponder befindet. Die Preise wurden aus mehreren Online-Shops bezogen und daraus ein Kostenbereich aufgestellt. Ein Preismaximum von 40 Euro ist in Kapitel 1.2 als Ziel der Arbeit angegeben und daran wird sich orientiert.

Unter dem Begriff *Erweiterbarkeit* steht zum einen die Nutzungsmöglichkeiten eines RFID-Transponders für mehrere Anwendungen, also eine mögliche Mehrfachverwendung des Transponders selbst. Zum andern steht der Begriff für die Möglichkeit die Länge des darauf gespeicherten Schlüssels ändern zu können.
Die Erweiterbarkeit ist vor allem auf lange Sicht hin notwendig, da davon auszugehen ist, dass die Rechenleistungen sich immer weiter erhöhen und sich damit aktuell theoretische Angriffe durch mehr Ressourcen realisieren lassen.

Der *Speicherkapaität* eines RFID-Transponders muss mindestens die Größe des privaten Schlüssels beinhalten und im besten Fall dynamisch an die benötigten Größe angepasst werden können. Das BSI empfiehlt in seiner technischen Richtlinie[14], für Daten eine Mindest-Schlüssellänge von 2.000 Bit. Bis 2019 sollen diese mit RSA sicher verschlüsselt sein, versichert das BSI darin. Darüber hinaus sollten Neuentwicklungen von kryptografischen Systemen, die längere Zeit eingesetzt werden sollen, über entprechende Erweiterungsmöglichkeiten für größere Schlüssel verfügen. In Tabelle 2.1 wird der Speicherbedarf je nach Schlüsselgröße dargestellt. Die Berechnung des Speicherbedarfs wird in Kapitel 4.1.4 erläutert.

Schlüsselgröße	**Benötigter Speicher**
1.024 Bits	436 Bytes
2.048 Bits	820 Bytes
4.096 Bits	1.588 Bytes
8.192 Bits	3.124 Bytes
16.384 Bits	6.196 Bytes
32.000 Bits	12.340 Bytes

Tabelle 2.1: Größe des benötigten Speicherplatzes nach Schlüsselgröße

Unter dem Begriff *Schnittstellen* steht die Nutzbarkeit des Lesegerätes und des RFID-Transponders durch existierende Software-Bibliotheken. Hierbei wird quelloffene Software bevorzugt, da es das Ziel der Arbeit ist, ebenfalls quelloffen zu sein. Die verfügbare Software sollte möglichst viele verschiedene RFID-Transponder und Lesegeräte unterstützen.

[14][bsi13]

Transponder-Typen

Folgende RFID-Transponder-Typen wurden verglichen:

- die Mifare Ultralight,
- die Mifare Ultralight C,
- die Mifare Plus X,
- die FeliCa RC-SA00,
- die Mifare Classic,
- die Mifare DES „Fast, Innovative, Reliable and Enhanced“ (DESFire) und
- die Mifare DESFire EV1.

Die *Mifare Ultralight* besitzt keine integrierte Verschlüsselungs-Methode. Sie ist für Einwegtickets in öffentlichen Transportnetzwerken gedacht. Der insgesamt zur Verfügung stehende Speicher beschränkt sich auf 512 Bit, welcher in 16 Seiten mit jeweils 4 Byte unterteilt ist. Der Nutzer selbst kann 12 dieser Seiten lesen und beschreiben. 32 Bit sind nur einmal änderbar, das bedeutet ein geschriebener Wert ist endgültig und unumkehrbar. Dazu verfügt sie über eine 7 Byte lange Unique Identifier (UID).

Eine Erweiterung zur *Mifare Ultralight* bietet die *Mifare Ultralight C*.[15] Sie ist vollständig abwärtskompatibel zu ihrem Vorgängermodell und ist für nutzungsbegrenzte Endsysteme konzipiert. Zu den Verbesserungen zählen das auf 1.536 Bit erweiterte Speichervolumen und die 3DES-Verschlüsselung[16]. 1.184 Bit sind dem Benutzer zugeordnet. Die Verschlüsselung wird optional angewandt.

Bei der *Mifare Classic* handelt es sich um eine weit verbreitete Art von Mifare-RFID-Transpondern. Sie ist in den Versionen 1K und 4K verfügbar.[17] Die Version beschreibt die Speichergröße. So steht die 1K für einen verwendbaren Speicher von einem Kibibyte (KiB). Die Adressierung des Speichers wird über Blöcke geregelt.

[15] Vergleiche [NXP09].
[16] Triple-DES, wobei Schlüssel 1 und 2 unabhängig sind und Schlüssel 1 gleich dem Schlüssel 3 ist (3DES)
[17] Vergleiche [NXP11b].

Hierzu wird bei der 1K ein MAD[18] 1 verwendet und bei der 4K kommt das MAD 2 zum Einsatz. Beide Varianten verwenden den Crypto1-Algorithmus, welcher als nicht sicher angesehen wird. Er weist kryptografische Lücken auf, die es ermöglichen ohne Wissen des Schlüssels an die Daten zu gelangen.[19]

Die *Mifare DESFire* bietet eine 3DES-Verschlüsselungsmöglichkeit. Sie verwendet das MAD 3.[20] Das bedeutet, der Speicher wird nicht direkt, wie in den Vorgängerversionen, angesprochen, sondern über ein virtuelles Zugriffssystem bestehend aus Applikationen und Dateien. Sie bietet einen Speicher von 4 KiB. Der RFID-Transponder kann bis zu 28 Applikation beinhalten. Jede dieser Applikationen kann bis zu 14 eigene Schlüssel besitzen und bietet Platz für bis zu 16 Dateien. Die Dateien spalten sich in fünf verschiedene, mit jeweiligen Spezifikationen und Anwendungsbereichen.

Eine neuere Art der *Mifare DESFire* bietet die *EV1*-Version.[21] Sie ist komplett abwärtskompatibel und erweitert den RFID-Transponder um zusätzliche Funktionen. Der RFID-Transponder existiert in verschiedenen Größenvarianten: 2K, 4K und 8K. Wie bei der *Classic* stehen hier die Namen für den dem Nutzer zur Verfügung stehenden Speicherplatz. Zudem wurden die kryptografischen Algorithmen um 3KDES[22] und AES[23] erweitert.

Die *Mifare Plus* ist zur einfachen Migrierung von verwendeten *Mifare Classic*-RFID-Transpondern auf eine sichere Verschlüsselungsart gedacht. Sie unterstützt den Crypto1-Algorithmus und AES-128. Es existieren die Varianten *Plus S* und *Plus X*.[24] Beide Varianten existieren jeweils mit einer Speichergröße von 2 KiB oder 4 KiB. Um Schrittweise die Migration leiten zu können, werden drei verschiedene Ebenen verwendet. Jede Ebene ermöglicht erweiterte Funktionalitäten und verringert die Unterstützung zu älteren RFID-Transponder-Versionen.

Die *FeliCa RC-SA00* beinhaltet AES- und Data Encryption Standard (DES) Verschlüsselungen.[25] Sie unterstützt mehrere Applikationen und hat einen 6 KiB großen Speicher.

[18] Mifare Application Directory (MAD)
[19] Vergleiche [cla12].
[20] Vergleiche [NXP13].
[21] Vergleiche [NXP10].
[22] Triple-DES, wobei alle Schlüssel unabhängig von einander sind (3KDES)
[23] Advanced Encryption Standard (AES)
[24] Vergleiche [NXP11a].
[25] Vergleiche [Fel12].

Wahl des Transpondertyps anhand der Kriterien

Typen	**Verschlüsselung**	**maximale Kosten**	**Erweiterbarkeit**	**Speicher (in Byte)**
Mifare Ultralight	keine Verschlüsselung	1 Euro	Block	384
Mifare Ultralight C	3DES	1 Euro	Block	1.184
Mifare Plus X	Crypto1 AES	3 Euro	Block	2.048 4.096
FeliCa RC-SA00	DES AES		Applikationen	6.144
Mifare Classic	Crypto1	3 Euro	Block	1.024 4.096
Mifare DESFire	DES	4 Euro	Applikationen	4.096
Mifare DESFire EV1	DES 3KDES AES	4 Euro	Applikationen	2.048 4.096 8.192

Tabelle 2.2: RFID-Transpondertypen anhand der Kriterien

Die *Mifare Ultralight C* hat eine 3DES-Verschlüsselung, die ausreichend wäre. Jedoch bietet ihre Speicherkapazität nicht genug Freiheiten, um auf lange Sicht nutzbar zu sein.[26] Das Gleiche gilt besonders für die *Mifare Ultralight*.

Die *Mifare Plus X* kann mit AES-Verschlüsselung genutzt werden. Sie hat ein Block-basiertes Speichersystem und ist groß genug, um einen privaten Schlüssel zu speichern. Jedoch fehlt es in den quelloffenen Bibliotheken an Unterstützung für diesen

[26] Vergleiche [bsi13].

RFID-Transponder. Aus diesem Grund wird sie in dieser Arbeit nicht verwendet.

Die *FeliCa RC-SA00* bietet eine AES-Verschlüsselung, Applikationsmöglichkeiten und enormen Speicherplatz. Sie wird jedoch nicht von quelloffener Software unterstützt. Erschwerend kommt hinzu, dass kein Onlineshop gefunden wurde, bei dem eine Privatperson diesen RFID-Transponder erwerben könnte.

Die *Mifare Classic* ist aufgrund ihres Block-Speichermanagements leicht zu bedienen. Sie würde auch in der 1K- und 4K-Version genügend Speicher zur Verfügung stellen. Sie bietet jedoch nicht die Möglichkeit Blöcke unterschiedlichster Größe zu verwalten. Dies ist besonders von Nachteil bei Informationen die größer als ein Block sind und deshalb in mehrere Blöcke aufgeteilt werden müssten. Zudem verwendet der RFID-Transponder den kryptografischen Algorithmus Crypto1, welcher bewiesen ungenügend Sicherheit bietet.[27]

Die *Mifare DESFire*, sowohl die Standard als auch die *EV1*-Version, bieten genügend Speicherkapazität von mindestens 2 KiB, hohe Sicherheit durch DES-, 3DES-, 3KDES- und AES-Verschlüsselung und ein ausgeklügeltes Speichersystem mit Applikationen und Dateien. Sie wird vollständig von quelloffener Software unterstützt. Damit erfüllt sie alle Anforderungen und gilt als beste Lösung der betrachteten RFID-Transpondern.

In Tabelle 2.2 ist dies nochmal übersichtlich zusammengefasst dargestellt.

2.5.2 Lesegerät

Aus den Zeilen dieser Arbeit können folgende Kriterien für das eingesetzte Lesegerät abgleitet werden:

- der Preis,
- der Treiber und
- die Verfügbarkeit.

[27] Vergleiche [cla12].

Zum einen spielt der Preis des Lesegerätes eine entscheidende Rolle, da er nicht den in den Zielen definierten Betrag überschreiten sollte. Zum anderen sollte es für dieses Lesegerät eine entsprechend mächtigen Treiber-API[28] geben. Am besten ist hier eine quelloffene Implementierung, da dies eine nachträgliche Eigen-Implementation von Schnitstellen ermöglicht. Des Weiteren sollte dieses Lesegerät einfach verfügbar sein, das heißt, es sollte von jedem ohne größeren Aufwand gekauft werden können. Dies ermöglicht, dass jeder den in dieser Arbeit entstandenen Prototypen testen und weiterentwickeln kann. Zusätzlich zu den aus den Zielen abgeleiteten Kriterien muss das RFID-Transponder-Lesegerät kompatibel zur *Mifare DESFire* sein, aus diesem Grund werden nur solche Lesegeräte betrachtet.

Omnikey 5321

Der *Omnikey 5321* ist für ca. 64 Euro erhältlich.[29] Eine quelloffene Bibliothek namens *Librfid* steht zur Verfügung, doch nach genauerer betrachtung ergab sich, dass die letzte veröffentlichte Version der *Librfid* von 2008 ist. Es ist davon auszugehen, dass dieses Projekt auch in Zukunft keinerlei Aktualisierungen erhält.

CL3711

Der von SCM Microsystems GmbH hergestellter *CL3711* ist ein Dongel und kostet ca. 40 Euro. Erhältlich ist dieses Lesegerät bei `chipdrive.de`.[30] Dieses Lesegerät ist vollständig mit der *Libnfc* kompatibel. [31]

Touchatag / ACR122U

Der *Touchatag* ist ein *ACR122U* von Advanced Card Systems Ltd. Dieser wird von der Firma Alcatel Lucent mit eigener Software und eigenen Tags für 40 Euro vertrieben.[32] Auch Advanced Card Systems Ltd bietet den *ACR122U* auf ihrer Webseite an, doch hier wird ein erheblich höherer Preis von 149 US Dollar verlangt.[33] Für

[28] Application Programming Interface (API)
[29] Vergleiche [CRY10].
[30] Vergleiche [CHI].
[31] Vergleiche [lib13].
[32] Vergleiche [Gro11].
[33] Vergleiche [Sys].

den *ACR122U* besteht die Möglichkeit der Verwendung der quelloffenen Bibliothek *Libnfc*. Dabei gibt es einige Einschränkungen, die aber für diese Arbeit nicht von Bedeutung sind.[34]

Wahl des Lesegerätes

	Omnikey	CL3711	Touchatag / ACR122U
Preis	64 €	40 €	40 €/ 149 €
Bibliotheken	veraltet	vollständig	unvollständig aber ausreichend
Verfügbarkeit	Verfügbar	Verfügbar	Verfügbar

Tabelle 2.3: Übersicht der Lesegeräte

In Tabelle 2.3 werden die betrachteten Lesegeräte verglichen. Zur Implementierung wurde ein *Touchatag* verwendet, da dieser den Autoren schon zur Verfügung stand und alle Kriterien vollständig erfüllt.

[34]Vergleiche [lib13].

Kapitel 3

Entwurf

Zu Beginn dieses Kapitels wird das verwendete Authentifikations-Protokoll erläutert. Im Anschluss werden die verschiedenen Komponenten dieses Protokolls näher betrachtet und die Kommunikation zwischen diesen Komponenten veranschaulicht.

3.1 Das verwendete Protokoll

In diesem Abschnitt wird das verwendete Authentifikations-Protokoll näher betrachtet. Es handelt sich hierbei um ein *Challenge Response*-Verfahren[35], welches RSA als asynchrone Verschlüsselungsmethode verwendet. Die Aufgabe dieses Protokolls ist es, einen Benutzer gegenüber einer Webapplikation zu authentifizeren. Der Benutzer soll dazu eine *Mifare DESFire*, auf der ein RSA-Schlüssel gespeichert ist, verwenden.

Die folgenden Punkte beschreiben den Ablauf des verwendeteten Authentifikations-Protokolls (dargestellt in Tabelle 3.1):

1. Der Benutzer B möchte sich an der Webapplikation W authentifizieren. Dafür sendet er der Webapplikation die UID seines RFID-Transponders.

2. Die Webapplikation verschlüsselt eine Zufallszahl r mit dem zur UID gehörenden öffentlichen Schlüssel und sendet diese Challenge zum Benutzer.

[35] Erläuterung zum *Challenge Response*-Verfahren befinden sich in Kapitel 2.2.

3. Der Benutzer entschlüsselt die Challenge mit dem auf dem RFID-Transponder gespeicherten privaten Schlüssels. Er sendet einen kryptografischen Hashwert des Ergebnisses als Response zur Webapplikation.

4. Die Webapplikation überprüft, ob der Response zur gesendeten Challenge passt und ob ein bestimmtes Zeitlimit eingehalten wurde.

	B		W
1	holt sich die UID des RFID-Transponders	$\xrightarrow{UID}$	
2		$\xleftarrow{challenge}$	Wählen und Verschlüsseln von r
3	Generierung der Response	$\xrightarrow{response}$	
4			Prüfen der Response und des Zeitlimits

Tabelle 3.1: Authentifikations-Protokoll, Quelle: Eigene Darstellung

Bei der Wahl der Zufallszahl sollte darauf geachtet werden, dass diese genügend Entropie besitzt, um die Sicherheit des Systems nicht zu gefährden. Das BSI empfiehlt derzeit, dass die verwendete Zufallszahl für die Erzeugung einer Challenge mindestens 100 Bit Entropie besitzt.[36] Diese Entropie ist definiert als der $-log_2(p)$ der zu testenden Zuffalszahl. p ist hierbei die Wahrscheinlichkeit des wahrscheinlichsten Wertes, der in den Testdaten enthalten ist.

3.2 Komponenten des Protokolls

Das von den Autoren beschriebene Authentifikations-System kann in fünf logische Komponenten aufgeteilt werden. Diese sind die *Mifare DESFire*, der *Daemon*, das *JavaScript*, die *Webseite* und die *Datenbank*. Diese werden in den nachfolgenden Unterkapiteln 3.2.1, 3.2.2 und 3.2.3 näher betrachtet. Das Unterkapitel 3.2.3 *Webapplikationen* beinhaltet das *JavaScript*, die *Webseite* und die *Datenbank*.

[36] Vergleiche [bsi13].

3.2.1 Mifare DESFire

Wie im Kapitel 2.5.1 begründet, verwendet das in dieser Arbeit beschriebene System eine *Mifare DESFire*. Diese dient nicht nur zum Speichern des privaten Schlüssels, sondern regelt auch den Zugriff auf den gespeicherten Schlüssel.

Sie kann als ein Zustandautomat angesehen werden. Jede Aktion versetzt den Zustand und ermöglicht eine andere Aktion. Entsprechend des Zustandes sind nur bestimmte Aktionen möglich. Der Initialisierungszustand ist das Antikollisionsverfahren um den RFID-Transponder eineindeutig ansprechen zu können. Dieses Verfahren existiert bei allen ISO/IEC 14443-3 kompatiblen RFID-Transponder-Typen. Der DESFire-spezifische Initialisierungszustand beginnt nach erfolgreicher Selektierung des RFID-Transponders. In diesem ist die Master-Applikation selektiert. Ein Ende stellt das Verlassen des Bezugsfeldes des RFID-Transponder-Lesers dar oder das explizite Abwählen des RFID-Transponders mit einem entsprechenden Kommando.

Mifare Application Directory

Das Mifare Application Directory der Version 3 ist die Speicherstruktur auf der DESFire. Das Ansprechen von Speicherbereichen erfolgt indirekt. Anstatt wie in früheren MAD-Versionen Speicherblöcke explizit anzusprechen, hat das MAD auf der DESFire ein Dateisystem bestehend aus Applikationen und Dateien, die intern auf die jeweiligen Speicheradressen umgewandelt werden. Der große Vorteil liegt hier bei den dynamischen Dateigrößen. In früheren Versionen lag die Unterteilung in größentechnisch festgelegten Blöcken, die jeweils einen Schreib- und einen Lese-Schlüssel haben. Das beschränkt die maximale Informationsgröße, ohne ein extra Protokoll zur mehrfachen Blocknutzung entwerfen zu müsssen. Zudem verbessert der MAD 3 die Möglichkeit der Verwendung des RFID-Transponders für verschiedene Anwendungszwecke.

Applikationen

Die DESFire kann bis zu 28 Applikationen besitzen, die jeweils eine eineindeutige Application Identifier (AID) haben. Diese AID ist eine 3 Byte lange Zahl, die als

Name der Applikation fungiert. In Applikationen sind Dateien gespeichert, in denen die eigentlichen Informationen stehen. Zur Zugangskontrolle besitzt jede Applikation bis zu 14 verschiedene Schlüssel. Diese sind den Zahlen 0 bis 13 zugeordnet. Der Schlüssel 0 wird Master-Schlüssel genannt und ist in jeder Applikation vorhanden.

Jede DESFire hat eine so genannte Master-Applikation, in ihr befindet sich alle Applikationen. Auf jedem Mifare DESFire-Transponder gibt es nur eine Master-Applikation. Sie regelt die Zugriffsrechte auf die einzelnen Applikationen. Der Schlüssel 0 der Master-Applikation ist, wie bei jeder Applikation, der Master-Schlüssel. Der Master-Schlüssel gibt die Berechtigung weitere Applikationen zu erzeugen, zu löschen oder Zugriffsberechtigungen zu ändern. In der Master-Applikation befinden sich keine weiteren Schlüssel.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	0	0	0	Konfiguration Veränderbar	Master Key benötigt zum Schreiben und Löschen von Applikationen	Freies Applikationen anzeigen	Erlaubtes Ändern des Master Keys

Tabelle 3.2: Master Applikations-Einstellungen, Quelle: Eigene Darstellung nach [Unb10]

Nachfolgend sind die Einstellungen der Master-Applikation dargestellt, die den Zugriff auf den RFID-Transponder regeln. In der Tabelle 3.2 wird eine Übersicht dieser Einstellungen dargestellt. Die Einstellungen werden durch 1 Byte ausgedrückt, bei dem die jeweilige Stelle des Bits die ensprechende Funktion ausdrückt.

Bit 7 - Bit 4: Bits müssen auf 0 gesetzt sein, da hier keine Funktion mit verbunden ist.

Bit 3: Bei gesetztem Wert können die Einstellungen der Master Applikation verändert werden. Im anderen Fall sind die gesetzten Einstellungen unumänderbar festgelegt.

Bit 2: Eine Setzung dieses Bits erlaubt es Applikationen zu erstellen oder zu löschen ohne eine vorherige Authentifikation mit dem Master Schlüssel. Ungesetzt wird eine erfolgreiche Authentifikation mit dem Master Schlüssel vorausgesetzt.

Bit 1: Dieses Bit erlaubt das Abrufen der verfügbaren Applikationen ohne eine vorherige Authentifikation mit dem Master Schlüssel. Sollte das Bit nicht gesetzt sein, wird ein Abfragen der Applikationen verweigert, solange keine erfolgreiche Authentifikation mit dem Master-Schlüssel erfolgte.

Bit 0: Um den Master-Schlüssel verändern zu dürfen, muss dieses Bit gesetzt sein, sonst ist der Schlüssel nicht abänderbar.

Alle Applikationen, außer der Master-Applikation, haben andere Einstellungen, um vorrangig den Zugriff auf die beinhaltenden Dateien zu regeln. In Tabelle 3.3 werden sie dargestellt und nachfolgend beschrieben. Sie werden durch 1 Byte ausgedrückt.

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Schlüssel zum Ändern anderer Schlüssel	Schlüssel zum Ändern anderer Schlüssel	Schlüssel zum Ändern anderer Schlüssel	Schlüssel zum Ändern anderer Schlüssel	Konfiguration änderbar	Daten erzeugen/entfernen ohne Master-Schlüssel	Freie Datei-Angabe	Erlaubtes Ändern des Master-Schlüssels

Tabelle 3.3: Applikations-Einstellungen, Quelle: Eigene Darstellung nach [Unb10]

Bit 7 - Bit 4: Diese Bits spezifizieren den benötigen Schlüssel zum Ändern von Applikationsschlüsseln. Hierbei gelten folgende Werte:
`0x0` bis `0xD`: Diese Werte stehen für den speziellen Schlüssel, der erforderlich ist, um einen Schlüssel abzuändern.
`0xE`: Dieser Wert erlaubt das Ändern des Schlüssels, mit dem sich an der Applikation authentifiziert wird. Es ist demnach nur möglich einen Schlüssel zu ändern, der vorher bekannt ist.
`0xF`: Dieser Wert verbietet das Ändern jedes Schlüssels, ausgenommen des Master-Schlüssels der Applikation.

Bit 3: Bei gesetztem Wert können die Einstellungen der ausgewählten Applikation verändert werden. Im anderen Fall sind die gesetzten Einstellungen unumänderbar festgelegt.

Bit 2: Eine Setzung dieses Bits erlaubt es Daten in der Applikation zu erstellen oder zu löschen ohne eine vorherige Authentifikation mit dem Master-Schlüssel

dieser Applikation. Ungesetzt wird eine erfolgreiche Authentifikation mit dem Master-Schlüssel der Applikation vorausgesetzt.

Bit 1: Dieses Bit erlaubt das Abrufen der verfügbaren Dateien in der Applikation ohne eine vorherige Authentifikation mit dem Master-Schlüssel der Applikation. Sollte das Bit nicht gesetzt sein, wird ein Abfragen der Dateien verweigert, solange keine erfolgreiche Authentifikation mit dem Master-Schlüssel der Applikation erfolgt.

Bit 0: Um den Master-Schlüssel der Applikation verändern zu dürfen, muss dieses Bit gesetzt sein. Anderenfalls ist der Schlüssel nicht abänderbar.

Dateien

Es gibt fünf verschiedene Datei-Varianten bei der DESFire:

- das Standard Data File,
- das Backup Data File,
- das Value File mit Backup,
- das lineare Record File mit Backup und
- das zyklische Record File mit Backup.

Ein *Standard Data File* ist zum Speichern von beliebigen unformatierten Daten. Sie besitzt eine feste Größe, die beim Erzeugen unveränderlich festgelegt wird. Beim Lesen der Daten kann ein beliebiger Anfangs- und Endpunkt gewählt werden. So kann die komplette Datei oder auch nur Teilbereiche ausgelesen werden.

Die *Backup Data File* gleicht der *Standard Data File*. Sie ist nur erweitert, um einen integrierten Mechanismus zur Datensicherung. Dieser Mechanismus sichert die Informationen über eine exakte Kopie der Datei implizit ab. Dadurch belegt eine *Backup Datei* den doppelten Platz.

Um Zahlen zu speichern, werden *Value Files* verwendet. Diese bieten besondere Vorteile für Geldbeträge, da diese spezielle Funktionen zur Inkrementierung und

Dekrementierung haben. Der Wert selber kann nur beim Erstellen der Datei gesetzt werden. Jede weitere Änderung muss duch Hinzugeben oder Abziehen des Wertes erreicht werden. Die Werte müssen sich zwischen einem beim Erzeugen angegebenen Minimal- und Maximalwert befinden.

Lineare und zyklische *Record Files* sind als Protokollierungs-Dateien gedacht. *Lineare Record Files* können je nach Einstellung verschieden oft beschrieben werden. Danach müssen sie zurückgesetzt werden, um sie weiter beschreiben zu können. *Zyklische Record Files* überschreiben sich selbst nach Erreichen der maximalen Eintragsmenge.

Alle Dateitypen haben Zugriffsrechte, die in Abbildung 3.1 dargestellt sind. Die jeweils 4 Bit großen Zahlen beziehen sich auf den benötigten Schlüssel für die jeweilige Option. Hierbei gilt, dass `0x0` bis `0xD` für die spezielle Schlüssel Nummer 0 bis 13 steht. `0xE` bedeutet, dass kein Schlüssel für diese Berechtigung benötigt wird. Bei `0xF` sind die Berechtigungen für niemanden erlaubt.

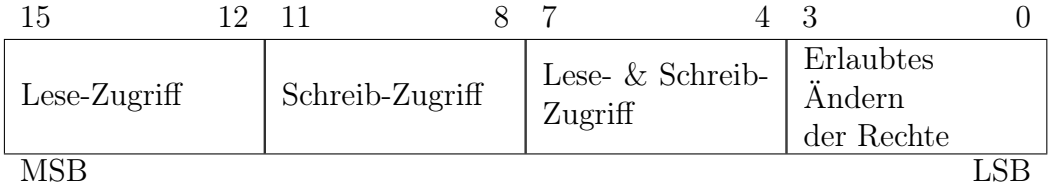


Abbildung 3.1: Datei-Zugriffs-Rechte, Quelle: Eigene Darstellung

3.2.2 Daemon

Der Daemon hat die Aufgabe über das Lesegerät mit dem RFID-Transponder zu kommunizieren. Um den privaten Schlüssel von dem Transponder zu lesen, muss der Daemon sich erst gegenüber diesem authentifizieren. Der hierfür verwendete Schlüssel wird aus einer vom Benutzer eingegebenen Geheimzahl generiert. Das Merken der Geheimzahl soll den Benutzer durch das Anzeigen von bekannten Portraits, die jeweils eine Zahl von 0 bis 9 repräsentieren, erleichtert werden.

Erzeugung des RFID-Transponder Schlüssels

Um zu verhindern, dass die Daten auf dem RFID-Transponder ausgespäht werden können, werden sie durch ein 3DES- oder AES-Authentifikationssystem gesichert.

Wäre dies nicht der Fall, wäre es zum Beispiel möglich den privaten Schlüssel beim Vorbeilaufen an einer Person auszuspähen.

Damit der Benutzer aber keinen 3DES- bzw. AES-Schlüssel erstellen oder sich merken muss, kann dem Daemon eine Zeichenfolge übergeben werden. Aus dieser wird dann der eigentliche Schlüssel errechnet.

Für die Entwicklung des Prototypen sind die Autoren noch einen Schrit weiter gegangen. Das Pinpad ist ein grafisches Benutzerinterface, um die Pinnummer des Benutzers zu erfassen. Da diese Pinnummern häufig vergessen werden, liegt es nahe den Benutzer das Erinnern zu erleichtern. Dies hat nicht nur Vorteile für den Nutzer, es ist auch relevant für die Sicherheit des Systems. Der Benutzer wird dadurch nicht verleitet, das System unsachgemäß zu benutzen.

Das menschliche Gehirn ist darauf trainiert sich tagtäglich Gesichter von Menschen zu merken und wieder zu erkennen. Daher liegt es nahe die Schaltflächen des Pinpads um jeweils ein Portraitfoto zu erweitern.
Es wird das Aufschreiben der Pin erschwert, insbesondere wenn, sich die Schaltflächen bei jedem Aufruf zufällig auf dem Pinpad verteilen und keine Zahlen neben dem Foto angezeigt werden.[37]

Ein für den Prototypen entwickelts Pinpad ist in Abbildung 3.2 gezeigt.

Diese Form der Schlüssel-Generierung bietet ein erhebliches Angriffspotential für eine Brutforce-Attacke. Um die Gefahren eines Bruteforce-Angriffs besser abschätzen zu können, wurden einige einfache Versuche durchgeführt. Diese werden in Kapitel 5.2.1 beschrieben.

Sollte jedoch die Sicherheit gegen einen Bruteforce-Angriff höhere Priorität als die Benutzerfreundlichkeit haben, so besteht die Möglichkeit den Daemon eine beliebig lange Zeichenfolge, als Berechnungsgrundlage für den 3DES- bzw. AES-Schlüssel, zu übermitteln.

[37] Vergleiche [Sch07].

Abbildung 3.2: Pinpad, Quellennachweise im Anhang A

3.2.3 Webapplikation

Die Webapplikation übernimmt die Rolle der Authentifikationsstelle. Sie muss also eine *Challenge* erstellen und einen *Response* auf seine Richtigkeit testen können. Hierfür benötigt die Webapplikation die Möglichkeit RSA-Verschlüsselungen vornehmen zu können. Außerdem benötigt sie eine Möglichkeit Benutzerdaten langfristig zu speichern.

Um eine *Challenge* zu erstellen, muss zuerst eine Zufallszahl erzeugt werden. Diese wird mit dem öffentlichen Schlüssel des zu Authentifizierenden verschlüsselt. Das Ergebnis dieser Verschlüsselung ist die *Challenge*. Der Programmablaufplan zum Erzeugen einer *Challenge* ist in Abbildung 3.3 dargestellt.

Damit die *Response* auf ihre Richtigkeit hin überprüft werden kann, muss die Zufallszahl als Hashwert in eine Datenbank geschrieben werden, da die *Response* ein Hashwert der Zufallszahl ist. Als Hashfunktion können prinzipiell alle kryptografi-

schen Hashfunktionen genutzt werden. Zusätzlich wird ein zugehöriger Zeitstempel in die Datenbank hinterlegt. Damit kann verhindert werden, dass das Lösen der *Challenge* eine bestimmte Zeitdauer überschreitet.

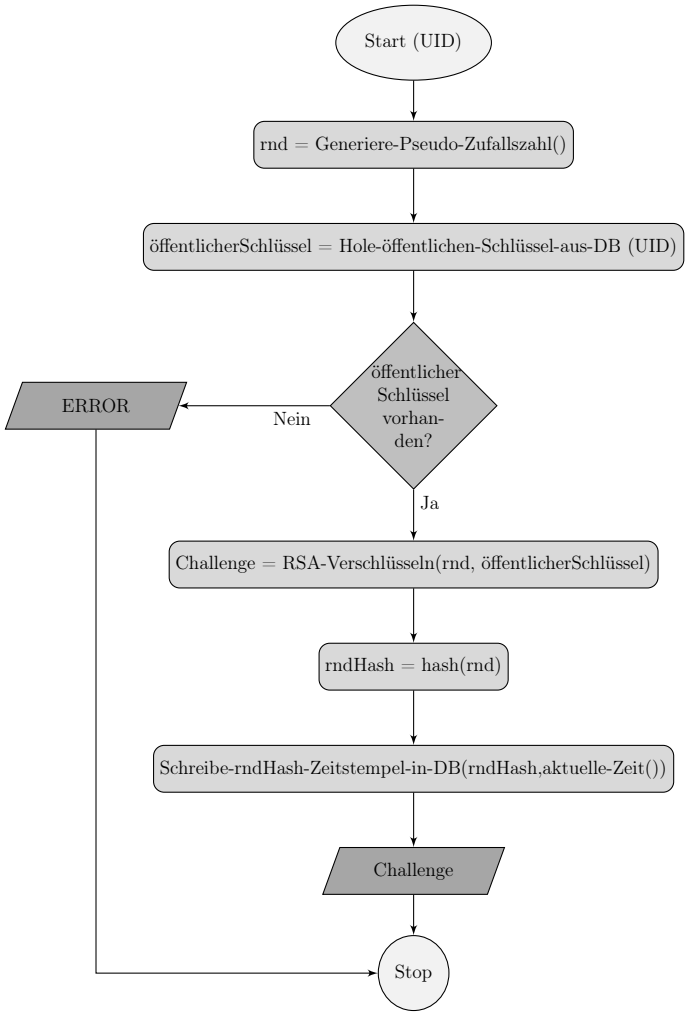


Abbildung 3.3: Programmablaufplan zum Erzeugen einer Challenge, Quelle: Eigene Darstellung

Die Webapplikation muss die *Response* auf ihre Richtigkeit hin überprüfen. Hierbei wird die *Response* mit dem in der Datenbank abgelegten Hashwert der Zufallszahl verglichen. Sind diese identisch, ist die *Challenge* richtig gelöst. Die dafür benötigte Zeit wird mit Hilfe des Zeitstempels in der Datenbank berechnet. Bei Überschreiten des Zeitlimits wird die Authentifikation untersagt. Das Überprüfen eines *Response* wird in dem Diagramm 3.4 dargestellt.

Um einer anderen Webapplikation zu ermöglichen, auf eine erfolgreiche oder nicht-erfolgreiche Authentifikation zu reagieren, gibt es zwei Template-Funktionen: Eine Funktion für den Erfolg und eine für das Scheitern der Authentifikation. Diesen

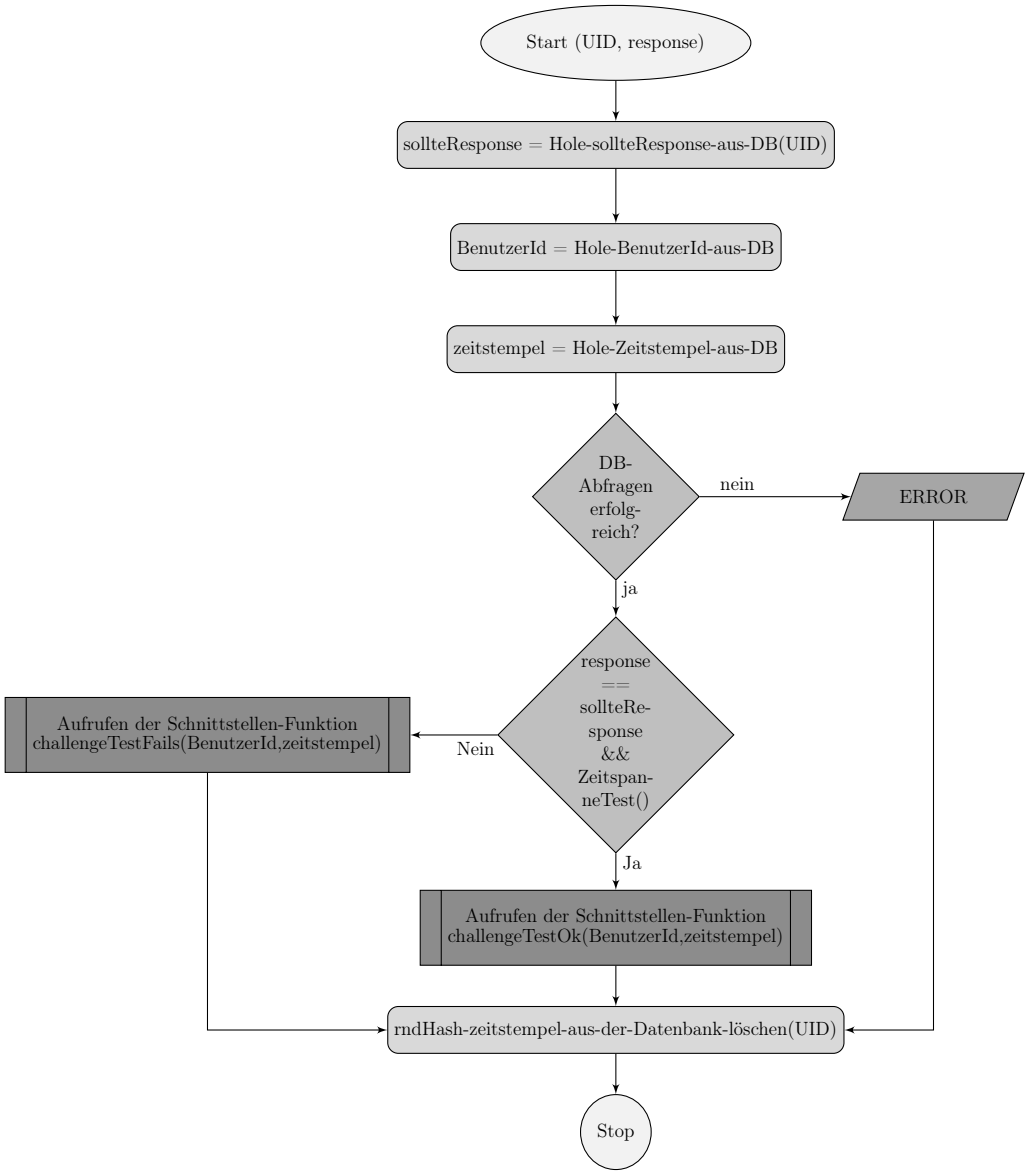


Abbildung 3.4: Programmablaufplan zum Überprüfen einer gelösten Challenge, Quelle: Eigene Darstellung

Funktionen wird eine Benutzerkennung und den Zeitpunkt der erfolgreichen oder nicht-erfolgreichen Authentifikation übergeben. Mit diesen Informationen ist es allen Webapplikation möglich, dass in dieser Arbeit beschriebene Authentifikations-System einzusetzen.

Das Datenbank-Schema ermöglicht die Speicherung eines Benutzers mit seinem Public Key und den gerade laufenden Authentifikationsvorgängen.

3.2.4 Kommunikation zwischen den Komponenten

Die Kommunikation zwischen dem Webserver und dem Daemon stellt ein Problem dar, da ein direktes Ansprechen nicht möglich ist. Hierbei wird ein Umweg über den Browser des Clients gemacht. Das Kommunikationsschema über den Browser wird in der Grafik 3.5 veranschaulicht.

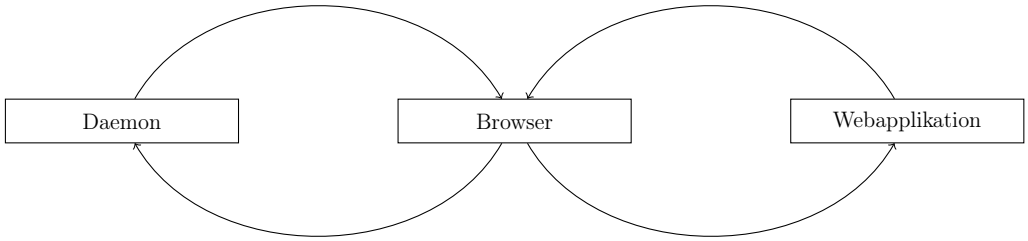


Abbildung 3.5: Vereinfachtes Kommunikationschema, Quelle: Eigene Darstellung

Im Folgenden werden verschiedene Lösungsansätze betrachtet und ihre Eignung für den Prototypen überprüft.

JavaScript und Websockets

Bei der Verwendung von Websockets mit JavaScript ist die Idee, ein JavaScript im Browser des Clients ausführen zu lassen. Dieses Script hat die Aufgabe zwei Websocket-Verbindungen aufzubauen. Eine Verbindung dient zur Verbindung des Webservers mit dem Script, die andere zur Verbindung mit dem Daemon. Dabei muss das Script die Daten vom Webserver an den Daemon weiterleiten und umgekehrt.

Durch dieses Konzept wird ein sehr modularer Aufbau des Systems ermöglicht und somit auch die Verwendung des Daemons von anderen Applikationen. Websockets werden aber noch nicht von allen Browsern unterstützt. Dies ist besonders bei vielen mobilen Browsern der Fall. Da immer mehr Smartphones eine NFC-Schnittstelle besitzen, wäre es besonders erstrebenswert hier eine bessere Lösung zu finden.

Daemon als Browser-Plugin

Bei diesem Lösungsansatz wird die Funktionalität des Daemon als Browser-Plugin implementiert. So kann dieses Plugin mit dem RFID-Transponder-Lesegerät kom-

munizieren und eine Verbindung mit dem Webserver aufbauen.

Bei der Verwendung eines Browser-Plugins können nur Browser verwendet werden, für die es auch ein Browser-Plugin gibt. Dies bedeutet einen immensen Aufwand. Es ist nicht möglich für alle Browser ein Plugin zu entwickeln, da nicht alle Browser diese Erweiterungsmöglichkeit unterstützen.

JavaScript und HTTP

Bei diesem Ansatz stellt der Daemon eine HTTP-Schnittstelle bereit. Die Aufgabe des JavaScript, welches im Browser des Benutzers ausgeführt wird, ist es, HTTP-Anfragen an den Webserver bzw. Daemon zu stellen und die Antwort an die richtige Stelle weiter zu leiten.

Diese Lösung bietet mehrere Vorteile. Der größte Vorteil ist der modulare Aufbau. Es gibt keine neuen Abhängigkeiten zu anderer Software. Des Weiteren kann der Daemon auch hier von einem anderen Programm über das HTTP-Protokoll angesprochen werden und ihn somit auch benutzen. Die Umsetzung in JavaScript gestaltet sich durch das Verwenden von HTTP für alle Verbindungen sehr einfach.

Diese Lösung wurde, aufgrund ihrer Vorteile gegenüber den Websockets und dem Browser-Plugin, für den entwickelten Prototypen verwendet. In der Grafik 3.6 wird eine beispielhafte Kommunikation dargestellt. Die Abbildung B.1 im Anhang zeigt die komplette Kommunikation zwischen den Komponenten bei einem typischen Authentifikationsvorgang.

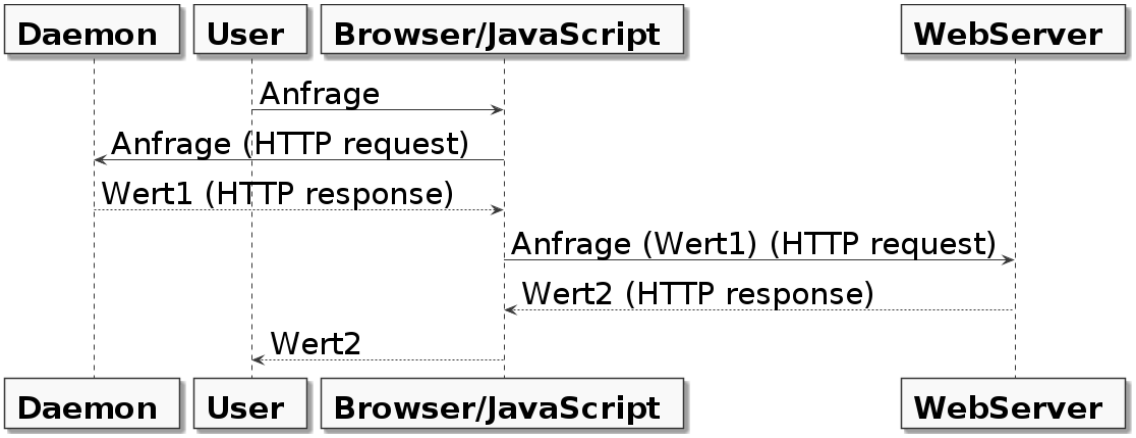


Abbildung 3.6: Beispielhafte Kommunikation der Komponenten mittels JavaScript, Quelle: Eigene Darstellung

Kapitel 4

Implementierung

Dieses Kapitel behandelt die Implementierung. Es wird unterschieden zwischen dem Daemon (in Kapitel 4.1) und der webbasierten Applikation (in Kapitel 4.2). Der Quellcode ist unter `http://code.google.com/p/edda/` verfügbar.

4.1 Daemon

Aufgrund der vorher getroffenen Hardware-Wahl wird die Programmiersprache *C* und die quelloffene Software-Bibliothek *Libnfc* und die darauf aufbauende *Libfreefare* gewählt. Diese Bibliotheken sind komplett für die Kommunikation mit der *Mifare DESFire* und weiterer RFID-Transponder der Firma NXP Semiconductors ausgelegt. Zudem unterstützen sie das, in Kapitel 2.5.2 gewählte, Lesegerät *ACR122U*. Andere Sprachen bieten nicht die entsprechend umfangreichen Bibliotheken.

Der Daemon soll die Kommunikation mit dem Lesegerät steuern, die Anfragen des Servers bearbeiten und beinhaltet die gesamte Client-seitige Programm-Logik in sich. Anfragen des Webservers erhält er über eine Socket-Schnittstelle und eingehende Anfragen im HTTP-Protokoll. Die Anbindung an den Reader und die dazu gehörigen RFID-Transponder wird über die *Libnfc* und die *Libfreefare* ermöglicht.

Zu Beginn wird eine Socketverbindung auf einem Port geöffnet, um die Kommunikation nach außen zu ermöglichen. Der Port ist frei wählbar über den ersten Parameter des Programms. In der Implementierung des Webservers wird von dem Port 12345 ausgegangen. Der Daemon wartet auf eine ankommende Verbindung und erwartet die Einhaltung des HTTP-Protokolls. Die erforderliche Aktion wird durch die

POST-Variable *cmd* ermittelt und eventuell durch weiterer Variablen, soweit erforderlich. *POST* wurde als Übertragungsmethode gewählt, da die *GET*-Methode in ihrer Länge begrenzt ist.[38]

4.1.1 HTTP-Schnittstelle

Im folgenden Kapitel wird die Schnittstelle des Daemon mit dem Browser betrachtet. Da der Server nicht direkt mit dem Daemon reden kann und der Browser als Zwischenstufe verwendet werden muss, existiert diese Schnittstelle, um die hohe Modularität zu erhalten. Wie schon in Kapitel 3.2.4 beschrieben, hat sich diese Art der Schnittstelle als die sinnvollste herausgestellt.

Mit der Implementierung wurde eine für ein UNIX-System funktionierende Lösung erstellt. Die Portierung auf ein anderes Betriebssystem ist möglich, jedoch im Prototyp nicht vorgesehen.

```
1  /* ... */
2
3  echoServPort = atoi(argv[1]);
4
5  if ((servSock = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP)) < 0)
6    DieWithError("socket() failed");
7
8  /* ... */
9
10 if (bind(servSock, (struct sockaddr *) &echoServAddr, sizeof(echoServAddr)) < 0)
11   DieWithError("bind() failed");
12
13 if (listen(servSock, MAXPENDING) < 0)
14   DieWithError("listen() failed");
15
16 for (;;)
17 {
18   clntLen = sizeof(echoClntAddr);
19   clntSock = accept(servSock, (struct sockaddr *) &echoClntAddr, &clntLen);
20   if (clntSock < 0)
21     DieWithError("accept() failed");
22   HandleTCPClient(clntSock);
23 }
```

Listing 4.1: Server-Socket-Implementierung

Im Beispielcode 4.1 werden die wichtigen Elemente des Anlegens eines Sockets und der Hauptprogramm-Schleife gezeigt. Nachfolgend werden die verwendeten Funktionen und einzelne Konstanten beschrieben.

[38] Vergleiche [Mic13].

Mit der Funktion `socket` wird das verwendete Kommunikationsprotokoll spezifiziert. Der Parameter `PF_INET` spezifiziert die entsprechende Internet-Protokoll-Familie. `SOCK_STREAM` definiert einen Stream-Socket und `IPPROTO_TCP` gibt die Verwendung des TCP-Protokolls an.

Durch `bind` wird das Socket-Objekt mit einem Port des Betriebssystems verbunden. Dieser Port ist von dem Zeitpunkt an ausschließlich vom Daemon nutzbar, bis er wieder freigegeben wird.

Die Funktion `listen` initiiert das Akzeptieren von ankommenden Verbindungen und beschränkt die maximale Anzahl gleichzeitig bestehender Verbindungen. Bei erreichter Maximalkapazität von bestehenden Verbindungen werden alle weiteren verworfen, bis wieder eine bestehende beendet wurde. In der Implementation ist mehr als eine Verbindung nicht anwendbar, da das Lesegerät nicht mehrere Anfragen zur gleichen Zeit handieren kann.

Mittels `accept` wird eine ankommende Verbindung bestätigt und kann behandelt werden. Sollte keine Verbindungsanfrage bestehen, wartet das Programm solange bis dies der Fall ist.

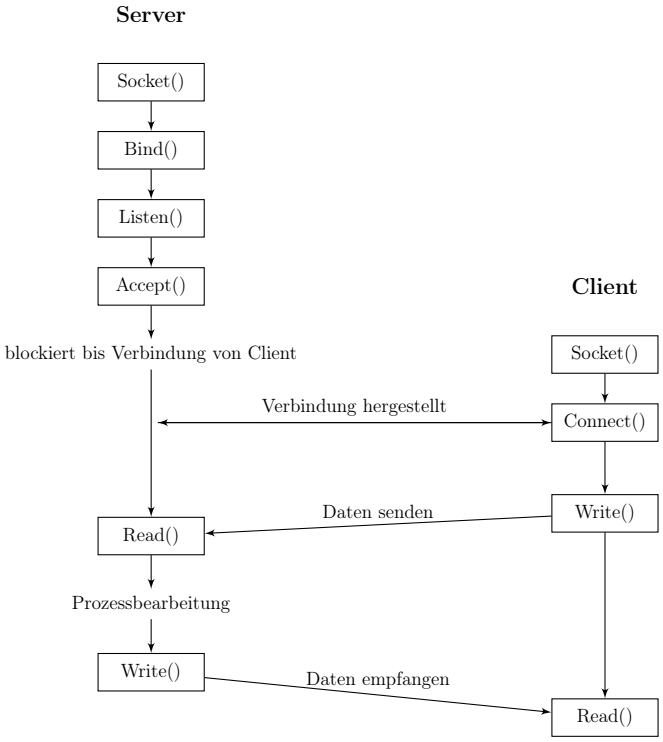


Abbildung 4.1: Systematische Darstellung einer Socket-Verbindung zwischen Server und Client, Quelle: Eigene Darstellung nach [Ste90]

Die in Abbildung 4.1 gezeigte Darstellung einer Server-Client-Verbindung ist eine symbolische Verhaltensweise einer Initialisierung und Kommunikation jedes Servers. In der Implementierung stellt die Prozessbearbeitung den Zugriff auf den RFID-Transponder über das Lesegerät dar. Der Datenaustausch ist das *Challenge Response*-Verfahren.

4.1.2 UID-Abfrage

Auf Anfrage des Webservers sendet der Daemon die UID des RFID-Transponders zurück. Hierbei verbindet sich der Daemon mit dem Lesegrät und fragt alle in der Nähe befindlichen RFID-Transponder ab. Die UID des ersten gefundenen RFID-Transponders, wird dem Webserver als Antwort übertragen.

Eine Auswahl der RFID-Transponder ist hierbei nicht vorgesehen, da sonst der Benutzer die UID des RFID-Transponders genau wissen und zuordnen müsste. Dies würde mehr Informationen bedeuten, die sich der Nutzer merken müsste und genau das soll minimiert werden.

Als Erweiterung wäre jedoch denkbar, alle gefundenen RFID-Transponder in einem Konstrukt zu übermitteln und den Server wählen zu lassen.

Das genaue Verfahren beinhaltet eine Verbindung mit dem Lesegerät und ein anschließendes Antikollisionsverfahren. Bei der Antikollision wird vom Lesegerät eine Anfrage an alle im Feld befindlichen RFID-Transponder nach der UID gestellt. Bei mehreren RFID-Transpondern, die sich im Feld des Lesegerätes befinden, erfolgt eine Überlagerung von Flanken und das Lesegerät erkennt eine ungültige Antwort.

Zur Übertragung wird eine *Manchester*-Kodierung verwendet. Bei dieser Übertragungskodierung wird jedes Bit durch ein Fallen oder Steigen einer Flanke ausgedrückt. Bei zwei unterschiedlichen Bits heben sich die beiden Flanken gegenseitig auf und es wird kein Flankenwechsel getätigt. Dies wird als Kollision gewertet und über ein Binary-Tree-Walk-Verfahren[39] ermittelt.

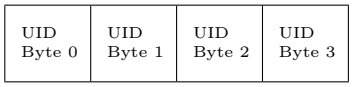
Eine UID kann eine Länge von 4, 7 oder 10 Byte haben. Bei einer ersten Anfrage der UID werden 4 Byte zurück geliefert. Das erste Byte gibt hierbei an ob die empfangene UID vollständig ist, oder nur ein Teil bisher versendet wurde.

[39]Für weiterführende Informationen vergleiche [Gro11].

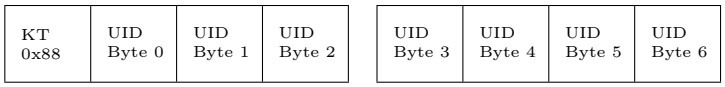
Kaskade-Level-Byte	Kaskade-Level	UID Länge
0x93	1	4
0x95	2	7
0x97	3	10

Tabelle 4.1: Verhältnis des Kaskade-Levels zur UID-Länge

UID Länge 4



UID Länge 7



UID Länge 10

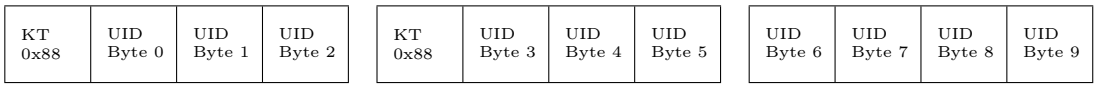


Abbildung 4.2: Kaskade Level und UID, Quelle: Eigene Darstellung nach [ISO99]

Sollte das erste Byte der erhaltenen Nachricht `0x88` betragen, sagt dies eine Weiterführung der UID aus. In diesem Fall wird das Byte Kaskade-Tag genannt, da die UID unvollständig und eine Kaskadierung der Länge erforderlich ist.

Das Lesegerät muss bei unvollständiger UID den verbleibenden Teil erfragen. Um einen expliziten Teil der UID anzusprechen, muss eine eineindeutige Zuordnung des Bereichs zur UID existieren. Dies wird über den Kaskadelevel geregelt. Der Kaskadelevel gibt den angefragten Bereich der UID an und wird über ein Byte ausgedrückt. Eine Zuordnung der UID-Länge zum Kaskadelevel und dem entsprechenden Kaskadelevel-Byte ist in Tabelle 4.1 dargestellt.

Falls im Kaskadelevel 2 das erste Byte den Kaskade-Tag `0x88` beinhaltet, folgt noch ein weiterer Teil der UID. Dies kann theoretisch unendlich lange weitergeführt werden und ermöglicht somit eine unendlich lange UID. Bildhaft ist dies in Abbildung 4.3 dargestellt.

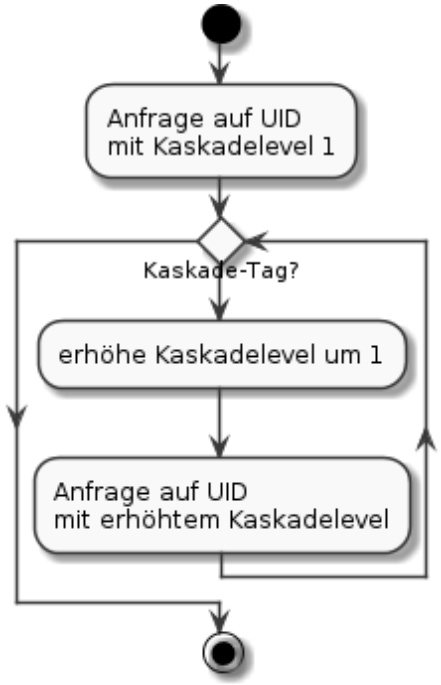


Abbildung 4.3: Aktivitätsdiagramm der UID-Abfrage, Quelle: Eigene Darstellung

Eine Begrenzung liegt dennoch vor, da das Lesegerät explizit den Kaskadelevel beim Anfragen angeben muss und dieser 1 Byte umfasst. ISO 14443-3[40] legt die maximale Tiefe bei drei fest, was eine maximale Länge von 10 Byte ermöglicht. Alle somit möglichen Arten einer UID werden in Abbildung 4.2 gezeigt.

Die *Mifare DESFire* unterstützt zudem eine Random-UID-Funktion. Diese hat datenschutzrechtlich gesehen Vorteile, da die eineindeutige Zuordnung von einer Person zu einem RFID-Transponder verhindert. Das ist aber genau die Zuordnung, die die Grundlage für eine Authentifizierung bildet, weil der Webserver einen Verweis von der UID zum öffentlichen Schlüssel benötigt.

4.1.3 Verifizierung

Der Schritt der Verifizierung dient der Authentifizierung des Benutzers am Server. Hierzu erhält der Daemon eine base64-kodierte und mit dem öffentlichen Schlüssel des Benutzers verschlüsselte Zufallszahl. Diese wird zur weiteren Verwendung dechiffriert.

[40][ISO99]

Das bedeutet, die empfangene Nachricht wird base64-enkodiert und mit dem privaten Schlüssel des Benutzers wird die Zufallszahl entschlüsselt. Der private Schlüssel wird vorher von dem RFID-Transponder gelesen. Vor dem Versenden wird die entschlüsselte Zufallszahl SHA1 gehasht und dieses Ergebnis wird im base64-Format übermittelt.

Um von dem RFID-Transponder lesen zu können, wird ein 3DES-Zugangsschlüssel benötigt. Dies wird in Bild B.1 dargestellt. Der Zugangsschlüssel wird über die Rückgabe des Pinpads und der OpenSSL-Funktion `EVP_BytesToKey` erzeugt. Der verwendete Algorithmus wird anhand des Verschlüsselungsnamens ermittelt, wie im Beispielcode 4.2 dargestellt.

```
1  unsigned char* createDesKey(char *password , char *keyType)
2  {
3      const EVP_CIPHER *cipher;
4      const EVP_MD *dgst = NULL;
5      unsigned char *key = calloc(1,EVP_MAX_KEY_LENGTH), iv[EVP_MAX_IV_LENGTH];
6      const unsigned char *salt = NULL;
7
8      OpenSSL_add_all_algorithms();
9
10     cipher = EVP_get_cipherbyname(keyType); // "for 3 key DES use des-ede3"
11     if(!cipher)
12     {
13       fprintf(stderr, "no such cipher\n");
14       return "1";
15     }
16
17     OpenSSL_add_all_digests();
18
19     dgst=EVP_get_digestbyname("sha1");
20     if(!dgst)
21     {
22       fprintf(stderr, "no such digest\n");
23       return "1";
24     }
25
26     if(!EVP_BytesToKey(cipher, dgst, salt, (unsigned char *) password, strlen(password), 1, key,
           iv))
27     {
28         fprintf(stderr, "EVP_BytesToKey failed\n");
29         return "1";
30     }
31
32     return key;
33 }
```

Listing 4.2: Verschlüsselung entsprechend dem übergebenen Algorithmusnamen

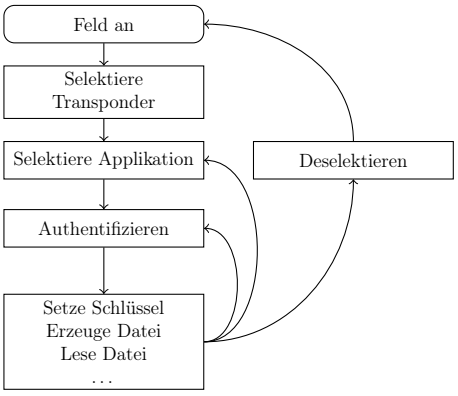


Abbildung 4.4: DESFire-Flussdiagramm, Quelle: Eigene Darstellung nach [Unb10]

4.1.4 DESFire-Funktionen

Authentifizieren an dem RFID-Transponder

Das Authentifizieren ermöglicht den Zugriff auf die in der Applikation beinhaltenden Daten. Wie in Abbildung 4.4 dargestellt, muss eine Applikation selektiert und sich an dieser dann authentifiziert werden. Die Wahl des zu verwendenden Schlüssels erfolgt bei der Authentifizierung. Je nach Einstellung wird das Abfragen der Applikationen oder Daten auch ohne Schlüssel gewährt.

Auf einem fabrikneuen *Mifare DESFire*-Transponder ist der Master-Schlüssel jeder Applikation ein DES-Schlüssel, der komplett aus Nullen besteht. In der Implementierung wurde der Master-Schlüssel durch einen über das Pinpad erzeugten 3DES-Schlüssel ersetzt. Die 3DES-Verschlüsselung wurde gewählt, da sowohl die *Mifare DESFire* als auch die *Mifare DESFire EV1* dies unterstützen. Durch eine leichte Änderung wäre ebenso eine AES-Verschlüsselung verwendbar, dafür kann aber nur die *Mifare DESFire EV1* verwendet werden.

Anlegen einer Applikation

Die Funktion `mifare_desfire_create_application` aus der *Libfreefare* wird zum Anlegen einer Applikation verwendet. Diese Funktion benötigt die Informationen des Applikationsnamens und der Zugriffsrechte. Der in der Arbeit verwendete Name der Applikation ist *010203*. Der Name ist für die Prototyp-Implementierung fest gewählt und könnte später dynamisch gewählt werden.

Die beim Erzeugen gesetzten Zugriffsrechte sind `0xED`. Wie der Tabelle 3.3 zu entnehmen ist, bedeutet dies, dass sich mit dem zu ändernden Schlüssel authentifiziert werden muss. Zudem ist durch das zweite Halbbyte der freie Dateizugriff untersagt.

Die Einstellungen der Master-Applikation wurden nicht verändert, da der RFID-Transponder für mehrere Applikation nutzbar sein soll.

Anlegen einer Datei

In den Dateien werden nacheinander die Informationen der BIGNUMS[41] gespeichert.

```
1 struct bignum_st
2 {
3   BN_ULONG *d;
4   int top;
5   int dmax;
6   int neg;
7   int flags;
8 };
```

Listing 4.3: BIGNUM-Struktur von *OpenSSL*

Die BIGNUM-Struktur ist zum Ausdrücken von Zahlen, die die Kapazität der Registergröße übersteigen. Eine BIGNUM-Struktur[42] steht dabei, für genau eine Zahl. Im Codebeispiel 4.3 ist die in der *OpenSSL*-Bibliothek für BIGNUMS verwendete Struktur dargestellt.[43] Die Hauptinformationen über die Zahl befinden sich im Feld `d`. Der im Feld `d` verwendete Datentyp `BN_ULONG` ist `size_t`, das ist die genaue Registergröße des Computers.[44] Die weiteren Strukturteile beschreiben die Interpretationen des Feldes.

Jede BIGNUM wird durch fünf Dateien dargestellt, davon sind vier *Value Files* und eine ist ein *Standard Data File*.

[41] BIGNUM ist eine in der *OpenSSL*-Bibliothek definierte Struktur.
[42] `d` steht für das Feld von Binärinformationen. `top` ist der Index vom letzten Element im Feld `d`. `dmax` ist die Länge des Feldes `d`. `neg` ist ein boolscher Indikator, ob die Zahl positiv oder negativ ist. `flags` ist der Bereich für besondere Kennzeichnungen.
[43] Vergleiche [ope13a].
[44] Vergleiche [fef06].

Value Files werden beim Erstellen auf den entsprechenden Wert gesetzt. Dies geschieht über die Funktion `mifare_desfire_create_value_file`. Sie dienen zum Speichern numerischer Werte. Die gespeicherten Informationen können nur über Inkrementierung oder Dekrementierung verändert werden.

Die Binär-Informationen der BIGNUMs werden in *Standard Data Files* gespeichert. Hierzu wird die Daten mit der Funktion `mifare_desfire_create_std_data_file` erzeugt und danach in die Datei geschrieben.

Alle Daten werden bei der Erzeugung auf einen verschlüsselten Übertragungsmodus eingestellt. Dies geschieht über die beim Erstellen jeder Datei gesetzten Konstante `MDCM_ENCIPHERED`. In dem Prototyp erlauben die Datei-Zugriffsrechte das völlige Bearbeiten der Dateien und deren Zugriffsberechtigung, da die überliegende Applikation durch den Schlüssel gesichert ist.

Privaten Schlüssel speichern

Der für den privaten Schlüssel zur Verfügung stehende Platz ist durch das Speichervolumen des RFID-Transponders begrenzt. Für eine *Mifare DESFire* bedeutet das im geringsten Fall eine Kapazität von 2 KiB.

Im Folgenden wird das Speichern im kodierten PEM-Format mit dem unkodierten Speichern der Informationen aus den BIGNUM-Strukturen verglichen. Aufgrund der starken Begrenzung von Speicherressourcen ist es wichtig so wenig wie möglich davon zu verbrauchen.

```
1  struct rsa_st
2  {
3    BIGNUM *n;
4    BIGNUM *e;
5    BIGNUM *d;
6    BIGNUM *p;
7    BIGNUM *q;
8    BIGNUM *dmp1;
9    BIGNUM *dmq1;
10   BIGNUM *iqmp;
11   // ...
12 };
```

Listing 4.4: RSA-Struktur von *OpenSSL*

Die RSA-Struktur[45] entstammt aus der *OpenSSL*-Bibliothek und ist im Beispielcode 4.4 teilweise dargestellt. Die für den privaten Schlüssel notwendigen Bestandteile sind die BIGNUMS `n`, `e` und `d`. Alle weiteren sind optional und werden, wenn sie nicht vorhanden sind, automatisch ausgerechnet.[46]

Das PEM-Format ist ein base64-kodiertes System mit markiertem Anfang und Ende (markiert mit `---BEGIN CERTIFICATE---` bzw. `---END CERTIFICATE---`). Zwischen diesen Bereichsmarkierungen sind die Informationen zu dem Schlüssel enthalten.

Schlüssellänge	**PEM-Format-Größe**	**BIGNUM-Größe**
1.024 Bit	891 Byte	436 Byte
2.048 Bit	1.679 Byte	820 Byte
4.096 Bit	3.243 Byte	1.588 Byte
8.192 Bit	6.363 Byte	3.124 Byte
16.384 Bit	12.603 Byte	6.196 Byte
32.768 Bit	25.083 Byte	1.2340 Byte

Tabelle 4.2: Größenvergleich zwischen PEM-Format und BIGNUMs

Die Tabelle 4.2 zeigt das Verhältnis der verschiedenen Speicherformen zu der Größe des verwendeten RSA-Schlüssels.

```
1  int writeBignum(MifareTag tag, MifareDESFireAID aid,const BIGNUM const *bignum,int start)
2  {
3    mifare_desfire_create_value_file (tag, start++, MDCM_ENCIPHERED, 0xeeee, 0, bignum->top+1,
         bignum->top, 0);
4    mifare_desfire_create_value_file (tag, start++, MDCM_ENCIPHERED, 0xeeee, 0, bignum->dmax+1,
         bignum->dmax, 0);
5    mifare_desfire_create_value_file (tag, start++, MDCM_ENCIPHERED, 0xeeee, 0, bignum->neg+1,
         bignum->neg, 0);
6    mifare_desfire_create_value_file (tag, start++, MDCM_ENCIPHERED, 0xeeee, 0, bignum->flags+1,
         bignum->flags, 0);
7
8    mifare_desfire_create_std_data_file(tag, start, MDCM_ENCIPHERED, 0x0000,sizeof(BN_ULONG)*(
         bignum->dmax));
9    mifare_desfire_write_data(tag,start,0,sizeof(BN_ULONG)*(bignum->dmax),bignum->d);
10
11   return 0;
12 }
```

Listing 4.5: Die Funktion `write_Bignum`

[45] `n` ist der öffentliche Modulus. `e` ist der öffentliche und `d` der private Exponent. `p` und `q` sind die geheimen Primfaktoren. `dmp1` berechnet sich aus $d \mod (p-1)$ und `dmq1` aus $d \mod (q-1)$. `iqmp` berechnet sich aus $q^{-1} \mod p$.

[46] Vergleiche [ope13b].

Wie im Beispiel-Quellcode 4.5 gezeigt, erwartet die Funktion `writeBignum`:

- einen Verweis auf den RFID-Transponder,
- den Namen der Applikation (die AID),
- die zu speichernde BIGNUM und
- den Startwert der Dateinummer.

Nacheinander werden die einzelnen Bereiche der BIGNUM-Struktur in die Daten geschrieben.

Privaten Schlüssel lesen

Zum Lesen des privaten Schlüssels wird eine Verbindung mit dem Lesegerät hergestellt. Es wird der erste sich in Reichweite befindliche RFID-Transponder gewählt und auf die Applikation `010203` zugegriffen. Da die Implementierung prototypisch ist, wurde auf eine Auswahl des RFID-Transponders, sowie der Applikation verzichtet. Danach wird jede benötigte BIGNUM einzeln ausgelesen.

```
1  nfc_device* getRfidDevice()
2  {
3    size_t device_count = 0;
4    nfc_connstring devices[8];
5    nfc_context *context;
6
7    nfc_init(&context);
8
9    device_count = nfc_list_devices(context,devices,8);
10
11   if (device_count > 0)
12     return nfc_open (context,devices[0]);
13   else
14   {
15     return NULL;
16   }
17 }
```

Listing 4.6: Herstellen der Verbindung mit dem Lesegerät

Der Beispiel-Quellcode 4.6 zeigt die Herstellung der Verbindung mit dem Lesegerät. Im positiven Fall wird eine gültige Adresse zurück gegeben. Dies geschieht über die Funktionen der *Libnfc*.

```
1  RSA* readRSA()
2  {
3    /* ... */
4
5    device = getRfidDevice();
6
7    if (!device)
8      return NULL;
9
10   tag = freefare_get_tags(device);
11
12   mifare_desfire_connect (tag[0]);
13
14   aid = mifare_desfire_aid_new(AID_NUMBER);
15
16   mifare_desfire_select_application (tag[0], aid);
17
18   if (authApplication(tag[0], keyNumber) < 0)
19   {
20     nfc_close(device);
21     return NULL;
22   }
23
24   rsa = RSA_new();
25   rsa->n = BN_new();
26   rsa->d = BN_new();
27   rsa->e = BN_new();
28
29   readBignum(tag[0],aid,rsa->n,0);
30   readBignum(tag[0],aid,rsa->d,5);
31   readBignum(tag[0],aid,rsa->e,10);
32
33   nfc_close(device);
34
35   return rsa;
36 }
```

Listing 4.7: RSA-Schlüssel auslesen – Teil 1

Nach der erfolgreichen Verbindung mit dem Lesegerät erfolgt das Selektieren des zu verwendenden RFID-Transponders. Dieser Bereich wird im Code-Beispiel 4.7 gezeigt. Das komplette Antikollisions- und Selektierungsprotokoll wird von den Bibliotheken *Libnfc* und *Libfreefare* übernommen. Um den Transponder anzusprechen, wird die Funktion `mifare_desfire_connect` aufgerufen. Die verwendete Applikation wird mit der Funktion `mifare_desfire_select_application` ausgewählt, die ebenfalls von der *Libfreefare* bereitgestellt wird.

```
1  int authApplication(MifareTag tag,int keyNumber)
2  {
3    MifareDESFireKey key = mifare_desfire_3des_key_new( createDesKey(makePinpad(),"des-ede"));
4    return mifare_desfire_authenticate(tag, keyNumber, key);
5  }
```

Listing 4.8: Authentifikation an der Applikation des RFID-Transponders

Vor dem Authentifizieren muss der Applikations-Schlüssel über das Pinpad erzeugt werden. Die implementierte Funktion ist im Beispielcode 4.8 dargestellt. Die eigentliche Authentifizierung an dem RFID-Transponder wird über die *Libfreefare*-Funktion

`mifare_desfire_authenticate` geregelt.

```
28
29   readBignum(tag[0],aid,rsa->n,0);
30   readBignum(tag[0],aid,rsa->d,5);
31   readBignum(tag[0],aid,rsa->e,10);
32
33   nfc_close(device);
34
35   return rsa;
36 }
```

Listing 4.9: RSA-Schlüssel auslesen – Teil 2

Nach erfolgreicher Authentifizierung an der Applikation kann auf die beinhaltenden Daten zugegriffen und der private Schlüssel ausgelesen werden. Dieser Teil ist im Beispielcode 4.9 zu sehen. Die grundlegende RSA-Struktur wird erstellt, die notwendigen BIGNUMs werden aus dem Transponder ausgelesen und in die Struktur geschrieben.

4.1.5 Pinpad

Für die Implementierung des Pinpads wurde das *GIMP Toolkit (GTK)* benutzt. *GTK* ist eine Komponenten-Bibliothek, die es ermöglicht in *C* grafische Benutzer-Oberflächen zu programmieren. *GTK* ist für Linux, Windows und MAC OS erhältlich.[47]

Das Pinpad ist als *C*-Funktion implementiert, die vor dem Authentifizieren des RFID-Transponders vom Daemon aufgerufen wird. Sie liefert eine Zeichenkette zurück, die den PIN enthält.

Zur Eingabe stehen dem Nutzer Druckknöpfe zur Verfügung.[48] In der Implementierung ist die Pinlänge auf vier beschränkt. Das Pinpad enthält selbst ein Feld aus vier Zahlen, die bei der Initialisierung auf `0` gesetzt sind. Mit Eingabe der ersten Zahl wird auch die erste Stelle im Feld überschrieben, mit der zweiten Zahl die zweite Stelle im Feld und weiterführend. Ab der fünften eingegebenen Zahl wird der Input verworfen und das Zahlenfeld ändert sich nicht mehr. Bei Bestätigung der Zahlenkombination wird das Zahlenfeld in eine Zeichenkette umgewandelt und zurück gegeben.

[47] Vergleiche [Tea].
[48] Siehe Abbildung 3.2.

4.2 Webbasierte Applikation

4.2.1 Wahl der Programmiersprache

Für die Implementierung der Webapplikation werden einige Anforderungen an die verwendete Programmiersprache gestellt. Sie muss für das Entwickeln einer Webapplikation geeignet sein. Das bedeutet, es sollten Module und Frameworks für die Webentwicklung bereitstehen. Des Weiteren ist die Benutzung von *OpenSSL*, oder einer anderen zu *OpenSSL* kompatiblen Bibliothek, erforderlich und muss für die ensprechende Programmiersprache vorhanden sein.

Die Verwendung eines bestimmten Datenbankmanagementsystem (DBMS) sollte nicht von der Programmiersprache beinflusst werden. Daher bietet sich eine ODBC-Unterstützung[49] an.

Vergleicht man beispielsweise *PHP*, *Python* und *Ruby* miteinander, stellt man schnell fest, dass jede dieser Programmiersprachen für die Implementierung der Webapplikation des Prototyps geeignet sind.

PHP bietet standardmäßig eine *OpenSSL*-Unterstützung an. Diese ist zwar nicht sehr umfangreich, kann aber leicht mit der Bibliothek *phpseclib* erweitert werden.[50]
In *Python* geschieht dies über die Bibliothek *M2Crypto*.[51]
In *Ruby* gibt es auch ein *OpenSSL*-Modul.

Bei der Unterstützung von ODBC gibt es keine nennenswerten Unterschiede. Alle drei Programmiersprachen unterstützen ODBC oder können darum erweitert werden. Ein Unterscheidungsmerkmal ist jedoch die Verbreitung. Laut *Tibo Software*[52] ist *PHP* die am meisten verwendete Sprache der drei. Danach folgen *Python* und *Ruby*. Daher wurde sich für die Umsetzung des Prototypen für *PHP* entschieden.

[49] Open Database Connectivity (ODBC)
[50] Vergleiche [php].
[51] Vergleiche [Fou].
[52] Vergleiche [BV].

4.2.2 Erzeugung der Challenge

Das `createChallenge.php`-Skript wird von dem JavaScript im Browser des Benutzers mit der UID des RFID-Transponders aufgerufen. Daraufhin wird, wie in Listing 4.10 von Zeile 1 bis 3 gezeigt, der entsprechende öffentliche Schlüssel aus der Datenbank geholt und in die Variable `pubKey` gespeichert. War dies nicht erfolgreich, wird in Zeile 6 und 7 das Programm abgebrochen.

Die in Listing 4.10 und 4.14 gezeigt Implementierung ist durch das direkte Einfügen einer übergebenen Variable in das SQL-Statement, anfällig für eine *SQL-INJECTION*. Dies kann durch den Einsatz geeigneter Filtertechniken, die hier nicht weiter erläutert werden, verhindert werden.

```
1 <?php
2 $uid = substr($_POST["uid"],0,-2);
3 $sqlCommand = 'select publicKey from publickey where userId=(select userId from uid where uid="'
      . $uid .'");';
4 $pubKey = substr(odbcSqlQuery($sqlCommand,1) , 1 , -1);
5
6 if($pubKey == NULL)
7   die("UID NOT FOUND");
```

Listing 4.10: *PHP*-createChallenge – Teil 1

In den Zeilen 8 und 9 wird die *Challenge* erzeugt. Die Zufallszahl wird mit Hilfe der *PHP*-Funktion `openssl_random_pseudo_bytes` erzeugt. Dieser Funktion werden zwei Argumente übergeben, zum einen die Länge der Pseudozufallszahl in Byte und zum anderen eine Variable, in der die Funktion die Qualität der Pseudozufallszahl hinterlegen kann. Um sicherzustellen, dass nur bessere Pseudozufallszahlen für die *Challenge* verwendet werden, wird wie in Listing 4.12 zu sehen ist, die Funktion so oft aufgerufen bis eine bessere Pseudozufallszahl gefunden wurde.[53]
Die so gefundene Zufallszahl wird anschließend in Zeile 9 des Listings 4.11 mit dem öffentlichen Schlüssel des Benutzers Verschlüsselt.

```
8 $rnd = getRND(100);
9 $challenge = encryptChallenge($pubKey,$rnd);
```

Listing 4.11: *PHP*-createChallenge – Teil 2

[53] Vergleiche [Gro13].

```
1  <?php
2  function getRnd ($length)
3    {
4    $cstrong = false;
5
6    while ($cstrong == false)
7      {
8      $bytes = openssl_random_pseudo_bytes($length, $cstrong);
9      }
10   return($bytes);
11   }
12 ?>
```

Listing 4.12: *PHP*-Definition der Funktion `genRnd()`

In den Zeilen 10, 11 und 12, des listings 4.13, wird ein base64-kodierter SHA1-Hash in die Datenbank geschrieben. Danach wird die erzeugte *Challenge* ausgegeben und somit als HTTP-Antwort an das JavaScript im Browser des Benutzers gesendet.

```
10 $ssessionId = uniqid(true,"FHSI");
11 $sqlCommand = 'INSERT INTO currentAuth (sessionId , challenge , userId) VALUES ("' . $sessionId .
       '" , "' . base64_encode(sha1($rnd,true)) . '" , (SELECT userId from uid WHERE uid="' . $uid
        . '") );';
12 odbcSqlQuery($sqlCommand,1);
13
14 echo $challenge;
15 ?>
```

Listing 4.13: *PHP*-createChallenge – Teil 3

4.2.3 Überprüfen der Response

Das JavaScript sendet die durch den Daemon berechnete *Response* an den Webserver. Dies gelingt durch einen HTTP bzw. HTTPS-Request. Dabei wird dem *PHP*-Script `checkChallenge.php` die UID des Benutzers und die *Response* übergeben.

Bei der Erzeugung der *Challenge* wird ein Datenbankeintrag geschrieben. Dieser wird mit der Hilfe der UID gefunden. Der Eintrag enthält einen Hashwert der Zufallszahl, der für die Erzeugung der *Challenge* verwendet wurde. Dieses geschiet in den Zeilen 6 bis 8, des Listings 4.14. Zusetzlich wird, in Zeile 10 bis 12, die externe Benutzer-ID, der zugehöigen UID aus der Datenbank gelesen. Sie dient später für die identifikation des Benutzers gegenüber eines exterenen Systems. Damit auch die, für den Authentifikationsversuch, benötigte Zeit überprüft werden kann, wird diese in den Zeilen 14 bis 16 aus der Datenbank gelesen.

```
<?php
  session_start();
  $uid = substr($_POST["uid"],0,-2);
  $solvedChallenge = base64_decode(substr($_POST["solvedChallenge"],0 , -2 )) ;

  $shouldSolvedChallenge = getShouldSolvedChallenge($uid);
  if($shouldSolvedChallenge == NULL)
    die("NO CURRENTAUTH ENTRY FOR IS UID");

  $externalUserId = getExternalUserId($uid);
  if($externalUserId == NULL)
    die("NO EXTERNALUSERID FOR THIS UID");

  $authRequestTime = getAuthRequestTime($uid);
  if($externalUserId == NULL)
    die("NO AUTH REQUEST TIME FOR THIS UID");
```

Listing 4.14: *PHP*-checkChallenge – Teil 1

Ist der Hashwert gleich der *response* und wurde das gegebene Zeiltlimit für einen Authentifikationsversuch nicht überschritten, so ist die Authentifikation erfolgreich. Daraufhin wird die Schnittstellenfunktion `challengeTestOk()` für eine erfolgreiche Authentifikation aufgerufen.

Passt die *Response* nicht zu der *Challenge*, oder wurde das Zeitlimit überschritten, wird die Funktion `challengeTestFaild()` aufgerufen. Hierbei wurde zu den in Kapitel 3.2.3 beschriebenen Übergabe-Variablen ein neuer Parameter `seesion_id()` eingeführt. Die Implementierung dieses Abschnittes ist in Listing 4.15 zu sehen.

```
 if ($solvedChallenge == $shouldSolvedChallenge && $maxAuthTime >= (time() - $authRequestTime) )
    {
    challengeTestOk($externalUserId , session_id(), $authRequestTime);
    }
  else
    {
    challengeTestFails($externalUserId  , session_id(), $authRequestTime);
    }
```

Listing 4.15: *PHP*-checkChallenge – Teil 2

Bevor der Authentifikations-Vorgang abgeschlossen werden kann, muss noch der Datenbankeintrag, der bei der Erstellung der *Challenge* angelegt wurde, aus der Datenbank entfernt werden. Dies zeigt das Listing 4.16.

```
$sqlCommand = 'delete from currentAuth where userId=(select userId from uid where uid="' . $uid
    . '");';
  odbcSqlQuery($sqlCommand,1);
```

Listing 4.16: *PHP*-checkChallenge – Teil 3

4.2.4 JavaScript-Kommunikation

Wie schon in Kapitel 3.2.4 beschrieben, dient das JavaScript der Kommunikation zwischen der Webapplikation und dem Daemon. Im folgenden Abschnitt soll die Funktionsweise an kleinen Quellcodeteilen und einer Beschreibung dieser erläutert werden.

Listing 4.17 zeigt wie das JavaScript die *Challenge* von der Webapplikation anfordert. Dafür wird in Zeile 1 bis 5 die UID vom Daemon geholt und in Zeile 7 bis 11 die Anfrage und die UID an die Webapplikation gesendet. Zur Übertragung wird ein *XMLHttpRequest* verwendet, welches eine API zur Übertragung von Daten über das HTTP-Protokoll bereitstellt.[54]

```
1  // UID vom Daemon holen
2  var httpUid = new XMLHttpRequest();
3  httpUid.open('POST','http://localhost:12345',false);
4  httpUid.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
5  httpUid.send('cmd='+'uid');
6
7  // Challenge von der Webapplikation anfordern mit Hilfe der UID
8  var httpCreateChallenge = new XMLHttpRequest();
9  httpCreateChallenge.open('POST','createChallenge.php',false);
10 httpCreateChallenge.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
11 httpCreateChallenge.send('uid=' + httpUid.responseText);
```

Listing 4.17: JavaScript-Anfordern der *Challenge*

Listing 4.18 zeigt das Versenden der *Challenge* an den Daemon und das Weiterleiten der *Response* an die Webapplikation. In den Zeilen 11 bis 15 wird die Challenge an den Daemon versendet und in den Zeile 17 bis 21 wird der vom Daemon erzeugte *Response* an die Webapplikation versendet.

```
12 // Challenge an den Daemon versenden
13 var httpSolvedChallenge = new XMLHttpRequest();
14 httpSolvedChallenge.open('POST','http://localhost:12345',false);
15 httpSolvedChallenge.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
16 httpSolvedChallenge.send('cmd='+'chg' + '&' +'rnd=' + httpCreateChallenge.responseText + '!');
17
18 // Response an die Webapplikation senden
19 var httpCheckChallenge = new XMLHttpRequest();
20 httpCheckChallenge.open('POST','checkChallenge.php',false);
21 httpCheckChallenge.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
22 httpCheckChallenge.send('solvedChallenge=' +  encodeURIComponent(httpSolvedChallenge.responseText
       ) + '&' + 'uid=' + httpUid.responseText);
```

Listing 4.18: JavaScript-Versenden des *Response*

[54]Für weitere Informationen vergleiche [ASSvK12].

4.2.5 Datenbank

Für die Implementierung der Datenbank wurde die ODBC-Datenbank-Schnittstelle verwendet. ODBC bietet eine API, welche es ermöglicht eine Anwendung, in diesem Fall den Prototypen, relativ unabhängig vom verwendeten Datenbankmanagementsystem zu entwickeln. Voraussetzung hierfür ist das Vorhandensein eines ODBC-Treibers für das verwendete DBMS.

Als DBMS wird bei der Implementierung der Prototypen *MySQL* verwendet. Dies ist durch die Verwendung der ODBC-Schnittstelle jederzeit mit geringem Aufwand durch ein anderes DBMS ersetzbar.

Das Datenbank-Schema besteht aus den folgenden Tabellen:

- user,
- uid,
- publickey und
- currentAuth.

Die Tabelle *user* dient zur Verknüpfung zweier Benutzer-IDs. Die eine ist für die Verwendung eines externen Systems, welches das Authentifikations-System nutzt, bestimmt und die andere ist für die Verwendung im Prototypen.

Die Tabelle *uid* verbindet eine Benutzerkennung mit der zu dem Benutzer gehörenden UID des RFID-Transponders.

Die Tabelle *publickey* dient der Speicherung eines öffentlichen RSA-Schlüssels und dessen Verküpfung mit der zugehörigen Benutzerkennung. Der öffentliche Schlüssel wird hierbei nicht wie auf dem RFID-Transponder in einem eigenen Format gespeichert, sondern im PEM-Format in die Datenbank gelegt.

Die Tabelle *currentAuth* speichert die *Response* zu den angeforderten *Challenges* und verknüpft diese mit der Benutzerkennung des Benutzers, der die *Challenge* angefordert hat. Zusätzlich wird der Zeitpunkt der Anforderung einer Challenge gespeichrt, um später die Zeitdauer ermitteln zu können.

Kapitel 5

Fazit

Das Fazit teilt sich in drei Teile. Das Kapitel 5.1 betrachtet die Leistung des Prototypen gegenüber den gestellten Zielen in Kapitel 1.2. In Kapitel 5.2 folgen einige Schwachstellen bzw. Sicherheitsrisiken bezogen auf dem in dieser Arbeit enstandenen Prototypen. Zuletzt in Kapitel 5.4 wird ein kleiner Ausblick auf ein System gegeben, welches nach Sicht der Autoren für die Zukunft wünschenswert wäre.

5.1 Leistung des Prototypen

Von den drei Anwendungsfällen[55] wurde in dieser Arbeit nur der primäre Anwendungsfall *Authentifikation* umgesetzt. Bei der Implementierung der Authentifikation an einer Webapplikation wurden alle wesentlichen Komponenenten eines solches Systems entwickelt.

Die Komponenten sind, wie in Kapitel 3.2.4 beschrieben, durch geeignete Schnittstellen sehr modular. Das bedeutet, dass eine Komponente einfach durch eine andere Komponente mit der gleichen Schnittstelle ersetzt werden kann. Desweiteren wurden, wie in Kapitel 4 gezeigt, geeignete Programmiersprachen und Bibliotheken verwendet, die mit einem breiten Spektrum von Geräten arbeiten können.

Die Kommunikation mit dem RFID-Transponder ist eine der wichtigsten Komponenten. Daher wurde bei der Prototyp-Umsetzung ein besonderes Augenmerk auf den Daemon gelegt. Dieser Teil wurde vollständig implementiert (siehe Kapitel 4).

[55] Die drei Anwendungsfälle sind im Kapitel 2.1 näher erläutert.

Hierbei wurde gezeigt, dass es möglich ist auch einen großen RSA-Schlüssel auf den verwendeten RFID-Transpondern zu speichern. Es wurde außerdem dargelegt, wie dieser Schlüssel vor dem unauthorisierten Auslesen geschützt werden kann.

Das in Kapitel 3.1 beschriebene und implementierte Authentifikations-Protokoll ermöglicht eine relativ hohe Sicherheit im Vergleich zu herkömmlichen Passwort-orientierten Systemen.

Besonders deutlich wird dies bei der Betrachtung von den folgenden zwei Szenarios. Zum einen könnten die auf dem Webserver gespeicherten Daten in falsche Hände geraten und somit auch die Benutzernamen und Passwörter. Bei dem in dieser Arbeit enstandenen System würden in diesem Fall nur die öffentlichen Schlüssel der Benutzer in die Hände des Angreifers fallen.

Zum anderen kann ein Angreifer den Datenverkehr zwischen der Authentifizierungsstelle und dem zu Authentifizierenden mitschneiden oder lesen. Bei herkömmlichen Passwortsystemen könnte ein Angreifer den Benutzernamen und in den meisten Fällen ein Hashwert des Passwortes abfangen, mit dessen Hilfe Rückschlüsse auf das eigentliche Passwort möglich sind. In dem in dieser Arbeit entstandenen Prototypen würde ein Angreifer lediglich die Challenge und den Hash der Response mitschneiden.

Der Preis für das Lesegerät und den RFID-Transponder liegen aktuell bei ca. 43 Euro. Dies liegt leicht über dem von den Autoren gesetzten Ziel. Da aber die Preise für Lesegeräte und RFID-Transpondern kontinuierlich sinken, kann davon ausgegangen werden, dass sich dieser Trend fortsetzt.

Die Handhabung des Systems aus Benutzersicht ist sehr einfach. Dies gelingt durch den Einsatz der RFID-Technik und auch durch das in Kapitel 3.2.2 beschriebene Pinpad.

Die Ziele der Arbeit können somit als erfüllt angesehen werden.

5.2 Schwachstellen

In diesem Unterkapitel werden die Schwachstellen bzw. Sicherheitsrisiken des im Rahmen dieser Arbeit entstandenen Prototypen näher betrachtet.

5.2.1 Bruteforce auf den RFID-Transponder

Auf dem RFID-Transponder ist der private Schlüssel gespeichert. Dieser wird durch ein Authentifikations-Mechanismus gesichert.

Verwendet man das in Kapitel 3.2.2 beschriebene Pinpad, wird der Raum aller möglichen 3DES- bzw. AES-Schlüssel stark eingeschränkt. Kommt ein Pinpad mit 10 Schaltflächen zum Einsatz, wobei die PIN vierstellig ist, so ergibt sich eine Variation mit $N = 10$ Schaltflächen und $k = 4$ Stellen. Das bedeutet, dass es $N^k = 10^4 = 10.000$ verschiedene Kombinationsmöglichkeiten gibt. Im Vergleich dazu besitzt ein 128 Bit AES-Schlüssel $2^{128} \approx 3,4 * 10^{38}$ verschiedene Kombinationsmöglichkeiten.

Möchte man einen PIN-Code in dem oben beschriebenen Szenario ermitteln, muss man im schlechtesten Fall 10.000 verschiedene PINs für das Authentifizieren an dem RFID-Transponder ausprobieren. Dafür muss sich nur die Karte im Empfangsbereich des Lesegerätes befinden.

Die Zielsetzung bei den durchgeführten Versuchen bestand darin, eine mittlere Zeitdauer für einen Bruteforce-Angriff zu ermitteln. Da RFID-Transponder prinzipiell immer betriebsbereit sind, besteht die Gefahr, dass ein potenzieller Angreifer versucht in Kontakt zum RFID-Transponder eines Benutzers zu gelangen, sei es ein Anrempler oder das längere nebeneinander Stehen, wie zum Beispiel im Zug. Hierbei sollte es nicht möglich sein einen solchen Angriff auf die Karte auszuführen.

Die größte Unbekannte in diesem Versuch war die Geschwindigkeit des Transponders. Es sollte gezeigt werden in welcher Geschwindigkeit der RFID-Transponder in der Lage ist eine Authentifikation durchzuführen.

Der Versuchsaufbau bestand aus einem Desktop-PC und einem Touchatag. Der Desktop-PC ist ein *Fujitsu Simens CELSIUS* mit einem Intel Pentium 4 Prozessor und einem Gigabyte RAM.

Es wurde ein entsprechendes Programm implementiert, welches versucht eine Authentifikation mittels 3DES-Schlüssels durchzuführen. Dieses Programm wurde mit verschieden großen Schlüsselräumen aufgerufen. Die Versuche und deren Ergebnisse sind in Tabelle 5.1 dargestelt.

Authentifikations-versuche	**entspricht**	**Dauer (in Sekunden)**	**Dauer (lesbar)**
10.000	4 aus 10 $\rightarrow 10^4$	439	$\approx$ 7,32 Minuten
1.000.000	6 aus 10 $\rightarrow 10^6$	43.924	$\approx$ 12,20 Stunden
65.635	4 aus 16 $\rightarrow 16^4$	2.877	$\approx$ 47,95 Minuten
16.777.216	6 aus 16 $\rightarrow 16^6$	736.773	$\approx$ 8,53 Tage

Tabelle 5.1: Durchgeführte Versuche mittels Bruteforce

Aus den Versuchen lässt sich eine durchschnittliche Zeitdauer für einen Authentifikationsvorgang berechnen. Diese beträgt $\approx$ 43.915 Millisekunden. Das bedeutet für eine 4-stellige PIN würde der Angreifer im Mittel nur ca. 3,66 Minuten benötigen. Würde es also ein Angreifer schaffen, sein Lesegerät lange genug in Reichweite des RFID-Transponders zu bringen, könnte er erfolgreich einen Bruteforce-Angriff ausführen und den privaten Schlüssel des Opfers kopieren.

Es ist zu vermuten, dass mit einem gewissen Mehraufwand auch bessere Ergebnisse erreicht werden können.

5.2.2 Verarbeitung des privaten Schlüssels im Daemon

Eine weitere Schwachstelle könnte das Entschlüsseln der Challenge auf dem Computer des Benutzers sein. Der private Schlüssel muss im Daemon bekannt sein. Eine Möglichkeit zum Entwenden des Schlüssels ist das Eindringen in den Computer des Nutzers.

Die Methoden, die hierfür eingesetzt werden, stammen zu meist aus der digitalen Forensik. Denkbar ist das Kopieren des Arbeitsspeichers über Schnittstellen, die einen direkten Zugang zum Arbeitsspeicher haben. Solch ein Zugang wird Direct Memory Access (DMA) genannt. Dies bietet zum Beispiel die Firewire-Schnittstelle, aber auch der PCMCIA-Anschluss an einem Laptop.

In Frage kommen auch Methoden, bei der keine spezielle Hardware benötigt wird. Darunter zählt zum einen das Kopieren des Arbeitspeicher über die vom Betriebs-

system gegebenen Schnittstellen, wie zum Beispiel im Unix- oder Linux-Umfeld die virtuellen Geräte */dev/mem* und */dev/kmem*. Jedoch werden hierfür die entsprechenden Rechte benötigt.

Zum anderen gibt es Methoden wie die Kaltstartattacke (auf Englisch *Cold Boot Attack*). Dieses Verfahren nutzt eine Eigenschaft des Arbeitsspeichers, welche es ermöglicht den Speicher nach völligem Wegfall der Stromversorgung wieder herzustellen. Hierfür muss lediglich das Computersystem ausgeschalten werden (kein Herunterfahren) und rasch wieder eingeschalten werden. Dabei besteht die Möglichkeit das System ein anderes kleines Betriebssystem laden zu lassen, welches den Arbeitsspeicher auf einen externen Datenträger kopiert.

Die Arbeit „Lest We Remember: Cold Boot Attacks on Encryption Keys“[56] hat gezeigt, dass dieser Angriff durchaus einfach und mit wenig Aufwand durchzuführen ist. Aus diesem Datensatz lassen sich grundsätzlich alle kryptografischen Schlüssel wiederherstellen.[57]

5.2.3 Man in the Middle

Bei einem *Man in the Middle*-Angriff muss der Angreifer die Möglichkeit besitzen, die Pakete, die zwischen der Authentifizierungstelle und den zu Authentifizierenden ausgetauscht werden, zu manipulieren.

Dafür könnte er den Datenverkehr des zu Authentifizierenden über sein Computersystem umleiten. Startet nun ein Benutzer ein Authentifikationsversuch, gibt sich der Angreifer als Authentifikationssystem aus und startet selbst ein Authentifikationsversuch zu dem eigentlichen Authentifkationssystem als der betroffene Nutzer. So bekommt er eine Challenge vom Authentifikationssystem, die er selber nicht lösen kann. Daher gibt er die Challenge an den betroffenen Benutzer weiter. Der Benutzer löst die Challenge und sendet die Response an das vermeintliche Authentifikationssystem, welches in diesem Fall der Angreifer ist. Der Angreifer hat jetzt eine gültige Response und kann sich gegenüber dem Authentifikationssystem als der betroffene Benutzer authentifizieren.

[56] [HSH+08]
[57] Vergleiche [Pet07].

5.2.4 Phishing

Die Authentizität des Servers wird nicht durch das in dieser Arbeit entstandene System gewährleistet bzw. überprüft. Dies bietet die Möglichkeit von Phishing. Würde es beispielsweise gelingen einen Benutzer auf eine gefälschte Webseite aufzurufen, würde dies nicht von dem in dieser Arbeit entstandenen System bemerkt. Der Benutzer könnte somit verführt werden geheime Informationen preis zugeben.

Dieses Problem könnte durch eine gegenseitige Authentifikation gelöst werden. Es bietet sich hier eine einfache Lösung mit Hilfe der Benutzung von HTTPS an.

5.3 Datenschutz

Die DESFire bietet die Möglichkeit eine zufällige UID zu verwenden. Hierbei wird bei jedem neuen Abfragen dieser eine neue zufällige UID generiert. Der Vorteil in Bezug auf den Datenschutz ist es, dass es nicht möglich ist eine Karte über die UID wieder zuerkennen. Dadurch ist es nicht möglich Bewegungsprofile oder den Standort einer Person zu ermitteln.

Der in dieser Arbeit beschriebene Prototyp kann diese Funktion leider nicht nutzen, da die eindeutige UID, wie in Abschnitt 4.1.2 beschrieben, benötigt wird.

5.4 Ausblick

Die in Kapitel 5.2 beschriebenen Probleme und Schwachstellen lassen sich durch ein System mit besserer Hardware lösen. Diese Hardware sollte nicht nur ein reiner Speicher sein, wie der RFID-Transponder in diesem System, stattdessen sollte die Hardware viel mehr die Speicherung des privaten Schlüssels sowie die Funktionalitäten des Daemon in sich vereinen.

Dies würde das Problem des Verarbeitens des privaten Schlüssels im Computer des Nutzers lösen. Das Schlüsselpaar in dem RFID-Transponder kann selber erzeugt werden. Alle weiteren Funktionen, wie das Berechnen der *Response*, können auch in dieser Hardware-Umgebung ausgeführt werden.

Dies würde einen enormen Sicherheitsgewinn bringen. Doch leider sind solche Hardwaresysteme schlecht bis gar nicht erhältlich. Da die Vorteile eines solchen Systems so gravierend sind, wird vermutetet, dass dieser Zustand sich in einiger Zeit verbessert.

Erste RFID-Transpondersysteme, die eine Funkschnittstelle mit einem RSA- bzw. DSA-Modul verbinden, gibt es bereits. Der *P5Cx012* von NXP Semiconductors bietet eine solche Unterstützung.

Ein anderes System sind die so genannten *Java Smart Cards*. Auf diesen *Java Smart Cards* läuft nicht, wie bei der *Mifare DESFire*, ein festverdrahtetes Program ab, sondern eine Java-Applikation, in der alle erforderlichen Funktionalitäten implementiert werden können.

Dies löst nicht das Problem des *Man in the Middle*-Angriffs. Dies muss durch ein anderes System erfolgen, Ein Ansatz ist einen Sitzungsschlüssel mit dem bereits bekannten öffentlichen Schlüssel des Benutzers zu verschlüsseln und mit diesem Sitzungschlüssel alle weiteren geheimen Informationen zu verschlüsseln. Dies verhindert, dass ein Angreifer die Möglichkeit hat, wie zum Beispiel beim Einsatz des *Diffie-Hellmann-Algorithmus*, eine *Man in The Middle*-Attacke zum Erfolg zu bringen.

Literaturverzeichnis

[ASSvK12] Aubourg, Julian, Jungkee Song, Hallvord R. M. Steen und Anne van Kesteren, Dezember 2012. http://www.w3.org/TR/XMLHttpRequest/ letzter Aufruf 20. Februar 2013.

[BFG+12] Beier, Martin, Beatrice Florat, Jonas Gross, Kai Trott und Volker Zeihs: *Leisefuchs Analyse eines MIFARE Classic Bezahlsystems*. Paper, FH Schmalkalden, Januar 2012. http://dscc.fh-schmalkalden.de/dokuwiki/lib/exe/fetch.php?media=public:leisefuchs.pdf letzter Aufruf 22. Februar 2013.

[bsi13] *Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Technischer Bericht 2013.02, Bundesamt für Sicherheit in der Informationstechnik, Januar 2013. https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102_pdf.pdf?__blob=publicationFile letzer Aufruf 20. Februar 2013.

[BV] BV, Tibo Software. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html letzter Aufrud 20. Februar 2013.

[CHI] CHIPDRIVE. http://www.chipdrive.de/de/chipkartenleser-schreibgeraete-guenstig-kaufen/scl3711-kontaktloser-chipkarten-und-nfc-leser.htm letzter Aufruf 20. Februar 2013.

[cla12] Dezember 2012. http://www.mifare.net/index.php/technology/security/mifare-classic/.

[CRY10] CRYPTAS: *OMNIKEY 5321 USB RFID*. Technischer Bericht, CRYPTAS it-Security GmbH AUSTRIA, 2010. http://www.cryptoshop.

com/de/products/reader/rfid/2010101023.php letzter Aufruf 22. September 2010.

[fef06] *Bignum Arithmetic.* Technischer Bericht, Felix von Leitner, Dezember 2006.

[Fel12] FeliCa: *Contactless IC Card RC-SA00 RC-SA01.* Februar 2012. http://www.sony.net/Products/felica/business/data/RC-SA00_E.pdf letzter Aufruf 20. Februar 2013.

[Fou] Foundation, Python Software. http://pypi.python.org/pypi/M2Crypto letzer Aufrud 20. februar 2013.

[Gro11] Gross, Jonas: *Mifare DESfire Eine Analyse der Implementierung.* Diplomarbeit, FH Schmalkalden, 2011. http://dscc.fh-schmalkalden.de/dokuwiki/lib/exe/fetch.php?media=public:diplomarbeit_jonas_gross_2011.pdf letzter Aufruf 20. Februar 2013.

[Gro13] Group, The PHP, Februar 2013. http://php.net/manual/en/function.openssl-random-pseudo-bytes.php Version vom Fri, 15 Feb 2013, letzter Aufruf 20. Februar 2013.

[HSH+08] Halderman, J. Alex, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrio, Ariel J. Feldman, Jacob Appelbaum und Edward w. Felten: *Lest We Remember: Cold Boot Attacks on Encryption Keys.* Proc. 2008 USENIX Security Symposium, Februar 2008. http://citp.princeton.edu/pub/coldboot.pdf letzter Aufruf 02. Februar 2013.

[ISO99] ISO/IEC: *ISO/IEC 14443-3 Identification cards - Contactless integrated circuit(s) cards - Proximity cards.* Juni 1999.

[lib13] libnfc, Februar 2013. http://nfc-tools.org/index.php?title=Devices_compatibility_matrix Revision vom 14 Februar 2013 20:33 Uhr.

[Lor13] Lord, Bob, Februar 2013. http://blog.twitter.com/2013/02/keeping-our-users-secure.html letzter Aufruf 20. Februar 2013.

[Mey13] Meyhöfer, Patrick, Januar 2013. http://stadt-bremerhaven.de/google-experimentiert-an-neuen-login-mechanismen/?utm_source=feedburner&utm_medium=feed&utm_campaign=Feed:+stadt-bremerhaven/dqXM+(Caschys+Blog) letzter Aufruf 20. Februar 2013.

[Mic13] Microsoft, März 2013. http://support.microsoft.com/kb/208427/en-us.

[NXP09] NXP: *MF0ICU2 MIFARE Ultralight C.* Mai 2009. http://www.nxp.com/documents/short_data_sheet/MF0ICU2_SDS.pdf letzter Aufruf 20. Februar 2013.

[NXP10] NXP: *MF3ICDx21_41_81 MIFARE DESFire EV1 contactless multi-application IC.* Januar 2010. http://www.nxp.com/documents/short_data_sheet/MF3ICDX21_41_81_SDS.pdf letzter Aufruf 20. Februar 2013.

[NXP11a] NXP: *MF1PLUSx0y1 Mainstream contactless smart card IC for fast and easy solution development.* Februar 2011. http://www.nxp.com/documents/short_data_sheet/MF1PLUSX0Y1_SDS.pdf letzter Aufruf 20. Februar 2013.

[NXP11b] NXP: *MF1S70yyX MIFARE Classic 4K - Mainstream contactless smart card IC for fast and easy solution development.* Mai 2011. http://www.nxp.com/documents/data_sheet/MF1S70YYX.pdf letzter Aufruf 20. februar 2013.

[NXP13] NXP: *AN10787 MIFARE Application Directory (MAD).* Januar 2013. http://www.nxp.com/documents/application_note/AN10787.pdf letzter Aufruf 20. Februar 2013.

[ope13a] Januar 2013. http://www.openssl.org/docs/crypto/bn_internal.html.

[ope13b] Januar 2013. http://www.openssl.org/docs/crypto/rsa.html.

[Pet07] Pettersson, Torbjörn: *Cryptographic key recovery from Linux memory dumps.* Technischer Bericht, Archive, CCC, 2007.

https://events.ccc.de/camp/2007/Fahrplan/attachments/1300-Cryptokey_forensics_A.pdf letzter Aufrud 02. Februar 2013.

[php] PHPSECLIB. http://phpseclib.sourceforge.net/ letzer Aufruf 20. Februar 2013.

[Sch01] SCHMEH, KLAUS: *Kryptografie und Public-Key-Infrastrukturen im Internet (2. Aufl.)*. ix edition. dpunkt.verlag, 2001.

[Sch07] SCHMEH, KLAUS: *Kryptografie - Verfahren, Protokolle, Infrastrukturen (3. Aufl.)*. dpunkt.verlag, 2007.

[Ste90] STEVENS, W. RICHARD: *Unix Network Programming*. Prentice-Hall, Englewood Cliffs, NJ, 1990.

[Sys] SYSTEMS, ADVANCED CARD. http://www.acs.com.hk/index.php?pid=product&prod_sections=0&id=ACR122-SDK letzter Aufruf 20. Februar 2013.

[Tea] TEAM, THE GTK+. http://www.gtk.org/features.php letzter Aufruf 20. Februar 2013.

[twi13] *Hacker spähen Nutzerdaten von Twitter aus*. ZEIT ONLINE, dpa, Reuters, AFP, nf, Februar 2013. http://www.zeit.de/digital/datenschutz/2013-02/twitter-nutzerkonten-hacker letzter Aufruf 20. Februar 2013.

[Unb10] UNBEKANNT: *Mifare Application Programming Guide for DESFire*. Januar 2010. ftp://ftp.maxatec-europe.com/Public/PDA/HH8010/Desfire/DesFireProgrammingGuide.pdf letzter Aufruf 25. Februar 2013.

[yah12] *Hacker veröffentlichen massenhaft Daten von Yahoo-Kunden*. ZEIT ONLINE, AFP, juli 2012. http://www.zeit.de/digital/internet/2012-07/hacker-daten-yahoo letzter Aufruf 20. Februar 2013.

Anhang A

Quellennachweis Pinpad

Abbildung A.1: Julian Assange

Author Espen Moe
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/7/75/Julian_Assange_cropped_(Norway,_March_2010).jpg`

Abbildung A.2: Steve Ballmer

Author Microsoft Sweden
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/2/21/Steve_Ballmer_at_CES_2010_cropped.jpg`

Abbildung A.3: Sergey Brin

Author enlewof
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/3/38/Sergey_Brin.JPG`

Abbildung A.4: Marissa Mayer

Author Mrgadget3000
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/c/c7/Marissa_Mayer.jpg`

Abbildung A.5: Jade Raymond

Author Gamerscore Blog
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/8/85/Jade_Raymond_-_E3_2007.jpg`

Abbildung A.6: Steve Jobs

Author Matthew Yohe
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/5/54/Steve_Jobs.jpg`

Abbildung A.7: Linus Torvalds

Author Unbekant
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/6/69/Linus_Torvalds.jpeg`

Abbildung A.8: Konrad Zuse

Author Wolfgang Hunscher
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/d/da/Konrad_Zuse_(1992).jpg`

Abbildung A.9: Alan Turing

Author Jon Callas
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/2/22/Alan_Turing_cropped.jpg`

Abbildung A.10: Bill Gates

Author World Economic Forum
Lizenz Creative Commons
letzer Aufruf 20.01.2013
Quelle `http://upload.wikimedia.org/wikipedia/commons/2/2a/Bill_Gates_in_WEF_,2007.jpg`

Anhang B

Sequenzdiagramm Authentifikation

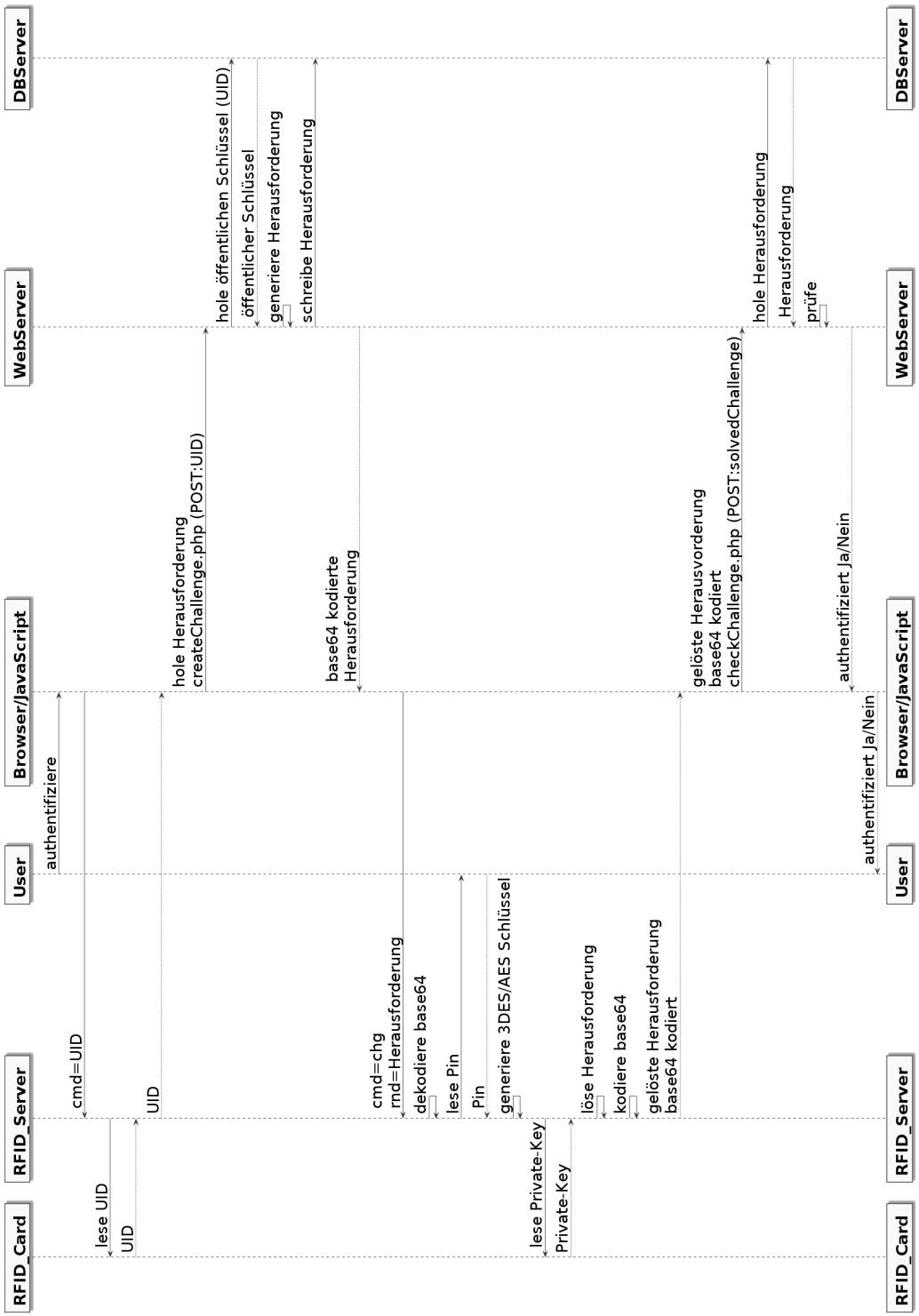


Abbildung B.1: Sequenzdiagramm zur Authentifikation, Quelle: Eigene Darstellung

XKCD

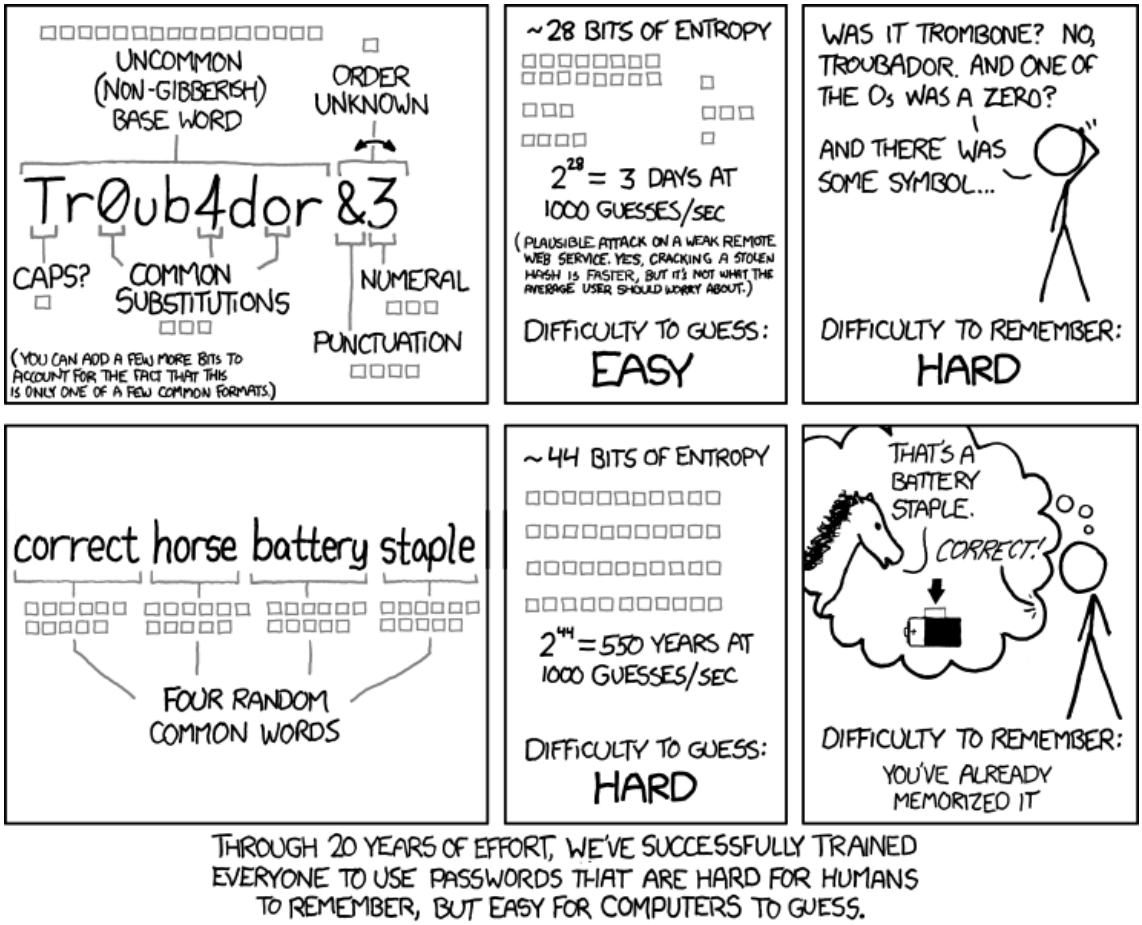


Abbildung B.2: XKCD - Password Strength

Author Randall Munroe
Lizenz Creative Commons
letzer Aufruf 21.01.2013
Quelle `http://imgs.xkcd.com/comics/password_strength.png`

Eidesstattliche Erklärung

Ich versichere an Eides Statt durch meine eigenhändige Unterschrift, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe angefertigt habe. Alle Stellen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Ich versichere außerdem, dass ich keine andere als die angegebene Literatur verwendet habe. Diese Versicherung bezieht sich auch auf alle in der Arbeit enthaltenen Zeichnungen, Skizzen, bildlichen Darstellungen und dergleichen.
Diese Arbeit ist eine Gemeinschaftsarbeit und wurde entsprechend Tabelle B.1 auf die Autoren aufgeteilt.
Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Kapitel	Kai Trott	Volker Zeihs
1	✓	✓
2.1 bis 2.4		✓
2.5.1	✓	
2.5.2		✓
3.1		✓
3.2.1 bis 3.2.2	✓	
3.2.3 bis 3.2.4		✓
4.1	✓	
4.2		✓
5	✓	✓

Tabelle B.1: Aufteilung der Arbeit auf die Autoren

Schmalkalden, den 15. März 2013 *Kai Trott*

Schmalkalden, den 15. März 2013 *Volker Zeihs*