Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorarbeit

Florian Stäps

Konzeption und Implementierung von Sicherheitsfunktionen für eine Publish-Subscribe-Architektur auf Basis einer agentenbasierten Middleware

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Florian Stäps

Konzeption und Implementierung von Sicherheitsfunktionen für eine Publish-Subscribe-Architektur auf Basis einer agentenbasierten Middleware

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Bachelor of Science Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr.-Ing. Martin Hübner
Zweitgutachter: M.Sc. Tobias Eichler

Eingereicht am: 02. Mai 2018

Florian Stäps

Thema der Arbeit

Konzeption und Implementierung von Sicherheitsfunktionen für eine Publish-Subscribe-Architektur auf Basis einer agentenbasierten Middleware

Stichworte

Multiagenten System, Middleware, verschlüsselte Kommunikation, IT - Sicherheit, verteiltes System

Kurzzusammenfassung

Diese Arbeit befasst sich mit einer agentenbasierte Middleware, welche unter anderem im Rahmen einer Masterarbeit erarbeitet wurde. Neben der Middleware wurde auch ein Framework erstellt, mit dem es ermöglicht wird eigene agentenbasierten Anwendungen zu entwicklen. Dieses System wird, im Rahmen dieser Arbeit, um einige Sicherheitsrelevante - Funktionen ergänzt.

Florian Stäps

Title of the paper

Design and implementation of security functions for a publish-subscribe architecture based on agent-based middleware

Keywords

Multiagen System, Middleware, encrypted communication, IT - Security, distributed system

Abstract

This thesis deals with an agent-based middleware, which was developed in the scope of a master thesis. In addition to the middleware, a framework was also created with which it is possible to develop own agent-based applications. In the context of this work, this system is extended by a number of safety-relevant functions.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Durch die stetige Weiterentwicklung von Technologie, dringt diese in immer mehr Bereiche vor und einer dieser Bereiche, ist das eigene zu Hause. Heutzutage ist es möglich immer mehr Technologie in die verschiedensten Dinge zu integrieren und diese so um weitere Funktionen zu ergänzen. Es gibt z. B. ein Thermostat, das selbständig einen Raum auf eine bestimmte Temperatur hält und dabei verschiedene weitere Informationen verarbeitet, um auf verschiedene Situationen richtig reagieren zu können. Ist z. B. auch das Fenster mit entsprechenden Sensoren und einer Möglichkeit zur Kommunikation ausgestattet und die Temperatur in einem Raum sinkt unter die eingestellte Temperatur im Thermostat, so kann dieser das Fenster fragen, ob es auf ist, um festzustellen, ob es gerade gewünscht ist, dass die Temperatur sinkt. Damit muss der Thermostat nicht die Heizungen nach regeln, sondern wartet damit, bis die Fenster wieder geschlossen sind.
Diese 'intelligenten' Objekte werden auch 'Smart Objects' genannt und finden sich in immer mehr sogenannten 'Smart Homes' wieder. Bei einem Smart Home handelt es sich um eine Umgebung in der ein oder mehrere dieser 'Smart Objects' in einem Netzwerk verbunden sind und verschiedene kleinere Aufgabe übernehmen. Das Ziel ist es, durch Fernsteuerung, Automatisierung und Überwachung von verschiedenen Komponenten im Haushalt, die Wohn- und Lebensqualität zu erhöhen Atzori u. a. (2010).
Allerdings ergeben sich durch die Tatsache, dass 'Smart Home' - Umgebungen mit dem Internet verbunden sind und persönliche Daten sammeln und verarbeiten, auch entsprechende Sicherheitsanforderungen an ein solches System Ali u. a. (2017).
Im Rahmen dieser Arbeit wird auf einige dieser Anforderungen, für eine 'Smart Home' - Laborumgebung, eingegangen und ein Sicherheitskonzept für die betrachteten Anforderungen erarbeitet und umgesetzt.

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist es, ein bestehendes Laborsystem, um eine verschlüsselte Kommunikation zu erweitern. Dafür wird das System analysiert und die bestehende Architektur um die dafür notwendigen Komponenten erweitert, so dass die neue Art der Kommunikation, auch mit den

bisherigen Anwendungen kompatibel ist bzw. bleibt.
Damit soll sichergestellt werden, dass die bisherigen und künftigen Benutzer des Systems ohne größeren Aufwand, auf die neue Art der Kommunikation zugreifen können und ihre bisherigen Anwendungen schnell erweitern bzw. umstellen können.
Mit der Ergänzung einer verschlüsselten Kommunikation zwischen verschiedenen Geräten, sollen auch weitere Anwendungsszenarien ermöglicht werden und damit das System attraktiver und sicherer für die Anwender zu machen.
Des Weiteren wird so ein Schritt in Richtung der Vervollständigung eine kompletten Sicherheitsarchitektur des Systems unternommen.

1.2 Zielgruppe der Arbeit

Diese Arbeit ist für Leser und Leserinnen gedacht, die an dem Bereich der IT - Sicherheit im Zusammenspiel mit einem verteilen System interessiert sind und ein bereits bestehendes System um weitere Funktionen aus diesem Bereich ergänzen wollen oder an einem Lösungsansatz für ein bestimmtes Problem aus diesem Bereich interessiert sind.

1.3 Voraussetzungen und Abgrenzungen

Diese Arbeit geht nicht darauf ein, eine komplette Sicherheitsarchitektur für eine verteilte agentenbasierten Middleware zu erschaffen. Sie versteht sich mehr als Schritt in Richtung einer solchen und lässt erahnen, welche Aspekte bei einer nachträglichen Ergänzung von Sicherheit zu berücksichtigen sind.
Ein anderer Aspekt, der im Rahmen dieser Arbeit nicht näher betrachtet wird, ist die Performance des Systems, wenn die verschlüsselte Kommunikation eingesetzt wird. So wird sich das System zwangsläufig verlangsamen, da bei der verschlüsselten Kommunikation mehr Schritte nötig sind, bevor eine Nachricht versendet wird oder eine erhaltene Nachricht vollständig bearbeitet wurde.

1.4 Struktur der Arbeit

Als erstes werden einige Grundlagen erläutert, die für die Arbeit relevant sind, danach wird im Kapitel Analyse auf den derzeitigen Stand des Systems eingegangen und erläutert, welche künftigen Anwendungsszenarien ergänzt werden sollen und welche technischen Anforderungen dafür wichtig sind. Im Kapitel Systemdesign wird näher erläutert, welche Architektur im

Rahmen dieser Arbeit geplant und umgesetzt wurde. Zudem wird näher darauf eingegangen, wie die bestehenden Nachrichten um neue ergänzt werden und wie der Ablauf dieser Nachrichten aussieht. Im Kapitel Fazit wird darauf eingegangen, ob das Ziel erreicht wurde und welche künftigen Anpassungen oder Verbesserungen gesehen werden.

2 Grundlagen

2.1 Verschlüsselung / Chiffrierung

Um zu verhindern, dass unbefugte Dritte den Inhalt einer Nachricht lesen können, allerdings Befugte in der Lage sind, entsprechende Nachrichten lesen zu können, ist es notwendig, dass solche Nachrichten verschlüsselt bzw. chiffriert werden. Bei einer Verschlüsselung oder Chiffrierung wird ein Text mithilfe von Algorithmen, welche entweder in Software oder in Hardware implementiert wurden, genutzt um den Klartext (Plaintext) in einen Geheimtext (Ciphertext) umzuwandeln. Damit dieser Prozess wieder rückgängig gemacht werden kann und somit der Geheimtext in den lesbaren bzw. verständlichen Klartext umzuwandeln, ist der Prozess des Ver- und Entschlüsselns von einem Schlüssel oder Geheimnis abhängig. Für die Ver- und Entschlüsselung gibt es zwei verschiedene Verfahren, die in folgenden Kapiteln näher erläutert werden.

2.2 Symmetrische Verschlüsselung

Bei der symmetrischen Verschlüsselung oder auch Secret-Key-Verfahren, wird für die Ver- und Entschlüsselung immer der gleiche Schlüssel benutzt.
Früher gab es dafür zwei verschiedene Ansätze, bei dem einen wurde jeder Buchstabe des Klartextes durch einen anderen desselben Alphabetes ersetzt (Substitution). Der andere Ansatz geht dabei ähnlich vor, nur werden hier nicht die einzelnen Buchstaben durch andere ausgetauscht, sondern es werden die Positionen der einzelnen Buchstaben innerhalb des Klartextes ausgetauscht (Transposition).
Die modernen bzw. heutzutage eingesetzten Verschlüsselungsalgorithmen werden in Stromchiffren und Blockchiffren unterteilt. Wobei auch bei diesen Algorithmen eine Kombination aus der Substitution und Transposition genutzt wird.

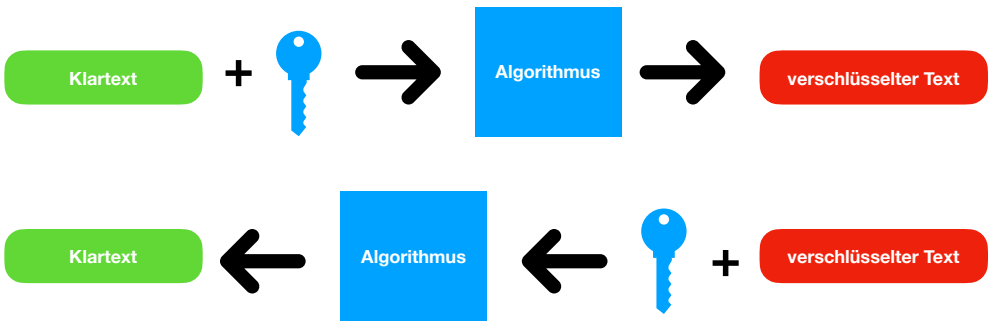

Abbildung 2.1: Veranschaulichung des Vorgangs

2.2.1 Stromchiffren

Bei den Stromchiffren wird jedes Zeichen des Klartextes einzeln verschlüsselt, um den Geheimtext (Ciphertext) zu erhalten.
Dabei werden die einzelnen Bits mit einem Schlüsselstrom und dem XOR - Operator verknüpft.
Der Vorteil hier ist, dass nicht erst gewartet werden muss, bis eine bestimmte Menge an Daten zusammengekommen ist, um diese zu Ver- und Entschlüsseln.
Der Nachteil hier ist, dass wenn der Angreifer den Klartext und den Chiffretext hat, dass er in der Lage ist, den Schlüsselstrom zu berechnen und somit alle weitere Kommunikation, die mit dem zu diesem Zeitpunkt verwendeten Schlüssel, mitlesen kann vgl. Schneier (1996).

2.2.2 Blockchiffren

Bei den Blockchiffren wird der Klartext in Blöcke aufgeteilt und anschließend werden die einzelnen Blöcke verschlüsselt.
Die Verarbeitung erfolgt dabei in Runden, die immer gleich aufgebaut sind und eine Eingabe und eine Ausgabe haben. Zu dieser Art gehört z. B. AES.
Da die Blöcke immer eine feste Größe haben, ist es nicht ungewöhnlich, dass der zu verschlüsselnde Text nicht gleichmäßig auf die feste Blockgröße aufgeteilt werden kann. In diesem Fall werden die fehlenden Stellen mit Nullen aufgefüllt.
Im Gegensatz zu den Stromchiffren soll so erreicht werden, dass der Zusammenhang zwischen Klar- und Geheimtext praktisch nicht ersichtlich ist. Und das durch eine Abhängigkeit der Blöcke untereinander, die Informationen möglichst breit über den Geheimtext verstreut werden vgl. Schneier (1996).

2.2.3 AES

AES steht für Advanced Encryption Standard und zählt zu den symmetrischen Verschlüsselungsalgorithmen, welche im Rahmen einer öffentlichen Ausschreibung, welche 1997 begann und von der NIST [1] ausgeschrieben wurde, zusammen mit verschiedenen Experten dieser Zeit erarbeitet wurde.
Das Ziel dieser Ausschreibung war es einen symmetrischen Verschlüsselungsalgorithmus zu erarbeiten, der als Blockchiffre arbeitet, dabei mindestens eine Blockgröße von 128 - Bit zu unterstützen und mit einer Schlüssellänge von 128 -, 192 - und 256 - Bits genutzt werden kann.
Im Rahmen dieser Ausschreibung wurde viele verschiedene Ideen und Algorithmen erarbeitet und eingereicht, wo sich am Ende die Rijndael - Variante durchgesetzt hat und von der NIST offiziell am 2. Oktober 2000 als offizieller Standard vorgestellt wurde. Dieser Algorithmus ist dabei frei verfügbar und kann ohne Lizenzgebühren genutzt werden vgl. NIST (2016).

2.3 Asymmetrische Verschlüsselung

Die asymmetrische Verschlüsselung nutzt im Gegensatz zur symmetrischen zwei Schlüssel, welche voneinander abhängig sind und für unterschiedliche Zwecke benötigt werden.
Dabei basiert dieser Art der Verschlüsselung auf Falltürfunktionen. Das sind Funktionen, die leicht zu berechnen sind, aber ohne ein Geheimnis praktisch unmöglich rückgängig zu machen sind.
Die Funktion hier ist der öffentliche Schlüssel, welche beschreibt, wie die Funktion aussieht und der geheime Schlüssel fungiert hier als Geheimnis, welches gebraucht wird, um die vorangegangene Berechnung rückgängig zu machen.
Da der öffentliche Schlüssel, wie der Name schon sagt, öffentlich ist und es auch sein muss, damit andere diesen nutzen können, um eine verschlüsselte Nachricht zuschicken, ist es zwingend notwendig, dass der geheime Schlüssel nicht aus dem öffentlichen Schlüssel berechnet werden kann.
Im Gegensatz zur symmetrische Verschlüsselung ist die asymmetrische Verschlüsselung deutlich langsamer, deswegen wird heutzutage erstere bevorzugt, wenn verschlüsselte Kommunikation eingesetzt wird. Da aber der Schlüssel beiden Kommunikationspartner bekannt sein muss und dieser Schlüssel entsprechend geheimgehalten werden muss, wird die asymmetrische Verschlüsselung genutzt, um einen Schlüssel für die symmetrische Verschlüsselung auszutauschen. Das ist vor allem dann wichtig, wenn der Weg, über den die verschlüsselte Kommunikation stattfindet als nicht vertrauenswürdig gilt, wie es z. B. im Internet der Fall ist.

[1] https://www.nist.gov/

Eine Kombination dieser beiden Vorgehen, wird dabei als Hybrides - Verschlüsselungsverfahren bezeichnet.
Ein weiteres Anwendungsszenario für die asymmetrische Verschlüsselung ist die Signierung und Prüfung dieser Signaturen vgl. Schmeh (2013).

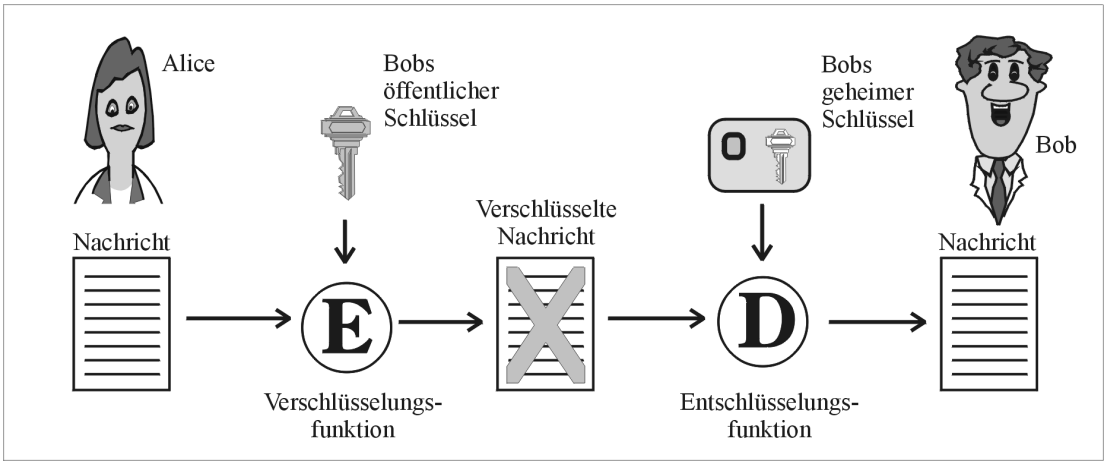

Abbildung 2.2: Ablauf gesicherte Kommunikation Wolf

Die Abbildung 2.2 zeigt den Ablauf einer asymmetrischen Verschlüsselung. Alice möchte Bob eine verschlüsselte Nachricht schicken, dafür erstellt sie eine Nachricht und nimmt den öffentlichen Schlüssel von Bob, um damit die Nachricht zu verschlüsseln. So kann nur noch der Besitzer, hier Bob, die Nachricht mit seinem privaten Schlüssel entschlüsseln und damit wieder lesbar machen.

2.3.1 RSA

Das RSA - Verfahren oder auch Rivest-Shamir-Adleman-Verfahren, ist ein asymmetrisches Kryptoverfahren, welches zurzeit, am verbreitetsten ist.
Die häufigsten Aufgaben von RSA, welche in der Praxis auftreten sind:

- "Verschlüsselung von kleinen Datenmengen, insbesondere für den Schlüsseltransport;"
- "digitale Signaturen [...], z. B. für Zertifikate im Internet."

Pelzl (2016)
Der Grund hierfür ist, das RSA, wie alle asymmetrische Kryptoverfahren, im Vergleich zur

symmetrischen Verschlüsselung, aufgrund des hohen Rechenaufwands, deutlich langsamer ist. Deshalb wird es vor allem für den Austausch von Schlüsseln für die symmetrische Verschlüsselung genutzt.
Die Einwegfunktion dieses Verfahrens, basiert auf dem Faktorisieren von ganzen Primzahlen, welche nach aktuellen Wissensstand äußerst aufwändig ist. Wobei die Multiplikation von Primzahlen einfach und schnell zu erledigen ist vgl. Pelzl (2016).

2.3.2 ECC - Elliptic Curve Cryptography

ECC ist ein Kryptoverfahren, das im Gegensatz zu RSA, auf einer Verallgemeinerung von Diskreter-Logarithmus Verfahren über endlichen Körpern basiert. So können Protokolle, die auf dem Diskreten-Logarihtmus aufsetzen, auch mit elliptischen Kurven umgesetzt werden, ohne das Nachteile entstehen.
Durch dieses Vorgehen ergeben sich verschiedene Vorteile, weil die Schlüssel die für ECC benötigt werden, kürzer sind.
Durch die kürzeren Schlüssel wird zum einen weniger Speicherplatz für die Schlüssel benötigt und zum anderen ist das Erstellen von Schlüsseln und das Ver- und Entschlüsseln von Informationen schneller, im Vergleich zu RSA. Allerdings ist das Verifizieren von Signaturen mit RSA schneller vgl. Pelzl (2016) und Agrawal und Mehrotra (2016).
Die nachfolgenden Tabellen (2.3 und 2.4) zeigen den zeitlichen Aufwand bei der Erstellung von Schlüsseln und der Ver-und Entschlüsselung der beiden Verfahren.
Als Ausgangslage wurden die folgenden Bedingungen festgelegt:

- "Die Nachricht ist: IEEE Conferences in India (25 Byte)"
- "Der private ECC Schlüssel und der öffentliche RSA Exponent ist: 569986981914167393707828428422173474150538 8 (zufällig gewählt) und 65537 (technisch gewählt)"
- "Die Algorithmen wurden in Java 7 umgesetzt"
- "Der Prozessor ist ein Intel mit 3.10 GHz und 4 GB RAM"
  Agrawal und Mehrotra (2016)

Security level(bits)	Algorithms	Public Key Generation Time (ns)
112	ECC-224	30239910
	RSA-2048	2825151941
128	ECC-256	32403603
	RSA-3072	10270890931
192	ECC-384	32457264
	RSA-7680	**102067678287**

Abbildung 2.3: Vergleich Ver- und Entschlüsselung Agrawal und Mehrotra (2016)

Die Überschrift Algorithms beschreibt, welcher Algorithmus, mit welcher Schlüssellänge genutzt wurde.
Die Punkte Encryption und Decryption Time stellen die benötigte Zeit für die Ent- und Verschlüsselung da.
Wird nun der oben genannte Text mithilfe von RSA und einer Schlüssellänge von 3072 Bits verschlüsselt, betrug die benötigte Zeit 593957338 Nanosekunden (0,593957338 Sekunden). Wird hingegen ECC mit einer Schlüssellänge von 256 genutzt, wird dafür lediglich 11607045 Nanosekunden (0,011607045 Sekunden) benötigt. Die Abbildung 2.4 zeigt die benötigte Zeit,

Security level(bits)	Algorithms	Public Key Generation Time (ns)
112	ECC-224	30239910
	RSA-2048	2825151941
128	ECC-256	32403603
	RSA-3072	10270890931
192	ECC-384	32457264
	RSA-7680	**102067678287**

Abbildung 2.4: Schlüsselerzeugung Agrawal und Mehrotra (2016)

um einen öffentlichen Schlüssel mit einem Algorithmus (RSA oder ECC) und einer entsprechenden Schlüssellänge zu erstellen.
So wird für die Erstellung eines RSA - 3072 Schlüssels, 10270890931 Nanosekunden (10,270890931 Sekunden) benötigt, wohingegen für die Erstellung eines ECC - 256 Schlüssels nur 32403603 Nanosekunden (0,032403603 Sekunden) gebraucht.

2.4 Signatur und Hash

Die digitale Signatur von Informationen gehört zu den wichtigsten Funktionen der asymmetrischen Kryptografie, die heutzutage eingesetzt werden.
Sie haben zum Ziel, dass der Empfänger einer signierten Nachricht prüfen kann, ob diese vom richtigen Absender kommt. Sie werden z. B. bei Zertifikaten eingesetzt, mit denen sich ein Server gegenüber einem Webbrowser ausweist.
Bei einer Signatur wird eine Nachricht oder Teile davon, mithilfe des privaten Schlüssels des Senders signiert. So ist der Empfänger in der Lage, mithilfe des öffentlichen Schlüssels des Absenders zu prüfen, ob die Nachricht auch wirklich von ihm stammt.
Allerdings wird in der Praxis nicht der komplette Klartext signiert, vor allem Texte die sehr lang sind, werden nicht komplett signiert.
Das hat mehrere Gründe, so ist der Klartext der signiert werden kann bei RSA und ECC beschränkt, bei RSA darf dieser nicht länger als der zu verwendende Schlüssel sein, welche in der Regel zwischen 1024 und 3072 Bit lang ist.

Ein weiterer Grund dafür ist der hohe Rechenaufwand der dafür benötigt wird. Die Signierfunktionen basieren auf den entsprechenden asymmetrischen Kryptoverfahren und diese sind von Natur aus langsam.
Außerdem entsteht dadurch ein Overhead bei den Nachrichten, da die Nachricht und die Signatur gemeinsam versendet werden, wodurch auch die Größe der Nachricht anwächst und damit die Bits die über das Netzwerk übertragen werden müssen. Deshalb wird nicht der Klartext selbst, sondern ein Hash davon signiert.
Bei einem Hash handelt es sich um ein kleines Abbild einer größeren Eingangsmenge. Ein Hash wird mithilfe einer Hash - Funktion gebildet, dabei wird eine Eingangsmenge wie z. B. ein Klartext genommen und auf eine kleine Zielmenge reduziert, die die Eingangsmenge repräsentiert. Es wird im Prinzip ein Fingerabdruck der Eingangsmenge erstellt, welcher einzigartig ist.
Dabei ist es wichtig, dass eine normale Hash - Funktion diese Einzigartigkeit nicht garantiert, was bedeutet, es kann passieren, dass zwei unterschiedliche Eingaben denselben Hash erzeugen, so was wird dann Kollision genannt.
Diese Kollisionen sind vor allem bei der verschlüsselten Kommunikation nicht gewünscht, da sonst ein Angreifer, unterwegs den Inhalt der Nachricht verändern kann, ohne das der Empfänger dies bemerkt.
Deshalb werden hier kryptografische Hash - Funktionen genutzt, welche nicht nur einen Fingerabdruck erstellen, sondern es praktisch unmöglich machen, zwei identische Hash - Werte für unterschiedliche Eingaben zu finden bzw. zu erstellen.
Dieser Hash wird dann der Nachricht angehängt und kann aufseiten des Empfängers überprüft werden, indem die gleichen Teile der Nachricht genommen werden und diese wieder mit einer Hash - Funktion gehasht werden. So wird sichergestellt, dass der Empfänger prüfen kann, ob die Nachricht auch vom richtigen Empfänger versendet wurde und ob die Nachricht bzw. der Inhalt der Nachricht unterwegs ausgetauscht oder verändert wurde vgl. Pelzl (2016).

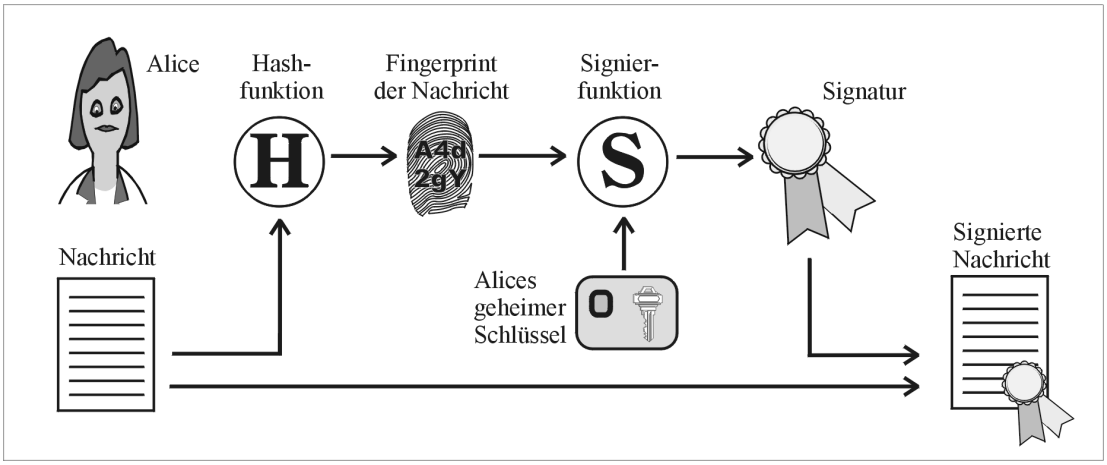

Abbildung 2.5: Veranschaulichung Verwendung und Erstellung einer Signatur Wolf

Dieser Prozess des Erstellens eines Hashes und signieren einer Nachricht, wird in der Abbildung 2.5 dargestellt. Alice nutzt hier eine Hashfunktion, welcher die Nachricht von Alice übergeben wird und einen Fingerabdruck (Fingerprint) der Nachricht (Hash) als Ergebnis liefert. Dieser Hash wird dann anschließend mit dem privaten Schlüssel von Alice und einer Signierfunktion signiert. Dadurch erhält Alice eine Signatur der Nachricht, welche der eigentlichen Nachrichten angehängt werden kann und mit der geprüft werden kann, ob die Nachricht verändert wurde und ob die Nachricht auch von Alice stammt.

2.5 Angriffe

In diesem Kapitel werden verschiedene Angriffsszenarios beschrieben, mit denen es möglich ist, eine verschlüsselte Kommunikation anzugreifen, um an Informationen zu kommen oder um die Kommunikation zu stören, in dem z.B. verschickte Datenpakete abgefangen und verändert werden.

2.5.1 Brute Force

Beim Brute Force - Angriff handelt es sich, vom Vorgehen her, um den simpelsten Angriff bzw. Lösungsansatz um ein Problem zu lösen.
Es werden dazu alle möglichen Lösungen ausprobiert bzw. berechnet die infrage kommen.
Aufgrund dieses Ansatzes findet er immer eine Lösung. Allerdings steigt der Zeitbedarf, der

nötigt ist, um die richtige Lösung zu finden sehr stark an und so dauert ein Angriff unter Umständen mehrere Jahre oder Jahrzehnte an vgl. Pelzl (2016).

2.5.2 Man-in-the-Middle

Beim Man-in-the-Middle Angriff wird versucht physisch bzw. hier logisch zwischen zwei oder mehreren Parteien zu stellen und die Kommunikation über den Angreifer laufen zu lassen. So wird erreicht, dass der Angreifer die Kommunikation der Parteien mitlesen kann oder auch manipulieren kann und die Parteien glauben, sie würden direkt miteinander kommunizieren vgl. Pelzl (2016). Dieser Angriff wird in der Abbildung 2.6 dargestellt, hier wollen Alice und

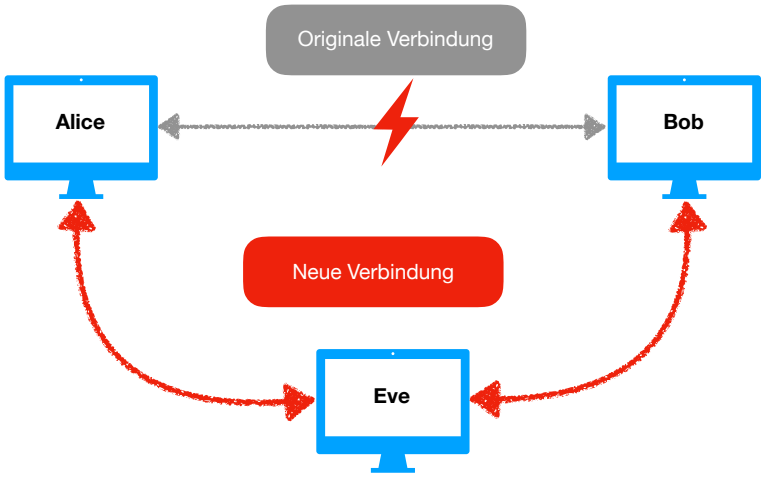

Abbildung 2.6: Veranschaulichung eines Man-In-The-Middle-Angriffes

Bob miteinander kommunizieren. Allerdings hat ein Angreifer (Eve), Alice und Bob davon überzeugen können, dass die mit dem jeweils anderen direkt kommunizieren. Jedoch geht sämtliche Kommunikation der beiden über Eve, welche somit die Kommunikation mitlesen und auch verändern kann, ohne dass Alice oder Bob dies bemerken.

2.5.3 Replay

Der Replay Angriff greift nicht direkt die Verschlüsselung von Nachrichten an, sondern die Authentizität der Nachrichten. Es wird versucht, durch das erneute Senden von zuvor abgefangen Nachrichten, eine Reaktion der Gegenseite zu erhalten und aufgrund dieser, an weitere Informationen zu kommen vgl. Pelzl (2016).

2.5.4 DoS und DDoS

Bei einem Denial-of-Service Angriff, geht es darum ein bestimmtes Ziel z. B. ein Dienst, der über ein Netzwerk erreichbar ist, außer Gefecht zu setzen, damit dieser nicht mehr genutzt werden kann.
Ein Beispiel für einen solchen Angriff wäre das SYN-Flooding bei der Verwendung von TCP. Bei diesem Angriff werden extrem viele SYN-Pakete an einen bestimmten Computer geschickt, der daraufhin versucht jedes dieser Pakete zu bearbeiten und irgendwann nicht mehr genug Kapazitäten hat, dieses zu tun. Daraufhin steht der Dienst für weitere Nutzer nicht oder nur sehr langsam zur Verfügung.
Greift nicht nur ein Angreifer an, sondern viele von verschiedenen Punkten, so spricht man von einem DDoS (Distributed-Denial-of-Service) - Angriff. Diese Art von Angriff wird vor allem genutzt, wenn hinter einem Dienst besonders viele Kapazitäten stecken, wie z.B. bei Google oder Github. Damit wird versucht die Erfolgschancen auf so große Dienste zu erhöhen vgl. McDowell (2009).

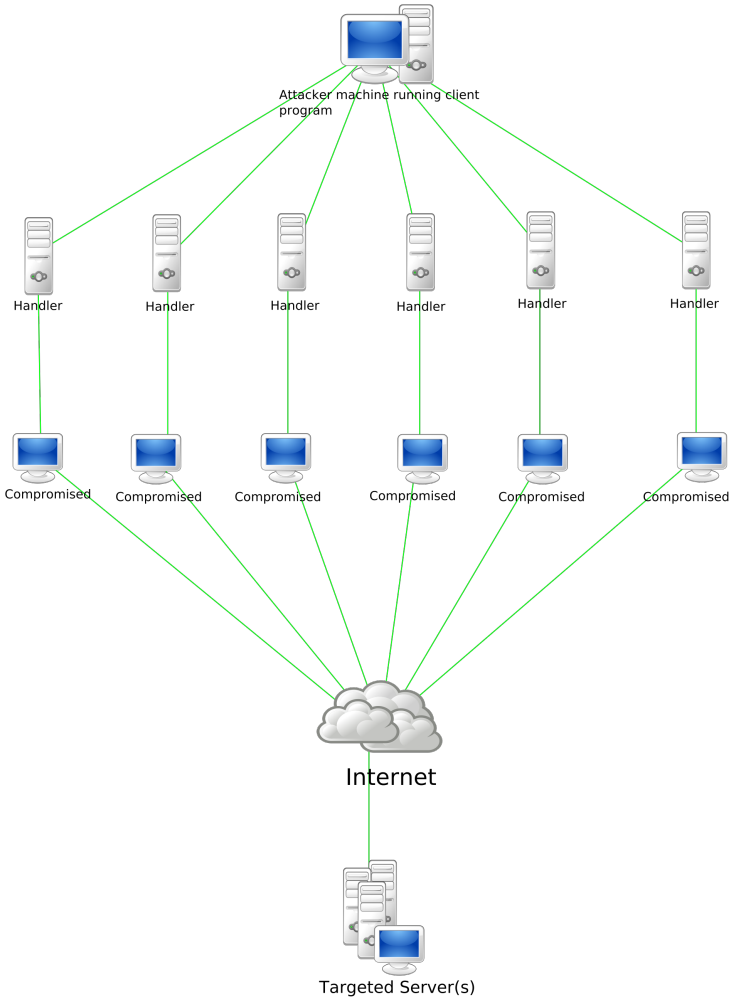

Abbildung 2.7: Darstellung eines Schemas für einen DDoS Angriff YellowIcon

In der Grafik 2.7 ist ein Schema dargestellt, wie mit der Hilfe von mehreren kompromittieren Computern ein Angriff über das Internet auf z.B. einen Webserver erfolgen kann.
Dafür signalisiert der Angreifer seinem Botnetz, dass diese anfangen sollen viele Anfragen an einen bestimmten Server zu schicken, um diesen mit der Bearbeitung dieser Anfragen so sehr zu beschäftigen, dass der Server nicht mehr in der Lage ist, diese und auch weitere Anfragen zu bearbeiten. Damit wird erreicht, dass der Server für weitere Anfragen gar nicht oder nur sehr zeitverzögert zur Verfügung steht.

2.5.5 Seitenkanalangriff

Bei Seitenkanalangriffen handelt es sich um einen Angriff auf die physikalische (Hardware) oder digitale (Software) Implementierung von Kryptoverfahren. Dabei wird eine bestimmte Implementierung angegriffen, was bedeutet, dass wenn der Angriff gelingt, dass nicht bedeutet, dass eine andere Implementierung ebenfalls betroffen ist.
Konkret werden bei dieser Art von Angriff die Hard- und/oder Software beobachtet und es wird versucht aus den dabei entstandenen Daten und den verwendeten Schlüsseln ein Zusammenhang herzustellen vgl. IT.
Diese Art des Angriffs wird allerdings nicht näher im Rahmen dieser Arbeit betrachtet, da dafür physischer Zugriff auf die Hardware erforderlich ist und damit weitaus größere Probleme, wie z. B. das Auslesen des Speichers möglich wird.

3 Analyse

Im Rahmen dieses Kapitels wird das System, welches im CSTI zur Verfügung gestellt wird, näher analysiert.
Dazu wird auf den aktuellen Stand eingegangen und beschrieben, wie es zurzeit aufgebaut ist, wie die Kommunikation der Agenten untereinander bzw. mit dem System aussieht und welche Rollen die Benutzer des Systems einnehmen können. Außerdem werden die künftigen Anforderungen und Schnittstellen beschrieben, die im Rahmen dieser Arbeit ergänzt werden.

3.1 Einleitung

Das System was im Rahmen dieser Arbeit analysiert wird, ist eine agentenbasierte und verteilte Middleware, welche für das CSTI entwickelt wurde und dort zur Verfügung steht vgl. Eichler (2014).
Bei einer Middleware handelt es sich um eine Zwischenschicht, die es verschiedenen Programmen erlaubt untereinander zu kommunizieren, ohne dabei auf eine bestimmte Programmiersprache festgelegt zu sein. Außerdem wird so versucht die Komplexität und Strukturen der einzelnen Programme voreinander zu verstecken und sich nur auf den Austausch von Informationen zu konzentrieren.
Beim Austausch der Informationen, muss sich lediglich an die zuvor definierten Nachrichten und Protokolle gehalten werden, damit ein reibungsloser Austausch stattfinden kann.
Die einzelnen Programme, die über die Middleware kommunizieren, werden als Agenten bezeichnet und agieren mehrere Agenten miteinander, wird von einem Multiagentensystem gesprochen.
Ein solches System versucht Problemstellungen durch die Zusammenarbeit der Agenten zu lösen. Dabei hat jeder Agent eine bestimmte Funktion und stellt diese den anderen Agenten, als Dienst über Nachrichten, zur Verfügung.
Das Ziel der Arbeit [Eichler (2014)] war es, ein System zu entwickeln, das zu den bisher entwickelten Anwendungen kompatibel bleibt und gleichzeitig den künftigen Anwendern (Studenten etc.) und deren Anforderungen, eine einfache Möglichkeit bietet, neue Anwendungen für diese Art des Systems zu entwickeln und zu testen. Ermöglicht wird dies unter anderem

durch die zur Verfügung Stellung eines Frameworks, welches die Komplexität des Systems für die Anwender / Programmierer reduziert. Allerdings wurde dieses System unter der Prämisse entwickelt, dass es sich um ein Laborsystem handelt, welches nach außen hin abgesichert ist und in einer Laborumgebung genutzt wird. Dementsprechend wurden verschiedene Aspekte der IT - Sicherheit, bei der Planung und Umsetzung des Systems, nicht näher betrachtet.
Die folgenden Kapitel gehen näher auf die Middleware selbst ein und beschreiben wie diese zurzeit aufgebaut ist, wie die Agenten miteinander über die Middleware kommunizieren, welche Probleme dadurch entstehen, welche Rollen die Anwender einnehmen können, wenn diese mit dem System interagieren und es wird auf die Anwendungsfälle eingegangen, die im Rahmen dieser Arbeit ergänzt werden und wie diese durch verschiedene Technische Maßnahmen realisiert werden sollen.

3.1.1 CSTI

CSTI steht für /* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */ und ist ein Labor der HAW - Hamburg, welche als Plattform für verschiedene Bereiche der angewandten Forschung und für den Austausch von Informationen innerhalb dieser Bereiche, aufgebaut wird.
Das langfristige Ziel des CSTI ist es, bei der Umsetzung von Idee in den Themenfeldern Interactive Virtual / Argumented Reality, Smart Objects & User Interface, Machine Learning & Data Mining und Science & Technology Studies zu unterstützen.
Dafür stellt das CSTI Werkzeuge, Laborfläche und Schlüsseltechnologien zur Verfügung, die für die Umsetzung der Ideen und Vorhaben nötig sind. Des Weiteren helfen Professoren und Studenten bei den Projekten, wodurch ein Austausch von Wissen und eine stetige Weiterentwicklung der Projekte erreicht werden soll und zudem nicht nur ein wissenschaftlicher Aspekt betrachtet wird, sondern auch in Richtung Produktentwicklung gedacht wird.
Damit möchte sich das CSTI unter anderem als Talentschmiede innerhalb dieser Themengebiete in Hamburg positionieren vgl. CSTI.

3.2 Aufbau des Systems

In der folgenden Abbildung 3.1 ist dargestellt, wie das System zurzeit aussieht.

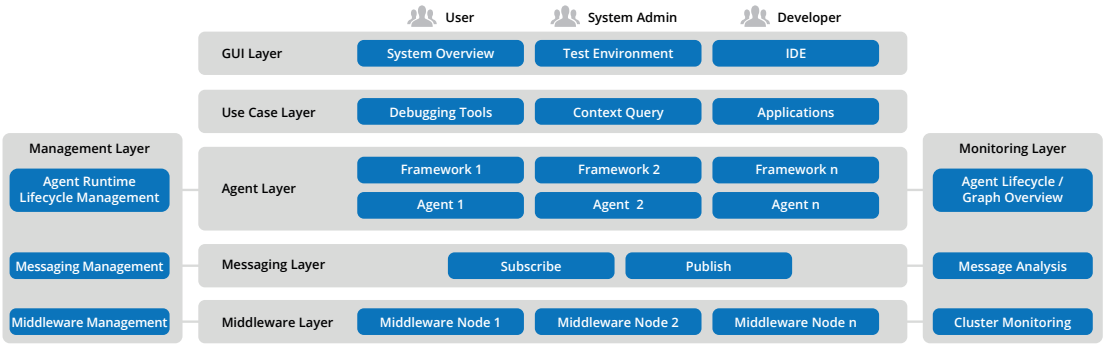

Abbildung 3.1: Aufbau des Systems Eichler u. a. (2017)

Auf der untersten Ebene (Middleware Layer) ist die Middleware zu sehen die im Rahmen der Masterarbeit [Eichler (2014)] konzipiert und umgesetzt wurde. Über ein Framework wird den Agenten der Zugriff auf die Kommunikationsschicht gewährt, was es ihnen ermöglicht Nachrichten über die Middlware zu Senden und zu Empfangen.

3.3 Middleware

Die Middleware dient als Grundlage des Systems und ist unter anderem für die Kommunikation der darüber liegenden Komponenten / Schichten verantwortlich. Sie ist dabei selbst ein verteiltes System, welches sich in verschiedene Komponenten aufteilt.
Eine dieser Komponenten ist für die Gruppenkommunikation der Agenten verantwortlich. Sie arbeitet nach dem Publish - Subscribe Ansatz und stellt dafür entsprechende Funktionen bereit. So existiert eine Schnittstelle, über die es möglich ist, sich bei verschiedenen Gruppen anzumelden um dort Nachrichten empfangen zu können.
Des Weiteren wird über diese Komponenten eine Schnittstelle angeboten welche es ermöglicht, Nachrichten in einer oder mehrere Gruppen zu veröffentlichen, damit die anderen Komponenten diese Empfangen und verarbeiten können.

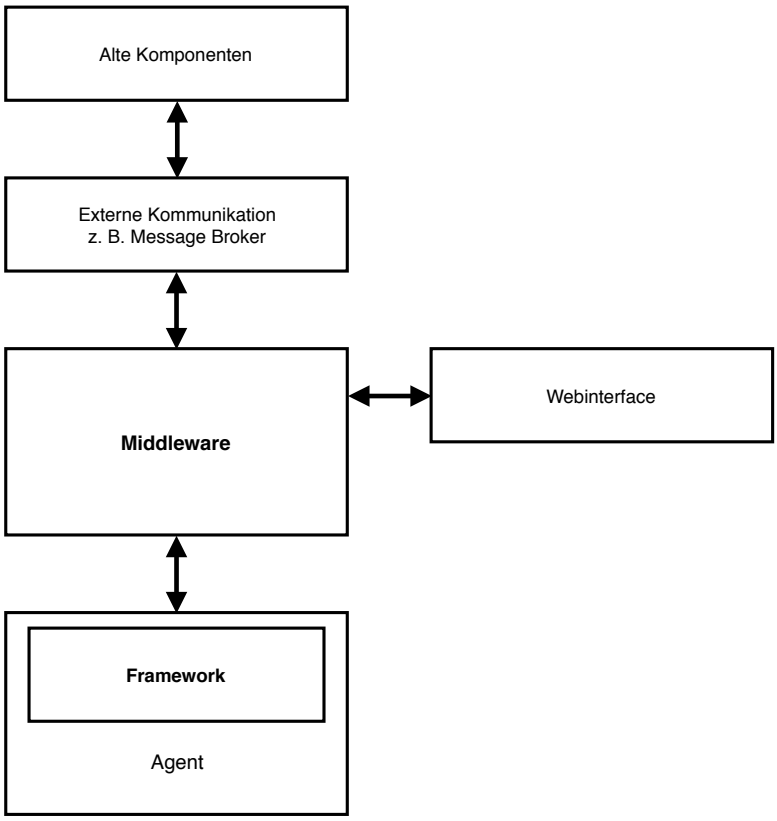

Abbildung 3.2: Blackbox - Ansicht der Middleware [Eichler (2014)]

In der Abbildung 3.2 sind die zuvor erwähnten Schnittstellen für die anderen Komponenten dargestellt.
Über ein Framework können Agenten Nachrichten veröffentlichen und empfangen.
Zudem können durch ein Webinterface die Anwender auf die Schnittstellen der Middleware zugreifen.
Außerdem ist es möglich, externe Komponenten bzw. alte Komponenten an die Middlware anzuschließen, damit diese ebenfalls mit den internen bzw. neuen Komponenten kommunizieren können.
Des Weiteren ermöglicht die Middleware das Monitoring von Agenten, damit können andere Agenten sich informieren lassen, wenn sich am Lebenszyklus eines anderen Agenten etwas ändert. Mit dieser Funktion wird es den Entwicklern einer Anwendung ermöglicht, eine Übersicht zu erhalten, in der der aktuelle Status ihrer Agenten aufgezeigt wird. Dies ist notwendig, damit festgestellt werden kann, ob die eigene Anwendung funktionstüchtig ist, wie viel Ressourcen die Anwendung in Anspruch nimmt und um die eigene Anwendung besser

debuggen zu können.
Allerdings ist das Monitoring von Agenten nicht eingeschränkt, so kann jeder Agent der sich über die Middleware registriert, von einem anderen Agenten überwacht werden. Ein Agent kann so feststellen, ob ein anderer Agent, welcher nicht zu eigenen Anwendung gehört, noch online ist oder nicht. Dadurch und durch die Möglichkeit einem Agenten beliebig viele Nachrichten zu schicken, kann ein automatisierter DoS / DDoS (siehe 2.5.4) Angriff auf Agenten gestartet werden.
Durch die Möglichkeit externe Komponenten an die Middleware anzuschließen, ergibt sich ein weiteres Problem: Externe Komponenten können ebenfalls überwacht werden, jedoch produziert das Überwachen dieser Komponenten deutlich mehr Nachrichten, als es bei einer internen Komponente der Fall wäre. Durch das überdurchschnittliche Aufkommen von Nachrichten ist es möglich die anderen Agenten zu beeinflussen. Die zusätzlichen Nachrichten belasten das Netz und die Middleware zusätzlich, wodurch es zu Verzögerungen bei der Zustellung von Nachrichten kommt vgl. Eichler (2014).

3.4 Kommunikation der Agenten

3.4.1 Protokoll

Nutzen Agenten zurzeit die Middleware zur Kommunikation untereinander, werden die Daten in zuvor definierten Nachrichten umgewandelt und dann über die Middleware verschickt. Die Agenten nutzen zur Kommunikation mit der Middleware, ein dafür erstelltes Framework, welches entsprechende Funktionen zur Verfügung stellt. Da sich die Agenten in der Regel nicht auf einem Computersystem befinden, sondern auf mehreren Systemen verteilt sind, kommuniziert die Middleware über ein Netzwerk und dafür wird das Protokoll SCTP verwendet. Bei SCTP handelt es sich um verbindungsorientiertes Netzwerkprotokoll, welches verschiedene Eigenschaften von UDP [1] und TCP [2] vereint.
So ist SCTP wie UDP Nachrichten orientiert und überträgt Nachrichten, wie TCP vollständig und je nach Einstellung, auch in der richtigen Reihenfolge.
TCP nutzt für die vollständige und Reihenfolgen Gesicherte Übertragung die Sequenznummer. Diese stellt sicher, dass jedes Paket identifizierbar ist und von der Empfängerseite bestätigt werden muss, bevor das nächste Datenpaket bearbeitet wird.
SCTP behandelt die vollständige Übertragung und die Reihenfolgen Gesicherte Übertragung separat. SCTP nutzt für die vollständige Übertragung eine 'transmission sequence number'

[1] https://tools.ietf.org/html/rfc768
[2] https://tools.ietf.org/html/rfc793

welche sicherstellt, dass alle Datenpakte korrekt und vollständig übertragen werden. Da SCTP allerdings bei der Übertragung auf mehrere Kanäle setzt, um Datenpakte zu übermitteln, ist es möglich, dass sich die Datenpakte innerhalb der unterschiedlichen Kanäle überholen. Damit die richtige Reihenfolge gewährleistet ist, wird eine 'stream sequence number' genutzt, mit der innerhalb eines Datenstroms sichergestellt wird, dass alle Datenpakte auch in der richtigen Reihenfolge ankommen. Allerdings ist es auch möglich, dass die Datenpakete direkt an die Anwendung weitergeleitet werden, ohne dass die richtige Reihenfolge sichergestellt wurde vgl. Stewart und Metz (2001).
Eine weitere Eigenschaft die SCTP von TCP übernommen hat, ist die Fluss- und Überlastkontrolle. SCTP nutzt Algorithmen die ähnlich zu den zuvor genannten Eigenschaften sind und stellt damit sicher das sich TCP und SCTP nicht gegenseitig behindern und damit die Kommunikation des jeweils anderen Protokolls lahmlegt bzw. beeinflusst.
Auch wenn SCTP verschiedene Eigenschaften von UDP und TCP in sich vereint, so werden dabei nicht alle Schwachstellen von diesen übernommen. So ist SCTP gegenüber SYN-Flooding resistent, was eine DoS - Attacke auf TCP ist und damit ein wichtiges Angriffsszenario von vornherein verhindert wird vgl. Stewart (2007).
Allerdings ist die Kommunikation nicht gegen die Manipulation der Nachrichten oder das Mitlesen der Nachrichten von Dritten gesichert. So ist es möglich, wenn ein Angreifer oder ein anderer Nutzer des Systems Nachrichten, welche nicht für ihn bestimmt sind mitzulesen oder auch zu verändern, ohne das es auffällt. Und damit ist es auch möglich, andere Agenten bei ihren Aufgaben gezielt zu stören oder die von diesen Agenten benötigten Daten so zu manipulieren, dass diese zu ganz falschen Ergebnissen führen.

3.4.2 Technische Umsetzung

Wenn ein Agent eine Nachricht über die Middlware verschickt, wird diese mit Hilfe eines Serialisierers in ein JSON - Nachricht umgewandelt und verschickt. Damit die Nachricht über das Netzwerk verschickt werden kann, wird das Netzwerkprotkoll SCTP genutzt, welches sicherstellt, dass die Nachricht auf den Empfänger Seite auch ankommt, sollte es unterwegs nicht einen Netzwerkausfall oder etwas ähnliches ereignen.
Die empfangende Nachricht, wird dann mit Hilfe eines Deserialisieres wieder in ein Objekt umgewandelt und dem empfangenden Agent zur Verfügung gestellt. Damit der Vorgang des Serialisieren und Deserialisieren stattfinden kann, muss einen Schnittstellenbibliothek genutzt bzw. importiert werden vgl. Eichler (2014).
Dieser Vorgang wird nocheinmal in der Abbildung 3.3 veranschaulicht.

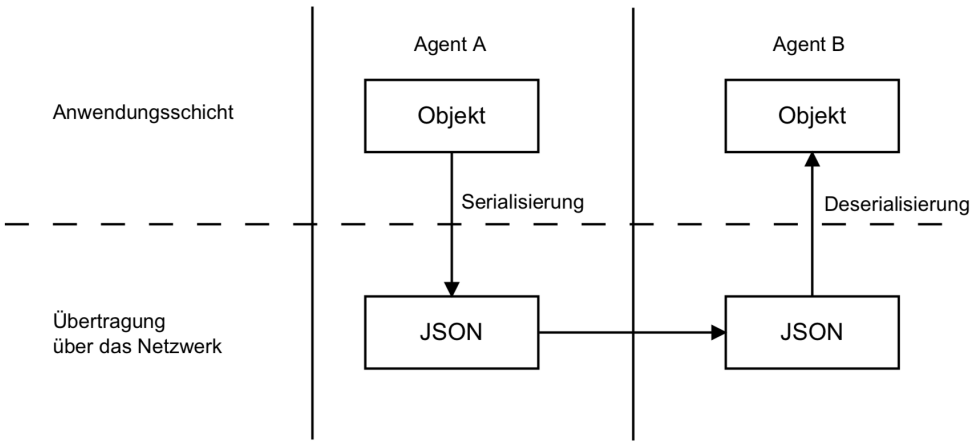

Abbildung 3.3: Protokollstack mit Serialisierung von JSON-Nachrichten [Eichler (2014)]

3.5 Anwendungsfälle

Im nachfolgenden Teil, werden die neuen Anwendungsfälle näher beschreiben. Die Anwendungsfälle ergeben sich zum einen aus dem aktuellen Stand des Systems bzw. der Kommunikation der Agenten und zum anderen durch Interviews mit den Benutzern des Systems.

3.5.1 Gesicherte Kommunikation über die Middleware (zwei Parteien)

Zwei Komponenten (Alice & Bob) möchten verschlüsselt über die Middleware kommunizieren, ohne dass eine weitere Komponente den Inhalt entschlüsseln oder verändern kann, ohne dass es auffällt. Dazu ist es erforderlich, dass die Informationen und Parameter der einzelnen Komponenten, öffentlich einsehbar sind, da diese zum einen über die Middleware übertragen werden müssen und so jeder die Identität des anderen überprüfen kann. Des Weiteren ist es wichtig, dass die Informationen über andere Komponenten jederzeit angefordert werden können, für den Fall, dass diese sich z.B. ändern oder noch nicht bekannt sind.
Der Ablauf dieses Anwendungsfalles ist wie folgt: Alice beschafft sich die notwendigen Informationen über Bob, damit die Kommunikation beginnen kann. Hat sie diese erhalten, prüft sie die Informationen auf Richtig- und Gültigkeit und versenden nach erfolgreicher Prüfung ihre verschlüsselte Nachricht an Bob.
Nachdem Bob die Nachricht erhalten hat, prüft er, ob die Nachricht von Alice ist und nicht verändert wurde. Wenn diese Überprüft erfolgreich verlief, entschlüsselt Bob die Nachricht von Alice und verarbeitet den Inhalt entsprechend weiter.
Sollte eine der oben genannten Überprüfungen einen Fehler hervorrufen, wenn z. B. die Nach-

richt unterwegs verändert wurde, dann wird diese Nachricht verworfen und eine entsprechende Meldung wird herausgegeben.

3.5.2 Gesicherte Kommunikation in Gruppen (Trusted Groups)

Mehrere Komponenten wollen miteinander, über die Middleware, verschlüsselt kommunizieren, sodass eine Partei, die nicht zur Gruppe gehört, den Inhalt nicht mitlesen kann oder diesen verändert, ohne dass es die Kommunikationsteilnehmern bemerken. Hierbei ist es wichtig, dass lediglich die berechtigten Parteien Zugriff auf die entsprechenden Schlüssel erhalten, und dass, wenn eine Partei nicht mehr zugriffsberechtigt ist, die weitere Kommunikation der verbleibenden Parteien nicht weiter entschlüsseln bzw. mit lesen kann.
Wenn eine Komponente einer Trusted Group beitreten möchte, dann meldet sie sich mithilfe des Frameworks bei der Komponente, welche für die Verwaltung der Gruppen zuständig ist, an. Diese prüft dann ob eine Freigabe für diese Gruppe vorliegt und antwortet mit den notwendigen Informationen. Sollte keine Freigabe vorliegen, dann enthält die Antwort lediglich die Information, dass die Anfrage abgelehnt wurde.

3.6 Rollen

Aus der Analyse des Systems und den neuen Anwendungsfällen, ergeben sich die nachfolgenden Rollen, welche zurzeit von einem Anwender des Systems eingenommen werden können. Dementsprechend ist es wichtig, dass bei der Planung der neuen Architektur, die Rollen berücksichtigt werden.

3.6.1 Administrator

Der Administrator hat vollen Zugriff auf das System. Er hat, neben dem Vollzugriff auf das System (Framework & Middleware), die Möglichkeit die einzelnen Komponenten zu überwachen und steuernden Einfluss auf das System zu nehmen. So kann er bestimmte Systemzustände herbeizuführen, den Nachrichtenfluss kontrollieren und diesen einsehen.

3.6.2 Entwickler

Der Entwickler nutzt die vom Framework gegebenen Funktionen ein, um seine eigene Ziele zu erreichen. Dabei hat er kompletten Zugriff auf das Framework, den dazugehörigen Quellcode und vorhandene Dokumentationen. Er kann das Verhalten der eigenen Software steuern und

hat keinen Zugriff auf die anderen Agenten oder die Middleware selbst. Kann allerdings die zur Verfügung gestellten Funktionen im Rahmen der Regeln nutzen.

3.6.3 Endbenutzer

Ein Endbenutzer ist jemand der die Hardware z.B. ein VR - Headset oder einen intelligenten Spiegel nutzt und sich am System mit Hilfe von z.B. Benutzername & Passwort oder mit Hilfe von biometrischen Daten anmeldet. Er hat keinen Zugriff auf das Framework oder die Middleware und kann dementsprechend keinen Einfluss auf das System nehmen.

3.7 Technische Maßnahmen zu Erreichung der verschlüsselten Kommunikation

Damit die neuen Anwendungsfälle mit den bestehenden Strukturen erreicht werden können, wurde neben verschiedenen Komponenten, neue Nachrichtentypen, samt entsprechenden Abläufen dieser Nachrichten und notwendige Kryptografische Funktionen geplant und umgesetzt.
Die neuen Komponenten werden näher im Kapitel 4.3.1 erläutert. Auf die neuen Nachrichtentypen und das dazugehörige Protokoll, wird im Kapitel 4.2 drauf eingegangen. Im folgenden werden die Kryptografischen Funktionen näher beschreiben, die für die verschlüsselte Kommunikation notwendig sind.

Anmerkung Wichtig ist es zu erwähnen, dass es keinen vollständigen Schutz gegen alle Angriffe (siehe 2.5) gibt. Es aber möglich, die Sicherheitsmaßnahmen soweit zu erhöhen, dass der zeitliche Aufwand so hoch ist, dass es sich nicht lohnt, einen Angriff auf die verschlüsselte Kommunikation der Agenten zu starten.

3.8 Verschlüsselung

Damit den Inhalt einer Nachricht, nur der vorgesehene Empfänger lesen kann, ist es erforderlich das dieser verschlüsselt wird. Dafür wird die symmetrische Verschlüsselung (siehe 2.2) genutzt. Der Standard dafür ist AES, allerdings gibt es verschiedene Modi, in denen AES genutzt werden kann. Im Rahmen dieser Arbeit, wird sich an den Empfehlungen der NIST und des BSI orientiert, welche unter anderen den GCM als ausreichend sicher darstellen vgl. Barker (2016).
Zudem ist es erforderlich, dass die Schlüssel für die symmetrische Verschlüsselung über die

Middleware ausgetauscht werden können. Dafür wird die asymmetrische Verschlüsselung genutzt. Beim Start generiert sich jeder Agent ein eigens asymmetrisches Schlüsselpaar, mit dem er sich zum einem beim öffentlichen Schlüssel Server (siehe: 4.3.1) registriert und zum anderen es ermöglicht, symmetrische Schlüssel über die Middleware auszutauschen.
Für die asymmetrische Verschlüsselung wird die ECC (siehe 2.3.2) mit der Kurve P-256 genommen. Verschlüsselt wird nach dem ECIES (Elliptic Curve Integrated Encryption Scheme). Hierbei handelt es sich um eine hybrides Verschlüsselungsverfahren, welches als Grundlage für die asymmetrische Verschlüsselung, das Diffie-Hellman-Problem, in der verwendeten elliptischen Kurve, nutzt, um eine ausreichende Sicherheit zu erreichen vgl. BSI (2010) und Barker (2015).
Im Rahmen des RFC 7748 wurden auch zwei neue Kurven (Curve25519 und Curve448) standardisiert und stehen für die nächste Generation von TLS zur Verfügung. Gerade die Curve25519 hat den Vorteil, dass diese für eine Software Implementation entwickelt und optimiert worden ist. Damit einher würden entsprechende Geschwindigkeitsvorteile einhergehen. Zusätzlich zielt dieser Kurve auch auf einen hohen Sicherheitsstandard, was sie zusätzlich interessant macht vgl. Koppermann u. a. (2016) und Langley u. a. (2016).
Allerdings unterstützt Scala 2.12 und Java 8, die Sprachen in denen das Framework geschrieben wurde, standardmäßig keine asymmetrische Verschlüsselung, weshalb Bouncy - Castle (siehe 4.4.1) hinzugezogen wurde. Diese Softwarebibliothek unterstützt zum Zeitpunkt dieser Arbeit, keine der neueren Kurven.

3.8.1 AES - GCM

Für die symmetrische Verschlüsselung wird AES im Galois / Counter Mode genutzt.
Dieses Verfahren gehört zu den Blockchiffren und nutzt den CTR - Modus und eine polynomiale Hash - Funktionen. Die Designer haben neben diesem Modus auch entsprechende Beweise für die Sicherheitsnachweise für die Vertraulichkeit als auch für die Authentizität vorgelegt vgl. McGrew und Viega (2004).
Der Vorteil von dieser Art von Algorithmen liegt darin, dass die Daten / Informationen nicht nur gut verschlüsselt werden, sondern durch Authenticated Encryption with Associated Data (AEAD), auch sichergestellt wird, dass die Daten gegenüber dem Empfänger authentifiziert werden können und so der Ursprung und die Integrität der Daten sichergestellt werden kann. Deshalb hat die NIST diesen Modus 2007 als Empfehlung für die Blockchiffren ausgesprochen vgl. Barker (2016).
Zudem wird AES - GCM in vielen Bereichen genutzt z.B. bei TLS oder IPSec.

Werte für Parameter

Für das Verfahren werden neben den Daten, welche Ver- und Entschlüsselt werden sollen, auch weitere Parameter gebraucht bzw. für das Projekt festgesetzt. Die hier verwendeten Werte beruhen zum einem auf Empfehlungen der NIST als auch den technischen Richtlinien des BSI.

Der Initialisierungsvektor ist 96 Bit lang und wird mit zufälligen Werten gefüllt. Er ist deshalb genau 96 Bits lang, weil AES-GCM intern im CTR Modus arbeitet und dieser erwartet einen 12 Byte IV, sollte dieser länger sein, wird gehashed, was zu Kollisionen führen kann und damit zu größeren Probleme führen kann, die darin resultieren, dass die Vertraulichkeit nicht mehr sichergestellt werden kann vgl. BSI (2010), McGrew und Viega (2004).
Die IV dient fungiert hierbei als Nounce. Der Schritt sieht dabei wie folgt aus:
Define a block, J_0 as follows:
If len(IV) = 96, then let $J_0 = IV \parallel 0^{31} \parallel 1$.
If len(IV) $\neq$ 96, then let s = 128 $\lceil$ len(IV) / 128 $\rceil$ - len(IV), and let
$J_0 = \text{GHASH}_H(IV) \parallel 0^{s+64} \parallel [\text{len}(IV)]_{64}$.

Ein weiterer wichtiger Punkt ist, dass sich der IV innerhalb eines Schlüsselintervalls nicht wiederholt, so wird in [Dworkin (2007)] sogar gefordert, dass während eines Intervalls nicht mehr als $\approx 2^{32}$ Aufrufe der GHASH - Funktion vorgenommen werden, welche an weiteren Stellen im Algorithmus genutzt wird.
Allerdings muss dieser nicht geheimgehalten werden und wird deshalb an den Anfang der verschlüsselten Informationen / Daten (4.2.2) gestellt, damit er beim entschlüsseln der Nachricht wieder genutzt werden kann.

Die AAD (additional authenticated data) dienen als zusätzlicher Faktor für die Authentifizierung oder können bestimmte Informationen enthalten, die für die Ver- oder Entschlüsselung benötigt werden. Die AAD können eine beliebige Zeichenkette (0 - 2^{64} Bits) sein. Diese muss dabei nicht geheim gehalten werden, da diese Daten nicht verschlüsselt werden und es wird vorher festgelegt, wie diese sich zusammensetzt bzw. ergibt.
So setzt sich die Zeichenkette z.B. bei TLS aus der Sequenznummer, TLSCompressed.type, TLSCompressed.version und TLSCompressed.length zusammen vgl. Dierks und Rescorla (2008).
In diesem Fall wird dafür die AgentenID des Senders verwendet. Der Grund dafür ist, das im Rahmen der Recherche nie eine Empfehlung oder etwas ähnliches gefunden wurde, an der es möglich ist, sich zu orientieren. Es gibt nur einige Beispiele, wie oben genannt, die als Orientierung dienten.

Ein weiterer Grund dafür ist, dass die Java - Implementierung es nicht zu lässt, dass diese Daten nicht gesetzt (null) sind.

3.8.2 Schlüssel

Ein Sitzungsschlüssel wird für die Ver- und Entschlüsselung des Inhaltes von den Encrypted-Messages genutzt und muss geheim gehalten werden.
Der Schlüssel ist 256 Bit lang und wird in regelmäßigen Abständen von den Agenten, welche Daten bereitstellen und diese verschlüsseln, erstellt und mit einer speziellen Nachricht verteilt (DistributeSessionKey).
Mit dieser gewählten Schlüssellänge sind die Nachrichten ausreichend verschlüsselt, neben der NIST gibt es noch weitere Einrichtungen die eine entsprechende Empfehlung für die zu verwendenden Schlüssellänge herausgeben.
So kann der Abbildung 3.4, welche auf [Giry (2017)] bereitgestellt wird, entnommen werden, dass z.B. die NSA empfiehlt für symmetrische Verschlüsselung einen Schlüssel zu verwendenden der 256-bit lang ist, wenn das System bis zum Jahr 2030, nach aktuellem Wissensstand, als sicher gelten soll.
Die Abbildung 3.4 ist wie folgt aufgebaut, unter dem Punkt 'Method' sind die Institute aufgeführt, die eine Empfehlung herausgeben. Hier werden vor allem die NIST, die NSA und das BSI berücksichtigt. Unter dem Punkt 'Date' ist das Jahr oder ein Zeitraum von Jahren zu finden, in denen die Empfehlung für die folgenden Algorithmen und deren empfohlene Schlüssellänge. Im Rahmen dieser Arbeit wird symmetrische- und asymmetrische Verschlüsselung mit elliptischen Kurven verwendet und Hashes, deshalb sind die Punkte 'Symmetric', 'Elliptic Curve' und 'Hash' relevant. Unter diesen Punkten sind die empfohlenen Schlüssellängen bzw. die empfohlene Länge des Hashes, für die jeweiligen Algorithmen in Bits aufgeführt.

Method	Date	Symmetric	Factoring Modulus	Discrete Logarithm Key	Discrete Logarithm Group	Elliptic Curve	Hash
Lenstra / Verheul	2030	93	2493 2016	165	2493	176	186
Lenstra Updated	2030	88	1698 2063	176	1698	176	176
ECRYPT II	2021 - 2030	112	2432	224	2432	224	224
NIST	2016 - 2030	112	2048	224	2048	224	224
ANSSI	> 2030	128	3072	200	3072	256	256
IAD-NSA	-	256	3072	-	-	384	384
RFC3766	-	-	-	-	-	-	-
BSI	> 2022	128	3000	250	3000	250	256

Abbildung 3.4: Übersicht der Empfehlungen Giry (2017)

Für die Verteilung des Sitzungsschlüssels an die anderen Agenten, wird der öffentliche Teil des speziellen Schlüsselpaares verwendet (4.3.4), welcher hier als Langzeitschlüssel dient.
Durch die regelmäßige Neuerstellung des Schlüssels, soll gewährleistet sein, dass wenn ein Angreifer einen Sitzungsschlüssel bekommt, dieser nur einen kleinen Teil der bisherigen Nachrichten entschlüsseln kann und keine der zukünftigen. Außerdem wäre die bisherige Kommunikation weiterhin geheim, sollte der Angreifer den privaten Teil des Langzeitschlüssels bekommen, da die Nachrichten, welche den Sitzungsschlüssel beinhalten, auch zusätzlich mit dem öffentlichen Schlüssel des Empfängers verschlüsselt sind. Damit wird sichergestellt, dass Programme die einmalig in einer Gruppe waren, nicht die weitere Kommunikation entschlüsseln können.
Dieses Prinzip nennt sich forward secrecy oder auch perfect forward secrecy vgl. Dworkin (2007) und Harkins und Carrel (1998).

3.8.3 Nounce

Damit der Nachrichtenaustausch gegen Replay - Angriffe (siehe 2.5) geschützt wird, werden die Nachrichten um einen Faktor ergänzt, der die Neuwertigkeit der Nachrichten sicherstellt. Dieser Faktor wird Nounce genannt, welche ein Wert ist, der nur einmal in einem bestimmten Kontext genutzt werden darf.
Bei einer Nounce handelt es sich entweder um einen zufälligen Wert oder er wird auf eine bestimmte Art berechnet (siehe nachfolgende Auflistung).

Challenge - Response

Bei der Challenge - Response Nounce, wird zuvor eine Aufgabe vom Empfänger festgelegt, welche der Sender erfolgreich lösen muss und das Ergebnis der Nachricht anhängen muss.

Fortlaufende Nummer

Bei der fortlaufenden Nummer wird ein Startwert festgelegt, welcher um einen bestimmten Wert erhöht wird.

Zeitstempel

Beim Zeitstempel wird die aktuelle Uhrzeit mit Datum genommen und der Nachricht angefügt.

3.9 Erstellung von Signatur und Hash

Für die Erstellung der Signaturen bzw. der Hash - Werte wird auf ECDSA (Elliptic Curve Digital Signature Algorithm) und SHA - 256 zurückgegriffen.
ECDSA ist die Übertragung des DSA (Digital Signature Algorithm), welcher auf dem diskretem Logarithmus in endlichen Kurven basiert.
Bei SHA - 256 handelt es sich um eine kryptografische Hashfunktion, welche neben ECDSA, von der NIST und dem BSI empfohlen wird vgl. BSI (2010).

3.10 Verwandte Arbeiten

Mit dem Thema der Verschlüsselung von Informationen in einem verteilten System wurde sich zuvor schon auseinandergesetzt und dabei wurde unter anderem die folgenden Ansätze erarbeitet, welche auch für die hier entwickelte Architektur genutzt wurden.

3.10.1 Foner - Matchmaking

In dieser Arbeit Foner und N. (1997) wird beschrieben, wie Multiagenten - System um eine gesicherte Kommunikation erweitertet werden. Diese Arbeit dient unter anderem als Grundlage für die hier vorgestellte Architektur. Sie beschreibt, wie Agenten die an einem bestimmte Thema (hier Matchmaking) in eine Gruppe zusammenfinden können und dort gesichert miteinander Kommunizieren können. Verwendet werden auch hier asymmetrische- und symmetrische Verschlüsselung.
Allerdings wird in dieser Arbeit von der Kommunikation über das Internet ausgegangen, was hier nicht der Fall ist.
Des weiteren wird auch in dieser Arbeit nicht auf alle Angriffe, vor allem DoS, eingegangen und wie diese verhindert bzw. erschwert werden können. Gerade die Erkennung eines DoS oder DDoS - Angriffs ist sehr aufwändig und geht auch über dieser Arbeit hier hinaus bzw. wird nicht näher erläutert.

3.10.2 Proxy-Re-Encryption

Bei diesem Ansatz, wird von einer dritten Instanz ein spezielles Schlüsselpaar erzeugt, mit Hilfe dessen der Sender seine Nachrichten an den Empfänger verschlüsselt, dabei muss der Sender nicht den öffentlichen Schlüssel des Empfängers kennen und nutzt lediglich den Schlüssel, welchen er von der dritten Instanz erhalten hat. Der Proxy nimmt dann die Nachricht entgegen und verschlüsselt diese neu, so dass nur noch der ursprüngliche Empfänger in der Lage ist, die

Nachricht zu entschlüsseln.
Dieses Vorgehen erfordert, dass die Schlüssel, welche der Sender und der Proxy benutzen über einen sicheren Weg wie z.B. offline bzw. händisch verteilt werden. Da es sich hier aber um ein verteiltes System mit vielen Komponenten und Agenten handelt bzw. handeln kann, ist dies nicht realistisch umsetzbar. Des weiteren soll jede Nachricht immer signiert werden, damit es möglich ist zu prüfen, von wem eine Nachricht stammt und ob dieser überhaupt eine Nachricht verschicken darf.
So wäre es auch möglich, die Schlüssel, die für die Proxy-Re-Encryption erforderlich sind, vertraulich zu verteilen. Dann allerdings käme der Ansatz, welcher mit Proxy-Re-Encryption verfolgt wird, hier nicht mehr zum Tragen vgl. Thangam und Chandrasekaran (2016).

3.10.3 Multiagent Security Architecutre

Sulaiman u. a. (2009) diese Arbeit diente auch als Grundlage für die hier verwendete Architektur. Es wird beschreiben, wie einem Arzt - Patienten Szenario die beiden Parteien Nachrichten und Code miteinander austauschen können, ohne dass eine Dritte Partei den Inhalt mitlesen kann oder verändern kann ohne dass es auffällt. Es wird eine Kombination von Verschlüsselung (asymmetrische und symmetrisch), Hash und digitaler Signatur verwendet. Diese Arbeit geht allerdings von einer Kommunikation über das Internet aus und nutzt zur Sicherung des Transportwegs SSL/TLS, was nicht auf das System anwendbar ist, welches in dieser Arbeit als Grundlage dient.

3.10.4 Trustwhorty and Referee Agents

In Bentahar und Khosravifar (2008) wird der Ansatz verfolgt, dass einige wenige vertrauenswürdige Agenten, dass System überwachen und die anderen Agenten regelmäßig an die vertrauenswürdigen berichten. Diese entscheiden dann anhand der ihnen vorliegenden Daten, ob ein Eindringling da ist oder ob jemand versucht ins System einzudringen.
Hier wird sich allerdings nicht auf die vertrauenswürdige Kommunikation zwischen den Agenten konzentriert, sondern es wird versucht, Angreifer zu entdecken und dann entsprechende Gegenmaßnahmen einzuleiten.

3.11 Zusammenfassung

Damit die Kommunikation der Agenten untereinander bzw. innerhalb der Gruppen vertraulich ist, muss das System in der Lage sein, mit asymmetrischer- und symmetrischer Verschlüsselung

umzugehen, damit die entsprechenden Nachrichten ver- und entschlüsselt werden können. Zusätzlich müssen die Agenten auch in der Lage sein, eine Nachricht zu signieren und Signaturen von anderen Nachrichten bzw. Agenten überprüfen zu können, um so festzustellen, dass diese auch mit den richtigen Agenten kommunizieren. Damit die künftigen Anwender, welche nicht zwangsläufig Kenntnisse über die verschlüsselte Kommunikation besitzen, diese in ihre Anwendungen integrieren können, müssen die neuen Schnittstellen dafür leicht zu nutzen bzw. zu implementieren sein.
Da die Funktionen für die verschlüsselte Kommunikation, sei es asymmetrische oder symmetrisch, immer Schlüssel benötigen, ist es erforderlich, dass die entsprechenden Schlüssel generiert und auch verwaltet werden. Dabei ist es wichtig, dass die Schlüssel immer zugeordnet werden können, damit die Nachrichten auch richtig geprüft, ver- und entschlüsselt werden können. Des Weiteren soll sich der Anwender nicht um die Generierung und Verwaltung dieser Schlüssel kümmern müssen.
Damit die Agenten, welche die verschlüsselte Kommunikation nutzen, nicht einfach ausfallen, sollten diese eine falsche oder ungültige Nachricht erhalten, ist es wichtig, dass diese nicht einfach ausfallen, sondern ihrer Aufgabe weiterhin nachgehen können. Allerdings ist es wichtig, dass der Anwender in einem solchen Fehlerfall benachrichtigt wird, damit dieser zum einen Bescheid weiß, warum Daten von anderen Agenten fehlen und zum anderen entsprechende Schritte zur Problemlösung einleiten kann.

4 Systemdesign

Damit die verschlüsselte Kommunikation über die Middlware möglich wird, wurden die bestehenden Strukturen so erweitert, dass zum einen die Benutzer die neue Art der Kommunikation ohne großen Aufwand nutzen können und zum anderen gegen verschiedene Angriffe (siehe 2.5) gewappnet sind, wurden neben den Kryptografischen Funktionen auch neue Nachrichten, ein Protokoll und verschiedene Komponenten konzipiert und umgesetzt.
Die Nachrichten, das Protokoll, die Erweiterung des Frameworks und die neuen Komponenten werden in den nachfolgenden Kapiteln näher erläutert.

4.1 Erweiterung der Technischen Umsetzung

Im Rahmen dieser Arbeit wurde technische Prozess des Versendens einer Nachricht (siehe 3.4.2) angepasst bzw. erweitert, diese Erweiterung wird in der Abbildung 4.1 dargestellt.
Um den Zugang zur verschlüsselten Kommunikation so unkompliziert wie möglich für die späteren Anwender zu gestalten, findet der Prozess des Verschlüsselns einer Nachricht, Hashen des Inhaltes und die anschließende Prüfung des Hashes und die Entschlüsselung der Nachricht, innerhalb des Frameworks statt. Außerdem werden dort die verschiedenen Schlüssel verwaltet und bei Erhalt einer DistributeSessionKey - Nachricht aktualisiert.
So soll sichergestellt werden, dass die Anwender lediglich eine Methode haben, über diese die entschlüsselten und geprüften Nachrichten zur Verfügung gestellt werden.

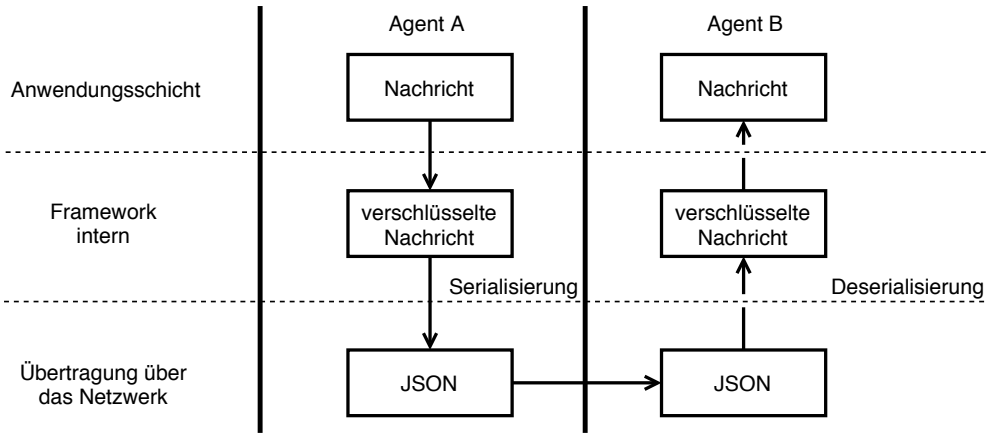

Abbildung 4.1: Erweiterung des Ablaufs

4.2 Nachrichten

Um die oben genannten Kryptografischen Funktionen zu ermöglichen und um Replay - Angriffe (siehe 2.5) zu verhindern, müssen die Nachrichten einen gewissen Aufbau haben und es muss ein gewisser Ablauf von Nachrichten eingehalten werden. Im nachfolgenden Teil wird näher auf die Nachrichten und deren Gemeinsamkeiten, sowie dem Ablauf des Austausches der Nachrichten eingegangen.

4.2.1 Aufbau und Ablauf

Damit die sichere Kommunikation zustande kommt, wurden neben der Architektur auch verschiedene Nachrichten definiert, die jeweils in verschiedenen Abläufen genutzt werden.
Es gibt einige Eigenschaften bzw. Gemeinsamkeiten, die sich bei allen Nachrichten finden lässt. So werden alle Nachrichten immer signiert, es unterscheidet sich zwar welche Teile der Nachricht signiert werden, aber eine Signatur ist immer vorhanden. Damit soll sichergestellt werden, dass falls ein Angreifer sich an den Nachrichten, während der Übertragung zu schaffen macht, dieses auf Seiten des Empfängers auffällt.
Des Weiteren ist immer die Agenten - ID in der Nachricht, dieser dient stets als zu Ordnung zu den öffentlichen Schlüsseln der Sender und ermöglicht so ein schnelles auffinden der benötigten Informationen, um die Nachricht korrekt weiterzubearbeiten.
Zudem wurde ein fester Ablauf von Nachrichten definiert, damit eine sichere und vertrauenswürdige Kommunikation zu Stande kommen kann.
Welche Nachrichten und in welcher Reihenfolge diese ausgetauscht werden, wenn ein Agent

versucht, einer Gruppe beizutreten, für die es noch keinen Gruppenverwalter gibt oder ein Gruppenproxy existiert, kann der Abbildung 4.2 entnommen werden. Hier wird der grundlegende Ablauf gezeigt, unabhängig von der Rolle des Agenten.

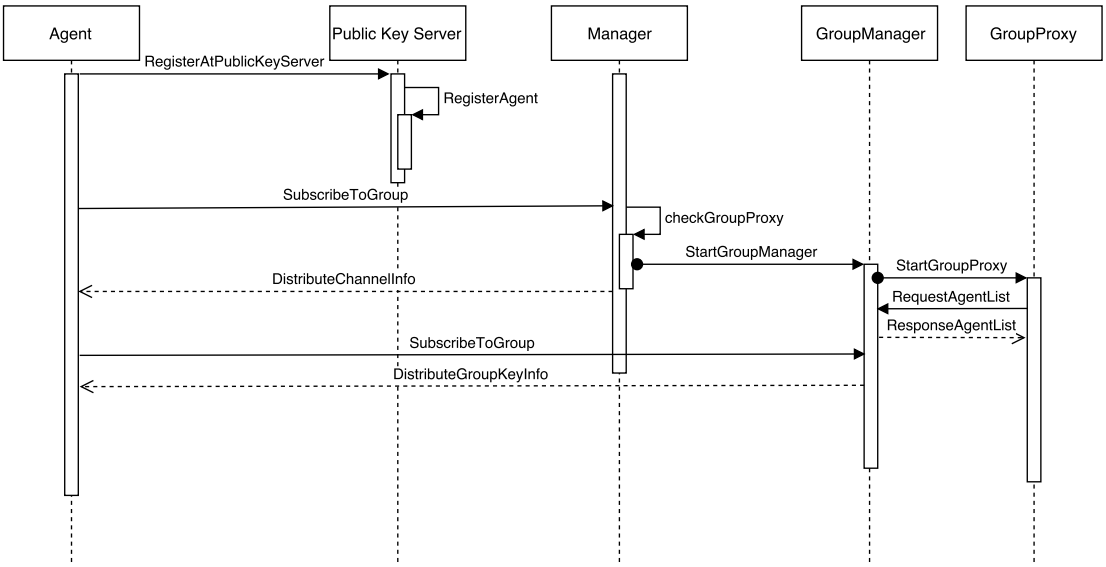

Abbildung 4.2: Ein Agent meldet sich am System, mit einer noch nicht existierenden Gruppe an (Rollen unabhängig)

Je nachdem welche Rolle der Agent hat (Senden oder Empfangen), erweitertet sich der Ablauf. In der Abbildung 4.3 wird näher beschrieben, welche Nachrichten und Informationen nötig sind, damit ein Agent der Rolle Empfangen nachkommen kann.

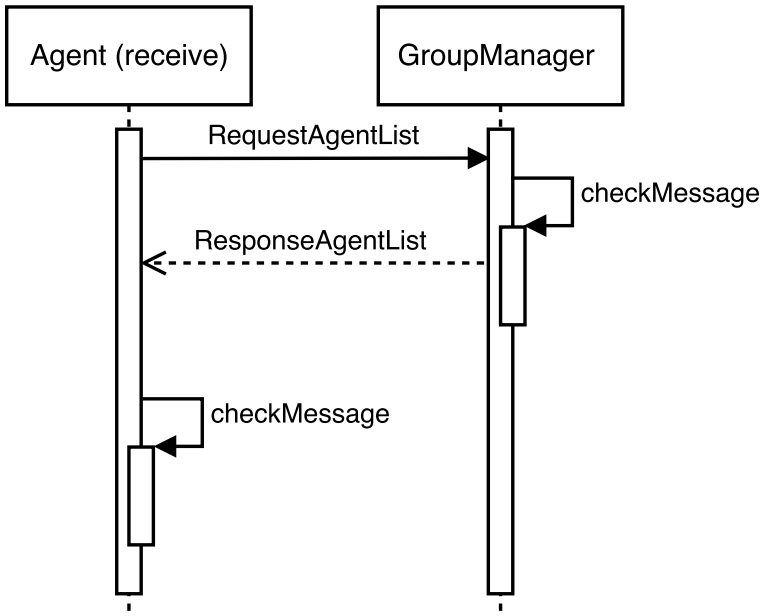

Abbildung 4.3: Erweiterung des Ablaufs

Nimmt der Agent die Rolle des Senders ein, erweitert sich der Ablauf um die Nachrichten, welche im Bild 4.4 zu sehen sind. Ein Agent kann auch beide Rollen einnehmen, dann werden

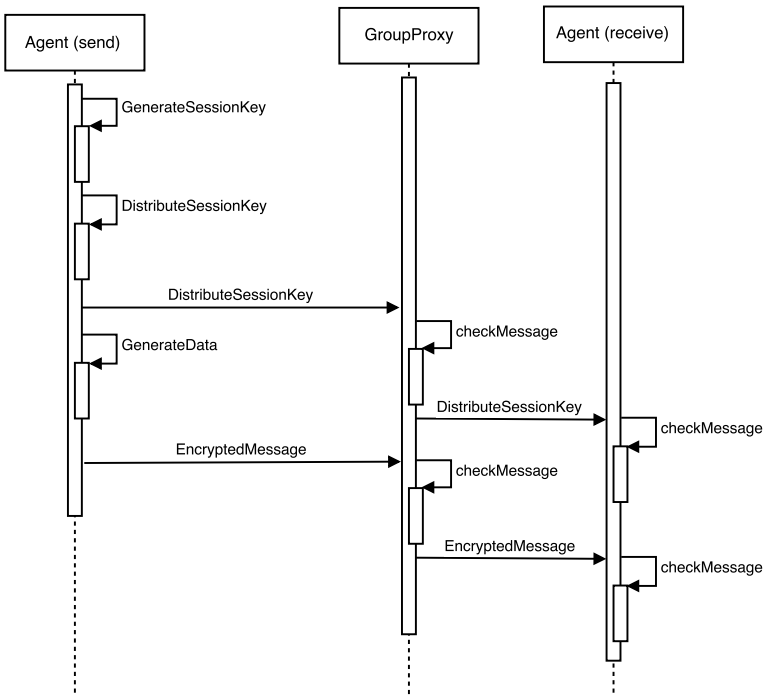

Abbildung 4.4: Erweiterung des Ablaufs

beide zusätzlichen Abläufe durchlaufen.
Nachdem die zusätzlichen Abläufe abgeschlossen wurden, sind die Agenten einsatzbereit und können ihre vorgesehenen Aufgaben bearbeiten.

4.2.2 Nachrichtenübersicht

Nachfolgenden sind die verwendeten Nachrichten aufgeführt und näher beschrieben.

EncrpytedMessage

Bei dieser Nachricht handelt es sich um die Nachricht, die die eigentlichen generierten Daten der Agenten beinhalten. Sie dient dabei nicht als Kontroll- oder Aufbaunachricht, sondern enthält neben der AgentID des Senders, den mit dem Sitzungsschlüssel verschlüsselten Daten, der Signatur der verschlüsselten Daten, die Signatur des Gruppenproxy und die Nounce als Zeitstempel.
Die Signatur des Gruppenproxy wird ergänzt, nachdem der Gruppenproxy diese Nachricht von einem Agenten erhalten hat und geprüft hat, ob der Agent überhaupt eine solche Nachricht verschicken darf. Der Gruppenproxy signiert dabei die komplette ursprüngliche Nachricht und leitet die an die Agenten der Gruppe weiter, welches sich als Empfängeragenten registriert haben.
Da durch das verwendete Protokoll der Middleware (siehe 3.4) sichergestellt ist, dass die Nachricht Reihenfolgen gesichert und zeitnah übertragen werden, ist es nur erforderlich einen Zeitstempel für die Nounce zu verwenden, um prüfen zu können, dass ein Nachricht nicht veraltet wird. Sollte sie es dennoch sein, dann wird die Nachricht verworfen.

DistributeSessionKey

Diese Nachricht zählt zu den Kontroll- oder Aufbaunachrichten. Sie enthält den Sitzungsschlüssel eines bestimmten Agenten, für eine bestimmte Gruppe.
Sie enthält neben der AgentID des Senders, den doppelt verschlüsselten Sitzungsschlüssel, den Kanalnamen, sowie die Signatur des Senders und des Gruppenproxy.
Der Sitzungsschlüssel wird einmal mit dem öffentlichen Teil des asymmetrischen Gruppenschlüssel verschlüsselt, damit sichergestellt ist, dass nur die empfangenden Agenten diesen entschlüsseln können. Zusätzlich ist der Sitzungsschlüssel noch mit dem öffentlichen Schlüssel des Empfängers verschlüsselt. Der sendende Agent pflegt für sich pro Gruppe eine Liste mit den Agenten, welche empfangen dürfen. Diese Liste wird immer, wenn sich an der Liste der Agenten beim Gruppenverwalter etwas ändert, vom Gruppenverwalter aktualisiert und an die

Agenten und an den Gruppenproxy geschickt. Damit soll sichergestellt werden, dass wenn ein Agent nicht mehr berechtigt ist, Nachrichten zu empfangen, er diese nicht weiterhin mit dem Gruppenschlüssel entschlüsseln kann und damit an den aktuellen Sitzungsschlüssel kommt, mit dem er die weiteren Nachrichten entschlüsseln kann.
Die AgentID ist wichtig, damit die Empfängeragenten den Sitzungsschlüssel einem bestimmten Agenten zu ordnen können. Der Kanalname erfüllt dabei einen ähnlichen Zweck und dient für die Zuordnung des Sitzungsschlüssels zu einem bestimmten Kanal und damit zu einer bestimmten Gruppe. Dies ist wichtig, weil es durchaus möglich ist, dass ein Empfängeragent in mehreren Gruppen ist, die verschlüsselte Nachrichten erhalten.
Die Signatur soll sicherstellen, dass der doppelt verschlüsselte Sitzungsschlüssel unterwegs nicht verändert wurde, ohne dass es an dieser Stelle auffallen würden. Der Gruppenproxy ergänzt seine Signatur, nachdem dieser geprüft hat, ob der sendende Agent diese Nachricht auch verschicken darf.

UpdateAgentRole

Diese Nachricht gehört zu den Kontrollnachrichten, sie enthält neben der AgentID des Senders, eine Liste mit den AgentenIDs und deren Rollen, welche zur einer Gruppe gehören, auch einen Hash und eine Signatur.
Diese Nachricht wird immer, wenn sich an dem Status eines Agenten etwas ändert, vom Gruppenverwalter an den Gruppenproxy versendet.
So soll sichergestellt werden, dass der Gruppenproxy immer über eine aktuelle Liste der Agenten und deren Rollen verfügt, damit der Gruppenproxy prüfen kann, ob eine Nachricht eines Agenten, auch weitergeleitet werden darf.

UpdateAgentList

Diese Nachricht gehört zu den Kontrollnachrichten. Sie enthält neben den AgentID des Senders, eine Liste mit AgentenIDs und deren öffentlichen Schlüsseln. Diese Nachricht wird, immer wenn sich an dem Status eines Agenten etwas ändert, vom Gruppenverwalter an den Gruppenproxy und an die Agenten verschickt, welche Daten bzw. Informationen zur Verfügung stellen. Damit wird sichergestellt, dass zum einen der Gruppenproxy den Hash einer empfangenden Nachricht prüfen kann und zum anderen bekommen nur die Agenten den aktuellen Sitzungsschlüssel, die auch vom Gruppenverwalter akzeptiert worden sind.

RequestPublicKey

Mit dieser Nachricht kann beim öffentlichen Schlüssel Server und einer AgentenID, der dazugehörige öffentliche Schlüssel angefragt werden.
Damit der Hash und die Signatur geprüft werden kann, muss auch die AgentID des Senders angegeben werden. Da der öffentliche Schlüssel Server über alle öffentliche Schlüssel verfügt, muss dieser in der Nachricht nicht mit angegeben werden.

ResponseRequestPublicKey

Diese Nachricht ist die Antwort auf die RequestPublicKey - Nachricht. Hier ist neben der AgentID des öffentlichen Schlüssel Servers, der öffentliche Schlüssel zur angefragten AgentID und eine Signatur, damit sichergestellt werden kann, dass die Antwort und der Inhalt der Antwort, auch vom öffentlichen Schlüssel Server kommt.
Sollte zur angefragten AgentID kein Schlüssel vorhanden sein, dann wird eine leere Zeichenkette als Antwort übertragen.

RegisterAtPublicKeyServer

Diese Nachricht ist die erste Nachricht die jeder Agent als erstes beim hochfahren verschickt. Sie geht an den öffentlichen Schlüssel Server und enthält neben der AgentID des Senders, auch den öffentlichen Schlüssel und eine Signatur.
Damit registriert sich ein Agent beim öffentlichen Schlüssel Server und stellt so sicher, dass anderen Agenten, falls diese nicht den öffentlichen Schlüssel eines anderen Agenten haben, diesen beim öffentlichen Schlüssel Server anfragen können.

SubscribeToGroup

Mithilfe dieser Nachricht kann sich ein Agent bei einer Gruppe registrieren. Sie geht zuerst an den Verwalter, der dann prüft, ob für die Gruppe, welche in der Nachricht angegeben wurde, schon ein Gruppenverwalter existiert. Sollte einer existieren, antwortet der Verwalter mit einer DistributeChannelName - Nachricht. Wenn noch kein Gruppenverwalter existiert, wird einer neuer für diese Gruppe gestartet.
Des Weiteren ist die AgentID, der öffentliche Schlüssel, die Rolle (Senden oder Empfangen), eine Signatur und ein Hash des öffentlichen Schlüssels enthalten.

DistributeChannelName

Diese Nachricht wird als Antwort auf eine SubscribeToGroup - Nachricht vom Verwalter an einen Agenten versendet. Sie enthält den Kanalnamen, auf den der Gruppenverwalter erreichbar ist. An diesen muss anschließend erneut eine SubscribeToGroup - Nachricht geschickt werden, damit der Gruppenverwalter zum einen prüfen kann, ob der Agent der Gruppe beitreten darf und zum anderen werden mit der Registrierung des Agenten die Nachricht UpdateAgentList und UpdateAgentRole versendet.

RequestAgentList

Diese Nachricht wird vom Gruppenproxy an den Gruppenverwalter beim hochfahren versendet. Damit fragt der Gruppenproxy die aktuellen Listen mit den Agenten samt deren Rollen und öffentlichen Schlüsseln an.
Als Antwort auf diese Nachricht wird eine UpdateAgentList - und UpdatenAgentRole - Nachricht vom Gruppenverwalter an den Gruppenproxy verschickt.

Verhalten im Fehlerfall

Um ein Fehlverhalten der Services oder Agenten zu verhindern, wird bei jeder Nachricht der Inhalt entsprechend geprüft, wenn eine Nachricht ankommt. Sollte eine dieser Überprüfungen fehlschlagen, dann wird diese Nachricht verworfen. Damit soll sichergestellt werden, dass keine Nachrichten verarbeiten werden, die möglicherweise unterwegs manipuliert wurden oder von der Sendeseite, falsch erstellte Nachricht zu Problemen auf der Empfängerseite führen. Allerdings gibt es keine Lastkontrolle bzw. Lastverteilung, sollte ein Agent oder ein bestimmter Service mit zu vielen Nachrichten bombardiert werden, kann das zu Verzögerungen oder zum Ausfall des Dienstes führen. Einen solchen Angriff allerdings zu erkennen, ist nicht Bestandteil dieser Arbeit.

4.3 Architektur

Zur Umsetzung der verschlüsselten Kommunikation und zur Einhaltung bestehender Anforderungen, wurde die folgende Sicherheitsarchitektur geplant und umgesetzt. Eine Übersicht der umgesetzten Komponente mit ihren Schnittstellen, ist in der Abbildung 4.5 zu sehen. Die

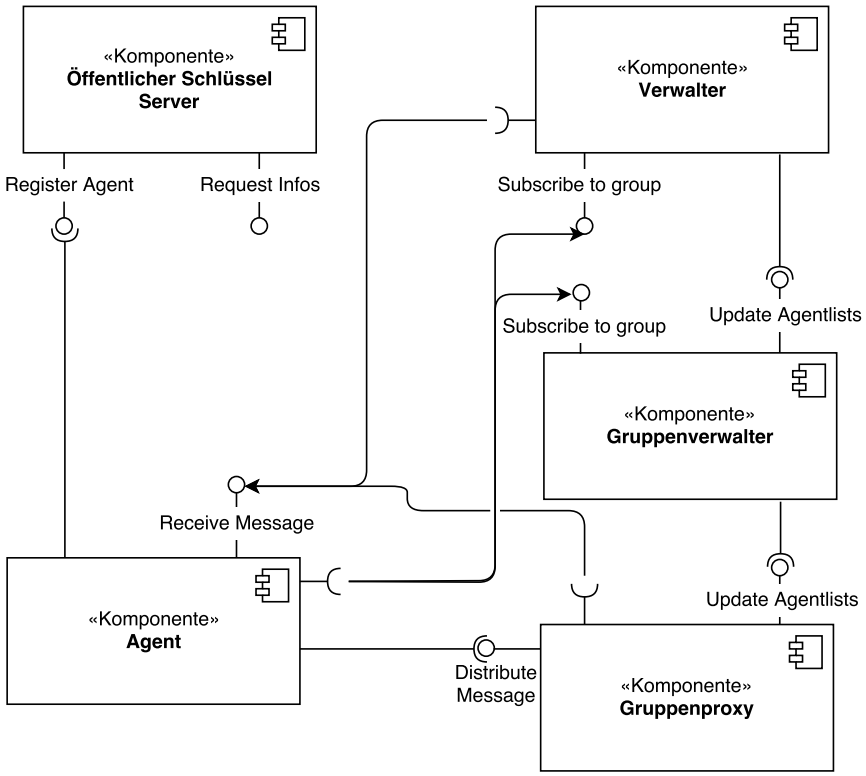

Abbildung 4.5: Übersicht der Komponenten

Architektur bietet neben der Einhaltung der gewünschten Sicherheitsanforderungen, auch den Vorteil, dass einer der Grundgedanken des ursprünglichen Projektes eingehalten und umgesetzt werden konnten. Bei dem Grundgedanken handelt es sich um den Ansatz, dass jeder Agent nur eine bestimmte Aufgabe hat und dass die einzelnen Komponenten einfach ausgetauscht bzw. den eigenen Bedürfnissen angepasst werden können.

4.3.1 Komponenten

OCSP - Dienst

Beim OCSP - Dienst handelt es sich um ein von der IETF standardisiertes Protokoll, um x.509 - Zertifikate auf ihren aktuellen Status zu prüfen.
Dabei wird eine Anfrage an einen Server geschickt, in der Regel vom Verteiler der Zertifikate, welcher mit good (Zertifikat ist nicht gesperrt), revoked (Zertifikat ist gesperrt bzw. ungültig) oder unknown (Zertifikat ist dem Server nicht bekannt) antwortet.

Antworten vom Server sind immer signiert, damit der Anfragende die Echtheit und Unverfälschtheit prüfen kann vgl. RFC 6960. [1]
Aus verschiedenen Gründen wurde dieser Dienst nicht umgesetzt, wie er eigentlich vorgesehen war. Allerdings wurde eine abgespeckte Version von diesem Dienst umgesetzt.
So gibt es im Framework eine Datei, welche sich application.conf nennt und Informationen wie z. B. die Standard IP - Adresse der Middleware beinhaltet. Diese Datei muss immer vorhanden sein und kann um weitere Informationen ergänzt werden. So wurde hier ein Eintrag hinzugefügt, der den öffentlichen Schlüssel des Schlüsselservers und des Verwalters beinhaltet, damit die anderen Dienste und Agenten die Möglichkeit haben, diesen auszulesen und zu prüfen, ob die Nachrichten, die vom Schlüsselserver und vom Verwalter kommen, auch wirklich von diesen Diensten kommen.

Öffentlicher Schlüssel Dienst

Beim öffentlichen Schlüssel Dienst müssen sich alle Parteien mit ihrer AgentID und ihrem öffentlichen Schlüssel registrieren, damit diese später von anderen Komponenten angefragt werden können.
Damit Man-in-the-Middle Angriffe verhindert werden können, war es angedacht, für diesen Dienst ein Zertifikat zu erstellen und dieses von der Certificat Authority unterschrieben zu lassen. Allerdings wurde der OCSP - Dienst nicht so umgesetzt, wie dieser geplant war. Deshalb wurde der öffentliche Schlüssel in der application.conf hinterlegt, damit die Agenten trotzdem prüfen können, ob sie mit dem echten Schlüssel Server kommunizieren.

Verwalter

Der Verwalter kümmert sich zum einen um die Verwaltung der Gruppenverwalter und zum anderen beantwortet er die Anfragen der Agenten, welche einer bestimmten Gruppe beitreten wollen.
Außerdem sorgt dieser Dienst dafür, dass falls eine Gruppe noch nicht existiert und ein Agent dieser Gruppe beitreten möchte, ein neuer Gruppenverwalter für diese Gruppe gestartet wird.
Außerdem kümmert sich der Verwalter um verschiedene Anfragen der Agenten. Bei der einen handelt es sich um die Anfrage, wenn ein Agent einer Gruppe beitreten möchte und die Information braucht, welcher Gruppenverwalter für diese Gruppe zuständig ist und wie er diesen Gruppenverwalter erreicht.
Bei der anderen Anfrage geht es darum, dass falls ein Gruppenverwalter nicht erreichbar ist

[1] https://tools.ietf.org/html/rfc6960

und einem Agent das auffällt, kann er sich an den Verwalter wenden und dieser prüft, ob der Gruppenverwalter erreichbar ist und leitet entsprechende Schritte ein, um das Problem zu lösen.

Gruppenverwalter

Der Gruppenverwalter dient dem Berechtigungsmanagement einer spezifischen Gruppe- Zu dem sorgt der Verwalter dafür, dass mindestens ein Gruppenproxy läuft, erreichbar ist und versorgt diesen in regelmäßigen Abständen mit einer Liste, in der die registrierten Agenten, deren Rolle, die AgentID und öffentlichen Schlüssel stehen. Außerdem dient er als Einstiegspunkt für die Agenten, welche einer Gruppe beitreten möchten und beantwortet die entsprechenden Nachrichten bzw. Anfragen dieser Agenten.
Zudem erzeugt der Gruppenverwalter auch das spezielle Schlüsselpaar (siehe 4.3.4) und kümmert sich um die Verteilung der einzelnen Teile des Paares an die Agenten.

Gruppenproxy

Der Gruppenproxy ist für die Verteilung und Überprüfung der Nachrichten innerhalb einer bestimmten Gruppe zuständig.
Nachdem der Proxy von dem Gruppenverwalter der Gruppe gestartet wird, erhält diese eine Liste mit den Agenten und deren Rolle in der Gruppe. Diese Liste wird in regelmäßigen Abständen vom Gruppenverwalter aktualisiert und an den Gruppenproxy weitergeleitet.
Erhält der Gruppenproxy eine Nachricht von einem Agenten, prüft der Proxy die Signatur der Nachricht und ob der Agenten als sendender Agent für die Gruppe eingetragen ist. Wenn diese Überprüfung erfolgreich verläuft, signiert der Proxy die Nachricht mit seinem eigenen Schlüssel, damit die Empfänger feststellen können, dass die Nachricht vom Proxy überprüft wurde.
Sollte die Überprüfung nicht erfolgreich verlaufen, wird die Nachricht einfach verworfen.

4.3.2 Implementierung der Komponenten

Für die Implementierung der Agenten, wurde zum einen das Framework genutzt, welches im Rahmen der Masterarbeit [Eichler (2014)] erarbeitet wurde und zum anderen wurden Komponenten des akka - Frameworks[2] genutzt.
Das akka - Framework dient unter anderem als Grundlage für das im Rahmen der Masterarbeit entwickelte Framework, allerdings stehen darüber nicht alle Funktionalitäten, welches das

[2] https://akka.io/

akka - Framework bietet, zur Verfügung.
Ein Grund dafür ist, dass zum einen so der hierarchische Aufbau der neuen Komponenten ermöglicht wird, denn wenn die Komponenten auf dem akka - Framework basieren, wird es möglich, dass andere Agenten innerhalb von Agenten gestartet werden können und damit in einer hierarchische Struktur angeordnet werden.
Zum anderen wird so auch der Zugriff auf die Logging - Funktionen des akka - Frameworks ermöglicht wird, welche wichtig werden, um zum einen den Überkick zu behalten, wenn mehrere verteilte Agenten auf einem System arbeiten und zum anderen wird es so leichter das Verhalten des Systems nachzuvollziehen und im Falle eines Fehlers oder Fehlverhaltens zu debuggen.
Des Weiteren wird so auch die Skalierbarkeit des Systems gewährleistet, da dass einer der Ziele des akka - Frameworks ist.

4.3.3 Hierarchischer Aufbau

Das System ist in verschiedene Dienste aufgeteilt, die jeweils eine entsprechende Funktion erfüllen, zusätzlich gibt es aber auch einen hierarchischen Aufbau der Komponenten.
Auf der ersteren Ebene befinden sich der öffentliche Schlüssel Server, der OCSP - Dienst und der Verwalter.
Von diesen Diensten gibt es immer nur einen und diese sind voneinander unabhängig, was bedeutet, dass z.B. der öffentliche Schlüssel Server nicht den OCSP - Dienst braucht um seine Aufgaben zu bewältigen.
Auf der zweiten Ebenen befinden sich die Gruppenverwalter, diese werden immer von dem Verwalter gestartet, wenn eine neue Gruppe dazukommt bzw. es Probleme mit dem bestehenden gibt.
Unter den Gruppenverwaltern befinden sich die Gruppenproxy, von diesen gibt es pro Gruppe immer mindestens einen. Von den Gruppenproxy könnten durchaus mehrere parallel zueinander existieren, um die Last der Nachrichten besser zu verteilen, allerdings ist dies im Rahmen dieser Arbeit nicht vorgesehen.
Und ganz unten befinden sich die Agenten, welche Daten bzw. Nachrichten generieren und empfangen. Diese Agenten sind vom restlichen System losgelöst, was bedeutet, sollte einer von den Agenten ausfallen oder es entstehen Probleme innerhalb der Abläufe des Agenten, dann sind die Komponenten darüber nicht für diese verantwortlich und versuchen nicht durch z.B. einen Neustart die Probleme zu beseitigen.
Bei den anderen Komponenten haben immer die darüber liegenden Systeme die Verantwortung bzw. dienen als Ansprechstelle, sollte einer der Dienste nicht ordnungsgemäß funktionieren,

kann sich an die darüber liegende Instanz gewandt werden und diese versucht dann, etwaige Probleme zu lösen.

4.3.4 Spezielles Schlüsselpaar

Bei dem speziellen Schlüsselpaar, handelt es sich um ein asymmetrisches Schlüsselpaar, bei dem die einzelnen Komponenten bei unterschiedlichen Instanzen sind, je nachdem welche Rolle diese einnehmen. Es wird vom Gruppenverwalter der jeweiligen Gruppe generiert und beim Registrieren der Agenten verteilt. Dabei wird, je nachdem welche Funktion ein Agent erfüllt (Senden oder Empfangen) der entsprechende Teil an den Agenten weitergeleitet.
Damit die sendenden Agenten einer Gruppe, nicht gegenseitig ihre Nachrichten lesen können, erstellen diese in regelmäßigen Abständen einen symmetrischen Schlüssel, mit dem die Nachrichten verschlüsselt werden. Verteilt wird dieser Schlüssel mit dem öffentlichen Teil des speziellen Schlüsselpaares, sodass die Agenten, welche Empfänger sind und damit den privaten Teil des Schlüsselpaares bekommen, diesen entschlüsseln können und somit auch nutzen können.
Damit nicht für jede Nachricht ein neuer Schlüssel generiert werden muss, wird ein Schlüssel alle 60 Minuten neu erstellt, sollte kein neuer Agent dazukommen und dann mithilfe einer Nachricht an die Agenten verteilt.

4.4 Software Bibliotheken

In diesem Kapitel wird auf Punkte eingegangen, die nicht direkt als Sicherheitsmaßnahmen gelten, dieser aber entweder ermöglichen, unterstützen oder besser zur Verfügung stellen.

4.4.1 Bouncy - Castle

In der hier umgesetzten Lösung wird Scala und Java verwendet. Diese bringen standardmäßig keine Unterstützung für die asymmetrische Verschlüsselung mit elliptischen Kurven mit sich. Da diese aber einen wichtigen Bestandteil der Architektur und der Abläufe innerhalb dieser darstellen, wurde Bouncy - Castle als externe Bibliothek hinzugezogen.
Bouncy - Castle enthält eine Vielzahl von kryptografischen Funktionen, darunter die Unterstützung für die asymmetrische Verschlüsselung und einen Sicherheitsprovider.
Bei einem Provider handelt es sich um eine Ansammlung von Klassen, die verschiedene Sicherheitsrelevante Funktionen bereitstellt [3]. Ein solcher Provider wird für Java benötigt, damit

[3] https://docs.oracle.com/javase/8/docs/api/java/security/Provider.html

die entsprechenden Ver- und Entschlüsselungs - Vorgänge möglich werden.
Ein weiterer Vorteil von Bouncy - Castle ist, dass alles unter einer Open Source - Lizenz zur Verfügung gestellt wird und damit jeder sich selbst von der Richtigkeit der Abläufe und des Quellcodes überzeugen kann. Außerdem wird Bouncy Castle von einer gemeinnützigen Organisation - Legion of the Bouncy Castle Inc. - verwaltet, welche die Bibliothek wartet und regelmäßig um neue Funktionen, wie neue Verschlüsselungs- oder Passwort Hash Algorithmen ergänzt. [4]

[4] https://bouncycastle.org/index.html

5 Fazit

5.1 Was wurde erarbeitet?

Im Rahmen dieser Arbeit, wurde eine neue Architektur mit den vorhanden Strukturen konzipiert und umgesetzt, die die verschlüsselte Kommunikation ermöglicht. Dafür wurden neue Nachrichten und ein Protokoll erarbeitet, welches die neue Art der Kommunikation möglich macht. Damit die neuen Nachrichten und das Protokoll genutzt werden können, wurden verschiedenen Komponenten (siehe 4.3.1) und Kryptografische - Funktionen (siehe 3.8) erarbeitet und umgesetzt. Des Weiteren wurde das bestehende Framework erweitert, damit die künftigen Anwender die neue Art der Kommunikation ohne größere Aufwände nutzen können.

5.2 Wurde das Ziel erreicht?

Die Ziele, die im Rahmen dieser Arbeit aufgestellt wurden, wurden durch die entwickelten Strukturen umgesetzt und damit erreicht.

5.3 Wo geht es weiter? Was müsste jetzt weiter gemacht werden?

Im Rahmen dieser Arbeit wurde keine vollständige Sicherheitsarchitektur entwickelt, sondern nur ein Schritt in Richtung einer gemacht. Es gibt verschiedene Punkte, die im Rahmen dieser Arbeit nicht betrachtet wurden.
So wurde die OCSP - Komponente und die dafür notwendigen Zertifikate nicht umgesetzt. Es wurden auch nur die typischen Angriffe auf die verschlüsselte Kommunikation betrachtet, so wurde z. B. nicht auf Schadsoftware, die korrekte Speicherung der verwendeten Schlüssel oder den Seitenkanalangriff eingegangen.
Zudem wurden einige Punkte nur oberflächlich angeschnitten, im Kapitel 2.5 wird näher auf den Dos / DDoS Angriff eingegangen und im Verlauf der Arbeit wird beschreiben, wie dieser theoretisch automatisiert innerhalb des Systems stattfinden kann. Allerdings wird nicht darauf

eingegangen, wie dieser erkannt werden kann oder welche Maßnahmen ergriffen werden müssten, sollte einer stattfinden.
Ein weiterer Punkt der nicht vollständig umgesetzt wurde, ist die Verwaltung von Benutzern. Es ist bei der Planung berücksichtigt worden, dass bestimmte Agenten nicht in bestimmte Gruppen beitreten dürfen und dementsprechend wäre es möglich, solche Agenten beim Gruppenverwalter auszufiltern. Allerdings wurde diese Funktion nicht vollständig umgesetzt. Des Weiteren wäre es auch möglich, eine Benutzerverwaltung mit externen Komponenten umzusetzen, welche mit dem Gruppenverwalter kommunizieren und entsprechende Listen bereitstellen würden.
Ein weiterer Punkt der bei der Planung nicht berücksichtigt wurde, ist die Skalierbarkeit und Performance des Systems. Es macht bei einem hohen Aufkommen von Nachrichten durchaus Sinn, dass nicht nur ein Gruppenproxy die Nachrichten prüft und weiterleitet, sondern mehrere Instanzen von diesem existieren, um die Last besser verteilen zu können. Damit einher müsste ein Konzept für die Synchronisierung und Ausfallsicherheit erstellt werden.

Literaturverzeichnis

[CSTI] */* CREATIVE SPACE FOR TECHNICAL INNOVATIONS */.* https://csti.haw-hamburg.de/. – Zugriff: 08.04.2018

[Agrawal und Mehrotra 2016] Agrawal, A. K. ; Mehrotra, S.: Application of elliptic curve cryptography in pretty good privacy (PGP). In: *2016 International Conference on Computing, Communication and Automation (ICCCA)*, April 2016, S. 924–929

[Ali u. a. 2017] Ali, W. ; Dustgeer, G. ; Awais, M. ; Shah, M. A.: IoT based smart home: Security challenges, security requirements and solutions. In: *2017 23rd International Conference on Automation and Computing (ICAC)*, Sept 2017, S. 1–6

[Atzori u. a. 2010] Atzori ; Luigi ; Iera ; Antonio ; Morabito ; Giacomo: The Internet of Things: A Survey. In: *Comput. Netw.* 54 (2010), Oktober, Nr. 15, S. 2787–2805. – URL http://dx.doi.org/10.1016/j.comnet.2010.05.010. – ISSN 1389-1286

[Barker 2015] Barker, Elaine: *FIPS 186-4 - Digital Signature Standard (DSS)*. National Institute of Standards and Technology, 2015. – Zugriff: 24.04.2018

[Barker 2016] Barker, Elaine: Guideline for Using Cryptographic Standards in the Federal Government: Cryptographic Mechanisms. In: *NIST Special Publication 800-175B* (2016), August, S. 73

[Bentahar und Khosravifar 2008] Bentahar, J. ; Khosravifar, B.: Using Trustworthy and Referee Agents to Secure Multi-Agent Systems. In: *Fifth International Conference on Information Technology: New Generations (itng 2008)*, April 2008, S. 477–482

[BSI 2010] BSI: BSI TR-02102 Kryptographische Verfahren: Empfehlungen und Schlüssellängen / BSI. URL https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf;jsessionid=E7A944870188F6EA59C11B4CAFF06C40.1_cid341?__blob=publicationFile&v=7, Januar 2010. – resreport

[Dierks und Rescorla 2008] Dierks, T. ; Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2 / RFC Editor. RFC Editor, August 2008 (5246). – RFC. – URL http://www.rfc-editor.org/rfc/rfc5246.txt. http://www.rfc-editor.org/rfc/rfc5246.txt. – ISSN 2070-1721

[Dworkin 2007] Dworkin, Morris: Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. In: *NIST Special Publication 800-38D* (2007), November

[Eichler u. a. 2017] Eichler, T. ; Draheim, S. ; Grecos, C. ; Wang, Q. ; Luck, K. von: Scalable context-aware development infrastructure for interactive systems in smart environments. In: *2017 IEEE 13th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*, Oct 2017, S. 147–150

[Eichler 2014] Eichler, Tobias: *Agentenbasierte Middleware zur EntwicklerunterstÃ¼tzung in einem Smart-Home-Labor*, HAW, Diplomarbeit, Oktober 2014

[Foner und N. 1997] Foner ; N., Leonard: Yenta: A Multi-agent, Referral-based Matchmaking System. In: *Proceedings of the First International Conference on Autonomous Agents*. New York, NY, USA : ACM, 1997 (AGENTS '97), S. 301–307. – URL http://doi.acm.org/10.1145/267658.267732. – ISBN 0-89791-877-0

[Giry 2017] Giry, Damien: *BlueKrypt - Cryptography Key Length Recommendation*. https://www.keylength.com/en/compare/. Februar 2017. – Zugriff: 02.04.2018

[Harkins und Carrel 1998] Harkins, Dan ; Carrel, Dave: The Internet Key Exchange (IKE) / RFC Editor. RFC Editor, November 1998 (2409). – RFC. – URL http://www.rfc-editor.org/rfc/rfc2409.txt. http://www.rfc-editor.org/rfc/rfc2409.txt. – ISSN 2070-1721

[IT] IT, Fraunhofer FOKUS Kompetenzzentrum O.: *Seitenkanalanalyse*. https://www.oeffentliche-it.de/trendsonar. – Zugriff: 01.04.2018

[Koppermann u. a. 2016] Koppermann, P. ; Santis, F. D. ; Heyszl, J. ; Sigl, G.: X25519 Hardware Implementation for Low-Latency Applications. In: *2016 Euromicro Conference on Digital System Design (DSD)*, Aug 2016, S. 99–106

[Langley u. a. 2016] Langley, A. ; Google ; Hamburg, M. ; Research, Rambus C. ; sn3rd ; Turner, S.: Elliptic Curves for Security / Internet Research Task Force (IRTF). URL https://tools.ietf.org/html/rfc7748, Januar 2016. – RFC. Zugriff: 24.04.2018

[McDowell 2009] McDowell, Mindi: *Security Tip (ST04-015) Understanding Denial-of-Service Attacks*. https://www.us-cert.gov/ncas/tips/ST04-015. November 2009. – Zugriff: 01.04.2018

[McGrew und Viega 2004] McGrew, David A. ; Viega, John: *The Security and Performance of the Galois/Counter Mode of Operation (Full Version)*. Cryptology ePrint Archive, Report 2004/193. 2004. – https://eprint.iacr.org/2004/193

[NIST 2016] NIST: AES Development / NIST. URL https://csrc.nist.gov/projects/cryptographic-standards-and-guidelines/archived-crypto-projects/aes-development, 2016. – techreport

[Pelzl 2016] Pelzl, Christof Paar; J.: *Kryptografie verständlich: Ein Lehrbuch für Studierende und Anwender (eXamen.press) (German Edition)*. 1. Springer Vieweg, 2016 (eXamen.press). – URL https://www.amazon.com/Kryptografie-verst%C3%A4ndlich-Lehrbuch-Studierende-eXamen-press-ebook/dp/B01KWSCGEW?SubscriptionId=0JYN1NVW651KCA56C102&tag=techkie-20&linkCode=xm2&camp=2025&creative=165953&creativeASIN=B01KWSCGEW. – ISBN 978-3-662-49297-0

[Schmeh 2013] Schmeh, Klaus: *Kryptograpfie - Verfahren, Protokolle, Infrastruktur*. dpunkt.verlag, 2013

[Schneier 1996] Schneier, Bruce: *Angewandte Kryptographie : Protokolle, Algorithmen und Sourcecode in C*. John Wiley & Sons, 1996

[Stewart 2007] Stewart, R.: Stream Control Transmission Protocol / RFC Editor. RFC Editor, September 2007 (4960). – RFC. – URL http://www.rfc-editor.org/rfc/rfc4960.txt. http://www.rfc-editor.org/rfc/rfc4960.txt. – ISSN 2070-1721

[Stewart und Metz 2001] Stewart, R. ; Metz, C.: SCTP: new transport protocol for TCP/IP. In: *IEEE Internet Computing* 5 (2001), Nov, Nr. 6, S. 64–69. – ISSN 1089-7801

[Sulaiman u. a. 2009] Sulaiman, R. ; Sharma, D. ; Ma, W. ; Tran, D.: A Multi-agent Security Architecture. In: *2009 Third International Conference on Network and System Security*, Oct 2009, S. 184–191

[Thangam und Chandrasekaran 2016] Thangam, V. ; Chandrasekaran, K.: Elliptic Curve Based Proxy Re-Encryption. In: *Proceedings of the Second International Conference on In-*

formation and Communication Technology for Competitive Strategies. New York, NY, USA : ACM, 2016 (ICTCS '16), S. 121:1–121:6. – URL http://doi.acm.org/10.1145/2905055.2905337. – ISBN 978-1-4503-3962-9

[Wolf] Wolf, Ruben: https://www.informatik.tu-darmstadt.de/BS/Lehre/Sem98_99/T11/index.html. – Zugriff: 30.03.2018

[YellowIcon] YellowIcon, Everaldo C.: https://de.wikipedia.org/wiki/Datei:Stachledraht_DDos_Attack.svg. – Zugriff: 01.04.2018

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 02. Mai 2018 Florian Stäps