Windthorst-Gymnasium Meppen

Facharbeit 2024

zum Thema:

Dezentralisierte asymmetrische Verschlüsselung über Tor – Die Lösung für sicheres Messaging?

Seminarfach:	Informatik - SF25_2
Verfasser/in:	Hendrik Lind
Fachlehrer/in:	Marco Geertsema
Abgabetermin:	22.02.2024

Inhaltsverzeichnis

1 Einleitung

Russland, China, Iran. In all diesen totalitären Staaten herrscht eine starke Zensur [vgl. Am23]. Rund 1,7 Milliarden Menschen sind allein in diesen drei Staaten von der Einschränkung der Meinungsfreiheit betroffen. Wie können Bürger dieser Staaten ihre Meinung verbreiten und andere Staaten auf staatskritische Probleme aufmerksam machen, ohne sich selbst in Gefahr zu bringen?

Bei herkömmlichen Messengern wie WhatsApp, Signal und Co. benötigt die Außenwelt die Telefonnummern von Bürgern und Journalisten in einem totalitären Staat, um mit ihnen in Kontakt treten zu können. Eine mögliche Gefahr besteht darin, dass sich ein totalitärer Staat als legitimer Empfänger ausgibt, wodurch Bürger und Reporter ihre privaten Telefonnummern an den Staat weitergeben, der dann in der Lage ist, diese Nummern zurückzuverfolgen [vgl. Fä23]. Genau hier liegt das Problem: Bürger und Reporter können nicht über gewöhnliche Messenger mit der Außenwelt kommunizieren, da der Staat ihre Nummern zurückverfolgen kann, was die Meinungsfreiheit weiter einschränkt und verhindert [vgl. ebd.].

Die zentralisierte Infrastruktur, die von den meisten Messengern wie WhatsApp und Signal genutzt wird, ermöglicht es auch totalitären Staaten wie China, die IP-Adressen von Servern zu blockieren und sie so für Bürger und Journalisten unzugänglich zu machen [vgl. Wu+23; Bh23]. Ein dezentraler, Ende-zu-Ende-verschlüsselter Messenger, der über das Tor-Netzwerk kommuniziert, könnte eine Lösung für diese Probleme darstellen. Ziel dieser Arbeit ist es, diese Lösung auf ihre Sicherheit und Praktikabilität hin zu untersuchen und zu implementieren.

Um ein Verständnis für die Sicherheit dieses Messengers zu erlangen, geht diese Arbeit im zweiten Kapitel auf die asymmetrische Verschlüsselung und die Ende-zu-Ende-Verschlüsselung (E2EE) ein und beschäftigt sich zunächst mit der mathematischen Betrachtung, um anschließend die Sicherheit des Verfahrens beurteilen zu können. Dabei wird nicht auf das Padding-Verfahren der asymmetrischen Verschlüsselung eingegangen. Im dritten Kapitel wird untersucht, ob das Tor-Netzwerk eine mögliche Lösung für das Problem der Anonymität im Internet darstellt und inwieweit es die Nutzer schützt. Das vierte Kapitel beschäftigt sich mit der Definition und der Sicherheit von Dezentralisierung. Im fünften Kapitel wird schließlich die Implementierung des Messengers beschrieben, während im sechsten Kapitel die

Nachteile des Messengers und deren Lösungsmöglichkeiten betrachtet werden. Im siebten Kapitel wird ein Fazit gezogen.

1.1 Motivation

Mein Interesse an Themenkomplex erstreckt sich über mehrere Ebenen. Neben der asymmetrischen Verschlüsselung und der dahinterliegenden Mathematik, ist die Programmierung selbst und die Implementierung eines sicheren Messengers im Tor-Netzwerk eine meiner Hauptmotivationen mich mit diesem komplexen Thema zu beschäftigen.

1.1.1 Tor-Netzwerk

Ich verfolge das Tor-Netzwerk und die Struktur schon seit ein paar Jahren mit großem Interesse. Das Tor-Netzwerk wird oft mit dem Begriff des Darknets in Verbindung gebracht, was bei den meisten Menschen zu negative Assoziationen führt [vgl. 24a]. Vor alldem die schwierige Rückverfolgung spielt eine große Rolle in dem Darknet [vgl. ebd.]. Sie ist der Grund für mein Interesse, die Struktur zu verstehen und darauf einen neuartigen Messenger zu programmieren.

1.1.2 Asymmetrische Verschlüsselung

Bei der asymmetrischen Verschlüsselung steht für mich ein mathematischer Erklärungsansatz mitunter im Vordergrund meines Interesses: „Wie schafft es der Angreifer, eine verschlüsselte Nachricht nur mit dem öffentlichen Schlüssel nicht zu entschlüsseln?“.

1.1.3 Programmatischer Aspekt

Aber auch zu der Programmierung habe ich ein hohes Interesse. Die vielen Facetten des Tor-Netzwerkes mit der asymmetrischen Verschlüsselung so zu verknüpfen, dass am Ende ein funktionierender Messenger entsteht. Die Dezentralisierung des Messengers tragen nochmals zu der Schwierigkeit bei, sodass fundierte Kenntnisse der Informatik erforderlich sind.

1.2 Relevanz

Durch die steigende Anzahl von Cyberangriffen geraten Messenger immer mehr in den Fokus von Angreifern, da durch die Kompromittierung eines Messagingdienstes viele persönliche Daten erlangt werden können [vgl. 23a]. Da China eine ausgefeilte

Analyse und Überwachung des Netzes betreibt, ist die eine Anonymität und Sicherheit von Messengern umso wichtiger [vgl. Ho16]. Durch einen solchen Messenger könnten oppositionelle Meinungen und Ansichten das autoritär geführte Land verlassen und in der Außenwelt Gehör finden. Und auch hier stellt sich wieder die Frage: „Schützt ein solcher Messenger Bürger und Journalisten in einem totalitären Staat vor möglichen Konsequenzen?“.

1.3 Methodisches Vorgehen

Zunächst werde ich die Grundlagen der asymmetrischen Verschlüsselung, des Tor-Netzwerks und der Dezentralisierung erläutern, um dann die Sicherheit des Verfahrens zu überprüfen. Danach werde ich die Verfahren in den Messenger implementieren und die Nachteile und mögliche Lösungen betrachten. Zum Schluss werde ich ein Fazit ziehen.

2 Asymmetrische Verschlüsselung

Um einen sicheren Nachrichtenaustausch zu gewährleisten, wird in dieser Arbeit E2EE implementiert. Bei E2EE wird die Nachricht vom Sender verschlüsselt, bevor sie an den Empfänger gesendet wird [vgl. Gr14]. Zwischengeschaltete Akteure wie Server oder mögliche Angreifer können die Nachricht somit nicht lesen [vgl. ebd.]. **Nur** der Empfänger bzw. Sender der Nachricht kann sie entschlüsseln [vgl. LB21]. Als Ver- und Entschlüsselungsverfahren der Nachrichten wird (unter anderem) die asymmetrische Verschlüsselung verwendet [vgl. ebd.]. Diese Arbeit beschränkt sich bei der asymmetrischen Verschlüsselung auf das RSA-Verfahren.

2.1 Grundlagen

Grundsätzlich wird bei der asymmetrischen Verschlüsselung ein Schlüsselpaar (Keypair) verwendet, das aus einem privaten Schlüssel (private key) und einem öffentlichen Schlüssel (public key) besteht [vgl. BSW15a]. Diese beiden Schlüssel sind mathematisch so miteinander verknüpft, dass der öffentliche Schlüssel Nachrichten **nur** verschlüsseln, aber nicht [vgl. ebd.]. entschlüsseln kann. **Nur** der zum Schlüsselpaar gehörende private Schlüssel kann die verschlüsselte Nachricht wieder entschlüsseln (siehe Abb. 1) [vgl. ebd.].

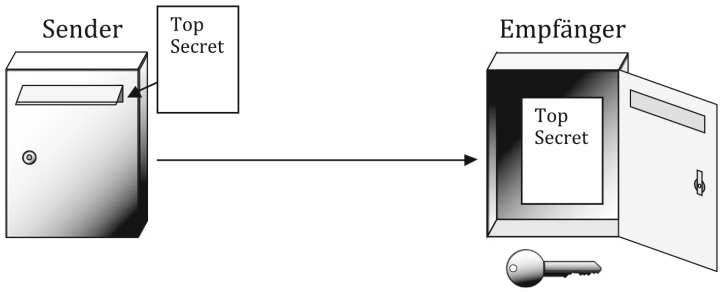

Abbildung 1: Jeder Sender kann mit dem öffentlichen Schlüssel die Nachricht „verschlüsseln“ (also eine Nachricht in den Briefkasten werfen), aber nur der Empfänger kann den Briefkasten mit seinem privaten Schlüssel öffnen und somit die Nachricht herausnehmen[vgl. BSW15b]

2.2 Mathematische Betrachtung

Alle Variablen der folgenden Berechnungen liegen im Bereich $\mathbb{N}$ [vgl. Is16].

Zur Erzeugung des Schlüsselpaares benötigen wir zunächst zwei große zufällige Primzahlen P und Q [vgl. ebd.]. Daraus ergibt sich $n = P * Q$, wobei $P \neq Q$ ist, so dass P bzw. Q nicht durch $\sqrt{n}$ bestimmt werden können [vgl. ebd.]. Der private Schlüssel besteht aus den Komponenten $\{n, d\}$, während der öffentliche Schlüssel aus $\{n, e\}$ besteht [vgl. Wa+13].

2.2.1 Eulersche Phi-Funktion

Die Eulersche Phi-Funktion spielt eine wichtige Rolle im RSA-Verfahren [vgl. Tu08]. Grundsätzlich gibt $\phi(x)$ an, wie viele positive teilerfremde Zahlen es bis x gibt (für wie viele Zahlen der größte gemeinsame Teiler (gcd) 1 ist). [vgl. Ta13]. So ergibt sich $\phi(6) = 2$ oder im Falle einer Primzahl $\phi(7) = 7 - 1 = 6$, also $\phi(x) = x - 1$, wenn x eine Primzahl ist, da jede Zahl kleiner als x teilerfremd sein muss [vgl. Tu08].

$$\phi(n) = \phi(P \cdot Q)$$
$$\phi(n) = \phi(P) \cdot \phi(Q)$$
$$\phi(P) = P - 1 \qquad \phi(Q) = Q - 1$$
$$\phi(n) = (P - 1) \cdot (Q - 1)$$

2.2.2 Generierung des Schlüsselpaares

Sowohl der private als auch der öffentliche Schlüssel bestehen unter anderem aus folgenden Komponenten: $n = P \cdot Q$ [vgl. Ta96]. Für den öffentlichen Schlüssel

benötigen wir die Komponente e, die zur Verschlüsselung einer Nachricht verwendet wird [vgl. ebd.]. e ist eine Zufallszahl, für die folgende Bedingungen gelten [vgl. ebd.]:

$$e = \begin{cases} 1 < e < \phi(n) \\ \text{gcd}(e, \phi(n)) = 1 \\ e \text{ kein Teiler von } \phi(n) \end{cases}$$

Mit der berechneten Komponente e, welche die Nachrichten verschlüsselt, kann nun der öffentliche Schlüssel an den Absender übermittelt werden. Zur Berechnung des privaten Schlüssels wird die Komponente d benötigt, die zur Entschlüsselung verwendet wird [vgl. Ta96].

$$\phi(n) = (P-1)(Q-1)$$
$$d = e^{-1} \mod \phi(n)$$

2.3 Sicherheit

Um die Sicherheit des RSA-Verfahrens betrachten zu können, müssen wir nun den Ver-/Entschlüsselungsprozess betrachten.

$c = m^e \mod n$	Verschlüsselung zu c mit m als Nachricht
$m = c^d \mod n$	Umkehroperation (Entschlüsslung) von c zu m

Um die verschlüsselte Nachricht c entschlüsseln zu können, bräuchte ein Angreifer die Komponente des privaten Schlüssels d. d ist jedoch sehr rechenintensiv, da, wie bereits gezeigt, $\phi(n)$ berechnet werden müsste. Man benötigt also eine Primfaktorzerlegung von n [vgl. Ta13]. Die Verschlüsselung von m nach c ist eine Trapdoor-Einwegfunktion [vgl. Kr16a]. Das bedeutet, dass es zwar einfach ist, $f(x) = i$ zu berechnen (im Falle von RSA: Verschlüsselung), es aber unmöglich ist, von i auf den ursprünglichen Wert x zu schließen, ohne eine weitere notwendige Komponente zu kennen (im Falle von RSA wäre die notwendige Komponente d) [vgl. ebd.]. Wichtig beim RSA-Verfahren ist, dass die Länge von n (die Schlüssellänge) mindestens 3000 Bit beträgt, da sonst die Primfaktorzerlegung von n mit modernen Rechnern möglich sein könnte [vgl. Si23].

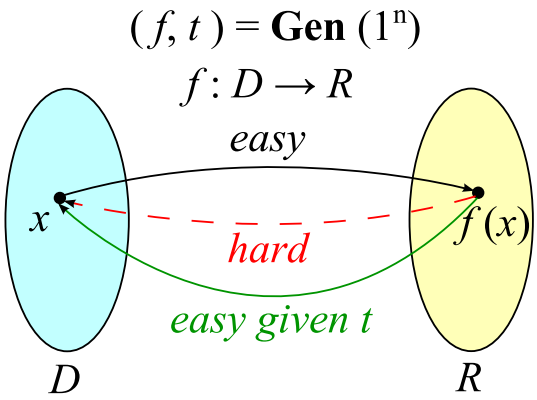

Abbildung 2: Die Trapdoor-Einwegfunktion bildlich dargestellt[vgl. Kr16b]

2.4 Vergleich zur symmetrischen Verschlüsselung

Bei der symmetrischen Verschlüsselung wird derselbe Schlüssel zum Ver- und Entschlüsseln verwendet [vgl. IBM21]. Im Vergleich zur asymmetrischen Verschlüsselung ist die symmetrische Verschlüsselung schneller und hat keine Beschränkung der Chiffretextlänge [vgl. HK20; Op24]. Allerdings muss der Schlüssel der symmetrischen Verschlüsselung dem anderen Kommunikationspartner sicher übermittelt werden, um die Nachrichten entschlüsseln zu können [vgl. ebd.].

3 Das Tor-Netzwerk

Die Anonymität des Messengers ist ein weiterer zentraler Aspekt, um die jeweiligen Kommunikationspartner zu schützen. Das Tor-Netzwerk ist hier eine mögliche Lösung, da bei normalem Routing, wie wir es täglich nutzen, der Client direkt mit dem Zielserver kommuniziert, der Zielserver also die IP-Adresse des Clients sehen kann [vgl. El16]. Eine Rückverfolgung der IP-Adresse ist somit möglich [vgl. Fä23]. Genau an diesem Punkt setzt das Tor-Netzwerk an. Das Tor-Netzwerk besteht dabei aus vielen *Nodes* , die eingehende Tor-Verbindungen annehmen und weiterverarbeiten [vgl. K 23]. Damit ein Client eine Anfrage über das Netzwerk senden kann, sucht er sich zunächst einen Weg durch das Netzwerk, einen sogenannten *Circuit* [vgl. To24d]. Der *Circuit* besteht in der Regel aus drei *Nodes* und ist 10 Minuten gültig, bis der Client den *Circuit* erneuert (also einen neuen Pfad „sucht“) [vgl. To24c]. Die drei *Nodes* werden in eine *Entry Node* , eine *Relay Node* und eine *Exit Node* gegliedert, über die später Anfragen an die Zielserver gesendet werden können [vgl. ebd.]. Dabei verschlüsselt der Client die eigentliche Anfrage mehrfach,

verpackt sie also in mehrere „Schalen“, die eine *Onion* bilden, und leitet diese über den *Circuit* an den Zielserver weiter [vgl. DMS04]. Zunächst wird die *Onion* vom Tor-Client an die *Entry Node* gesendet [vgl. ebd.]. An jeder *Node* , wie der *Entry Node* , wird eine Schale der *Onion* „abgeschält“ (die *Onion* also einmal entschlüsselt), die Informationen über die nächste Node enthält [vgl. Au18]. Die Node leitet nun die um eine Schale „geschälte“ *Onion* an den nächsten Knotenpunkt weiter [vgl. ebd.]. Sobald die Anfrage den *Exit Node* erreicht hat, entfernt dieser die letzte „Schale“ von der Anfrage, die nun vollständig entschlüsselt ist und an den Zielserver gesendet werden kann [vgl. ebd.]. Dieses Routing-Verfahren ist auch als *Onion-Routing* bekannt [vgl. DMS04].

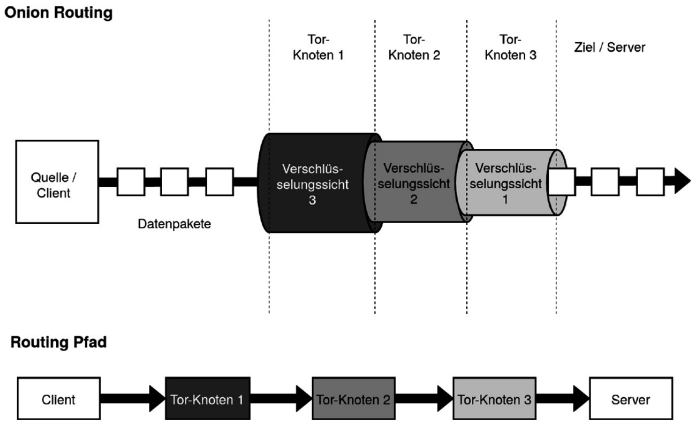

Abbildung 3: Das Prinzip des Onion-Routings [vgl. Wi+22a]

Nur die *Entry Node* kennt also die echte IP-Adresse des Clients und **Nur** die *Exit Node* weiß, an welchen Zielserver die Anfrage geschickt wurde [vgl. Au18]. Zwischengeschaltete *Nodes* wie die *Relay Node* erkennen nur eine verschlüsselte Anfrage und haben keine Informationen über den eigentlichen Client oder den Zielserver. Der *Exit Node* hingegen sendet die Anfrage ohne Verschlüsselung des Tor-Netzwerks an den Zielserver, wodurch Informationen über den Benutzer ausgelesen werden können (wenn die Anfrage nicht über das HTTPS-Protokoll gesendet wurde) und zusätzlich die IP-Adresse des Zielservers gespeichert werden kann [vgl. Ch+11]. Durch die Einrichtung von *Exit Nodes* im Tor-Netzwerk könnte ein Angreifer somit die Anonymität des Netzwerks bzw. der Benutzer gefährden [vgl. ebd.].

3.1 Onion Services

Onion Services sind eine mögliche Lösung für den *Exit Node* -Angriff [vgl. To24g]. Sie sind nicht auf *Exit Node* angewiesen und agieren nur innerhalb des Tor-Netzwerks mit anderen Tor-Clients und verhalten sich wie normale Tor-Clients [vgl. ebd.]. Onion Services sind nicht wie die Zielserver im vorherigen Beispiel über das normale Internet erreichbar, sondern nur über das Tor-Netzwerk [vgl. To24f]. Im Gegensatz zu normalen öffentlichen Servern müssen bei Onion Services keine Ports geöffnet werden, damit sich ein Client mit dem Server verbinden kann, da der Onion Service direkt mit dem Tor-Netzwerk über ausgehende Verbindungen kommuniziert (auch bekannt als *NAT-Punching*) und alle Daten darüber geleitet werden [vgl. ebd.].

3.1.1 Verbindungsaufbau

Zunächst erzeugt der Onion Service ein Schlüsselpaar, bestehend aus einem öffentlichen und einem privaten Schlüssel [vgl. Ge23b]. Aus dem öffentlichen Schlüssel wird nun unter anderem die Adresse des Onion Service generiert, die mit „.onion“ endet [vgl. ebd.]. Ein Beispiel für eine solche Adresse ist die der Suchmaschine DuckDuckGo:
duckduckgogg42xjoc72x3sjasowoarfbgcmvfimaftt6twagswzczad.onion [vgl. Du24].
Der Onion Service verbindet sich nun wie ein normaler Client mit dem Tor-Netzwerk über einen *Circuit* , der aus drei *Nodes* besteht [vgl. To24f]. Der Dienst sendet eine Anfrage an das letzte Relay im *Circuit* , sodass dieses als *Introduction Point* fungiert und eine Langzeitverbindung zu diesem aufbaut (der *Circuit* erneuert sich also nicht alle 10 Minuten wie beim Tor-Client) [vgl. ebd.]. Dieser Vorgang wird zweimal wiederholt, bis der Onion Service drei *Introduction Points* auf drei verschiedenen *Circuits* gefunden hat [vgl. ebd.]. Damit andere Clients den Onion Service erreichen können, erstellt der Onion Service einen *Onion Service descriptor*, der die Adressen der *Introduction Points* und die Authentifizierungsschlüssel enthält, signiert diesen mit seinem privaten Schlüssel und schickt den *Descriptor* an die Directory Authority [vgl. To24a; To24e]. Die Directory Authority ist ein Server, der Informationen über das Tor-Netzwerk speichert und verteilt, wie z.B. die des Onion Services [vgl. To24b]. Übertragen auf die Arbeit hat sich nun der Onion Service von Person A mit dem Tor-Netzwerk verbunden, aber es braucht noch Person B, die sich über ihren Tor-Client mit dem Onion Service von Person B verbindet. Damit sich der Tor-

Client von Person A verbinden kann, fordert der Client nun die Directory Authority auf, den signierten *Descriptor* des Onion Services von Person B an den Client zu senden [vgl. K 23]. Der Client besitzt nun die Adressen der *Introduction Points* und die Signatur des *Descriptor*, sodass er die Signatur mit dem öffentlichen Schlüssel der Onion-Adresse überprüfen kann [vgl. ebd.]. Der Client generiert nun 20 zufällige Bytes (Secret) und sendet diese an eine zufällig ausgewählte *Relay Node* , die nun als *Rendezvous Point* dient [vgl. To23b]. Der Client sendet das Secret auch an einen der *Introduction Points* , damit der Onion Service mit dem gleichen Secret eine Verbindung zum *Rendezvous Point* aufbauen kann [vgl. To23a]. Wenn der Client und der Onion Service über ihre *Circuits* verbunden sind, leitet der *Rendezvous Point* die Nachrichten zwischen den beiden weiter [vgl. To23b]. Der Client und der Onion Service kommunizieren nun nur noch über das Tor-Netzwerk miteinander und sind nicht mehr auf die *Exit Node* angewiesen.

3.2 Sicherheit

In der Sicherheitsbetrachtung beziehe ich mich ausschließlich auf die Verbindung zwischen den Onion Services und den Tor-Clients (also nicht zwischen Tor-Client und *Exit Node*). Das Tor-Netzwerk in Verbindung mit den Onion Services bietet, wie bereits erläutert, durch *Circuits* und *Rendezvous Point* ein hohes Maß an Anonymität. Im Gegenzug führt die hohe Anonymität, die durch die vielen *Nodes* und die mehrfache Verschlüsselung erreicht wird, auch dazu, dass das Tor-Netzwerk im Vergleich zum normalen Routing um das 120-fache langsamer ist [vgl. LSS10]. Das Tor-Netzwerk ist in der Praxis kaum zu blockieren, da es über 6.500 verschiedene *Nodes* gibtund somit kein zentraler Hauptserver existiert, von dem das Tor-Netzwerk abhängig ist[1] [vgl. Wi+22b]. Theoretisch ist eine Deanonymisierung eines Onion Services möglich, wenn der ISP (Internet Service Provider, also z.B. Telekom, EWE, etc.) eingehende und ausgehende Verbindungen zwischen Client und Onion Service bzw. zu deren *Relay Node* aufzeichnet und mithilfe von Machine Learning analysiert [vgl. Lo+24]. Die Umsetzung der theoretischen Grundlagen war bisher unter idealisierten Bedingungen erfolgreich, jedoch sind weitere Untersuchungen notwendig, um die potenziellen Gefahren dieses Algorithmus einordnen zu können

[1] Es ist zwar möglich die IP-Adressen jeglicher *Relay Nodes* zu blockieren, jedoch ist dies durch Pluggable Transports umgänglich [vgl. Pa+16]

[vgl. ebd.]. Bei einer Zusammenarbeit der Regierungen Deutschlands, der USA und Frankreichs könnten 78,34 % der *Circuits* deanonymisiert werden [vgl. ebd.]. Dieses Szenario erscheint jedoch unrealistisch, da die Bürger in Deutschland, den USA und Frankreich das Recht auf freie Meinungsäußerung haben und daher höchstwahrscheinlich keine Deanonymisierungsangriffe starten werden, um die Privatsphäre z.B. von Whistleblowern zu schützen [vgl. Am23; Re24].

4 Dezentralisierung

Mit den vorgeschlagenen Konzepten ist der Messenger imstande anonym (durch das Tor-Netzwerk und Onion Services) und sicher (durch asymmetrische Verschlüsselung) zu kommunizieren, jedoch wurde die Infrastruktur des Messengers noch nicht behandelt. Dafür ist die Betrachtung und Abwägung eines zentrales bzw. dezentrales Netzwerkes notwendig.

Ein zentrales Netzwerk, zum Beispiel das Netzwerk von WhatsApp oder Signal, besteht aus einem Hauptserver, welcher Daten verarbeitet und mit welchem sich Clients verbinden können [vgl. La22; Si16; Ge21]. Der Client muss bei dieser Netzwerkstruktur dem Hauptserver „vertrauen“ und kann nicht überprüfen, ob der Server von einem Hacker kompromittiert wurde [vgl. Se19]. Dezentrale Netzwerke bestehen hingegen aus mehreren Servern, auf welche Rechenoperationen und oder Daten aufgeteilt werden [vgl. Li23].

4.1 Sicherheit

Durch die Abhängigkeit des zentralen Netzwerks von einem Hauptserver kann ein Angreifer gezielter, z.B. durch DDoS-Angriffe oder Kompromittierung, das gesamte Netzwerk außer Betrieb setzen oder Nutzerdaten (z.B. Telefonnummern bei Messengern) auslesen [vgl. Bu24].

Ein dezentrales Netzwerk mit einer Peer-to-Peer-Architektur[2] bietet dagegen eine größere Angriffsfläche, sodass mehrere Knoten im Netzwerk kompromittiert oder abgeschaltet werden müssen, um das Netzwerk für deren Benutzer unzugänglich zu machen [vgl. Ge23a]. Es ist jedoch wichtig, dass die Knoten im Netzwerk vertrauenswürdig sind (im Beispiel des Messengers, dass nur die Gesprächsteilnehmer Teil

[2] Peer-to-Peer bedeutet, dass jeder Knoten im Netzwerk sowohl Client als auch Server sein kann, die bidirektional kommunizieren [vgl. HD05]

des Netzwerks sind), um zu verhindern, dass ein Angreifer die Onion-Adressen der Gesprächsteilnehmer durch einfaches Beitreten zum Netzwerk auslesen kann.

5 Umsetzung des Messengers

Folgende Kriterien müssen erfüllt sein, um den Messenger sicher, anonym und einfach zu gestalten:

Einfache Handhabung und Installation - Für die Installation und Nutzung des Messengers sollten möglichst wenig technische Kenntnisse nötig sein

Peer-to-Peer-Verbindung - Die Kommunikation zwischen Messengern sollte nur über eine direkte Verbindung erfolgen

Anonymität - Verbindungen zwischen Messengern nur das Tor-Netzwerk, keine Anfragen über das normale Internet

Sicherheit - Verschlüsselte Speicherung von Nachrichten und Schlüsselpaaren, um Auslesen der Daten (z.B. durch totalitäre Staaten) zu verhindern

E2EE - Verschlüsselung der Nachrichten und Verifikation des Kommunikationspartners

Damit der Benutzer keine Konsole verwenden muss, um den Messenger zu benutzen, habe ich eine Desktopanwendung mit Rust und Tauri entwickelt. Tauri erfordert hierbei keine zusätzlichen Interpreter oder Bibliotheken, was die Installation vereinfacht. Außerdem kann die Anwendung für Windows, Linux und macOS kompiliert werden [vgl. Ru24]. Rust bietet eine hohe Performance und ist Memory Safe, wodurch viele Sicherheitslücken, wie Buffer Overflows, verhindert werden können [vgl. Ma23]. Das Tauri Framework bietet hier zwischen dem User Interface (UI), welches ich in Typescript und React programmiert habe, und dem Rust-Backend eine Schnittstelle, um Daten (*payloads*) zwischen Frontend und Backend zu versenden [vgl. 23b]. Ich werde in diesem Kapitel nicht auf die Implementierung des Frontends eingehen, da dieser nur für die Benutzeroberfläche zuständig ist und keine weiteren Funktionalitäten bietet.

5.1 Peer-to-Peer-Verbindung

Damit Messenger sowohl als Client als auch als Server fungieren kann, startet jeder Messenger einen HTTP-Server, der einen HTTP-Endpunkt anbietet (in diesem Messenger */ws/*), mit welcher Clients eine Websocket[3]-Verbindung aufbauen können. Der Server ist somit in der Lage, sowohl Packets zu empfangen als auch zu senden.

Analog dazu hat jeder Messenger auch einen Websocket-Client, der sich direkt über das Tor-Netzwerk mit den anderen Messengern verbinden kann. Der Verbindungsaufbau zu anderen Messengern sieht wie folgt aus: Damit zwei Benutzer über den Messenger Nachrichten versenden können, muss ein Messenger die Rolle des Clients und der andere die Rolle des Servers übernehmen. Dadurch, dass jeder Messenger sowohl Client als auch Server sein kann, müssen empfangene Nachrichten von Client und Server zusammengeführt und an das Frontend übermittelt werden, welches in der Bibliothek *messaging* implementiert ist: Der *MessagingManager* speichert sowohl Client-Server-(C2S) als auch Server-Client(S2C)-Verbindungen in einem allgemeinen Konstrukt, der *Connection*. Diese *Connection*s werden in einer *HashMap* gespeichert. Der *MessagingManager* stellt dem Frontend eine Schnittstelle zum Senden und Empfangen von Nachrichten zur Verfügung. Wenn in die-

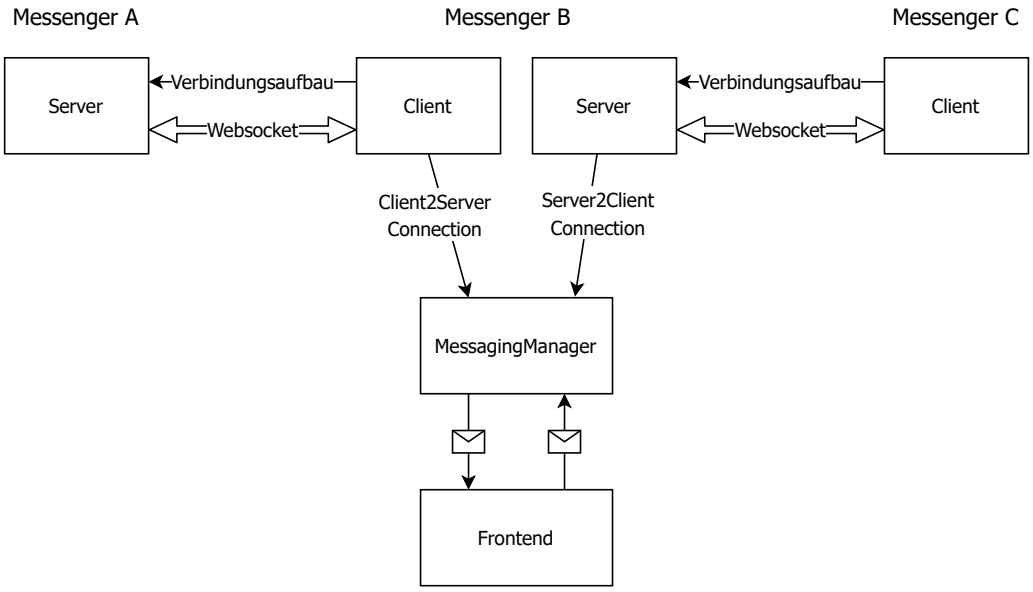

Abbildung 4: Mögliches Beispiel für Messenger B, welcher eine Verbindung zu Messenger A aufgebaut hat, und Messenger C, welcher mit Messenger B verbunden ist

[3] Ein Protokoll für bidirektionale Kommunikation zwischen Server und Client über HTTP [vgl. Mi23]

sem Beispiel das Frontend eine Nachricht an Messenger A senden möchte, wird diese Nachricht zunächst an den *MessagingManager* gesendet, der die Nachricht verschlüsselt an den entsprechenden Client weiterleitet. Der Client sendet nun die Nachricht über den Websocket an den Server von Messengers A.

5.2 Tor-Netzwerk

Eine mögliche Lösung für das Kriterium der Anonymität ist das Tor-Netzwerk. Dazu muss eine Verbindung zum Tor-Netzwerk hergestellt werden, was über die *tor-proxy* Bibliothek geschieht. Auch der HTTP-Server wird über den Tor-Proxy in das Tor-Netzwerk als Onion Service eingebunden. Damit sich der Tor-Proxy mit dem Tor-Netzwerk verbinden kann, wird eine Konfigurationsdatei benötigt, welche durch die *tor-proxy* Bibliothek erstellt wird: Eingehende Verbindungen zum Onion Service dieses Messengers werden nun auf den Port des HTTP-Servers umgeleitet. Nun muss nur noch der Tor-Proxy gestartet werden:

Der Messenger ist nun mit dem Tor-Netzwerk verbunden und der HTTP-Server ist über den Onion Service erreichbar. Andere Messenger können sich nun anonym über das Tor-Netzwerk mit diesem Messenger verbinden.

5.3 Sicherheit

Ein symmetrisch verschlüsselter Datenspeicher wird durch die Bibliothek *secure-storage* implementiert. Mit dieser Bibliothek können beliebige *Structs* in Text umgewandelt und verschlüsseln bzw. ausgelesen und entschlüsselt werden. Die *storage-internal* Bibliothek verwendet hierbei die Methoden der *secure-storage* Bibliothek und dient als Wrapper, um die Daten auf der Festplatte zu speichern und wieder auszulesen. Der Datenspeicher enthält hierbei eine *HashMap*, die als Schlüssel die Onion Adresse anderer Messengers und als Wert die Chatinformationen enthält. Chatinformationen sind zum Beispiel gesendete oder empfangene Nachrichten, Schlüsselpaare und öffentliche Schlüssel. Der *StorageChat* enthält die an den Empfänger gesendeten die Nachrichten oder von ihm empfangenen Nachrichten (*messages*), den öffentlichen Schlüssel des Kommunikationspartners (*rec_pub_key*) und den privaten Schlüssel des Messengers *priv_key*. Um die Nutzerdaten vor weiteren möglichen Angriffen zu schützen, verwende ich die Bibliothek *zeroize*, die sensible Daten (wie private Schlüssel) aus dem Arbeitsspeicher löscht (Zeroization) [vgl.

24b]. Zudem ist es durch die symmetrische Verschlüsselung des Datenspeichers für Angreifer nicht möglich (sofern die Applikation geschlossen ist), private Schlüssel oder Chatverläufe auszulesen, wodurch die Privatsphäre der Nutzer geschützt wird.

5.4 Ende-zu-Ende-Verschlüsselung

Das letzte Kriterium, das der Messenger nun erfüllen muss, ist die Ende-zu-Ende-Verschlüsselung. Dazu muss zunächst die Identität des anderen Kommunikationspartners überprüft werden.

5.4.1 Identity verification

Um das Konzept der E2EE umzusetzen, muss der Sender einer Nachricht sicher sein, dass der Empfänger tatsächlich derjenige ist, für den er sich ausgibt. Zum Zwecke der Erklärung gehen wir nun davon aus, dass Messenger A eine Nachricht an Messenger B senden möchte. Messenger A wäre in diesem Fall der Client und Messenger B der Server. Messenger A und Messenger B erzeugen zunächst (falls noch nicht vorhanden) private Schlüssel und erstellen ein neues *StorageChat*-Konstrukt, in welchem die privaten Schlüssel gespeichert werden. Damit sowohl Client als auch Server sicher sein können, dass der Kommunikationspartner tatsächlich der ist, für den er sich ausgibt, wird vom Client zunächst ein *Identity* -Packet versendet. Dieses besteht aus dem eigenen Hostnamen, der Signatur des Hostnames und dem Zielhostname (durch den privaten Schlüssel des Chats signiert) und dem öffentlichen Schlüssel des Schlüsselpaares (vom Chat). Das *Identity* -Packet wird also wie folgt generiert: Ein wichtiges Detail spielt hierbei die Signatur des *Identity* -Packets. Sie besteht sowohl aus dem eigenen Hostname als auch dem Ziel des Hostnames, um zu verhindern, dass ein Angreifer das *Identity* -Packet abfängt und sich damit bei anderen Messengern als dieser Messenger ausgibt. Nachdem der Server das Packet empfangen hat, überprüft er es. Dazu ruft er den öffentlichen Schlüssel des Kommunikationspartners ab, und überprüft damit die Signatur des *Identity* -Packets: Ist die Identität gültig, sendet der Server seine Identität an den Client. Der gleiche Validierungsprozess der Validation findet nun auch auf dem Client statt. Wenn der Client die *Identity* als gültig betrachtet, ist eine sichere Verbindung zwischen Client und Server hergestellt.

5.5 Nachrichten versenden

Gehen wir davon aus, dass der Client an den Server über die sichere Verbindung eine Nachricht versenden möchte. Dazu ruft er zunächst den öffentlichen Schlüssel des Empfängers aus dem Datenspeicher ab und verschlüsselt die Nachricht mit dem RSA-Verfahren. Anschließend sendet er die verschlüsselte Nachricht an den Server. Der Server empfängt die verschlüsselte Nachricht vom Client über die Websocket-Verbindung und entschlüsselt sie anschließend mit dem privaten Schlüssel des Chats. Damit ist die Nachricht nun empfangen und entschlüsselt, sodass sie an das Frontend übermittelt werden kann. Benutzer können nun sicher und anonym miteinander kommunizieren.

6 Nachteile und Lösungsmöglichkeiten

Der Messenger bietet zwar hohe Anonymität und Sicherheit, jedoch ist es unerlässlich auch die Nachteile des Messengers und deren Lösungsmöglichkeiten zu betrachten. Durch die Struktur des Tor-Netzwerkes können Datenmengen um das 120-fache langsamer übermittelt werden, welches bei einfachen Textnachrichten nicht auffallen mag, bei größeren Datenmengen, wie zum Beispiel Bildern, jedoch ein Problem darstellen könnte [vgl. LSS10]. Dadurch, dass der Client eine Websocket-Verbindung mit dem Onion Service aufbaut, herrscht hier ein *long-term-circuit*, wodurch der *Circuit* zwischen Onion Service und Client nicht erneuert wird, somit für Angriffe anfälliger sein könnte [vgl. To24c]. Zusätzlich ist die Nachrichtenlänge durch das Padding der asymmetrischen Verschlüsselung begrenzt [vgl. Op24]. Eine hybride Verschlüsselung, also eine Kombination aus asymmetrischer und symmetrischer Verschlüsselung, ist eine mögliche Lösung für die limitierte Nachrichtenlänge [vgl. KW11]. Dabei generiert bereits bei dem Verbindungsaufbau der Server einen zufälligen symmetrischen Schlüssel [vgl. ebd.]. Dieser wird mit dem öffentlichen Schlüssel des Clients verschlüsselt und an den Client übermittelt, welcher diesen entschlüsselt und abspeichert, sodass sowohl Server als auch Client den gleichen symmetrischen Schlüssel teilen [vgl. ebd.]. Somit werden die Vorteile der symmetrischen Verschlüsselung (schnell und keine Nachrichtenlängenbegrenzung) mit den Vorteilen der asymmetrischen Verschlüsselung (sicherer Austausch von Schlüsseln) kombiniert [vgl. 22]. Aufgrund der Infrastruktur dieses Messengers, müssen beide

Kommunikationspartner den Messenger geöffnet haben, damit diese miteinander kommunizieren können (der Onion Service wird nur durch Öffnen des Messengers gestartet).

7 Fazit

Literatur

[Am23] Amnesty International. *Amnesty International Report 2022/23*. London WC1X 0DW, United Kingdom: International Amnesty Ltd, 2023, S. 307–312, 122–128, 196–201. ISBN: 978-0-86210-502-0. URL: https://www.amnesty.org/en/wp-content/uploads/2023/04/WEBPOL1056702023ENGLISH-2.pdf (besucht am 13. 01. 2024).

[Au18] Autumn. „How does Tor ∗really∗ work?“ In: (Feb. 2018). URL: https://hackernoon.com/how-does-tor-really-work-c3242844e11f (besucht am 23. 01. 2024).

[BSW15a] Albrecht Beutelspacher, Jörg Schwenk und Klaus-Dieter Wolfenstetter. „Kryptologische Grundlagen“. In: *Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 14. ISBN: 978-3-8348-2322-9. DOI: 10.1007/978-3-8348-2322-9_2. URL: https://doi.org/10.1007/978-3-8348-2322-9_2.

[Bh23] Jasdeep Bhatia. *Understanding System Design Whatsapp & Architecture*. Mai 2023. URL: https://pwskills.com/blog/system-design-whatsapp/#Client-Server_Architecture (besucht am 22. 01. 2024).

[23a] *BKA - Listenseite für Pressemitteilungen 2023 - Veröffentlichung Bundeslagebild: über 130.000 Fälle von Cybercrime in 2022*. Aug. 2023. URL: https://www.bka.de/DE/Presse/Listenseite_Pressemitteilungen/2023/Presse2023/230816_PM_BLB_Cybercrime.html (besucht am 19. 02. 2024).

[Bu24] Bundesamt für Sicherheit in der Informationstechnik. *DoS- und DDoS-Attacken*. Feb. 2024. URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Cyber-Sicherheitslage/Methoden-der-Cyber-Kriminalitaet/DoS-Denial-of-Service/dos-denial-of-service_node.html (besucht am 08. 02. 2024).

[Ch+11] Sambuddho Chakravarty, Georgios Portokalidis, Michalis Polychronakis und Angelos D. Keromytis. „Detecting Traffic Snooping in Tor Using Decoys“. In: *Recent Advances in Intrusion Detection*. Hrsg. von Robin Sommer, Davide Balzarotti und Gregor Maier. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, S. 222–241. ISBN: 978-3-642-23644-0.

[24a] *Darknet und Deep Web – wir bringen Licht ins Dunkle*. Feb. 2024. URL: https://www.bsi.bund.de/DE/Themen/Verbraucherinnen-und-Verbraucher/Informationen-und-Empfehlungen/Darknet-und-Deep-Web/darknet-und-deep-web_node.html (besucht am 19. 02. 2024).

[DMS04] Roger Dingledine, Nick Mathewson und Paul Syverson. „Tor: The Second-Generation Onion Router“. In: *13th USENIX Security Symposium (USENIX Security 04)*. San Diego, CA: USENIX Association, Aug. 2004. URL: https://www.usenix.org/conference/13th-usenix-security-symposium/tor-second-generation-onion-router.

[Du24] DuckDuckGo. *duckduckgo onion at DuckDuckGo*. Feb. 2024. URL: https://duckduckgo.com/?q=duckduckgo+onion&atb=v160-7&ia=web (besucht am 02. 02. 2024).

[El16] Elektronik Kompendium. *TCP/IP*. Nov. 2016. URL: https://www.elektronik-kompendium.de/sites/net/0606251.htm (besucht am 23. 01. 2024).

[23b] *Events | Tauri Apps*. März 2023. URL: https://tauri.app/v1/guides/features/events (besucht am 15. 02. 2024).

[Fä23] Jan Fährmann. „Rechtliche Rahmenbedingungen der Nutzung von Positionsdaten durch die Polizei und deren mögliche Umsetzung in die Praxis–zwischen Strafverfolgung und Hilfe zur Wiedererlangung des Diebesguts“. In: *Private Positionsdaten und polizeiliche Aufklärung von Diebstählen*. Nomos Verlagsgesellschaft mbH & Co. KG. 2023, S. 142. ISBN: 978-3-8487-5905-7.

[Ge23a] GeeksforGeeks. *Comparison Centralized Decentralized and Distributed Systems*. Sep. 2023. URL: https://www.geeksforgeeks.org/comparison-centralized-decentralized-and-distributed-systems (besucht am 09. 02. 2024).

[Ge23b] GeeksforGeeks. *What are onion services in Tor Browser*. Okt. 2023. URL: https://www.geeksforgeeks.org/what-are-onion-services-in-tor-browser (besucht am 04. 02. 2024).

[Ge21] Gemini. *Networks: Decentralized, Distributed, & Centralized | Gemini*. Juli 2021. URL: https://www.gemini.com/cryptopedia/blockchain-network-decentralized-distributed-centralized#section-what-is-a-centralized-network (besucht am 08. 02. 2024).

[Gr14] Andy Greenberg. „Hacker Lexicon: What Is End-to-End Encryption?“ In: *WIRED* (Nov. 2014). URL: https://www.wired.com/2014/11/hacker-lexicon-end-to-end-encryption (besucht am 16. 01. 2024).

[HK20] Aljaafari Hamza und Basant Kumar. „A Review Paper on DES, AES, RSA Encryption Standards“. In: *2020 9th International Conference System Modeling and Advancement in Research Trends (SMART)*. 2020, S. 337. DOI: 10.1109/SMART50582.2020.9336800.

[HD05] Manfred Hauswirth und Schahram Dustdar. „Peer-to-peer: Grundlagen und Architektur“. In: *Datenbank-Spektrum* 13.2005 (2005), S. 1.

[Ho16] Chris Hoffman. „How the ‚Great Firewall of China‘ Works to Censor China’s Internet“. In: *How-To Geek* (Sep. 2016). URL: https://www.howtogeek.com/162092/htg-explains-how-the-great-firewall-of-china-works (besucht am 19. 02. 2024).

[22] *Hybrid Encryption*. Jan. 2022. URL: https://www.techopedia.com/definition/1779/hybrid-encryption (besucht am 15. 02. 2024).

[IBM21] *IBM Documentation*. März 2021. URL: https://www.ibm.com/docs/en/ztpf/2020?topic=concepts-symmetric-cryptography (besucht am 23. 01. 2024).

[Is16] Ni Made Satvika Iswari. „Key generation algorithm design combination of RSA and ElGamal algorithm“. In: *2016 8th International Conference on Information Technology and Electrical Engineering (ICITEE)*. 2016, S. 1–2. DOI: 10.1109/ICITEED.2016.7863255.

[K 23] Arun K. L. „Detailed Anatomy of the Tor Network | Structure of the Tor Network“. In: *Sec Master* (Okt. 2023). URL: https://thesecmaster.com/detailed-anatomy-of-the-tor-network-structure-of-the-tor-network (besucht am 23. 01. 2024).

[Kr16a] N. Krzyworzeka. „Asymmetric cryptography and trapdoor one-way functions“. In: *Automatyka / Automatics* 20.2 (2016), S. 40–42. ISSN: 1429-3447. DOI: 10.7494/automat.2016.20.2.39.

[KW11] Ralf Küsters und Thomas Wilke. „Asymmetrische Verschlüsselung“. In: *Moderne Kryptographie: Eine Einführung*. Wiesbaden: Vieweg+Teubner, 2011, S. 175. ISBN: 978-3-8348-8288-2. DOI: 10.1007/978-3-8348-8288-2_6. URL: https://doi.org/10.1007/978-3-8348-8288-2_6.

[La22] Lakhwinder. *Understanding WhatsApp Architecture*. Aug. 2022. URL: https://hackernoon.com/understanding-whatsapp-architecture (besucht am 08. 02. 2024).

[Li23] Max (Chong) Li. „What A Decentralized Infrastructure Is And How It Actually Works“. In: *Forbes* (Mai 2023). URL: https://www.forbes.com/sites/digital-assets/2023/05/07/what-a-decentralized-infrastructure-is-and-how-it-actually-works (besucht am 08. 02. 2024).

[LSS10] Tomáš Liška, Tomáš Sochor und Hana Sochorová. „Comparison between normal and TOR-Anonymized Web Client Traffic“. In: *Procedia - Social and Behavioral Sciences* 9 (2010). World Conference on Learning, Teaching and Administration Papers, S. 542–546. ISSN: 1877-0428. DOI: https://doi.org/10.1016/j.sbspro.2010.12.194. URL: https://www.sciencedirect.com/science/article/pii/S1877042810022998.

[Lo+24] Daniela Lopes, Jin-Dong Dong, Pedro Medeiros, Daniel Castro, Diogo Barradas, Bernardo Portela, João Vinagre, Bernardo Ferreira, Nicolas Christin und Nuno Santos. „Flow Correlation Attacks on Tor Onion Service Sessions with Sliding Subset Sum“. In: *Network and Distributed System Security Symposium*. Internet Society, Feb. 2024. ISBN: 1-891562-93-2. URL: https://www.ndss-symposium.org/ndss-paper/flow-correlation-attacks-on-tor-onion-service-sessions-with-sliding-subset-sum/ (besucht am 08. 02. 2024).

[LB21] Ben Lutkevich und Madelyn Bacon. „end-to-end encryption (E2EE)“. In: *Security* (Juni 2021). URL: https://www.techtarget.com/searchsecurity/definition/end-to-end-encryption-E2EE (besucht am 16. 01. 2024).

[Ma23] Giorgio Martinez. „Rust: Exploring Memory Safety and Performance - Giorgio Martinez - Medium“. In: *Medium* (Okt. 2023). ISSN: 9598-0120. URL: https://medium.com/@giorgio.martinez1926/unlocking-the-power-of-rust-exploring-memory-safety-and-performance-9afd5980c120 (besucht am 11. 02. 2024).

[Mi23] Microsoft. *WebSockets - UWP applications*. Juli 2023. URL: https://learn.microsoft.com/de-de/windows/uwp/networking/websockets (besucht am 13. 02. 2024).

[Op24] Openssl Foundation, Inc. */docs/man3.1/man3/RSA_public_encrypt.html*. Jan. 2024. URL: https://www.openssl.org/docs/man3.1/man3/RSA_public_encrypt.html (besucht am 04. 02. 2024).

[Pa+16] Ioana-Cristina Panait, Cristian Pop, Alexandru Sirbu, Adelina Vidovici und Emil Simion. „TOR - Didactic Pluggable Transport“. In: *Innovative Security Solutions for Information Technology and Communications*. Hrsg. von Ion Bica und Reza Reyhanitabar. Cham: Springer International Publishing, 2016, S. 225–239. ISBN: 978-3-319-47238-6.

[Re24] Reporter ohne Grenzen, e. V. *USA | Reporter ohne Grenzen für Informationsfreiheit*. Feb. 2024. URL: https://www.reporter-ohne-grenzen.de/usa (besucht am 08. 02. 2024).

[Ru24] Rust Foundation. *cargo build - The Cargo Book*. Feb. 2024. URL: https://doc.rust-lang.org/cargo/commands/cargo-build.html (besucht am 11. 02. 2024).

[Se19] Session. *Centralisation vs decentralisation in private messaging - Session Private Messenger*. Dez. 2019. URL: https://getsession.org/blog/centralisation-vs-decentralisation-in-private-messaging (besucht am 08. 02. 2024).

[Si23] Bundesamt für Sicherheit in der Informationstechnik. *BSI TR-02102-1 Kryptographische Verfahren: Empfehlungen und Schlüssellängen*. Bundesamt für Sicherheit in der Informationstechnik, Jan. 2023, S. 41. URL: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publikationen/TechnischeRichtlinien/TR02102/BSI-TR-02102.pdf?__blob=publicationFile&v=9 (besucht am 21. 01. 2024).

[Si16] Signal Messenger. *Reflections: The ecosystem is moving*. Mai 2016. URL: https://signal.org/blog/the-ecosystem-is-moving (besucht am 08. 02. 2024).

[Ta96] D. Taipale. „Implementing the Rivest, Shamir, Adleman cryptographic algorithm on the Motorola 56300 family of digital signal processors". In: *Southcon/96 Conference Record*. Juni 1996, S. 10–11. DOI: 10.1109/SOUTHC.1996.535035.

[Ta13] Marius Tarnauceanu. *A generalization of the Euler's totient function*. 2013. DOI: 10.48550/arXiv.1312.1428. arXiv: 1312.1428 [math.GR]. URL: https://doi.org/10.48550/arXiv.1312.1428.

[To24a] Tor Project. *Deriving blinded keys and subcredentials [SUBCRED] - Tor Specifications*. Jan. 2024. URL: https://spec.torproject.org/rend-spec/deriving-keys.html (besucht am 04. 02. 2024).

[To24b] Tor Project. *directory authority | Tor Project | Support*. Feb. 2024. URL: https://support.torproject.org/glossary/directory-authority (besucht am 02. 02. 2024).

[To24c] Tor Project. *How can we help?* | *Tor Project* | *Support*. Feb. 2024. URL: https://support.torproject.org/about/change-paths/ (besucht am 04. 02. 2024).

[To24d] Tor Project. *Kanal* | *Tor Project* | *Hilfe*. Jan. 2024. URL: https://support.torproject.org/de/glossary/circuit (besucht am 31. 01. 2024).

[To24e] Tor Project. *src/core/or · main · The Tor Project / Core / Tor · GitLab*. Feb. 2024. URL: https://gitlab.torproject.org/tpo/core/tor/-/blob/main/src/feature/hs/hs_descriptor.c?ref_type=heads (besucht am 04. 02. 2024).

[To23a] Tor Project. *The introduction protocol [INTRO-PROTOCOL] - Tor Specifications*. Dez. 2023. URL: https://spec.torproject.org/rend-spec/introduction-protocol.html (besucht am 04. 02. 2024).

[To23b] Tor Project. *The rendezvous protocol - Tor Specifications*. Nov. 2023. URL: https://spec.torproject.org/rend-spec/rendezvous-protocol.html (besucht am 04. 02. 2024).

[To24f] Tor Project. *Tor Project* | *How do Onion Services work?* [Online; accessed 2. Feb. 2024]. Jan. 2024. URL: https://community.torproject.org/onion-services/overview.

[To24g] Tor Project. *Tor Project* | *Talk about onions*. Jan. 2024. URL: https://community.torproject.org/onion-services/talk (besucht am 31. 01. 2024).

[Tu08] Clay S Turner. „Euler's totient function and public key cryptography". In: *Nov* 7 (2008), S. 138.

[Wa+13] Hongjun Wang, Zhiwen Song, Xiaoyu Niu und Qun Ding. „Key generation research of RSA public cryptosystem and Matlab implement". In: *PROCEEDINGS OF 2013 International Conference on Sensor Network Security Technology and Privacy Communication System*. 2013, S. 126–127. DOI: 10.1109/SNS-PCS.2013.6553849.

[Wi+22b] Sandra Wittmer, Florian Platzer, Martin Steinebach und York Yannikos. „Deanonymisierung im Tor-Netzwerk – Technische Möglichkeiten und rechtliche Rahmenbedingungen“. In: *Selbstbestimmung, Privatheit und Datenschutz : Gestaltungsoptionen für einen europäischen Weg*. Hrsg. von Michael Friedewald, Michael Kreutzer und Marit Hansen. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, S. 154–155. ISBN: 978-3-658-33306-5. DOI: 10.1007/978-3-658-33306-5_8. URL: https://doi.org/10.1007/978-3-658-33306-5_8.

[Wu+23] Mingshi Wu, Jackson Sippe, Danesh Sivakumar, Jack Burg, Peter Anderson, Xiaokang Wang, Kevin Bock, Amir Houmansadr, Dave Levin und Eric Wustrow. „How the Great Firewall of China Detects and Blocks Fully Encrypted Traffic“. In: *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, S. 2666. ISBN: 978-1-939133-37-3. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/wu-mingshi (besucht am 14. 01. 2024).

[24b] *zeroization - Glossary | CSRC*. Feb. 2024. URL: https://csrc.nist.gov/glossary/term/zeroization (besucht am 15. 02. 2024).

Anhang

[BSW15b] Albrecht Beutelspacher, Jörg Schwenk und Klaus-Dieter Wolfenstetter. „Kryptologische Grundlagen“. In: *Moderne Verfahren der Kryptographie: Von RSA zu Zero-Knowledge*. Wiesbaden: Springer Fachmedien Wiesbaden, 2015, S. 14. ISBN: 978-3-8348-2322-9. DOI: 10.1007/978-3-8348-2322-9_2. URL: https://doi.org/10.1007/978-3-8348-2322-9_2.

[Kr16b] N. Krzyworzeka. „Asymmetric cryptography and trapdoor one-way functions“. In: *Automatyka / Automatics* 20.2 (2016), S. 41. ISSN: 1429-3447. DOI: 10.7494/automat.2016.20.2.39.

[Wi+22a] Sandra Wittmer, Florian Platzer, Martin Steinebach und York Yannikos. „Deanonymisierung im Tor-Netzwerk – Technische Möglichkeiten und rechtliche Rahmenbedingungen“. In: *Selbstbestimmung, Privatheit und Datenschutz : Gestaltungsoptionen für einen europäischen Weg*. Hrsg. von Michael Friedewald, Michael Kreutzer und Marit Hansen. Wiesbaden: Springer Fachmedien Wiesbaden, 2022, S. 153. ISBN: 978-3-658-33306-5. DOI: 10.1007/978-3-658-33306-5_8. URL: https://doi.org/10.1007/978-3-658-33306-5_8.

Quellcode: https://github.com/sshcrack/enkrypton

Die Bestimmungen zur Seminararbeit am Windthorst-Gymnasium Meppen habe ich zur Kenntnis genommen.

Ich versichere, dass ich die vorgelegte Arbeit selbstständig und ohne fremde Hilfe angefertigt und alle benötigten Hilfsmittel und Quellen ordnungsgemäß angegeben und gekennzeichnet habe.

Der Veröffentlichung der nicht korrigierten Fassung der Facharbeit in der Mediathek der Windthorst-Gymnasiums

stimme ich zu ☒ stimme ich nicht zu ☐

Meppen	17.02.2024	
Ort	Datum	Unterschrift Verfasser/in