HAW HAMBURG

BACHELORTHESIS
Marc Kreutzmann

Sicherheitsanalyse des Instant-Messengers Riot unter Verwendung des Matrix-Protokolls

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Marc Kreutzmann

Sicherheitsanalyse des Instant-Messengers Riot unter Verwendung des Matrix-Protokolls

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. Klaus-Peter Kossakowski
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 15. Juni 2020

Marc Kreutzmann

Thema der Arbeit

Sicherheitsanalyse des Instant-Messengers Riot unter Verwendung des Matrix-Protokolls

Stichworte

Sicherheitsanalyse, Riot, Matrix, Ende-zu-Ende-Verschlüsselung

Kurzzusammenfassung

Diese Arbeit prüft das Matrix Protokoll, unter Verwendung der von Matrix.org entwickelten Software Riot und Synapse, auf die Umsetzung sicherheitsrelevanter Anforderungen. Um die Anforderungen zu klassifizieren wurde auf Checklisten vom Bundesamt für Sicherheit in der Informationstechnik zurückgegriffen und weitere Punkte durch Brainstorming ergänzt. Die daraus resultierenden Anforderungen beschäftigen sich mit der Sichtung des Netzverkehrs durch Wireshark, der Umsetzung der Verschlüsselung, dem Schlüsselmanagement und der Speicherung der Daten auf dem Server.

Marc Kreutzmann

Title of Thesis

Security analysis of instant Messenger Riot using the matrix protocol

Keywords

Security Analysis, Riot, Matrix, End-to-End-Encryption

Abstract

This work examines the matrix protocol, using the Riot and Synapse software developed by Matrix.org, for the implementation of security-relevant requirements. To classify the requirements, checklists from the Bundesamt für Sicherheit in der Informationstechnik were used and further points were added through brainstorming. The resulting requirements deal with the screening of network traffic through Wireshark, the implementation of encryption, key management and the storage of data on the server.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Das Internet bietet viele unterschiedliche Möglichkeiten, um mit seinen Freunden, der Familie oder anderen Menschen zu kommunizieren. Dabei kommt eine große Anzahl an unterschiedlichen Anbietern zusammen, wobei jeder Anbieter den Schwerpunkt seiner Anwendung anders definiert. Eine Kommunikation per E-Mail bietet sich für einen Austausch an Informationen und Dokumenten an [8], ist aber für die Echtzeit-Kommunikation meist ungeeignet. Chatplattformen für Gruppenchats (z. B. IRC, Slack) bieten die Möglichkeit, sich mit einer beliebigen Menge an Personen über Themen auszutauschen. Bei Gesprächen mit Bekannten bieten Chatplattformen wie WhatsApp oder Telegram eine Option. Die beiden zuvor genannten bieten die Möglichkeit, seine Kommunikationspartner anhand der Handynummer zu identifizieren und danach mit diesen private Gespräche zu führen. Soziale Netzwerke hingegen bieten die Möglichkeit, mit Bekannten in Kontakt zu bleiben oder allgemeine Informationen auszutauschen [8].
Nahezu alle Dienste haben allerdings eines gemeinsam: Beide Seiten der Kommunikation müssen Nutzer des Dienstes sein. Ein Blick auf die Menge der unterschiedlichen Dienste für das Chatten über ein Smartphone veranschaulicht das Problem: Facebook Messenger, WhatsApp, Telegram, Signal, WeChat uvm.. Wenn sich innerhalb eines Bekanntenkreises nicht alle auf eine bestimmte Anwendung einigen können, da jeder einen anderen Dienst aufgrund persönlicher Präferenzen bevorzugt, benötigt jeder in der Gruppe die Schnittmenge der benutzten Dienste, sofern man nicht dazu bereit ist, Abstriche zu machen.
Ein weiteres Problem wird bei der Betrachtung von der Speicherung der Daten deutlich. Diese wird zentral auf den Servern des Anbieters vorgenommen, was den Vorgang für den Benutzer nicht wirklich transparent macht. Der Benutzer erhält keine gesicherten Informationen was und in welcher Form der Anbieter diese Daten speichert und wer auf diese Zugriff hat.

Die gemeinnützige Matrix.org Foundation nimmt sich dieser Probleme an. So sieht deren Manifest vor, dass Personen die volle Kontrolle über ihre Kommunikation haben sollen. Außerdem soll es keine Einschränkungen in der Kommunikation geben, nur weil einer der Gesprächspartner einen anderen Anbieter verwendet [21]. Das Matrix-Protokoll soll dieses Manifest umsetzen. Die Entwicklung eines Clients hierfür ("Riot") wird ebenfalls durch die Foundation vorangetrieben. Dieser Client läuft als Webanwendung, Desktopanwendung oder als App für Android oder iOS.

1.2 Ziele der Arbeit

Ziel dieser Bachelorarbeit ist es sicherheitsrelevante Funktionen im Riot-Client und Matrix-Protokoll zu prüfen. Dazu soll zunächst eine Liste aus der Sicht des Anwenders erstellt werden, welche die an das Instant-Messaging gestellten sicherheitsrelevanten Anforderungen enthält. Anschließend soll für jeden einzelnen dieser Punkte geprüft werden, wie die Umsetzung in Riot oder Matrix erfolgte und ob die Anforderung ganz, teilweise oder gar nicht erfüllt wurde. Für die Analyse kann die Dokumentationen der Software und gleichermaßen der Quellcode der Software herangezogen werden. Außerdem ist die Verwendung externer Tools (bspw. zum Sniffen des Netzwerkverkehrs) vorgesehen.
Ziel der Arbeit soll es dabei explizit nicht sein, Implementierungsvorschläge oder Handlungsanweisungen zu erstellen, welche die Sicherheitslücken schließen oder das Risiko verringern.

1.3 Abgrenzung

Betrachtet werden in dieser Arbeit nur die technischen Sicherheitsmerkmale bei einem Kommunikationsablauf zwischen zwei Riot Clients über das zugrundeliegende Matrix-Protokoll.
Die Matrix-Foundation bietet unter Verwendung der React SDK ein Frontend (Riot) an, welches über das Matrix-Protokoll kommuniziert. Diese Arbeit betrachtet nur das React SDK. Riot bietet seinen Benutzern die Möglichkeit an, Chats zu verschlüsseln. Dabei handelt es sich um eine Einstellung welche explizit aktiviert werden muss. Soweit dies in dieser Arbeit nicht anders gekennzeichnet wird, bezieht sich jede Analyse in dieser Arbeit auf eine aktivierte Verschlüsselung im Chat.

1.4 Verwendete Tools und Werkzeuge

Diese Arbeit untersucht die sicherheitsrelevanten Anforderungen mit Hilfe der folgenden Tools und Werkzeugen. Die Produkte von Matrix.org wurden alle über ihr offizielles GitHub-Repository bezogen (https://github.com/matrix-org):

- **matrix-js-sdk** in der Version 2.0.1
- **matrix-react-sdk** in der Version 1.7.4
- **synapse** in der Version 1.8.0

Bei der Kommunikation zwischen mehreren Homeservern wird zusätzlich der offizielle matrix.org-Homeserver verwendet. Dieser verwendet immer die aktuellste, stabilste Version. Synapse als Homeserver bietet eine öffentliche Schnittstelle, über die es möglich ist, die Version des Homeservers zu prüfen (https://matrix.org/_matrix/federation/v1/version).

Für die Analyse des Quellcodes oder des Netzwerkverkehrs wurden folgende Werkzeuge verwendet:

- **Wireshark** in der Version 3.0.6.
- **IntelliJ** in der Version 2019.1.3.

1.5 Struktur der Arbeit

Die vorliegende Arbeit ist in acht Kapitel aufgeteilt. Das erste Kapitel ist die Einleitung.

Das zweite Kapitel behandelt die Grundlagen für diese Arbeit. Dabei wird auf den generellen Aufbau von Sicherheitsanalysen eingegangen. Zusätzlich werden grundlegende Informationen zum Matrix-Protokoll sowie zum Riot-Client untersucht und es werden andere Messaging-Protokolle hinsichtlich bekannter Sicherheitsprobleme betrachtet.

Das dritte Kapitel legt die zu prüfenden Anforderungen der Arbeit fest. Dabei handelt es sich um die sicherheitstechnischen Anforderungen, welche für einen Benutzer bei Verwendung des Instant-Messaging von Bedeutung sind. Diese Anforderungen entstehen

durch Methoden, die im zweiten Kapitel vorgestellt wurden.

Das vierte Kapitel prüft welche Daten ein passiver Angreifer aus der Überwachung des Netzwerkverkehrs ziehen kann. Dafür wird der Netzwerkverkehr, welcher während der Kommunikation zwischen zwei verschiedenen Clients über das Matrix-Protokoll auftritt, mit Wireshark geprüft und Rückschlüsse gezogen, wie wertvoll gewonnene Informationen sein können.

Das fünfte Kapitel prüft die Umsetzung der Sicherheitsziele, *Vertraulichkeit und Integrität*, die an das Protokoll gestellt werden. Dabei wird ein kurzer Einblick in die Verschlüsselungstechniken von Riot gegeben und geprüft, wie sicher diese umgesetzt wurden.

Das sechste Kapitel prüft die Umsetzung der Authentizität und des Schlüsselmanagements. Dabei wird untersucht wie nutzerrelevante Daten (z. B. die geheimen Schlüssel) im Client gespeichert werden. Außerdem wird auch die Umsetzung der Verifizierung anderer Nutzer betrachtet.

Das siebte Kapitel betrachtet den Homeserver, also den Gegenpart, mit dem der Client Daten austauscht. Dieser wird hinsichtlich der sicheren Speicherung der serverrelevanten Daten geprüft (z. B. Benutzerkonten) und welche Maßnahmen z. B. präventiv getroffen werden, um Angreifern einen Angriff zu erschweren (z. B. Umgang mit sicheren Passwörtern).

Das achte und letzte Kapitel zieht ein Fazit aus den Ergebnissen der Arbeit und gibt einen Ausblick über geplante Umsetzungen im Protokoll oder im zugehörigen Client.

2 Grundlagen

2.1 Durchführung von Sicherheitsanalysen

2.1.1 Grundlegendes

Softwaresicherheitslücken stellen ein großes Risiko dar, besonders für Unternehmen, welche diese Produkte verwenden. Daher ist es wichtig, fortlaufende Prozesse zu etablieren, welche die Sicherheit der eingesetzten Software dauerhaft garantieren. Dabei besteht allerdings immer ein Risiko, dass Schwachstellen unentdeckt bleiben. Durch das Ausnutzen dieser Schwachstellen kann dem Unternehmen ein großer Schaden entstehen - bis hin zur Existenzbedrohung. So zeigt folgendes Beispiel, aus dem Dezember 2019, eine als "kritisch" eingestufte Sicherheitslücke:
Die Citrix Software enthielt eine Schwachstelle, die es einem Angreifer erlaubte spezifischen Code in das System einzuschleusen und auszuführen. Der Angreifer erhielt damit vollständigen Zugriff auf das entsprechende System. Dies ist mit einer Übernahme gleichzusetzen [14][CVE-2019-19781]. Betroffen von dem Problem war die IT tausender Firmen. Durch konkrete Anleitungen zum Ausnutzen der Sicherheitslücke, welche im Netz veröffentlicht wurden, benötigte der Angreifer kein besonderes Fachwissen. Auch das Bundesamt für Sicherheit in der Informationstechnik warnte die deutschen Unternehmen mit Nachdruck [52].

Anhand des vorstehenden Beispiels lassen sich wesentliche Begriffe aus der IT-Sicherheit definieren. Der "**Wert** (asset) ist etwas, das für ein Unternehmen wichtig und unabdingbar ist, um die Geschäftsziele zu erreichen. Werte (assets) müssen daher vor Risiken geschützt werden" [39][S. 209]. Ein **Risiko** entsteht dann, wenn eine Sicherheitslücke bzw. Schwachstelle besteht, auf die eine Bedrohung abzielen kann. Das entstehende Risiko kann den Geschäftsbetrieb beeinträchtigen [39][S. 209]. Es wird als die Kombination aus Wahrscheinlichkeit und Konsequenz eines Ereignisses bezeichnet [47][S. 19]. **Bedrohungen** richten sich auf die Werte des Unternehmens und sind absehbare Szenarien, welche

darauf abzielen, vorher festgelegte Sicherheitsrichtlinien zu verletzen. Eine Bedrohung wirkt sich allerdings nur dann negativ auf den Geschäftsbetrieb aus, wenn dazu eine passende Schwachstelle existiert. **Sicherheitsmaßnahmen** sind vorbeugende Maßnahmen, um Risiken zu minimieren. Diese können in verschiedenster Form (administrativ, technisch, organisatorisch etc.) durchgeführt werden. Bevor Sicherheitsmaßnahmen umgesetzt werden können, müssen diese zunächst durch **Sicherheitsanforderungen** beschrieben werden. Diese ergeben sich aus den festgestellten Bedrohungen [39][S. 207 ff.].

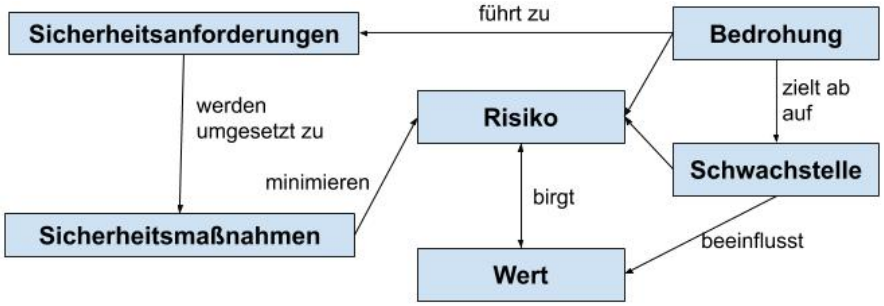

Abbildung 2.1: Übersicht der wichtigsten Begriffe und deren Verhältnis untereinander

2.1.2 Risikoidentifikation

Um sich einen Überblick der Risiken zu verschaffen, müssen diese im ersten Schritt zunächst identifiziert werden. Dieser Schritt ist Teil des Risiko-Assessments, welches zur Ermittlung von nötigen und unnötigen Risiken durchgeführt wird. Für die Identifikation der Risiken gibt es verschiedene Methoden mit jeweils eigenen Merkmalen sowie Vor- und Nachteilen. Die Auswahl der richtigen Methode hängt von verschiedenen Faktoren ab und kann von Unternehmen zu Unternehmen variieren - dies kann auch durch persönliche Vorlieben begründet werden [47][S. 109 ff.]. Auch eine Kombination verschiedener Methoden ist möglich. Im Folgenden werden Methoden aus ISO/IEC 31010 vorgestellt.

Das **Brainstorming** wird nach ISO/IEC 31010 als eine unterstützende Methode definiert und steht meistens zu Beginn einer Analyse. Die Anwendung dieser Methode ist sehr einfach und benötigt keinen großen Ressourcenbedarf. Das Ergebnis der Methode ist sehr stark von den Teilnehmern abhängig, die an dem Prozess beteiligt sind. Diese sollten

mit dem Thema bereits vertraut sein und auch den Mut haben eigene Ideen einzubringen. Im Brainstorming-Prozess sollte jede Idee zunächst einmal notiert werden, auch wenn es Zweifel aus der Gruppe heraus an diesem Punkt gibt. Das Aussortieren und bewerten der einzelnen Punkte kann zu einem späteren Zeitpunkt durchgeführt werden [47][S. 112 ff.].

Eine nach ISO/IEC 31010 definierte Nachschlagemethode ist die Verwendung von **Checklisten** [47][S. 112]. Auch bei dieser Methode ist die Komplexität und der benötigte Ressourcenbedarf eher gering. Checklisten bilden zudem eine gute Ergänzung zu anderen Methoden. "Die Idee, die hinter der Verwendung von Checklisten steht, ist es, auf den Erfahrungen der Vergangenheit oder Best Practices aufzubauen. Checklisten laufen unter dem Motto, das Rad nicht neu zu erfinden"[39][S. 120]. Auch das Bundesamt für Sicherheit in der Informationstechnik (BSI) bietet im Rahmen des IT-Grundschutzes Checklisten für verschiedenste Anwendungen zum Download an [11]. Das Ziel bei der Verwendung von Checklisten ist die Gewissheit, Fehler, welche in der Vergangenheit gemacht wurden, nicht erneut zu wiederholen. Checklisten bieten eine sehr einfache Form der Dokumentation, mit der Information darüber, bekannte Probleme im Griff zu haben [47][S. 120 f.].

Das **Delphi**-Verfahren ermittelt Expertenurteile in einem iterativen Prozess. Dadurch "sollen zufällige Abweichungen, unterschiedliches Verständnis der Fragestellungen [...] von tatsächlichen Differenzen in der Beurteilung eines Sachverhaltes oder einer Zielsetzung getrennt werden" [53][S. 7]. Dazu wird eine These aufgestellt (Schritt 1), welche dann den Experten vorgelegt wird (Schritt 2). Diese beantworten die These anonym. Anschließend wird der Fragebogen ausgewertet und es wird versucht einen Konsens herbeizuführen (Schritt 3). Der gleiche Fragebogen wird den Experten erneut vorgelegt (Schritt 4), allerdings mit den Ergebnissen aus Schritt 3. Solange es noch Meinungsverschiedenheiten zwischen den Experten gibt, werden die Schritte 2-4 erneut durchgeführt. Ziel ist es eine These aufzustellen, mit der alle einverstanden sind. Die Anonymität ist bei diesem Verfahren besonders wichtig, um einzelne Personen nicht durch den Status anderer Personen zu beeinflussen [53][S. 7 ff.].

2.1.3 Risikoabschätzung

Der zweite Schritt des Risiko-Assessments umfasst die Abschätzung der Risiken. Wie der Name bereits verrät, ist dies eine Schätzung, die jedes Unternehmen selbst beurteilen muss. Letztendlich haben aber zwei große Parameter großen Einfluss auf die Schätzung. Zunächst muss beurteilt werden, wie groß die Auswirkung auf die Geschäftstätigkeit ist. Dies sind zum einen die Wiederherstellungskosten, also das System auf den vorherigen, korrekten Zustand zurückführen und der zu entstehende Gesamtschaden. Dem Gegenüber steht die Abschätzung der Wahrscheinlichkeit, welche durch Statistiken und eigene Erfahrungen ermittelt werden können [47][S. 76 f.].

Eine mögliche Methode zur Risikoabschätzung ist die Ursachenanalyse. Bei der **Ursachenanalyse** (Root Cause Analysis, kurz: RCA) geht es darum, einen tatsächlichen Vorfall zu untersuchen und anschließend zu prüfen, wie es zu diesem kommen konnte. Um diese Analyse durchführen zu können, werden alle verfügbaren Informationen zu einem Vorfall herangezogen. Diese werden durch vorhandene Daten und Erkenntnisse ähnlicher Vorfälle ergänzt. Das Ziel ist es hierbei eine lückenlose Dokumentation durch ein Expertenteam zu erstellen, aus denen sich Empfehlungen für die Zukunft ableiten lassen. In Unternehmen gestaltet es sich oftmals als schwierig, Ressourcen für einen bereits abgeschlossenen Sicherheitsvorfall zur Verfügung zu stellen [47] [S. 136 f.].

2.1.4 Risikobewertung

Das bereits vorgestellte Delphi-Verfahren beinhaltet neben der Identifizierung von Risiken auch die Bewertung dieser durch die Experten. Die Ursachenanalyse nimmt ebenfalls eine Risikobewertung mit vor. Eine weitere Methode, welche nur im Bereich der Risikobewertung Anwendung findet, wird im Folgendem erklärt.

Um Risiken zu priorisieren bieten die **Risikoindizes** eine Möglichkeit. Diesem muss ein Verfahren der Risikoidentifikation zuvorgehen. Ziel ist es Risiken auf einen numerischen Wert zu abstrahieren. Für jedes Kriterium wird eine drei- bis zehnstufige Skala festgelegt (z.B. gering - mittel - hoch), welche in einen Zahlenwert übertragen wird (1,0 - 5,0 - 10,0). Unter Berücksichtigung des Gewichtungsfaktors pro Kriterium berechnet sich ein

Gesamtindex aus diesen Werten. Ein Problem an diesem Verfahren ist es, dass der Index für außenstehende nicht besonders transparent hinsichtlich seines Zustandekommens ist [47][S. 148 f.].

2.1.5 Risikobehandlung

Sind die Risiken abschließend bewertet, folgt im nächsten Schritt die Behandlung dieser [47] [S. 81 ff.]. Zur Behandlung gibt es vier Möglichkeiten, auf die im Folgenden kurz eingegangen wird. "Die Entscheidung für eine der vier Alternativen sollte jeweils auf den Ergebnissen des Risiko-Assessments beruhen und das Kosten- Nutzen-Verhältnis berücksichtigen." [47] [S. 82] Auch eine Kombination von verschiedenen Lösungen sollte in Betracht gezogen werden.

Bei der **Risikoreduktion** wird durch Änderungen an den Implementierungen versucht das Risiko zu reduzieren. Dies hat jedoch große Auswirkungen auf alle Beteiligten und sollte genauestens anhand einiger Punkte geprüft werden. Kriterien, die dafür beachtet werden müssen, sind unter anderem: Zeit (z. B. "Ausreichender Zeitaufwand der Implementierung"), Geld, Gesetze, Personal (z.B. "Know-How der Mitarbeiter vorhanden?")

Bei der **Risikoübernahme** entscheidet man sich aktiv den Eintritt eines Schadens in Kauf zu nehmen. Dabei ist es kein Ziel, das Risiko zu ignorieren, sondern sich ganz bewusst für diese Maßnahme zu entscheiden.

Die Unterlassung von risikobehafteten Aktivitäten kann auch eine Maßnahme sein, welche als **Risikovermeidung** definiert wird. Möglich wird dies z. B. durch neuere Geschäftsprozesse, die einen alten ablösen und das Risiko nicht mehr beinhalten.

Die letzte Möglichkeit sieht einen **Risikotransfer** vor. Dadurch wird das Risiko auf andere transferiert (z. B. durch entsprechende Verträge mit anderen Unternehmen oder durch Versicherungen). In der Praxis entstehen dem Unternehmen dennoch immer wieder negative Folgen durch bspw. Kundenverluste.

2.2 Instant Messaging

2.2.1 Bekannte Protokolle

Um einen Vergleich zu aktuell verwendeten Chat-Protokollen ziehen zu können, werden im Folgenden zwei andere Instant-Messaging-Protokolle hinsichtlich ihrer Architektur beschrieben. Im darauffolgenden Kapitel werden dann mögliche Sicherheitsprobleme bei diesen Protokollen thematisiert.

Das **Internet Relay Chat-Protokoll** (kurz: IRC) ist ein offener Standard, welcher in den RFCs 2810-2813 genauer beschrieben wird. Dabei wird ein IRC-Netzwerk als eine Gruppe von einem oder mehreren Servern beschrieben. Die Server eines Netzwerkes sind als Spannbaum untereinander vernetzt. Die Kommunikation zwischen zwei Clients findet immer über den Server und niemals direkt statt. Versendet ein Benutzer eine Nachricht, so leitet der Server diesen an den jeweiligen Server des Benutzers über den kürzesten Weg im Spannbaum weiter. Wird der Empfänger nicht gefunden (z.B. nicht online), dann wird dem Absender eine jeweilige Meldung zurückgegeben. Da im Netzwerk keine Nachrichten gespeichert werden, wird diese Nachricht auch nicht zu einem späteren Zeitpunkt zugestellt. Das IRC-Protokoll unterstützt sowohl Channels (One-To-Many) als auch private Gespräche (One-To-One). Um ein Netzwerk zu betreten ist kein Benutzeraccount notwendig, kann aber erstellt werden [45].

Ein weiteres Protokoll ist das **Extensible Messaging and Presence Protocol** (kurz: XMPP). Dieses wird durch die RFCs 6120-6122 genauer dokumentiert. XMPP verwendet eine Client-Server-Architektur, bei der die Daten im XML Format übertragen werden. Ein Benutzerkonto wird benötigt um den Service verwenden zu können. Usernamen sind ähnlich wie bei Mail-Adressen aufgebaut: *localpart@domainpart*. Der Domainpart gibt dabei an, auf welchem Server der Benutzer registriert ist. XMPP unterstützt laut Protokoll auch das Versenden von Nachrichten an Empfänger, die zur Zeit der Nachricht nicht vom Server erreicht werden (Offline-Nachrichten). Der Server des Empfängers hält in diesem Fall die Nachricht so lange vor, bis der Benutzer sich an diesem wieder anmeldet [57]. Erweitert werden die RFCs durch *XEPs* [26], welche Features behandeln, die nicht zur Kernfunktionalitäten aus den RFCs zählen wie bspw. der Multi-User-Chat (ähnlich dem zum IRC), welcher im XEP-0045 [25] behandelt wird.

2.2.2 Bekannte Sicherheitsprobleme

Im Bezug auf die Architektur des **IRC-Protokolls** existieren Sicherheitsprobleme. Die Authentifizierung zwischen den verschiedenen Servern, sowie die optionale Authentifizierung zwischen einem Client und dem Server, erfolgt auf Basis eines Klartextpasswortes. Dieses kann ein Angreifer leicht mitlesen. Durch die TCP-basierte Verbindung ist es jedoch möglich die Passwörter, durch Nutzung von TLS, für außenstehende zu verschlüsseln. Auch eine Ende-zu-Ende-Kommunikation sieht dieses Protokoll nicht vor, wodurch die Sicherheitsziele Authentizität, Integrität und Vertraulichkeit nicht gegeben sind.
Um z.B. Suchoperationen beantworten zu können, werden lokale Benutzerinformationen an die anderen IRC-Server im Netzwerk weiterverschickt, was zu einer sehr hohen Verbreitung der Daten führt.
Durch den Aufbau des Netzwerkes als Spanning-Tree und der daraus resultierenden fehlenden Redundanz wird jeder Server zu einem Single-Point-of-Failure. Das Ausführen einer DoS-Attacke führt dazu, dass das geschlossene Netzwerk in zwei einzelne gespalten wird, wodurch - je nach Client - die Kommunikation zweier Benutzer in dem ehemaligen Netzwerk nicht mehr möglich ist. Dieses kann dazu führen, dass der Angreifen in bestimmten Channels höhere Rechte erhält, da die verbleibenden Server davon ausgehen, dass dieser der Channelersteller ist [34][S. 480].

Die Architektur des **XMPP-Protokolls** ist hinsichtlich der sicherheitsrelevanten Designentscheidungen bei der Architektur deutlich robuster als das IRC-Protokoll. Daher lohnt es sich hier eher einen Blick auf die weit verbreitete Server-Software **ejabberd** zu werfen. Im Folgenden werden Sicherheitsprobleme aus der ejabberd Version 18.06 beschrieben. Die Standard-Konfiguration unterstützt TLS in den Versionen 1.0, 1.1 und 1.2 und aktiviert diese auch standardmäßig. Das BSI empfiehlt jedoch dringend den Einsatz von TLS erst ab der Version 1.2 [7]. Auch die Schlüssellänge von 1024 Bits, die für das Diffie-Hellman Verfahren standardmäßig angegeben wurde, unterschreitet die vom BSI vorgeschlagene Minimallänge von 2000 Bits [6]. Auch der Logmodus enthält eine unschöne Eigenschaft. So werden im Debug-Modus auch die Passwörter der Benutzer im Klartext mitgeloggt, welche sie beim Anmelden oder beim Passwortwechsel verwenden. Außerdem ist es möglich sämtliche Benutzerdaten zu ändern, auch von Benutzern die nicht auf dem Server existieren. Die persönlichen Benutzerdaten bleiben dabei auch nicht auf einem Server, sondern werden an alle Server dupliziert, mit denen kommuniziert

wird (also durch Chaträume oder Gespräche mit Benutzern auf anderen Servern). Somit ist es auch anderen Server-Admins möglich die Daten auszulesen oder zu manipulieren. [29]

2.3 Instant Messaging mit Matrix/Riot

2.3.1 Matrix-Protokoll

Bei dem Matrix-Protokoll handelt es sich um einen offenen Standard, welcher eine dezentralisierte Echtzeitkommunikation über IP ermöglicht. Matrix definiert hierbei einen Standard und bietet Referenz-Implementierungen in verschiedenen Programmiersprachen als Vorlage an, wodurch Clients implementiert werden können.

Ein Hauptziel von Matrix ist die Unabhängigkeit des Dienstes, die die Anwender verwenden um untereinander zu kommunizieren. Auch Anwender unterschiedlicher Dienste sollen problemlos untereinander kommunizieren können. Als Vergleich ist auf der Matrix-Webseite der Versand einer E-Mail aufgeführt. Dieser kann über die unterschiedlichsten Programme erfolgen, ohne sich mit dem Empfänger auf einen Dienst zu einigen.
Der Protokollname leitet sich von der mathematischen Matrix ab, in der einzelne Ausdrücke ebenfalls als eine eigene Einheit betrachtet werden [19].

Zentraler Bestandteil der Architektur des Protokolls sind **Homeserver**, welche den gesamten Kommunikationsverlauf und Accountdaten speichern, für den sie zuständig sind. Jeder Homeserver kann eine beliebige Anzahl von Benutzern verwalten. Soll eine Nachricht an einen Benutzer eines anderen Homeservers gesendet werden, so übernimmt der Homeserver des Benutzers die Aufgabe, diese an den richtigen Homeserver zu versenden, wie Abbildung 2.2 zeigt [20]. Nachrichten, die mehrere Homeserver betreffen, werden auf allen betroffenen Homeservern repliziert. Mehrere Homeserver sind innerhalb eines Raumes dann betroffen, sobald zwei unterschiedliche Gesprächsteilnehmer unterschiedliche Homeserver verwenden. Dadurch hat kein Homeserver die alleinige Macht über die Daten, die er hält.

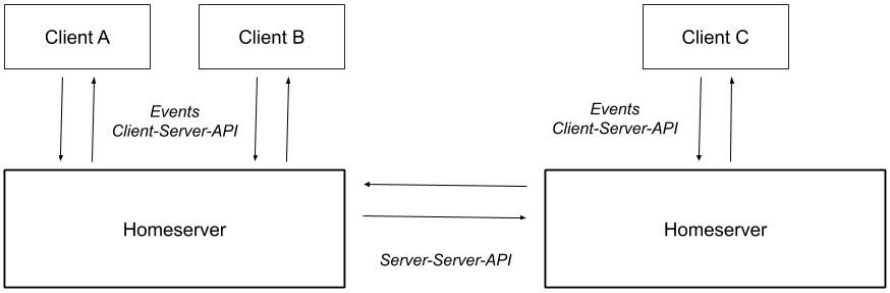

Abbildung 2.2: Vereinfachte Client-Server-Architektur im Matrix Netzwerk

Die Implementierung eines solchen Homeservers wird über das GitHub-Repository angeboten. **Synapse** heißt das in Python geschriebene Produkt der Referenz-Implementierung eines Homeservers von Matrix. Jeder kann einen solchen Server selber installieren und somit ins Matrix-Netzwerk aufnehmen [1].

2.3.2 Riot-Client

Bei Riot handelt es sich um einen Frontend-Client für das Matrix-Protokoll. An der Entwicklung ist größtenteils die Firma beteiligt, welche auch am Matrix-Protokoll beteiligt ist. Riot wird sowohl als Webanwendung als auch als Desktop, Android oder iOS Anwendung angeboten. Der Quellcode ist öffentlich über GitHub zugänglich. Für die Web- und Desktopanwendung unterteilt sich dieser in folgende GitHub-Projekte:

- **vector-im/riot-web** - Enthält das Aussehen (Skin) des Clients
- **matrix-org/matrix-react-sdk** - Enthält den eigentlichen Client
- **matrix-org/matrix-js-sdk** - Enthält die wesentlichen Funktionalitäten für den Client. Das *matrix-react-sdk* basiert auf diesem.

Riot enthält die wichtigsten Funktionalitäten die im Matrix-Netzwerk möglich sind. Dazu zählen Videochat, Telefonie oder eine Ende-zu-Ende-Verschlüsselung.
Jedem ist es möglich einen eigenen Client, als Alternative zu Riot, zu schreiben. Dieser kann ebenfalls auf Basis der Javascript-SDK geschrieben werden.
Durch die Verwendung des Electron-Frameworks unterscheiden sich die Web- und Desktopversion nicht. Electron ermöglicht Desktop Anwendungen mit HTML, CSS und JavaScript auf Basis des Chromium-Browsers und der JavaScript Runtime node.js zu bauen [2].

3 Ziele der Sicherheitsanalyse

3.1 Identifikation

3.1.1 Checklisten

Um die Ziele der Sicherheitsanalyse festzulegen, wird auf Methoden der Risikoidentifizierung zurückgegriffen, welche im vorherigen Kapitel thematisiert wurde. Sicherheitsrisiken, die nicht direkt die Software betreffen, allerdings die Sicherheit dennoch beeinträchtigen können, werden im Scope dieser Arbeit nicht berücksichtigt. Dazu zählt z. B. der Webserver, auf dem der Homeserver läuft oder das Endgerät, auf dem der Client installiert ist. Eine Checkliste für den Einsatz von Instant-Messaging bietet das BSI nicht an. Zwei andere Checklisten zielen jedoch auf Bereiche ab, die für diese Arbeit von Bedeutung sind. Dies sind die Checklisten zu Webanwendungen und zum Kryptokonzept.

Durch die Kommunikation zwischen dem Client und seinem als Webanwendung auf einem Server laufendem Homeserver, wurde zunächst die Checkliste des BSIs für Webanwendungen [5] als relevant betrachtet und gesichtet. Diese sieht eine geeignete Authentisierung am Webserver vor und definiert diese in wesentlichen Anforderungen. Demnach müssen die Zugangsdaten angemessen geschützt werden. Eine Möglichkeit hierzu ist die Verwendung von Salted Hashes für das Passwort. Dabei wird das Passwort zunächst um eine definierte Zeichenkette erweitert und anschließend erst zusammen gehasht. Zusätzlich muss der Benutzer dazu gezwungen werden, ein sicheres Passwort zu verwenden. Das Programm sollte Grenzwerte für fehlgeschlagene Anmeldeversuche definieren [5][APP.3.1.A1, APP.3.1.A14] um Brute-Force Attacken durch das Probieren verschiedener Kombinationen stark einzugrenzen. Diese Punkte werden in der Anforderungsliste zusammengefasst unter dem Punkt “Authentisierung am Homeserver”.

Um SQL-Injections zu vermeiden, müssen alle Interaktionen mit der Datenbank abgesichert werden. Dies geschieht durch die Verwendung von Stored Procedures oder Prepared SQL Statements [5][APP.3.1.A19]. SQL-Injections bieten einem Angreifer die Möglichkeit, innerhalb von gewollten SQL-Abfragen auch weitere, ungewollte, anzufügen, die vom Programm nicht beabsichtigt sind. Inwieweit SQL-Injections eine Rolle spielen wird unter dem Punkt "Schutz vor SQL-Injections" geprüft.

Für die Kommunikation zwischen dem Client und dem Server werden Tokens verwendet, welche den Client am Server authentifizieren. Diese müssen, nach der BSI-Checkliste, zufällig und in ausreichender Länge erstellt werden. Der Austausch der Daten zwischen Client und Server muss ausreichend geschützt werden, da ein Angreifer, der einen Token abfängt, ebenfalls Aktionen mit diesem ausführen kann. Außerdem muss der Benutzer diese Sitzung beenden können [5][APP.3.1.A3]. Geprüft wird dieses durch "Sicheres Session-Management".

Die folgenden Punkte beziehen sich auf die Checkliste zur Erstellung eines Kryptokonzeptes des BSIs [13]. Dies soll besonders das Sicherheitsziel "Vertraulichkeit" gewähren. Dafür muss die Verschlüsselung der Nachrichten geeignete kryptograpfische Verfahren und aktuell empfohlene Schlüssellängen aufweisen, um den gegenwärtigen optimalsten Schutz zu erreichen [13][CON.1.A1]. Das BSI gibt sowohl zu geeigneten Verfahren als auch zu den Schlüssellängen Empfehlungen an, mit denen die Implementierung verglichen werden kann. Der Punkt "Auswahl geeigneter kryptografischer Verfahren" prüft dieses.

Die Kommunikation zwischen dem Client und dem Server sowie die Kommunikation der Server untereinander sollte ebenfalls verschlüsselt werden. Daraus sollten keine Informationen zu ziehen sein, wer mit wem und in welcher Art kommuniziert. Geprüft wird dies unter dem Punkt "Verschlüsselung der Kommunikationsverbindungen" [13][CON.1.A3].

Zusätzlich beschreibt die Checkliste ein geeignetes Schlüsselmanagement, welches unter dem Punkt "Geeignetes Schlüsselmanagement" geprüft wird. Dieses sieht vor, dass Schlüssel möglichst nur für einen Einsatzzweck verwendet werden sollen. Die Verschlüsselung und die Signaturbildung sollte mit unterschiedlichen Schlüsseln geschehen. Zusätzlich sollten die Schlüssel häufig gewechselt werden, sowie sicher aufbewahrt und verwaltet werden [13][CON.1.A4].

3.1.2 Brainstorming

Die meisten Anforderungen an eine sichere Kommunikation über einen Instant-Messenger wurden unter Zuhilfenahme der Checklisten bereits aufgeführt. Zwei weitere Punkte wurden durch Brainstorming ermittelt. Diese wurden in den vorherigen Anforderungen nicht explizit genannt.
Dabei handelt es sich neben dem Sicherheitsziel der Vertraulichkeit auch um die Ziele der Integrität und Authentizität von Nachricht welche beide ebenfalls als Anforderung aufgenommen wurden. Veränderungen an einer Nachricht sollten durch Dritte nicht unbemerkt vorgenommen werden können. Außerdem sollte der Empfänger / Sender einer Nachricht immer die Person sein, die sie angibt zu sein. Der Status der Person wird hierbei allerdings als "zuvor verifiziert" angenommen. Besonders geht es darum einen Man-in-the-Middle zu erkennen, welcher möglicherweise Ursprung oder Empfänger einer Nachricht sein kann. Zusätzlich muss besonderes Augenmerk auf mögliche Replay-Attacken gelegt werden, bei denen sowohl die Integrität als auch die Authentizität einer Nachricht verletzt wäre.

3.2 Zusammenfassung

Die vorher definierten Sicherheitsanforderungen an die Software werden, in der nachfolgenden Übersicht, noch einmal tabellarisch zusammengefasst. Wird in den weiteren Kapiteln Bezug auf einen dieser Punkte genommen, so wird stets der Identifikator angeführt. Die Reihenfolge der Liste ergibt sich aus der chronologischen Reihenfolge in den folgenden Kapiteln.

Identifikator	Beschreibung
S1	**Verschlüsselung der Kommunikationsverbindungen**
a.)	Zuordnung Absender & Empfänger
b.)	Art der Kommunikation
c.)	Zeitverhalten
S2	**Gewährleistung der Authentizität der Nachrichten**
S3	**Schutz vor unbemerkten Veränderungen einer Nachricht**
S4	**Auswahl geeigneter kryptografischer Verfahren**
a.)	Verwendung geeigneter kryptografischer Verfahren
b.)	Verwendung empfohlener Schlüssellänge
S5	**Geeignetes Schlüsselmanagement**
a.)	Schlüssel wird häufig gewechselt
b.)	Sichere Aufbewahrung der Schlüssel
c.)	Schlüssel nur für einen Einsatzzweck
S6	**Schutz vor SQL-Injections**
S7	**Authentisierung am Homeserver**
a.)	Ausreichender Schutz der Zugangsdaten
b.)	Benutzung eines sicheren Benutzerpasswortes
c.)	Anzahl fehlgeschlagener Anmeldungen begrenzen
S8	**Sicheres Session-Management**
a.)	Verfall nach Sitzungsbeendigung
b.)	Zufällige und ausreichende Länge der Tokens
c.)	Schutz beim Austausch zwischen Client & Server

Tabelle 3.1: Übersicht festgelegter Sicherheitsanforderungen an das Matrix-Protokoll

4 Verkehrsflussanalyse

4.1 Problemstellung

Eine Verkehrsflussanalyse wertet Metadaten und nicht verschlüsselte Informationen einer Nachricht aus. Dies hat das Ziel, Rückschlüsse aus dem sichtbaren Datenverkehr zu ziehen. Bei den Metadaten kann es sich um die Länge, den Zeitpunkt oder den Aufbau der Nachricht handeln.
Dieses Kapitel geht dabei stets von einem passiven Angreifer aus. Passiv bedeutet, dass der Angreifer die Kommunikation zwar mitlesen kann, allerdings ohne die Möglichkeit Verschlüsselungen zu brechen [54, S.46].
Bei Riot erfolgt die Kommunikation mit dem Homeserver über eine TLS-Verschlüsselung, somit erhält ein passiver Angreifer keinen Zugriff auf die einzelnen Nutzdaten der Kommunikation.
Welche möglichen Rückschlüsse ein Angreifer aus den Metadaten gewinnen kann und welche Auswirkung dies haben kann, wird in den folgenden Kapiteln genauer beschrieben.

4.2 Zuordnung von Absender und Empfänger

4.2.1 Versuchsaufbau

Für den Versuch wurden zwei Accounts auf unterschiedlichen Homeservern verwendet. Beide Accounts kommunizierten über verschiedene Endgeräte durch den Riot-Client miteinander. Auf beiden Endgeräten lief Wireshark mit, um den Netzwerkverkehr aufzuzeichnen (vgl. Abbildung 4.1). Die beiden Accounts kommunizierten über einen Zeitraum von circa zehn Sekunden miteinander und tauschten dabei Nachrichten unterschiedlichster Länge aus. In diesem kurzen Zeitraum genügen schon wenige Pakete um Rückschlüsse

aus diesen zu ziehen. Innerhalb von zehn Sekunden lassen sich genügend Nachrichten verschiedenster Länge versenden, um eine Analyse vorzunehmen. Da die IP-Adressen der jeweiligen Homeserver bekannt sind, konnte Wireshark auf diese gefiltert werden, um anderen angefallenen Netzverkehr aus der Analyse auszuschließen und das Ergebnis übersichtlicher zu halten.

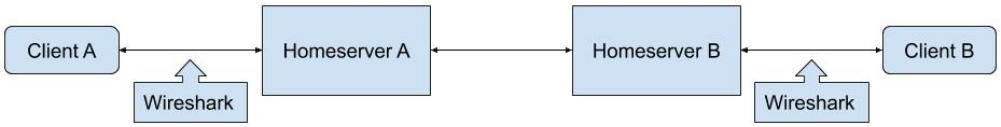

Abbildung 4.1: Grafische Darstellung des Versuchsaufbaus

4.2.2 Ergebnis

Dass ein Nachrichtenverkehr stattgefunden hat, lässt sich durch die Wireshark Logs belegen. Es erfolgt in dem Fall eine Nachricht vom Homeserver zum jeweiligen Client. Den zugehörigen Homeserver zu einem Nutzer kann man anhand seiner Nutzeridentifikation herausfinden. Dies grenzt zwar den Kreis der möglichen Absender und Empfänger einer Nachricht ein, allerdings lässt sich das nicht auf einen Benutzer festlegen.
Dadurch, dass der Client den Homeserver ständig nach Updates fragt (Pull-Prinzip), findet eine ständige Kommunikation zwischen den beiden statt. Besonders Nachrichten kleinerer Länge fallen so in den Logs nicht auf, da anhand der Logs nicht ersichtlich wird, ob es sich um eine Nachricht von einem anderen Nutzer oder eine technische Nachricht handelt.
Nachrichten, die deutlich mehr Text beinhalten, stechen jedoch in den Logs besonders heraus. Zwar ist auch hier der Absender und Empfänger nur anhand des Homeservers identifizierbar, allerdings kann man anhand der Zeit und des deutlich längeren Payloads mit einer hohen Wahrscheinlichkeit sagen, dass hier eine Kommunikation stattgefunden hat.

Über die Paketlänge lassen sich keinerlei Informationen ziehen, da der Payload der ankommenden Nachricht immer deutlich größer ist, als die der ausgehenden Nachricht. Für die Kommunikation mit dem Homeserver verwendet Riot JSON-Arrays. Für das Versenden der Nachricht wird ein Array mit zwei Schlüsseln erzeugt, wovon einer die eigentliche Nachricht beinhaltet und der andere die Information, dass es sich um eine Textnachricht handelt. Für das Empfangen einer Nachricht, wird ein deutlich umfangreicheres

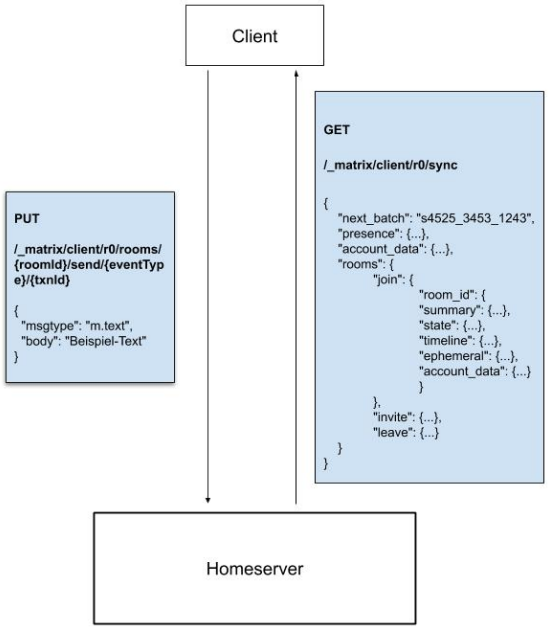

Abbildung 4.2: Client-Server Austausch beim Versand einer Nachricht und anschließender Synchronisierung (unverschlüsselte Sicht)

Array an den Client gesendet, welches alle Änderungen seit der letzten Synchronisierung beinhaltet. Auch wenn alle Werte leer sind, handelt es sich hierbei um ein sehr langes JSON-Array. Dadurch entsteht eine unterschiedliche Länge zwischen dem Ausgangspaket und dem Eingangspaket, was das zuordnen der Pakete unmöglich macht. Abbildung 4.2 verdeutlicht dies.

4.3 Zeitverhalten

4.3.1 Versuchsaufbau

Unter Verwendung der Javascript SDK wurde ein Programm geschrieben, welches automatisert (in diesem Fall alle fünf Minuten) Nachrichten an einen festgelegten Empfänger schreibt. Die Nachricht enthält dabei immer den gleichen Text. Ziel ist es dabei festzustellen, ob Nachrichten technischer Art in der Kommunikation auffallen. Ein Beispiel für ein solchen Fall wäre ein Monitoring-System. Dieses meldet dem Benutzer ob das System in Ordnung ist.

Besonderes Augenmerk wird hierbei auch auf die Uhrzeit gelegt, da technische Nachrichten meistens über ein längeren Zeitraum Sekundengenau regelmäßig versendet werden (Bspw. stündlich um xx:15:00 Uhr).
Absender und Empfänger verwenden hierbei einen unterschiedlichen Homeserver.

4.3.2 Ergebnis

Abbildung 4.3 zeigt die TLS-Pakete an, welche Teil der beim Empfänger zugestellten Nachricht sind. Die Uhrzeit zeigt leichte Differenzen auf, welches durch Latenzen zu begründen ist, die bspw. auch bei der Kommunikation zwischen den beiden Homeservern auftreten können. Die Länge der einzelnen Pakete ist jedoch nur minimal anders. Über einen größeren Zeitraum betrachtet, kann ein Angreifer mit hoher Wahrscheinlichkeit sagen, dass es sich hierbei um automatisierte Nachrichten handelt.

180	13:45:41,050076	173.212.237.230	192.168.178.20	TLSv1.2	997 Application Data
184	13:45:41,187566	173.212.237.230	192.168.178.20	TLSv1.2	386 Application Data
441	13:50:40,116064	173.212.237.230	192.168.178.20	TLSv1.2	992 Application Data
445	13:50:40,221085	173.212.237.230	192.168.178.20	TLSv1.2	386 Application Data
639	13:55:40,135316	173.212.237.230	192.168.178.20	TLSv1.2	994 Application Data
643	13:55:40,290956	173.212.237.230	192.168.178.20	TLSv1.2	386 Application Data
820	14:00:42,831166	173.212.237.230	192.168.178.20	TLSv1.2	999 Application Data
824	14:00:42,973201	173.212.237.230	192.168.178.20	TLSv1.2	386 Application Data
1002	14:05:40,449183	173.212.237.230	192.168.178.20	TLSv1.2	996 Application Data
1006	14:05:40,562426	173.212.237.230	192.168.178.20	TLSv1.2	386 Application Data

Abbildung 4.3: Auszug aus den Wireshark-Logs beim Eintreffen der Nachrichten

4.4 Art der Kommunikation

4.4.1 Versuchsaufbau

Um zu prüfen, ob die Art der Kommunikation durch die Wireshark-Logs erkennbar ist, wurden zwischen zwei Clients wieder Nachrichten ausgetauscht. Zusätzlich wurde über beide Clients ein Anruf per VoIP getätigt, welchen Riot in seinem Funktionsumfang ebenfalls anbietet.

4.4.2 Ergebnis

Riot verwendet für VoIP, aufbauend auf dem Interactive Connectivity Establishment-Protokoll (kurz: ICE), den WebRTC-Standard. Das ICE-Protokoll ist dabei bemüht, den effektivsten Weg zwischen zwei Endpunkten zu finden. Für das ICE-Protokoll geben beide Endpunkte die IP-Adresse und den Port an, über den sie erreichbar sind. Innerhalb eines Netzwerkes führt dies dann zu einer direkten Verbindung. Steht zwischen den beiden Endpunkten jedoch ein NAT, so würde der Versuch die private IP-Adresse des anderen Endpunktes zu erreichen scheitern.
Nach den zuvor erfolglosen Versuchen, wurde als nächstes versucht über das STUN-Protokoll eine Verbindung zwischen den beiden Endgeräten herzustellen. Der STUN-Server wurde dabei angefragt, die öffentliche IP-Adresse und den Port bekannt zu geben, unter welchem der Endpunkt von außerhalb des Netzwerkes zu erreichen ist [37] [56].
Bei der Nutzung eines Sprachanrufes über Riot, wurde die Anfrage an den STUN-Server allerdings in den Wireshark Logs sichtbar. Dadurch kann ein Angreifer schließen, dass eine Kommunikation per VoIP stattfindet. Außerdem steigt die Anzahl der Pakete, die bei einem Anruf an den Homeserver versendet werden, im Vergleich zu einem reinem Textchat deutlich an.

4.5 Bewertung

Erfolgt die Kommunikation zwischen dem Client und dessen Homeserver TLS-Verschlüsselt, so können nur noch die Metadaten durch einen Angreifer gewonnen werden. Die Zuordnung von Absender und Empfänger, welche sich aus der Anforderung S1a ergibt, ist durch die unterschiedlichen Paket-Größen beim Versenden und beim Empfangen von Nachrichten für einen Angreifer sehr schwierig. Lediglich eine Vermutung mit Hilfe des Zeitpunktes vom Versand und dem anschließendem Empfang kann der Angreifer anstellen. Allerdings ist dies auch sehr ungenau, da ein Nutzer bei allen Interaktionen in seinen Räumen ein Paket erhält und dies bei vielen oder sehr großen Räumen einen größeren Datenstrom erzeugen kann. Zusätzlich kann der Zeitpunkt durch Latenzen leicht schwanken. Eine Möglichkeit um dies ganz zu umgehen, wäre es bei jeder Nachricht “Fake-Pakete” an alle anderen Nutzer zu senden (die dann innerhalb des Clients nicht verarbeitet werden, sondern nur in Wireshark sichtbar sind). Da hierbei allerdings die Effizenz in Frage gestellt werden kann, wird die Zuordnung von Absender und Empfänger mit einem minimalen Restrisiko aufgrund des Zeitpunktes als Anhaltpunkt versehen.

Durch die, in den Wireshark-Logs sichtbare, Adressierung eines STUN-Servers kann ein Angreifer deutlich erkennen ob ein VoIP-Anruf erfolgt ist oder nicht. Die Anforderung S1b ist damit nicht erfüllt. Kritische Auswirkungen hat dies jedoch nicht, da sämtlicher Payload dennoch verschlüsselt übertragen wird.
Ähnlich sieht es beim Zeitpunkt der Nachricht, für die Anforderung S1c, aus. Liest ein Angreifer über einen längeren Zeitraum den Netzwerkverkehr mit, kann er - trotz Latenzen - von automatisieren Nachrichten ausgehen, wodurch die Anforderung nicht erfüllt ist. Jedoch ist es hierbei ebenfalls so, dass die Auswirkungen nicht besonders hoch sind.

5 Verschlüsselung

5.1 Grundbegriffe

5.1.1 Curve25519 - Schlüsselpaar

In Matrix besitzt jedes Gerät einen Curve25519-Identifizierungsschlüssel. Dieser ermöglicht es mit anderen Geräten ein "shared secret" zu vereinbaren. Möglich wird dies durch das Diffie-Hellman-Verfahren unter Verwendung von elliptischen Kurven (ECDH) [30] [18].
Die Verwendung elliptischer Kurven bietet den Vorteil einer deutlichen Reduzierung des Rechen- und Speicheraufwandes. Dabei wurden bei der herkömmlichen Lösung, also ohne der Verwendung von elliptischen Kurven, Parameter mit einer Länge von über 4096 Bit verwendet. Die elliptischen Kurven "kommen mit Parametern der Länge von 160-256 Bit aus" [38, S. 347]. Die Reduktion von Speicher- und Rechenaufwand geht dabei nicht auf Kosten der Sicherheit. Diese ist in beiden Verfahren gleicher Maßen gegeben [1].

5.1.2 Ed25519 - Fingerprint

Ed25519 ist eine Public-Key Signatursoftware um die Nachrichten in Matrix zu signieren. Jedes Gerät besitzt ein Schlüsselpaar, wobei der öffentliche Schlüssel für andere Geräte im Netzwerk publiziert wird. Das Verfahren setzt dabei auf einen sehr schnellen Algorithmus zum Signieren und zum Verifizieren der Signaturen. Die Geschwindigkeitsvorteile ergeben sich auch durch kurze, komprimierte Signaturen (64 bytes) und Schlüssel. Wie

[1] Das originale Paper sah zwar Curve25519 als Bezeichnung für das DH Verfahren vor, dies hat D. Bernstein allerdings später nochmal korrigiert und neu festgelegt: So heißt das Verfahren eigentlich **X25519** für das DH Verfahren und **Ed25519** für das Signaturensystem. Beide verwenden **Curve25519** als elliptische Kurve [10]. In Matrix ist dies allerdings nicht komplett angepasst, so dass man bei Curve25519 eigentlich von X25519 spricht.

beim Curve25519 werden die Geschwindigkeitsvorteile ohne Sicherheitskompromisse erzielt.
Sowohl das Curve25519- als auch das Ed25519-Schlüsselpaar werden bei der Registrierung neuer Geräte erstellt. Durch den Ed25519 Fingerprint wird der öffentliche Teil des Curve25519-Schlüssels signiert.[31] [18]

5.2 Megolm-Verschlüsselung

5.2.1 Aufbau

Die Megolm-Verschlüsselung wird bei Nachrichten verwendet, welche theoretisch an mehrere Empfänger versendet werden können. Nachrichten, welche mit Megolm verschlüsselt wurden, werden intern als *m.megolm.v1.aes-sha2* gekennzeichnet. Der Name ist dabei so gewählt, dass dieser die wichtigsten Grundelemente des Algorithmus beschreibt und dabei noch gut lesbar bleibt [12, 13.11.4.1 Messaging Algorithm Names].

```
{
  "type": "m.room.encrypted",
  "content": {
    "algorithm": "m.megolm.v1.aes-sha2",
    "sender_key": "<sender_curve25519_key>",
    "device_id": "<sender_device_id>",
    "session_id": "<outbound_group_session_id>",
    "ciphertext": "<encrypted_payload_base_64>"
  }
}
```

Abbildung 5.1: Event im JSON-Format, welches durch Megolm verschlüsselt wurde

Abbildung 5.1 zeigt den grundlegenden Aufbau eines Events, welches verschlüsselt übertragen wird. Der unverschlüsselte Aufbau vom *ciphertext* wird in Abbildung 5.2 gezeigt. Einige Events erweitern dieses Grundgerüst um weitere Objekte. So wird im Falle eines Nachrichtenevents der betreffende Raum auch im unverschlüsselten Part übertragen. Diese redudante Übertragung (vgl. Abbildung 5.2) schützt davor, dass der Homeserver die Möglichkeit erhält den Raum zu verändern [12, 13.11.4.3] und so Nachrichten in einen anderen Raum zu senden.

```
{
  "type": "<event_type>",
  "content": "<event_content>",
  "room_id": "<the room_id>"
}
```

Abbildung 5.2: Unverschlüsselte Sicht auf den *ciphertext* aus Abbildung 5.1

Abbildung 5.3 zeigt den Aufbau der versendeten Nachricht. Diese besteht aus einem Byte für die Versionsnummer, gefolgt von einer flexiblen Länge für den eigentlichen Payload, einer fixen Länge für den MAC und einer fixen Signatur-Länge.
Der Payload besteht aus zwei key-value Paaren: Der erste gibt den Message-Index an, also eine fortlaufende Nummer. Der zweite beinhaltet den verschlüsselten Text der Nachricht.
Der MAC schützt alle Bytes, die vor dem MAC stehen. Er ist auf acht Bytes begrenzt, d.h. bei einem längeren MAC werden nur die ersten acht Bytes genommen.
Die Signatur schützt ebenfalls alle vorstehenden Bytes. Dieser hat eine Begrenzung von 64 Bytes, was bei längeren Signaturen nur die ersten 64 Bytes berücksichtigen würde [22].

Version	Payload	MAC	Signature
1 Byte	n Byte	8 Byte	64 Byte

Abbildung 5.3: Aufbau einer Megolm-Nachricht

5.2.2 Anwendung

Die Megolm-Verschlüsselung (Schutzziel: Vertraulichkeit) findet Anwendung bei einer Kommunikation zwischen mehr als zwei Parteien. Da jeder Raum im Matrix-Netzwerk jederzeit um weitere Teilnehmer erweitert werden kann und es außerdem üblich ist als Benutzer mehrere Endgeräte für den Chat zu verwenden (unterschiedliche Browser, per App) ist Megolm die Standardverschlüsselung für alle Gespräche.

Um die Verschlüsselung in einem Raum zu aktivieren, muss ein *m.room.encryption*-Event in den Raum gesendet werden. Dieses Event enthält zusätzlich die Information, welcher Verschlüsselungs-Algorithmus angewendet werden soll. Aktuell ist nur der Megolm-Algorithmus vorhanden. Für Clients ist dies ab dem Event der Hinweis, dass Nachrichten absofort nur noch verschlüsselt in den Raum gesendet werden sollen. Sobald die Verschlüsselung für einen Raum einmal aktiviert wurde, bleibt diese in Riot für den Raum immer bestehen [2] [18].

Eine verschlüsselte Nachricht hat immer das *m.room.encrypted*-Events. Dieses enthält die Information, welche Verschlüsselung für diese Nachricht angewendet wurde. Das tatsächliche Event und der weitere Payload verbirgt sich hinter dem verschlüsselten *ciphertext*-Key (vgl. Abbildung 5.1 und 5.2).

Damit später die Nachricht von allen Beteiligten entschlüsselt werden kann, wird zu Beginn der Kommunikation eine Session zwischen zwei Geräten etabliert. Dafür gibt es sowohl eine Session-ID als auch einen geheimen Session Key. Jedes von einem Benutzer verwendete Gerät ist öffentlich einsehbar und besitzt eine eindeutige Geräte-ID. Zusätzlich zur ID ist einsehbar, um was für ein Gerät es sich handelt. Es ist jedem Benutzer möglich, andere Geräte durch einen Prozess zu verifizieren oder zu blockieren. Blockierte Geräte erhalten niemals einen Session Key und können folgerichtig auch keine Nachrichten des Senders entschlüsseln. Der Austausch des Session Keys erfolgt durch den Olm-Algorithmus, welcher für die Kommunikation zwischen genau zwei Agenten entwickelt wurde. Der Session Key wird durch das X25519-Verfahren etabliert (vgl. Kapitel "Curve25519 - Schlüsselpaar") [18].

[2] Das Matrix-Protokoll würde auch unverschlüsselte Nachrichten in diesem Raum erlauben. Riot verhindert dies allerdings und markiert Nachrichten signifikant, welche unverschlüsselt in den Raum übertagen wurden, nachdem die Verschlüsselung aktiviert wurde.

5.3 Einhaltung der weiteren Schutzziele

5.3.1 Authentizität

Um die Authentizität von Nachrichten zu gewährleisten, werden diese durch das Ed25519-Schlüsselpaar signiert. Der öffentliche Schlüssel wurde dafür mit allen anderen Teilnehmern der Konversation, über einen sicheren Kanal, geteilt. [22].

Besonderes Augenmerk sollte hierbei auf **Replay-Angriffe** gelegt werden. Dabei handelt es sich um einen Angriff, bei dem der Angreifer eine Nachricht (Paket) abfängt, und diese zu einem beliebigen Zeitpunkt erneut einstellt. Für den Empfänger sieht es daher so aus, dass der Absender die gleiche Nachricht erneut versendet hat, was aber nicht der Fall ist. Den Inhalt des (verschlüsselten) Paketes muss er dabei nicht genau kennen, kann sich aber sicher sein, dass dieses gültig (korrekte Signaturen) ist. Dies würde die Authentizität von Nachrichten verletzen.
Megolm verhindert diesen Angriff durch fortlaufende Nummern (Message Index), die innerhalb der verschlüsselten Nachricht übertragen werden. Erhält ein Client eine Nachricht mit einer Nummer, die er bereits bearbeitet hat, so sollte diese nicht erneut verarbeitet werden. Zu beachten ist, dass diese Prüfung im Client vorgenommen werden muss. Riot prüft dies und stellt im Falle doppelter Indizes die Verarbeitung der Nachricht ein und gibt eine entsprechende Meldung aus [18].

Ein weitere Angriff auf die Authentizität einer Nachricht ist die sogenannte **Unknown Key-Share Attack**. Dabei glaubt A, dass er einen Schlüssel mit B geteilt hat. Obwohl dies wirklich der Fall ist, glaubt B, dass dieser Schlüssel mit jemand anderem geteilt wurde, der nicht A ist [33]. Um diesen Angriff zu verhindern, wertet Riot beim Aufbau der Olm-Session Informationen über den Empfänger und Absender der Nachricht aus. Will Alice nun mit Eve eine Nachricht austauschen, so verschlüsselt Alice diese mit einem von Eves One-Time-Keys. Innerhalb der Nachricht sind Informationen über Absender und Empfänger enthalten, welche in Riot ausgewertet werden. Da Eve allerdings, in Vorbereitung auf den Angriff, nur Bobs One-Time-Keys angeboten hat, kann sie diese Nachricht zwar nicht entschlüsseln, aber an Bob weiterleiten. Bob erkennt beim Entschlüsseln der Nachricht, dass diese eigentlich von Alice kommen sollte, tatsächlich aber von Eve kommt. Würde diese Überprüfung fehlen, würde Bob denken, dass er einen Schlüsselaustausch mit Alice getätigt hat, Alice jedoch denkt, sie hätte diesen mit Eve durchgeführt [3].

Die Authentizität der Nachricht ist aufgrund der vorstehenden Überprüfungen sichergestellt. Somit wird die Sicherheitsanforderung S2 mit “kein Risiko” bewertet und ist damit erfüllt.

5.3.2 Integrität

Die Integrität der Nachricht, also das unbemerkte Verändern von Informationen, wird durch HMAC-SHA-256 (RFC2104) sichergestellt [48]. Dieser MAC sichert die Version und das gesamte Payloadformat (vgl. Abbildung 5.3) einer Nachricht.
Bei dem HMAC-Verfahren handelt es sich um eine bestimmte Form des MACs, welcher jedoch eine Schwachstelle, die beim MAC-Verfahren auftreten kann, unwirksam macht. Dabei besteht bei manchen Hash-Verfahren die Möglichkeit, aus einem bereits gehashten Wert (H’) und einem weiteren verketteten Wert (A) den gleichen Hash zu erhalten, wie aus dem eigentlichen Wert (H) und dem verketteten Wert (A). “HMAC-Verfahren verschleiern den internen Zustand der genutzten Hash-Funktion” [38, S. 378].
Die Abbildung 5.4 [38, S. 378 f.] zeigt die Unterschiede der beiden Verfahren. So wird beim MAC der gemeinsame, geheime Schlüssel $K_{A,B}$ zusammen mit der eigentlichen Nachricht verkettet, so das anschließend geprüft werden kann ob das Ergebnis mit dem erwarteten übereinstimmt. Beim HMAC wird zweimal eine Hashfunktion ausgeführt. Hier wird der geheime Schlüssel durch die Konstanten *ipad* und *opad* auf die Blocklänge der Hashfunktion aufgefüllt.

$$\text{mac} = \text{MAC}(M, K_{A,B})$$
$$\text{HMAC}(M,K) = H(K \oplus opad, H(K \oplus ipad, M))$$

Abbildung 5.4: Vergleich von HMAC und MAC

Die Integrität einer Nachricht, hinsichtlich der Anforderung S3, ist somit gewährleistet und die Anforderung ist damit erfüllt.

5.4 Bewertung

Zuvor wurde der Umgang mit Verschlüsselungen thematisiert. Um nun zu prüfen ob die eingesetzten Algorithmen und Schlüssellängen ausreichend sind wird zunächst auf Empfehlungen des BSIs zurückgegriffen. Die Ergebnisse fließen in den Punkt S4 “Auswahl geeigneter kryptografischer Verfahren” mit ein.

Die Verwendung von den **Curve25519**-Kurven wird nicht explizit durch die BSI-Dokumente erwähnt. Stattdessen wird auf Brainpool-Kurven verwiesen, welche im RFC 5639 spezifisiert wurden. Die Brainpool-Kurven werden durch das BSI favorisiert, da diese u.a. von Mitarbeitern des BSIs für den elektronischen Personalauswas entwickelt wurden. Um das Jahr 2014 wurde innerhalb der IETF darüber diskutiert, weitere elliptische Kurven zu standardisieren [35]. Als Ergebnis kam die Curve25519-Kurve heraus, welche im Jahr 2016 als RFC 7748 standardisiert wurde.

Auch das **ed25519-Verfahren** wird beim BSI nicht explizit erwähnt, allerdings enthält das Paper zum ed25519-Verfahren einen Vergleich zum RSA Verfahren. Hierbei wird angegeben, dass dieses ähnlich schwierig ist zu brechen, wie RSA mit einer Schlüssellänge von ca. 3000 bits [31]. Zu dem RSA Verfahren legt das BSI Empfehlungen fest. So sollte beim RSA Verfahren eine Schlüssellänge von 2000 Bit bis zum Jahre 2022 reichen. Ab 2023 sollte eine Schlüssellänge von mindestens 3000 Bit verwendet werden [6]. Demnach ist das ed25519-Verfahren zumindest für die nächsten Jahre eine sichere Wahl. Auf dieses Verfahren wird auch im RFC 7748 verwiesen [50].

Das **X25519**-Verfahren zum Schlüsselaustausch wird ebenfalls durch den RFC 7748 gestützt. Jeder bekannte Angriff ist teurer, als eine Brute-Force-Attacke auf eine typische 128bit Verschlüsselung [30], welche auch beim AES noch als sicher gilt [6].

Das, für die Ver- und Entschlüsselung von Nachrichten, verwendete Blockchiffre-Verfahren **AES-256** wird vom BSI auch hinsichtlich der Schlüssellänge von 256 bits empfohlen. Selbiges gilt für das **HMAC-SHA-256** Verfahren [6].

Verwendet werden nur standardisierte Verfahren bzw. vom BSI unterstützte Verschlüsselungsalgorithmen. Auch die jeweils verwendeten Schlüssellängen weisen keine Probleme auf, wodurch kein Risiko bei der Verwendung der kryptografischen Verfahren, hinsichtlich der Anforderungen S4a und S4b, ausgeht und beide Anforderungen damit erfüllt sind.

6 Schlüsselmanagement

6.1 Anwendung

Meldet sich ein neues Gerät an, so wird im Client ein Curve25519-Schlüsselpaar (Identity Key) und ein Ed25519-Fingerprint Key erzeugt. Dieser Vorgang wird nach jedem Login ausgeführt, also technisch ausgedrückt, wenn eine neue Session erzeugt wird. Der öffentliche Schlüssel wird jeweils durch die REST-API an den Homeserver übermittelt, der private Schlüssel bleibt auf dem Gerät. Zusätzlich wird zu jedem Gerät eine Liste von unterstützten Algorithmen übermittelt, um in der Zukunft dynamisch auch weitere Algorithmen zur Verschlüsselung unterstützen zu können und sich nicht auf bestimmte festzulegen.

Jeder Client übersendet seinem Homeserver außerdem eine bestimmte Menge an Curve25519-One-Time-Keys. Bei den übersendeten Schlüsseln handelt es sich nur um den öffentlichen Schlüssel. Der private Schlüssel der One-Time-Keys verbleibt auf dem jeweiligen Gerät. Es liegt in der Verantwortung vom Client genügend One-Time-Keys zur Verfügung zu stellen. Diese Keys werden später von anderen Nutzern verwendet, um eine Session mit dem Eigentümer der Keys aufzubauen (Schlüsselaustausch).

Will Alice also eine Session mit Bob aufbauen, benötigt sie zunächst eine Liste mit allen Geräten und deren öffentlichen Schlüssel, die Bob verwendet. Diese erhält sie über einen REST-Aufruf an ihren Homeserver. Anschließend kann sie mit allen Geräten, welche sie nicht blockiert hat, ein Shared-Secret aushandeln. Dafür benötigt sie den Identity-Key und einen One-Time-Key des Geräts. Sie selber muss ebenfalls einen One-Time-Key generieren. Aus diesen Daten kann sie dann mit Hilfe von Diffie-Hellman (X25519) ein Shared-Secret erstellen. Damit Bob das gleiche Secret berechnen kann, muss sie zunächst

noch an Bob ihren Identity-Key, ihren One-Time-Key und Bobs verwendeten One-Time-Key senden (da Bob mehrere hat, muss er wissen, welcher benutzt wurde). Bei allen Schlüsseln handelt es sich jeweils um den öffentlichen Schlüssel. Aus diesen Daten kann Bob den Schlüssel ebenfalls berechnen.

Der vereinbarte Schlüssel wird dann verwendet um Nachrichten per Megolm zwischen den Geräten austauschen zu können [18] [17] [23].

6.2 Schlüsselaufbewahrung

Riot speichert die wichtigsten Daten (dazu zählen z.B. die Sitzungsschlüssel, welche für eine verschlüsselte Kommunikation benötigt werden) standardmäßig in einer IndexedDB ab. Dabei handelt es sich um eine, in HTML 5 ergänzte, clientseitige Datenbank, welche direkt über den Browser per JavaScript adressierbar ist. Geschwindigkeitsvorteile ergeben sich durch das schnelle Heranziehen der benötigten (lokal gespeicherten) Daten, ohne eine externe Abfrage starten zu müssen. Außerdem bleiben dadurch sicherheitsrelevante Daten (z.B. Private Schlüssel) immer auf dem Gerät und müssen nicht über das Internet übertragen und verbreitet werden. Die Same Origin Policy (SOP) ermöglicht es, dass jede Domain nur auf die Datenbank zugreifen kann, mit der dieser assoziiert ist. Demnach können Daten, welche im WebClient von Riot (https://riot.im/app/) entstehen, nicht von einer anderen Webseite verwendet oder gelesen werden.

Die IndexedDB sieht dabei keine standardmäßige Verschlüsselung von Daten vor. In Riot hingegen werden Session-relevante Daten (wie bspw. der private Schlüssel) mit einem statischen Schlüssel verschlüsselt und anschließend base64-codiert abgespeichert. Ein ausreichender Schutz entsteht hierdurch jedoch nicht, da ein Angreifer diesen Schlüssel aus dem Quellcode herauslesen kann.

Eine Schwachstelle stellt das Cross Site Scripting (XSS) dar. Dabei handelt es sich um einen Angriff, bei dem der Angreifer durch fehlende Validierung der Benutzereingaben sämtlichen Code in das System einschleusen kann, welcher im Client dann ausgeführt wird. Dieser Angriff kann dazu verwendet werden gespeicherte Daten auszulesen (wie z.B. die IndexedDB) und an den Angreifer weiterzuleiten [46] [49] [4] [18].

6.3 Verifizierung von Geräten

Um den Gesprächspartner korrekt zu identifizieren, ist es wichtig seine Geräte zu verifizieren. Unter der Annahme, dass ein Benutzer dem Administrator des Homeservers nicht vertraut, kann diese Verifizierung nicht gänzlich in Matrix abgeschlossen werden. Riot erzeugt während des Verifizierungsprozesses für beide Endpunkte (Geräte) eine Zeichenkette (dargestellt durch Emojis). Den Vergleich der Zeichenkette sollte über einen anderen, vertrauenswürdigen, Kanal durchgeführt werden (z.B. durch ein persönliches Treffen). Auf die Verwendung eines sicheren Kanals weist Riot die Anwender im Dialog hin. Ist die Zeichenkette bei beiden Gesprächspartnern gleich, so kann ein Man-in-the-Middle (MitM) ausgeschlossen werden.

Abbildung 6.1: In Riot angezeigte Emoji-Kette. Die Verifizierung über die identische Kette bei beiden Gesprächsteilnehmern muss über einen sicheren Kanal erfolgen.

Der *Short Authentication String* (SAS), zur Erzeugung der Emojikette wie in Abbildung 6.1 dargestellt, ist angelehnt an den Key Agreement Handshake von Phil Zimmermann (ZRTP), welcher im RFC 6189 genauer definiert wird. Dieser zeichnet sich besonders durch einen *hash commitment* aus, wodurch der erste öffentliche Schlüssel erst gehasht übertragen wird, und erst nach Erhalt des öffentlichen Schlüssels des anderen im Klartext übertragen wird. Dadurch hat der Angreifer nur eine Möglichkeit den Austausch anzugreifen ohne entdeckt zu werden, wodurch der SAS Schlüssel relativ kurz sein kann. Schon bei einem 16-bit SAS hat der Angreifer von 655536 Möglichkeiten eine Chance nicht entdeckt zu werden [60] [17].

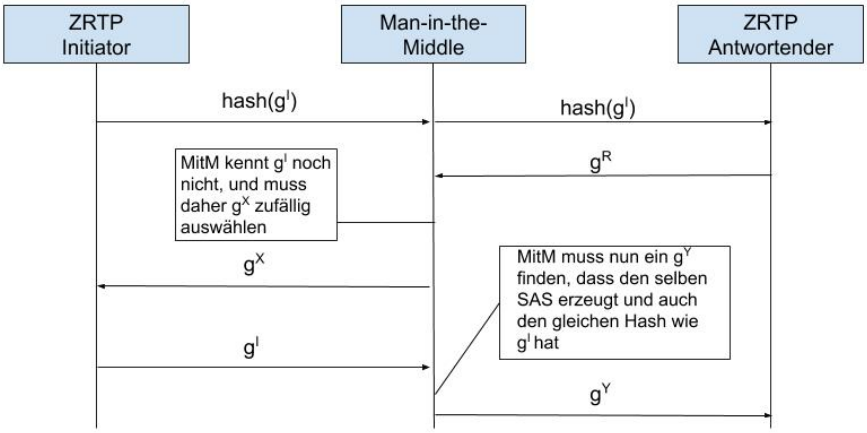

Abbildung 6.2: Problematik des MitM bei einem DH-Schlüsselaustausch mit einem hash commitment [40]

Um aus dem erhaltenen Hash eine Emoji-Kette zu erzeugen, bildet Riot aus den ersten 42 Bits insgesamt sieben Gruppen von jeweils 6 Bits. Jede Gruppe wird dann anhand der Anzahl der Bits einem Emoji zugeordnet (Insgesamt können daher $64_{(10)}$ verschiedene Emojis angezeigt werden, um von $000000_{(2)}$ bis $111111_{(2)}$ alles abzudecken) [17].

Die von Riot in diesem Prozess angebotene Funktion zum Hashen, verwenden den *sha*-Algorithmus mit 256 Bits. Dabei handelt es sich um eine Hashfunktion, welche vom BSI in dieser Stärke empfohlen wird [6].

6.4 Bewertung

Durch die Tatsache, dass die Schlüssel nur zu Beginn einer Session erzeugt werden, die Session jedoch kein Ablaufdatum hat, ist kein Mechanismus zum häufigen Wechsel der Schlüssel implementiert worden. Dies fließt in die Bewertung der Anforderung S5a hinein. Die Dokumentation zur Verschlüsselung von matrix.org weist auf dieses Problem hin: “Theoretisch sollten wir den Curve25519-Schlüssel von Zeit zu Zeit wechseln, allerdings haben wir dieses noch nicht implementiert” [18]. Es ist also davon auszugehen, dass dieser Punkt in einem der zukünfigten Releases mit aufgenommen wird.

Eine sichere Aufbewahrung der Schlüssel ist zur Zeit nicht gewährleistet. Die privaten

Schlüssel werden mit einem Standard-Schlüssel verschlüsselt und dann in einer unverschlüsselten lokalen Datenbank abgelegt. Solange keine XSS-Lücken existieren, ist diese Methodik über Riot nicht wirklich angreifbar. Allerdings kann niemand ein 100 Prozent sicheres System garantieren. Es gibt zudem zwei Überlegungen, welche den statischen Schlüssel ersetzen könnten:

Der Schlüssel wird vom Benutzer selber festgelegt. Dann muss der Benutzer diesen jedoch auch bei jeder Datenbank-Transaktion oder zumindest in jeder Browser-Session erneut angeben. Auch diese Angabe kann von einem Angreifer über eine XSS-Lücke abgefangen werden.

Der Schlüssel leitet sich aus Benutzerdaten wie bspw. dem Benutzernamen, Registrierungszeitpunkt o.ä. ab. Jedes deterministische Verfahren kann von einem Angreifer zurückberechnet werden. Hervorzuheben wäre hierbei jedoch, dass dann nicht alle Benutzer den gleichen Schlüssel hätten.

Eine Speicherung im Browser birgt immer die Gefahr eines XSS-Angriffes auf diese Daten. Eine wirklich benutzerfreundliche Speicherung ist daher nur schwer herbeizuführen. Die Anforderung S5b ist damit allerdings nicht erfüllt.

Die Anforderung S5c erwartet, dass Schlüssel nur für einen Einsatzzweck verwendet werden. Diese Anforderung wurde zum Teil in diesem Kapitel und im Kapitel zur Verschlüsselung behandelt. Es wurde deutlich gemacht, dass für sämtliche Einsatzzwecke ein anderer Schlüssel verwendet wird. Somit ist diese Anforderung erfüllt.

7 Homeserver

7.1 Datenspeicher

7.1.1 Systeme

Synapse unterstützt die Verwendung von SQLite und PostgreSQL, wobei in der Konfiguration standardmäßig SQLite konfiguriert wurde. Dies hat den Vorteil, dass man nach der Installation eines Servers diesen direkt verwenden kann, ohne sich um die Installation und die Konfiguration von PostgreSQL zu bemühen. Allerdings wird allen Administratoren von Synapse empfohlen, zeitnah auf PostegreSQL umzustellen, da SQLite wirklich nur für Demonstrationszwecke verwendet werden sollte. Der dafür argumentierte Hauptnachteil liegt in der Performance: Während PostgreSQL die Installation eines Datenbankservices benötigt, um Operationen für die Datenbank auszulagern, benötigt SQLite einen solchen Service nicht, sondern kann direkt aus dem Programm heraus ausgeführt werden - auf Kosten der Performance [44].

7.1.2 SQL-Injections

Webanwendungen, die Schwachstellen für SQL-Injections aufweisen, können einem Angreifer den Zugriff auf die gesamte Datenbank ermöglichen. Da in der Datenbanken meistens sehr sensitive Daten gespeichert werden, sind die Auswirkungen von z.B. eines Datendiebstahls verheerend.
Bei einer SQL-Injection wird eine Benutzereingabe in eine bestehende SQL-Query eingefügt, allerdings in dem Wege, dass die Eingabe ebenfalls als ausführbarer SQL-Code behandelt wird. Dies ermöglicht dem Angreifer SQL-Befehle direkt an die Datenbank zu richten. Diese Gefahr besteht also bei allen Webanwendungen, welche Benutzereingaben abfragen und an die Datenbank richten [41].

Dafür genügt schon ein einfacher Login auf der Seite. Unter Verwendung des Benutzernamens *'; drop table users-* - wird die gesamte Benutzertabelle gelöscht. Der ausgeführte SQL-Befehl würde dabei wie folgt aussehen: *SELECT * FROM users WHERE username = ''; drop table users- -' AND password = ''*. Insgesamt werden also zwei SQL-Befehle ausgeführt: Ein *Select* auf einen leeren Benutzernamen und ein *Drop table*. SQL verwendet - - als Kommentar, daher wird der hintere Teil der Abfrage komplett ignoriert [28].

Um SQL-Injections zu verhindern muss also jede Benutzereingabe validiert werden. Synapse greift dafür auf die Verwendung von Prepared Statements zurück. Dabei handelt es sich um eine Art von kompilierten Templates für SQL, welches Platzhalter für variable Parameter enthält. Prepared Statments liefern Schutz gegen SQL-Injections, da der Datenbanktreiber eine Maskierung der Eingaben (Platzhalter) vornimmt. Durch diese Maskeriung werden Zeichen, welche das SQL als Befehle interepretieren könnte (wie z.B. das Semikolon oder Anführungsstriche) unwirksam gemacht. Neben diesem gravierenden Sicherheitsvorteile bieten Prepared Statements auch Geschwindigkeitsvorteile, da eine Abfrage nur einmal in der Datenbank vorbereitet werden muss und anschließend mehrmals mit verschiedenen Parametern durchgeführt werden kann [24].
Von der Anforderung S6 der Anforderungstabelle geht somit keine Gefahr aus und es besteht kein Risiko, da sämtliche SQL-Abfragen innerhalb von Synapse als ein Prepared Statement behandelt werden und somit ausreichend geschützt sind.

7.1.3 Schutz sensibler Daten

Die Datenbank in Synapse enthält alle personenbezogene Daten der Benutzer, die diesen Homeserver verwenden. Besonders das Passwort der Benutzer sollte sicher gespeichert werden. Schlüssel für kryptografische Funktionen werden in diesem Abschnitt nicht behandelt, da die Datenbank nur öffentliche Schlüssel enthält, die keinen weiteren Schutz benötigen (sämtliche private Schlüssel verbleiben auf den Geräten).

Das BSI versteht unter der sicheren Datenablage von Passwörtern die Verwendung von sicheren kyptografischen Algorithmen unter der Verwendung von Salted Hashes. Salted Hashes bezeichnet eine Technik, welche das eigentliche Passwort um einen weiteren String ergänzt. Dies soll den Hash sicherer machen, da im Falle eines Datendiebstahls der Hashwert mit einer Liste bekannter Hashwerten verglichen werden kann und so Rückschlüsse auf das Passwort gezogen werden kann [5][APP.3.1.A14]. Verwendet ein Benutzer also

ein sehr einfacheres Passwort, so wird dieses zusammen mit dem String gehasht (salted) und in der Datenbank gespeichert. Im besten Fall sollte der Wert des Strings geheim sein und für jeden Benutzer neu generiert werden.

Synapse speichert die Benutzerpasswörter unter Verwendung von bcrypt. Bcrypt bezeichnet eine Hash-Funktionalität, die auf das Hashen von Passwörtern spezialisiert ist und besonders resistent gegen Brute-Force-Attacken ist [55]. Das Benutzerpasswort wird mit einem, von Bcrypt erzeugten salt, vermischt und es entsteht ein salted Hash. Zusätzlich hat der Administrator des Homeservers über die Konfigurationsdatei die Möglichkeit, einen sogenannten *pepper* festzulegen, welchen ebenfalls noch mit dem Passwort vermengt wird, bevor es gehasht wird. Dieser Pepper muss allerdings konfiguriert werden, bevor der erste Benutzer sich auf dem Server registriert und darf anschließend auch nichtmehr geändert werden. Synapse hat die Verwendung dieses Strings standardmäßig in der Konfiguration deaktiviert.
Durch die Verwendung von bcrypt, über welches aktuell noch keine Sicherheitsprobleme bekannt sind, und des salted Hashes erfüllt Synapse die Anforderungen an die Speicherung eines Passwortes und es geht somit kein Risiko vom Punkt S7a aus.

7.2 Schutz des Benutzerkontos

7.2.1 Sichere Passwörter

Die Ablage der Benutzerpasswörter wurde im vorherigen Kapitel bereits thematisiert. Um Brute-Force-Angriffe allerdings einzuschränken, sollte das Benutzerpasswort sicher gewählt sein. Was ein gutes Passwort auszeichnet, hat das BSI in einer Übersicht dargestellt. Daraus ergeben sich folgende Hinweise, für ein gutes Passwort [9]:

- Je länger das Passwort umso besser - mindestens acht Zeichen sollte ein gutes Passwort besitzen
- Die Verwendung aller verfügbarer Zeichen wie z.B.: Kleinbuchstaben, Großbuchstaben, Sonderzeichen, Ziffern
- Das Passwort sollte kein Name (bspw. eines Familienmitglieds) sein oder in einem Wörterbuch auffindbar
- Es sollte keinem Tastaturmuster wie "qwertz" gleichen

Um zu prüfen, ob ein Passwort geeignet ist, verwendet Riot das OpenSource Programm zxcvbn von Dropbox Inc.. Dieses wird über GitHub [15] zur Verfügung gestellt. Hauptfunktionalität hierbei ist die Überprüfung, wie gebräuchlich ein eingegebenes Passwort ist. Dafür vergleicht das Tool das Passwort mit Passwörtern aus geleakten Passwortlisten, häufigen Vor- und Nachnamen oder mit der Worthäufigkeit in Wikipedia-Artikeln (vgl. Abbildung 7.1). Außerdem werden auch mögliche Pattern abgefangen (wie z.B. *P@ssw0rd*, welches andere Tools als "starkes Passwort" einstufen würden), Tastaturkombinationen oder bekannte Leetspeak (3 als e, oder 0 als o). Das Tool liefert für jedes übergebene Passwort einen Score zwischen 0 (sehr schwach) und 4 (sehr stark), welcher von der erwarteten Zeit zum Knacken des Passwortes abhängig ist. Ein Nachteil besteht jedoch in den Daten die diesem Tool zur Verfügung stehen: Die bestehen hauptsächlich aus englischen Wörtern und Name [59] [58].
Zwar bietet das Tool eine Schnittstelle an, um die Listen um weitere Einträge zu erweitern, allerdings wird die Liste in Riot nur um die Begriffe *riot*, *matrix* und den Benutzernamen des jeweiligen Benutzers ergänzt. Die Prüfung gegen bekannte (geleakte) Passwörter und die Patternerkennung kompensieren jedoch die fehlende deutschen Begriffe. Ein wirklich schwaches Passwort ist daher nichtmehr möglich.
Riot erwartet von seinen Benutzern ein Passwort mit einem Score von mindestens drei

um den Registrierungsprozess abschließen zu können.

Abbildung 7.1: Meldung bei Eingabe eines bekannten Passworts

Durch diese Implementierung ist die Anforderung S7b erfüllt, da die Verwendung schwacher Passwörter von Riot abgewiesen wird.

7.2.2 Fehlgeschlagene Anmeldungen begrenzen

Passwortbasierte Authentifikationen sind im Internet sehr weit verbreitet. Durch den Verzicht auf zusätzliche Hardware ist es sehr einfach in Systemen einzubauen und wird von den Benutzern sehr gut angenommen. Allerdings hat diese sehr einfache Lösung auch sicherheitsbedenkliche Nachteile. So ist es einem Angreifer durch einen gezielten Angriff (*"targeted attack"*) auf einem Benutzernamen möglich, das Passwort zu erraten. Dabei können mehrere Techniken verfolgt werden, wie bspw. das Verwenden von Begriffen aus dem Wörterbuch als Passwort etc.. Der Benutzer kann durch die Verwendung eines sicheren Passwortes den Erfolg des Angriffes verringern. Allerdings kann auch seitens des Systems etwas dagegen getan werden, weil das Ergebnis jeder Brute-Force-Attacke eine große Anzahl an Login-Versuchen ist. Um einen Angriff einzuschränken, verweigern die meisten Systeme, nach einer bestimmten Anzahl an fehlgeschlagenen Anmeldeversuchen, temporär weitere Anmeldeversuche [27].
Auch der Administrator von Synapse kann die Anzahl der Login-Versuche durch Konfigurationseinstellungen begrenzen. Dabei kann er festlegen, wieviele fehlgeschlagene Anmeldeversuche innerhalb eines Zeitraums möglich sind. Weitere Anmeldeversuche werden dann zunächst abgewiesen. Der Administrator kann dabei für folgende drei Gruppen eigene Einstellungen festlegen:

- **IP-Adresse** - Begrenzungen pro IP-Adresse
- **Account** - Begrenzungen pro Benutzeraccount
- **Account generell** - Begrenzungen pro Benutzeraccount allgemein übergreifend auf Versuche von anderen

Durch die Limitierung fehlgeschlagener Anmeldungen durch definierbare Grenzwerte besteht beim Punkt S7c kein Risiko und die Anforderung ist erfüllt.

7.3 Sicheres Session-Management

7.3.1 Verfall nach Sitzungsbeendigung

Das BSI hat festgelegt, dass Sessions durch den Benutzer explizit beendet werden können [5][APP.3.1.A3]. Riot verwendet solche Sitzungsschlüssel für die Kommunikation zwischen dem Client und dem Server. Dieser Sitzungsschlüssel bleibt für die ganze Sitzung gleich und ist für den Benutzer über Riot einsehbar. Die zu überprüfende Erwartung ist also, dass der Sitzungsschlüssel durch das Abmelden des Benutzers (dies stellt die explizite Beendigung der Sitzung durch den Benutzer dar) als inaktiv markiert wird bzw. verworfen wird. Meldet sich der Benutzer mit diesem Benutzernamen wieder an, so ist die Erwartung, dass dieser einen neuen Sitzungsschlüssel erhält. Dieses Verhalten ist zwingend erforderlich, da ein Angreifer mit dem Sitzungsschlüssel ebenfalls alle Aktionen im Namen des Benutzers ausführen kann, daher muss dem Benutzer zwingend eine Möglichkeit eingeräumt werden, diesen Sitzungsschlüssel (durch das Beenden der Sitzung) zu vernichten.

Um zu überprüfen, ob dies nach den Vorgaben des BSIs implementiert wurde, wurde ein REST-Aufruf an den Homeserver des Nutzers, unter Verwendung des bekannten Sitzungsschlüssel gestartet. Da der REST-Aufruf zum Synchronisieren der Client-Daten (*/_matrix/client/r0/sync*) per HTTP-GET erfolgt und zusätzlich eine Client-Authorisierung erfordert, ist dieser zur Überprüfung der Funktionalität gut geeignet und kann in jedem Browser erfolgen. Der Sitzungsschlüssel wird dafür noch zusätzlich an den Pfad ergänzt, so dass der Aufruf an dem matrix.org-Homeserver unter Verwendung des Sitzungsschlüssel wie folgt aussieht: *https://matrix.org/_matrix/client/r0/sync?access_token=access_token* [12]. Wie erwartet liefert dieser Aufruf sämtliche Informationen an

den Client zurück und zeigt diese auch im Browser an, da der Sitzungsschlüssel einem Nutzer zugeordnet werden kann.
Nun meldet sich der Benutzer über die Oberfläche ab und beendet diese Sitzung. Danach wird der gleiche Pfad erneut aufgerufen. Da der Sitzungsschlüssel nun nichtmehr bekannt ist, liefert der Aufruf statt den Daten einen Fehler mit dem Hinweis, dass der Sitzungsschlüssel nicht existiert:

```
{"errcode":"M_UNKNOWN_TOKEN","error":"Unrecognised access token
    ","soft_logout":false}
```

Daraufhin hat sich der Benutzer erneut angemeldet und es wurde der selbe Aufruf erneut probiert. Auch dies führte zu dem obenstehenden Fehler. Daher wurde die Vorgabe vom BSI eingehalten, wodurch der Benutzer die Session explizit beenden kann. Ergänzend listet Riot eine Übersicht aller aktiven Sitzungen auf und zeigt diese dem Nutzer an. Daher ist es auch möglich andere Sitzungen als die eigene, aktuelle zu beenden. Dies ist sinnvoll wenn der Nutzer sich an einem öffentlichen Rechner vergessen hat abzumelden. Die Anforderung S8a ist somit erfüllt, da von dieser kein Risiko ausgeht.

7.3.2 Token-Handling

Für die Erstellung der Token werden Macaroons verwendet. Diese sind ähnlich wie Cookies, allerdings mit erweiterten Möglichkeiten besonders für dezentralisierte Systeme. In Synapse werden diese unter der Verwendung der Bibliothek PyMacaroons [16] benutzt. Macaroons wurden von Google-Mitarbeiter entwickelt und bieten im Vergleich zu Cookies kontextbezogene Einschränkungen. Dieser Macaroon setzt sich dann aus verschiedenen Werten zusammen. Zum einen aus einer *location*, welche angibt wo dieser Token generiert wurde, welche Berechtigungen dieser Token besitzt (Standardmäßig verwendet Synapse hier die Bezeichnung *access*), zu welchem Benutzer dieser Token gehört (*user_id*) und einer Nonce aus 16 zufälligen Zeichen, um zu verhindern, dass ein Benutzer bei verschiedenen Logins einen gleichen Token erhält. Unter Verwendung eines geheimen Keys (welcher nur dem Server bekannt sein darf), wird in dem Macaroon ein HMAC erzeugt. Diese Signatur ist ebenfalls Teil des Macaroons und stellt die Integrität des Macaroons sicher. Diese Werte werden dann serialisiert und können für den Austausch zwischen Client und Server verwendet werden. Erhält der Server einen Request, kann der den Access-Token unter Verwendung der Signatur und des geheimen Schlüssels verifizieren [32].

Die Verbindung zwischen Riot und seinem Homeserver läuft TLS verschlüsselt ab. Daher kann ein Angreifer den Verkehr nicht mitlesen, und es ist ihm daher auch nicht möglich, den Access-Token durch die Kommunikation zwischen dem Client und seinem Homeserver abzuhören. Wie die Schlüssel für die Ende-zu-Ende-Verschlüsselung wird auch dieser Access-Token im Speicher des Browsers (in diesem Fall im *local storage*) gespeichert. Die Gefahren, welche aus so einer Speicherung resultieren könnten, wurden im Kapitel 6.2 zur Schlüsselaufbewahrung erläutert.

Die Token werden somit zufällig, unter Berücksichtigung der 16-stelligen Nonce und einer ausreichender Länge, erzeugt. Der direkte Austausch zwischen Client und Server ist geschützt. Daher besteht bei den Punkten S8b und S8c kein Risiko und beide Anforderungen sind erfüllt. Das angesprochene Risiko ist identisch mit dem Punkt S5c und wird dort behandelt. Daher fließt die Risikobewertung aus der Speicherung nicht in S8b mit ein um redundante Risiken zu vermeiden.

8 Fazit

Die vorliegende Arbeit hat sich mit den sicherheitsrelevanten Eigenschaften des Matrix Protokolls beschäftigt. Die Entwickler des Protokolls haben dafür einen Client (Riot) und einen Server (Synapse) als Referenzimplementierung entwickelt. Diese waren Hauptbestandteil der Untersuchungen. Alle entwickelten Produkte sind Open Source. Für die Analyse zur Umsetzung der sicherheitsrelevanten Anforderungen wurde der Quelltext der jeweiligen Produkte herangezogen.
Um die sicherheitsrelevanten Anforderungen festzulegen, wurde zum größten Teil auf Checklisten des BSIs zurückgegriffen. Diese sind für verschiedenste Anwendungsbereiche vorhanden, wobei für diese Arbeit die Checklisten für Webserver und Kryptographie besonders bedeutsam waren und somit viele Anforderungen daraus gezogen werden konnten. Die daraus gezogenen Anforderungen wurden durch weitere Ergebnisse eines Brainstorming-Prozesses erweitert. Anschließend entstand eine Tabelle mit den wichtigsten Anforderungen an das Matrix-Protokoll (s. Tabelle 3.1), welche in dieser Arbeit geprüft und abgearbeitet wurden.
Diese Anforderungen wurden dann durch die einzelnen Kapitel nacheinander hinsichtlich der Umsetzung überprüft. Dies geschah meistens durch Quellcode Analysen unter Berücksichtigung aktueller Standards (z.B. Empfehlungen des BSI), aber im Bereich der Verkehrsflussanalyse auch durch die Verwendung von Wireshark.
Die Anforderungen aus S1 die sich aus der Verkehrsflussanalyse ergeben, zeigen einige Schwächen auf. So ist es, besonders bei längeren Nachrichten, unter Umständen möglich die beiden Gesprächspartner durch die Aufzeichnung des Netzwerkverkehrs, unter Berücksichtigung des Versand- und Empfangzeitpunktes, zu erahnen. Eine eindeutige Zuordnung anhand der Paketlänge, kann jedoch nicht getroffen werden, da die Paketlänge des Empfängers immer deutlich umfangreicher ist, als die des Absenders.
Ein weiteres großes Problem stellt die Speicherung der privaten Daten in einer unverschlüsselten, lokalen Datenbank im Browser dar (Anforderung S6c). Dadurch kann ein Angreifer, wenn eine dementsprechende Lücke existiert, Zugriff auf diese Daten (wie z.B. private Schlüssel) erhalten und somit auch verschlüsselte Nachrichten entschlüsseln.

Zusammengefasst weisen die Produkte ein hohes Sicherheitsniveau auf. Die sicherheitsrelevanten Implementierungen zeigen einen hohen Standard und die verwendeten kryptographischen Funktionen haben sich in der Form etabliert. Die Empfehlungen des BSIs sind kompatibel mit der eingesetzten Implementierung.

Für die Zukunft erwartet das Protokoll noch eine ständige Weiterentwicklung, gerade durch eine immer größer werdende Community. So entscheiden sich immer mehr Institutionen ihre interne Kommunikation auf das Matrix Protokoll umzustellen. Zwei prominente Beispiele sind Mozilla und die französische Regierung. Mozilla hat im Dezember 2019 bekannt gegeben von IRC, als Kommunikationsplattform für sämtliche Community-Angelegenheiten, auf Matrix umzusteigen und die IRC Server spätestens zum März 2020 herunterzufahren. Besonders für diese Entscheidung hob Mozilla die Barrierefreiheit in Riot und die Community-Sicherheit (durch das Melden unangebrachter Beiträge, die gegen die Mozilla Richtlinien verstoßen) hervor [51]. Die französische Regierung begann im Februar 2020 mit dem Test der Anwendung *Tchap*, welche auf Riot basiert und Nachrichten über das Matrix-Netzwerk austauscht [36]. Auf Matrix stieß die Regierung auf der Suche nach Alternativen zu WhatsApp und Telegram. Besonders wichtig - und das garantiert Riot und somit auch Tchap - war die Unterstützung von Ende-zu-Ende verschlüsselten Nachrichten [43].
Aber auch die Entwicklung der eigentlichen Produkte geht stetig weiter. So ist es für die Zukunft geplant standardmäßig jede Kommunikation Ende-zu-Ende zu verschlüsseln. Bevor dies jedoch standardmäßig aktiviert wird, soll zunächst eine Cross-signing Funktionalität zur Verfügung gestellt werden. Die sieht nach der Verifizierung eines Gerätes vor, andere Geräte (die dem verifiziertem Gerät vertraut), automatisch ebenfalls zu vertrauen. Somit wird in Zukunft der Verifizierungsprozess für die einzelnen Geräte eines anderen Nutzers deutlich einfacher fallen und nicht so zeitaufwendig für die Nutzer sein [42].
Die Entwicklung ist noch lange nicht am Ende und man darf gespannt sein, welche weiteren prominente Organisationen Mozilla und der französischen Regierung dem Wechsel ins Matrix-Netzwerk folgen werden.

Literaturverzeichnis

[1] Website. – URL: https://github.com/matrix-org/synapse; (abgerufen am 05.06.2020)

[2] Website. – URL: https://github.com/vector-im/riot-web; (abgerufen am 05.06.2020)

[3] Website. – URL: https://github.com/vector-im/riot-web/issues/2483; (abgerufen am 22.02.2020)

[4] Website. – URL: https://gitlab.matrix.org/matrix-org/olm/-/blob/master/README.md; (abgerufen am 05.03.2020)

[5] *APP.3.1 Webanwendungen*. Website. – URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKompendium/bausteine/APP/APP_3_1_Webanwendungen.html; (abgerufen am 17.02.2020)

[6] *BSI - Technische Richtlinien des BSI - BSI TR-02102-1 „Kryptographische Verfahren: Empfehlungen und Schlüssellängen“ Version: 2020-01*. Website. – URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf; (abgerufen am 16.02.2020)

[7] *BSI - Transport Layer Security (TLS)*. Website. – URL: https://www.bsi.bund.de/DE/Themen/StandardsKriterien/Mindeststandards_Bund/TLS-Protokoll/TLS-Protokoll_node.html; (abgerufen am 16.02.2020)

[8] *BSI für Bürger - Kommunikation über das Internet*. Website. – URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/DigitaleGesellschaft/KommunikationUeberInternet/kommunikationInternet_node.html; (abgerufen am 11.11.2019)

[9] *BSI für Bürger - Passwörter.* Website. – URL: https://www.bsi-fuer-buerger.de/BSIFB/DE/Empfehlungen/Passwoerter/passwoerter_node.html; (abgerufen am 16.03.2020)

[10] *[Cfrg] 25519 naming.* Website. – URL: https://mailarchive.ietf.org/arch/msg/cfrg/-9LEdnzVrE5RORux3Oo_oDDRksU (abgerufen am 24.01.2020)

[11] *Checklisten zum IT-Grundschutz-Kompendium der Edition 2019.* Website. – URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Grundschutz/Kompendium/checklisten_2019.html; (abgerufen am 17.02.2020)

[12] *Client-Server API.* Website. – URL: https://matrix.org/docs/spec/client_server/latest; (abgerufen am 23.01.2020)

[13] *CON.1 Kryptokonzept.* Website. – URL: https://www.bsi.bund.de/DE/Themen/ITGrundschutz/ITGrundschutzKompendium/bausteine/CON/CON_1_Kryptokonzept.html; (abgerufen am 20.02.2020)

[14] *CVE-2019-19781 - Vulnerability in Citrix Application Delivery Controller, Citrix Gateway, and Citrix SD-WAN WANOP appliance.* Website. – URL: https://support.citrix.com/article/CTX267027; (abgerufen am 08.02.2020)

[15] *dropbox/zxcvbn: Low-Budget Password Strength Estimation.* Website. – URL: https://github.com/dropbox/zxcvbn; (abgerufen am 16.03.2020)

[16] *ecordell/pymacaroons: A Python Macaroon Library.* Website. – URL: https://github.com/ecordell/pymacaroons; (abgerufen am 03.05.2020)

[17] *End-to-End Encryption.* Website. – URL: https://github.com/matrix-org/matrix-doc/blob/6911171e83fbf322e42565f1f6a217ea24d73924/specification/modules/end_to_end_encryption.rst; (abgerufen am 05.03.2020)

[18] *End-to-End Encryption implementation guide.* Website. – URL: https://matrix.org/docs/guides/end-to-end-encryption-implementation-guide/ (abgerufen am 16.01.2020)

[19] *FAQ | Matrix.org.* Website. – URL: https://matrix.org/faq; (abgerufen am 01.11.2019)

[20] *Matrix Specification*. Website. – URL: https://matrix.org/docs/spec/#id2; (abgerufen am 18.11.2019)

[21] *The Matrix.org Foundation*. Website. – URL: https://matrix.org/foundation/; (abgerufen am 22.01.2020)

[22] *Megolm group ratchet*. Website. – URL: https://gitlab.matrix.org/matrix-org/olm/blob/master/docs/megolm.md; (abgerufen am 24.01.2020)

[23] *Olm: A Cryptographic Ratchet*. Website. – URL: https://gitlab.matrix.org/matrix-org/olm/blob/master/docs/olm.md; (abgerufen am 06.03.2020)

[24] *Prepared Statements und Stored Procedures*. Website. – URL: https://www.php.net/manual/de/pdo.prepared-statements.php; (abgerufen am 14.03.2020)

[25] *XEP-0045: Multi-User Chat*. Website. – URL: https://xmpp.org/extensions/xep-0045.html; (abgerufen am 03.05.2020)

[26] *XMPP Specifications*. Website. – URL: https://xmpp.org/extensions/; (abgerufen am 03.05.2020)

[27] Adams, Carlisle ; Jourdan, Guy-Vincent ; Levac, Jean-Pierre ; Prevost, Francois: Lightweight protection against brute force login attacks on Web applications, 09 2010, S. 181 – 188

[28] Anley, Chris: *Advanced SQL Injection In SQL Server Applications*. Website. 2002. – URL: https://www.cgisecurity.com/lib/advanced_sql_injection.pdf (abgerufen am 12.03.2020)

[29] Benjamin: *XMPP: Admin-in-the-middle*. Website. – URL: https://infosec-handbook.eu/blog/xmpp-aitm/; (abgerufen am 16.02.2020)

[30] Bernstein, Daniel J.: *Curve25519: new Diffie-Hellmann speed records*. Website. 2006. – URL: https://cr.yp.to/ecdh/curve25519-20060209.pdf (abgerufen am 16.01.2020)

[31] Bernstein, Daniel J. ; Duif, Niels ; Lange, Tanja ; Schwabe, Peter ; Yang, Bo-Yin: *High-speed high-security signatures*. Website. 2011. – URL: http://ed25519.cr.yp.to/ed25519-20110926.pdf (abgerufen am 16.01.2020)

[32] Birgisson, Arnar ; Politz, Joe ; Erlingsson, Úlfar ; Taly, Ankur ; Vrable, Michael ; Lentczner, Mark: Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud, 01 2014. – ISBN 1-891562-35-5

[33] Blake-Wilson, Simon ; Menezes, Alfred: *Unknown Key-Share Attacks on the Station-to-Station (STS) Protocol.* Website. 1999. – URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.123.1798&rep=rep1&type=pdf; (abgerufen am 22.02.2020)

[34] Bless, Roland ; Mink, Stefan ; Blass, Erik-Oliver ; Conrad, Michael ; Hof, Hans-Joachum ; Kutzner, Kendy ; Schöller, Marcus: *Sichere Netzwerkkomunikation.* Heidelberg : Springer-Verlag Berlin Heidelberg, 2005

[35] Böck, Hanno: *Die Suche nach neuen Kurven.* Website. Dezember 2014. – URL: https://www.golem.de/news/bernstein-gegen-microsoft-die-suche-nach-neuen-kurven-1412-110935.html; (abgerufen am 28.02.2020)

[36] Dussutour, Chloé: *French government launches in-house developed messaging service, Tchap.* Website. – URL: https://joinup.ec.europa.eu/collection/open-source-observatory-osor/document/french-government-launches-house-developed-messaging-service-tchap; (abgerufen am 30.03.2020)

[37] Dutton, Sam: *WebRTC in the real world: STUN, TURN and signaling.* Website. 2013. – URL: https://www.html5rocks.com/en/tutorials/webrtc/infrastructure/ (abgerufen am 12.01.2020)

[38] Eckert, Claudia: *IT-Sicherheit.* 10. München : De Gruyter Oldenbourg, 2018

[39] Faber, Eberhard von ; Behnsen, Wolfgang: *Joint Security Management: organisationsübergreifend handeln.* Wiesbaden : Springer Vieweg, 2018

[40] Floroiu, John ; Sisalem, Dorgham: A comparative analysis of the security aspects of the multimedia key exchange protocols, 01 2009, S. 2

[41] Halfond, William ; Viegas, Jeremy ; Orso, Alessandro: A Classification of SQL Injection Attacks and Countermeasures. (2006), 01

[42] Hodgson, Matthew: *The 2019 Matrix Holiday Update!* Website. – URL: https://matrix.org/blog/2019/12/24/the-2019-matrix-holiday-update; (abgerufen am 30.03.2020)

[43] Hodgson, Matthew: *Matrix and Riot confirmed as the basis for France's Secure Instant Messenger app.* Website. – URL: https://matrix.org/blog/2018/04/26/matrix-and-riot-confirmed-as-the-basis-for-frances-secure-instant-messenger-app; (abgerufen am 30.03.2020)

[44] Johnson, Neil: *Synapse 1.7.0 released.* Website. 12 2019. – URL: https://matrix.org/blog/2019/12/13/synapse-1-7-0-released (abgerufen am 12.03.2020)

[45] Kalt, C.: Internet Relay Chat: Architecture / RFC Editor. RFC Editor, 04 2000 (2810). – RFC. – URL https://www.rfc-editor.org/rfc/rfc2810.txt. – ISSN 2070-1721

[46] Kimak, Stefan ; Ellman, Jeremy ; Laing, Christopher: Some Potential Issues with the Security of HTML5 IndexedDB, 10 2014

[47] Klipper, Sebastian: *Information Security Risk Management.* Wiesbaden : Springer Vieweg, 2011

[48] Krawczyk, H. ; IBM ; Bellare, M. ; UCSD ; Canetti, R. ; IBM: HMAC: Keyed-Hashing for Message Authentication / RFC Editor. RFC Editor, 02 1997 (6189). – RFC. – URL https://www.rfc-editor.org/rfc/rfc2104.txt. – ISSN 2070-1721

[49] Laing, Christopher ; Kimak, Stefan ; Ellman, Jeremy: An Investigation into Possible Attacks on HTML5 IndexedDB and their Prevention, 09 2012

[50] Langley, A. ; Google ; Hamburg, M. ; Research, Rambus C. ; Turner, S. ; sn3rd: Elliptic Curves for Security) / RFC Editor. RFC Editor, 01 2016 (7748). – RFC. – URL https://www.rfc-editor.org/rfc/rfc7748.txt. – ISSN 2070-1721

[51] mhoye: *Synchronous Messaging at Mozilla: The Decision.* Website. – URL: https://discourse.mozilla.org/t/synchronous-messaging-at-mozilla-the-decision/50620; (abgerufen am 30.03.2020)

[52] Muth, Max: *Citrix-Sicherheitslücke: Tausende Firmen betroffen.* Website. – URL: https://www.sueddeutsche.de/digital/citrix-sicherheitsluecke-1.4755894; (abgerufen am 08.02.2020)

[53] Niederberger, Marlen ; Renn, Ortwin: *Das Gruppendelphi-Verfahren.* Wiesbaden : Springer VS, 2018

[54] Petrlic, Ronald ; Sorge, Christoph: *Datenschutz - Einführung in technischen Datenschutz,Datenschutzrecht und angewandte Kryptographie*. Wiesbaden : Springer Vieweg, 2017

[55] Provos, Niels ; Mazieres, David: *A Future-Adaptable Password Scheme*. 1999

[56] Rosenberg, J. ; Cisco ; Mahy, P. ; Matthews, P. ; Wing, D.: Session Traversal Utilities for NAT (STUN) / RFC Editor. RFC Editor, 10 2008 (5389). – RFC. – URL https://www.rfc-editor.org/rfc/rfc5389.txt. – ISSN 2070-1721

[57] Saint-Andre, P. ; Cisco: Extensible Messaging and Presence Protocol (XMPP): Core / RFC Editor. RFC Editor, 03 2011 (6120). – RFC. – URL https://www.rfc-editor.org/rfc/rfc6120.txt. – ISSN 2070-1721

[58] Wheeler, Daniel L.: Website. 2012. – URL: https://dropbox.tech/security/zxcvbn-realistic-password-strength-estimation; (abgerufen am 16.03.2020)

[59] Wheeler, Daniel L.: zxcvbn: Low-Budget Password Strength Estimation. In: *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX : USENIX Association, August 2016, S. 157–173. – URL https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/wheeler. – ISBN 978-1-931971-32-4

[60] Zimmermann, P. ; Project, Zfone ; Johnston, A. ; Avaya ; Callas, J. ; Inc., Apple: ZRTP: Media Path Key Agreement for Unicast Secure RTP / RFC Editor. RFC Editor, 04 2011 (6189). – RFC. – URL https://www.rfc-editor.org/rfc/rfc6189.txt. – ISSN 2070-1721

A Anhang

Glossar

Event Alle Nachrichten die über Matrix ausgetauscht werden, werden als Event bezeichnet (wie z.B. eine normale Textnachricht).

Homeserver Server, mit dem der Client im Matrix-Protokoll kommuniziert. Jedem Benutzeraccount ist ein Homeserver zugeteilt, welchen der Benutzer bei der Registrierung frei wählen kann..

MAC Der Message Authentication Code (MAC) schützt vor dem Verändern von Informationen innerhalb einer Nachricht (Integrität).

Megolm Verschlüsselungsalgorithmus innerhalb Riots, der den Austausch von verschlüsselten Nachrichten ermöglicht (1:N).

Olm Verschlüsselungsalgorithmus innerhalb Riots, der den Aufbau einer Megolm-Session ermöglicht (Schlüsselaustausch) (1:1).

STUN-Protokoll Session Traversal Utilities for NAT (STUN) ist ein Protokoll, das Tools für andere Protokolle im Umgang mit NATs anbietet.

Synapse Von Matrix implementierte Version eines Homeservers: https://github.com/matrix-org/synapse.

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Sicherheitsanalyse des Instant-Messengers Riot unter Verwendung des Matrix-Protokolls

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort Datum Unterschrift im Original