*


# Dezentralisierte asymmetrische Verschlüsselung über Tor

## Die Lösung für sicheres Messaging?

---

Hendrik Lind

Facharbeit

Windthorst-Gymnasium Meppen

Seminarfach Informatik

8. Februar 2024

# Inhaltsverzeichnis

# 1 Einleitung

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

Bei herkömmlichen Messengern, wie WhatsApp, Signal und Co., braucht die Außenwelt die Telefonnummern der im totalitären Staat wohnenden Bürgern und Reportern, um diese zu kontaktieren. Allerdings könnte ein totalitärer Staat, sich als Empfänger ausgeben, sodass Bürger/Reporter ihre private Nummer an den Staat überreichen und dieser somit jene Nummer rückverfolgen kann [**LocPolice**]. Und genau hier liegt das Problem: Bürger und Reporter können nicht durch alltägliche Messenger mit der Außenwelt kommunizieren, da der Staat deren Nummer zurückverfolgen kann und somit weiter die Meinungsfreiheit einschränkt und unterbindet [vgl. ebd.].

Durch die zentrale Infrastruktur, welche die meisten Messenger, wie zum Beispiel WhatsApp und Signal verwenden, ist es für totalitäre Staaten, wie China, möglich, die IP-Adressen jener Server zu blockieren und somit für Bürger und Reporter unzugänglich zu machen [**ChinaFirewall**; **CentralizedWhatsapp**].

Ein dezentralisierter Messenger, welcher Ende-zu-Ende verschlüsselt ist und über das Tor-Netzwerk kommuniziert, könnte bei diesen Problemen eine Lösung sein. Die Frage, ob ein solcher Messenger die Lösung für Bürger eines totalitären Staates ist, soll in dieser Arbeit geklärt werden.

Um diese Frage beantworten zu können, beschäftigt sich diese Arbeit in dem zweiten Kapitel mit der asymmetrischen Verschlüsselung, welche benötigt wird um die Ende-zu-Ende-Verschlüsselung (E2EE) umzusetzen und die Definition der E2EE, sowie der Sicherheitsbetrachtung der asymmetrischen Verschlüsselung [**E2EE-Method**]. Die Arbeit geht dabei nicht auf weitere Padding-Verfahren ein. Eine mögliche Lösung, um eine Anonymität über das Internet zu gewährleisten wird in Kapitel drei vorgeschlagen, wobei das Tor-Netzwerk eine wichtige Rolle spielt. Diese Arbeit befasst sich im vierten Kapitel mit einer Dezentralisierung der Infrastruktur, um eine weitere Sicherheitsebene zu schaffen. Zuletzt werden im fünften Kapitel die Vor-

und Nachteile eines solchen Messengers betrachtet, im sechsten Kapitel wird eine mögliche Umsetzung des Messengers beschrieben und im siebten Kapitel wird ein Fazit gezogen.

# 2 Asymmetrische Verschlüsselung

Um einen sicheren Nachrichtenaustausch zu gewährleisten, wird in dieser Arbeit die E2EE implementiert. Bei der E2EE wird von dem Sender die Nachricht, bevor sie an den Empfänger geschickt wird, verschlüsselt [**E2EE**]. Dazwischenliegende Akteure, wie zum Beispiel Server oder mögliche Angreifer, können demzufolge die Nachricht nicht lesen [vgl. ebd.]. **Nur** der Empfänger der Nachricht kann diese auch entschlüsseln. Als Ent- und Verschlüsselungsverfahren der Nachrichten wird die asymmetrische Verschlüsselung verwendet [vgl. ebd.]. Diese Arbeit beschränkt sich bei der asymmetrischen Verschlüsselung auf das RSA-Verfahren.

## 2.1 Grundlagen

Grundsätzlich gibt es bei der asymmetrischen Verschlüsselung ein Schlüsselpaar (Keypair), welches aus einem privaten Schlüssel (private key) und einem öffentlichen Schlüssel (public key) besteht [**Rsa-Basics**]. Diese beiden Schlüssel hängen mathematisch zusammen, sodass der öffentliche Schlüssel Nachrichten **nur** verschlüsseln aber nicht entschlüsseln kann [vgl. ebd.]. **Nur** der zum Schlüsselpaar dazugehörige private Schlüssel ist in der Lage, die verschlüsselte Nachricht wieder zu entschlüsseln (siehe Abb. 1) [vgl. ebd.].

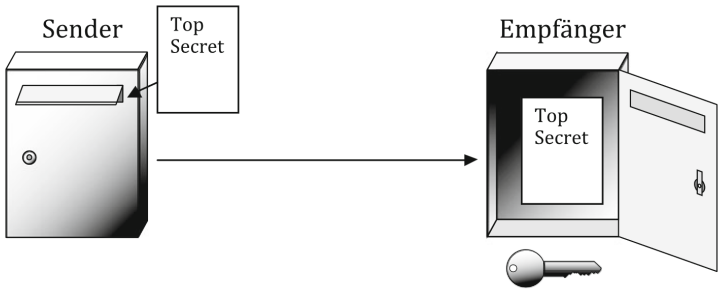


Abbildung 1: Jeder Sender kann mit dem öffentlichen Schlüssel die Nachricht „verschlüsseln“ (also eine Nachricht in den Briefkasten werfen), aber nur der Empfänger kann den Briefkasten mit seinem privaten Schlüssel öffnen und somit die Nachricht herausnehmen[**fig:Rsa-Cryptography**]

## 2.2 Mathematische Betrachtung

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

Für die Generierung des Schlüsselpaares benötigen wir zuerst zwei große zufällige Primzahlen, $P$ und $Q$ [vgl. ebd.]. Daraus ergibt sich $n = P*Q$, wobei $P \neq Q$, sodass $P$ bzw. $Q$ nicht durch $\sqrt{n}$ ermittelt werden kann [vgl. ebd.]. Der private Schlüssel besteht aus den Komponenten $\{n, d\}$ währenddessen der öffentliche Schlüssel aus $\{n, e\}$ besteht [**RsaVariables**].

### 2.2.1 Eulersche Phi-Funktion

Die Eulersche Phi-Funktion spielt eine wichtige Rolle in dem RSA-Verfahren [**TotientFuncMultiplicative**]. Grundsätzlich gibt $\phi(x)$ an, wie viele positive teilerfremde Zahlen bis $x$ existieren (bei wie vielen Zahlen der größte gemeinsamer Teiler (gcd) $1$ ist) [**EulersTotientFunction**]. Somit ergibt $\phi(6) = 2$ oder bei einer Primzahl $\phi(7) = 7 - 1 = 6$ somit $\phi(x) = x - 1$, wenn $x$ eine Primzahl ist, da jede Zahl kleiner als $x$ teilerfremd sein muss [**TotientFuncMultiplicative**].

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

### 2.2.2 Generierung des Schlüsselpaares

Sowohl der private als auch der öffentliche Schlüssel besteht unter anderem aus folgender Komponente: $n = P \cdot Q$ [**RsaMaths1**]. Für den öffentlichen Schlüssel benötigen wir die Komponente $e$, die zur Verschlüsselung einer Nachricht verwendet wird [vgl. ebd.]. $e$ ist hierbei eine zufällige Zahl, bei welcher 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 errechneten Komponente $e$, welche Nachrichten verschlüsselt, kann der öffentliche Schlüssel nun an den Sender übermittelt werden.

Um den privaten Schlüssel zu berechnen, benötigen wir die Komponente $d$, wel-

che zur Entschlüsselung verwendet wird [**RsaEncryptionDecryption**].

$$\phi(n) = (P-1)(Q-1)$$
$$e \cdot d = 1 \mod \phi(n)$$

## 2.3 Sicherheit

Um die Sicherheit des RSA-Verfahrens betrachten zu können, müssen wir nun den Ver/-Entschlüsselungsvorgang 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 $c$ zu entschlüsseln, bräuchte ein Angreifer die Komponente des privaten Schlüssels $d$. $d$ ist allerdings mit einem starken Rechenaufwand verbunden, da, wie schon vorher bereits gezeigt, dafür $\phi(n)$ kalkuliert werden müsste. Somit wird eine Primfaktorzerlegung von $n$ benötigt wird [**EulersTotientFunction**]. Bei der Verschlüsselung von $m$ zu $c$ liegt eine Trapdoor-Einwegfunktion vor [**RsaTrapdoor**]. Das bedeutet, dass es zwar leicht ist $f(x) = i$ zu berechnen (bei RSA: Verschlüsselung), es jedoch unmöglich ist von $i$ auf den Ursprungswert $x$ zu schließen, ohne dass weitere dafür notwendigen Komponente bekannt sind (bei RSA wäre die benötigte Komponente $d$) [vgl. ebd.].

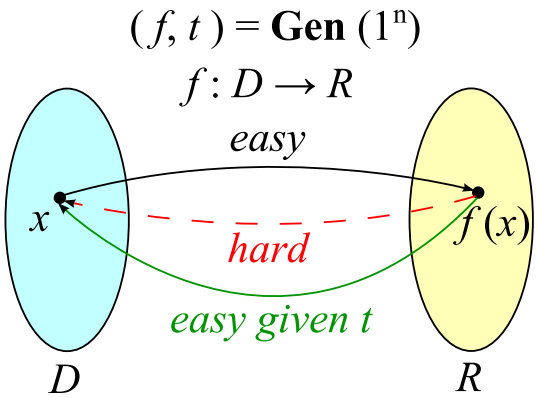


Abbildung 2: Die Trapdoor-Einwegfunktion bildlich dargestellt[**fig:TrapdoorPermutation**]

Wichtig bei dem RSA-Verfahren ist, dass die Länge von $n$ (die Schlüssellänge) mindestens 3000 Bit betragen sollte, da sonst die Primfaktorzerlegung von $n$ mit modernen Computern möglich sein könnte [**RsaKeyLength**].

## 2.4 Vergleich zur symmetrischen Verschlüsselung

Bei der symmetrischen Verschlüsselung wird der gleiche Schlüssel sowohl für die Verschlüsselung als auch für die Entschlüsselung verwendet [**GeneralSymmetricCryptography**]. Im Vergleich zu der asymmetrischen Verschlüsselung, ist die symmetrische Verschlüsselung schneller und keine Beschränkung des Chiffretextes [**RsaAESAnalysis**; **OpensslRsaMaxLength**]. Jedoch muss der Schlüssel der symmetrischen Verschlüsselung sicher an den jeweils anderen Kommunikationspartner übermittelt werden, um Nachrichten zu entschlüsseln [vgl. ebd.].

# 3 Anonymität des Tor-Netzwerkes

Die Anonymität des Messengers ist ein weiterer zentraler Aspekt, um die jeweiligen Kommunikationspartner zu schützen. Das Tor-Netzwerk ist hierbei ein möglicher Lösungsansatz, da normalen Routing, wie wir es tagtäglich nutzen, der Client kommuniziert direkt mit dem Zielserver, somit der Zielserver die IP-Adresse des Clients einsehen kann [**TCP_IP**]. Eine Rückverfolgung der IP-Adresse ist somit möglich [**LocPolice**]. Und genau bei diesem Problem setzt das Tor-Netzwerk an. Das Tor-Netzwerk besteht hierbei aus vielen *Nodes* , welche eingehende Tor-Verbindungen akzeptieren und weiterverarbeiten. Damit ein Client eine Anfrage über das Netzwerk verschicken kann, sucht er zunächst einen Pfad durch das Netzwerk, genannt *Circuit* [**TorCircuits**]. Der *Circuit* besteht dabei meist aus drei *Nodes* und ist für 10 Minuten gültig, bis der Client das Circuit erneuert (also einen neuen Pfad „sucht“) [**FAQCircuitLifetime**]. Die drei *Nodes* werden klassifiziert in einer *Entry Node* , einer *Relay Node* und einer *Exit Node* , worüber später Anfragen an die Zielserver geschickt werden können [vgl. ebd.]. Dabei verschlüsselt der Client verschlüsselt die eigentliche Anfrage mehrmals, hüllt sie also in ein mehrere „Schalen“ ein, welche eine *Onion* bilden, und leitet diese über den *Circuit* an den Zielserver weiter[**TorDesign**]. Anfangs wird die *Onion* von dem Tor-Client an die *Entry Node* geschickt [vgl. ebd.]. Bei jeder *Node* , wie der *Entry Node* , wird eine Schale der *Onion* „geschält“ (die *Onion* also einmal entschlüsselt), welche Informationen zu dem nächsten Knotenpunkt enthält [**TorStructure2**]. Die Node leitet nun die um eine Schale „geschälte“ *Onion* an den nächsten Knotenpunkt weiter [vgl. ebd.]. Sobald die Anfrage bei der *Exit Node* angekommen ist, entfernt diese die letzte „Schale“ der Anfrage, welche

nun vollständig entschlüsselt ist und an den Zielserver geschickt werden kann [vgl. ebd.]. **Nur** die *Entry Node* weiß somit die reale IP-Adresse des Clients und **nur** die *Exit Node* weiß, an welchen Zielserver die Anfrage geschickt wurde [vgl. ebd.]. Dazwischenliegende *Nodes* , wie die *Relay Node* erkennen nur eine verschlüsselte Anfrage und haben keinerlei Informationen über den eigentlichen Client oder den Zielserver. Da die *Exit Node* die Anfrage an den Zielserver verwendet, könnte diese Informationen der Nutzer auslesen (wenn die Anfrage nicht über das HTTPS-Protokoll versendet wurde) und IP-Adressen der Zielserver speichern [**TorExitNodeVulnerability**]. Durch Etablieren von *Exit Nodes* in dem Tor-Netzwerk könnten Angreifer somit die Anonymität des Netzwerkes bzw. der Nutzer gefährden [vgl. ebd.].

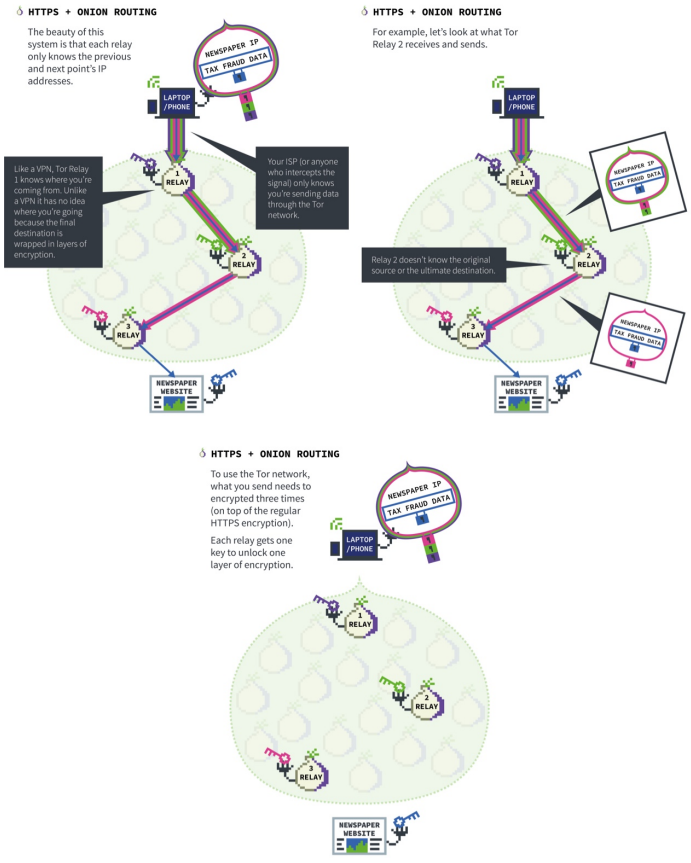


Abbildung 3: Tor Entschlüsselungslayer [**fig:Tor-Structure**]

## 3.1 Onion Services

Onion Services sind eine mögliche Lösung für den *Exit Node* -Angriff [**TorOnionServiceTalk**]. Diese nicht auf die *Exit Node* angewiesen sind und agieren nur innerhalb des Tor-Netzwerkes mit anderen Tor-Clients und verhalten sich wie normale Tor-Clients [vgl.

ebd.]. Onion Services sind nicht über das normale Internet erreichbar, wie die Zielserver im vorherigen Beispiel, sondern nur über das Tor-Netzwerk [**TorOnionService**]. Bei Onion Services müssen im Vergleich zu normalen öffentlichen Servern keine Ports geöffnet werden, damit ein Client sich mit dem Server verbinden kann, da der Onion Service direkt mit dem Tor-Netzwerk über ausgehende Verbindungen kommuniziert (auch bekannt als *NAT punching*) und darüber sämtliche Daten geleitet werden [vgl. ebd.].

### 3.1.1 Verbindungsaufbau

Zunächst generiert der Onion Service ein Schlüsselpaar, bestehend aus einem öffentlichen und einem privaten Schlüssel [**GeeksOnionService**]. Unter anderem wird nun aus dem öffentlichen Schlüssel die Adresse des Onion Services generiert und endet mit „.onion“ [vgl. ebd.]. Ein Beispiel für eine solche Adresse ist die der Suchmaschine DuckDuckGo:

*duckduckgogg42xjoc72x3sjasowoarfbgcmvfimaftt6twagswzczad.onion* [**DuckDuckGoLink**].

Der Onion Service sich nun mit dem Tor-Netzwerk wie ein normaler Client über einen *Circuit* , welcher aus drei *Nodes* besteht [**TorOnionService**]. Der Service sendet eine Anfrage an das letzte Relay im *Circuit* , sodass es als *Introduction Point* dient und etabliert eine Langzeitverbindung jenem (der *Circuit* erneuert sich also nicht alle 10 Minuten wie bei dem Tor-Client) [vgl. ebd.]. Dieser Vorgang wiederholt sich zwei Mal, 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*, welcher die Adressen der *Introduction Points* und Authentifizierungsschlüssel enthält, signiert diesen mit seinem privaten Schlüssel und schickt den *descriptor* an die Directory Authority [**TorSpecDerivingKeys**; **TorSpecDirectoryInf**]. Die Directory Authority ist ein Server, welcher Informationen über das Tor-Netzwerk, wie zum Beispiel die des Onion Services, speichert und verteilt [**TorDirectoryAuthority**]. Im Sinne dieser Arbeit, hat der Onion Service sich nun von Person A mit dem Tor-Netzwerk verbunden, jedoch braucht es noch Person B, welche sich über ihren Tor-Client mit dem Onion Service der Person B verbindet. Damit der Tor-Client von Person A sich allerdings verbinden kann, fragt der Client nun die Directory Authority an den signierten *descriptor* des Onion Services von Person B an den Client zu schicken

[**TorStructure**]. Der Client besitzt nun die Adressen der *Introduction Points* und die Signatur des *descriptors*, sodass dieser mit dem öffentlichen Schlüssels der Onion Adresse die Signatur überprüfen kann [vgl. ebd.]. Der Client generiert nun 20 zufällige Bytes (Secret) und schickt diese an eine zufällig ausgewählte *Relay Node* , welche nun als *Rendezvous Point* dient [**TorSpecRendezvous**]. Das Secret wird von dem Client auch an eine von den *Introduction Points* geschickt, sodass der Onion Service mit dem gleichen Secret eine Verbindung zu dem *Rendezvous Point* aufbauen kann [**TorSpecIntroP**]. Der *Rendezvous Point* leitet, wenn der Client und der Onion Service über deren *Circuits* miteinander verbunden sind, die Nachrichten zwischen den beiden weiter [**TorSpecRendezvous**]. Der Client und der Onion Service kommunizieren nun also nur über das Tor-Netzwerk miteinander und sind nicht mehr auf die *Exit Node* angewiesen.

## 3.2 Vor- und Nachteile

In diesen Vor- und Nachteilen beziehe ich mich ausschließlich auf die Verbindung zwischen Onion Services und Clients. Nicht auf die Verbindung über Tor zwischen Client und Zielserver über *Exit Node* .

Vorteile der Kommunikation zwischen Client und Onion Services, sind, wie bereits in diesem Kapitel erläutert, die Anonymität. Die verschiedenen Verschlüsselungsebenen der *Onion* sorgen jedoch dafür, dass das Tor-Netzwerk um das 120-fache langsamer ist im Vergleich zum normalen Routing [**TorPerformance**]. 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 diese mit Hilfe von Machine Learning analysiert [**OnionServiceFingerprinting**]. Dabei war die Umsetzung der theoretischen Grundlage bis jetzt mit idealisierten Bedingungen erfolgreich, jedoch braucht es noch weiterer Studien, um die möglichen Gefahren dieses Algorithmus einzustufen [vgl. ebd.]. Dabei könnten, wenn die deutsche, amerikanische und die französische Regierung zusammenarbeiten, 78,34 % der *Circuits* entanonymisieren [vgl. ebd.].

# 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.

## 4.1 Sicherheit

# 5 Vor- und Nachteile

## 5.1 Kriminalität

## 5.2 Freie Meinungsäußerung

# 6 Umsetzung des Messengers

# 7 Fazit

*

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

*