HAW HAMBURG

Masterarbeit

Denis Lugowski

Eine Vergleichsstudie zu TLS-1.3-Bibliotheken in einer QUIC-Simulationsumgebung

Fakultät Technik und Informatik
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

Denis Lugowski

Eine Vergleichsstudie zu TLS-1.3-Bibliotheken in einer QUIC-Simulationsumgebung

Mastertarbeit eingereicht im Rahmen der Masterprüfung
im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Martin Becke
Zweitgutachter: Prof. Dr.-Ing. Martin Hübner

Eingereicht am: 23. April 2019

Denis Lugowski

Thema der Arbeit

Eine Vergleichsstudie zu TLS-1.3-Bibliotheken in einer QUIC-Simulationsumgebung

Stichworte

Transportprotokoll, Informationssicherheit, QUIC, TLS 1.3, OMNeT++

Kurzzusammenfassung

Das Transportprotokoll QUIC basiert auf dem Verschlüsselungsprotokoll TLS 1.3 und ermöglicht den sicheren Austausch von Schlüsselinformationen sowie eine sichere Datenübertragung über die QUIC Packet Protection. Im Gegensatz zur QUIC Packet Protection finden gängige Implementierungen des TLS-1.3-Verschlüsselungsprotokolls eine weite Verbreitung und sind Gegenstand zahlreicher Forschungsarbeiten geworden. Ein möglicher alternativer Verschlüsselungsansatz für QUIC besteht aus einer vollständigen Verschlüsselung über eine TLS-Bibliothek. In einer Vergleichsstudie werden beide Verschlüsselungsansätze in einem Simulationsmodell von QUIC in OMNeT++ entwickelt und im Anschluss gegenübergestellt. Dabei werden die Vorzüge und Unterschiede beider Ansätze hervorgehoben und diskutiert, ob eine vollständige Verschlüsselung über eine TLS-Bibliothek als eine ernstzunehmende Alternative zur derzeitigen Spezifikation in Erwägung gezogen werden kann.

Denis Lugowski

Title of Thesis

A comparative study on TLS 1.3 libraries in a QUIC simulation environment

Keywords

Transport protocol, information security, QUIC, TLS 1.3, OMNeT++

Abstract

The transport protocol QUIC is based on the cryptographic protocol TLS 1.3 and allows a secure exchange of keying material and data over the QUIC Packet Protection. In contrast to the QUIC Packet Protection some common implementations of TLS 1.3 are widely distributed and were subject of numerous research works. A possible alternative cryptographic approach for QUIC consists of a complete encryption over a TLS library. In a comparative study both cryptographic approaches will be developed in a simulation model of QUIC in OMNeT++ and compared to each other. The advantages and differences of both approaches will be pointed out and discussed, whether a complete encryption over a TLS library could be a serious alternative to the current specification.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Die Kommunikation über das Internet befindet sich im stetigen Wandel und hat in den Jahrzehnten seit dem Bestehen große Veränderungen erfahren. In den Anfangszeiten wurde das Internet ausschließlich für das militärische Umfeld konzipiert. Das Ziel der DARPA[1] lag primär darin, eine effektive Technik für eine Kommunikation zwischen verbundenen Netzwerken zu entwickeln. Dabei wurde großen Wert auf eine hohe Zuverlässigkeit und Flexibilität gelegt, die besonders im Kriegsumfeld essentiell sind [1]. Heutzutage besitzt ein Großteil der Weltbevölkerung Zugang zum Internet und damit auch zu zahlreichen Diensten. Die Nutzung dieser neuen Dienste hat eine Zunahme der Anforderungen an das Internet zur Folge, die von etablierten Protokollen, wie z.B. dem weit verbreiteten Transportprotokoll TCP [2] nicht berücksichtigt werden.

QUIC [3] ist ein von Google initiiertes Transportprotokoll, das mit dem Ziel entwickelt worden ist, die Netzwerkperformance zu erhöhen [4]. Der Schwerpunkt in der Entwicklung von QUIC liegt in der Reduktion der Latenz bei der Übertragung von Daten. Dies wird beispielsweise durch eine Reduktion der *Round Trip Time* (RTT) beim Verbindungsaufbau erzielt (*1-RTT handshake*, *0-RTT handshake*). Besonders Dienste, die gegenüber größeren Verzögerungen in der Datenübertragung anfällig sind, wie z.B. HTTP profitieren am meisten von den Verbesserungen, da die darüber ausgelieferten Webseiten aus zahlreichen kleinen Ressourcen bestehen, welche innerhalb kürzester Zeit nach Anforderung beim Benutzer antreffen müssen. Seit dem Jahr 2016 beschäftigt sich eine *Working Group* (QUIC WG) bei der IETF[2] um eine Realisierung des Protokolls QUIC als Internet-Standard [5].

Das Simulationsframework OMNeT++[3] ist ein Werkzeug, um Netzwerke zu simulieren und Analysen von Protokollen, Topologien usw. zu erstellen. Es stellt eine Entwicklungsumgebung bereit, mit der eigene Simulationsmodelle entwickelt werden können.

[1] Abkürzung für *Defense Advanced Research Projects Agency*
[2] Abkürzung für *Internet Engineering Task Force* – (https://www.ietf.org/)
[3] https://omnetpp.org/

Das INET-Framework[4] ist eine Sammlung von zahlreichen Implementierungen von Internetprotokollen (IP, TCP, UDP usw.) und wird, sofern benötigt, in das eigene Simulationsmodell eingebunden. Trotz der bevorstehenden Finalisierung von QUIC existiert bis zum heutigen Zeitpunkt noch kein Simulationsmodell des Protokolls. Im Rahmen des Masterprojekts an der HAW Hamburg bestand das Ziel darin, ein Simulationsmodell von QUIC für das INET-Framework zu entwickeln, um Analysen am Protokoll in einem kontrollierten Umfeld durchzuführen. Dabei stellt die Verschlüsselung der Kommunikation eine bedeutende Funktion von QUIC dar, die bisher nicht im Modell umgesetzt worden ist. Auf Basis des Verschlüsselungsprotokolls TLS 1.3 werden die benötigten Pakete zum TLS-Handshake und die danach zur Verfügung stehenden Schlüsselinformationen nach QUIC exportiert und verwaltet. Aus den Schlüsselinformationen leitet QUIC die konkreten Schlüssel zur Verschlüsselung ab und übernimmt ab diesem Zeitpunkt das Ver- und Entschlüsseln von Paketen.

Die Idee, die endgültige Ver- und Entschlüsselung durch QUIC durchführen zu lassen (QUIC-mit-TLS), führt zu der Frage, inwiefern es sich von einem Ansatz unterscheidet, in dem eine TLS-1.3-Bibliothek den gesamten Prozess der Verschlüsselung übernimmt (QUIC-über-TLS). Zahlreiche TLS-1.3-Bibliotheken sind bereits weltweit im praktischen Einsatz und haben sich im Prozess der vollständigen Ver- und Entschlüsselung etabliert, während QUIC noch den Standardisierungsprozess durchläuft und bisher einen weniger breiten Einsatz findet.

Zur Lösung dieser Problemstellung ist die Masterarbeit wie folgt aufgebaut: Die Grundlagen zu den Protokollen QUIC, TLS 1.3 und den verwendeten Frameworks werden in Kapitel 2 vorgestellt und es wird ein Überblick über die verwandten Arbeiten gegeben, die im Zusammenhang mit QUIC und der Verschlüsselung stehen. Danach folgt in Kapitel 3 eine Beschreibung der methodischen Vorgehensweise zur Lösung der Forschungsfrage. Zuerst wird hierfür das bestehende Simulationsmodell von QUIC beschrieben. Daran schließt die Auswahl einer geeigneten TLS-Bibliothek für die Verschlüsselungsansätze an und die Vorstellung der Architekturen und des Vorgehens bei der Implementierung in das bestehende Simulationsmodell. Die Methode zum Test der Implementierungen schließt das Kapitel ab. Kapitel 4 hat die Definition und Durchführung der im Rahmen der Vergleichsstudie durchgeführten Experimente zum Inhalt und stellt Kriterien zur Bewertung auf. Der abschließende Abschnitt des Kapitels befasst sich mit der Vorstellung und Diskussion der Ergebnisse. In Kapitel 5 werden schließlich die wesentlichen Ergebnisse und Schlüsse zusammengefasst und ein Ausblick auf mögliche zukünftige Arbeiten gegeben.

[4] https://inet.omnetpp.org/

1.1 Motivation

Das QUIC-Protokoll nutzt den TLS-Handshake, um eine sichere Kommunikation zwischen zwei Hosts herzustellen. Nach Abschluss des Handshakes werden die bereitstehenden Schlüsselinformationen nach QUIC exportiert und daraus die benötigten *secret keys* abgeleitet. Der Prozess der Verwaltung der *secret keys* und der Ver- und Entschlüsselung von Datenpaketen übernimmt ab diesem Zeitpunkt QUIC durch die *QUIC Packet Protection* (QPP). Einzig für die konkreten Funktionen zur Ver- und Entschlüsselung greift QUIC noch auf TLS zurück. Ansonsten bleibt die TLS-Komponente passiv, solange keine Daten auf dem dafür vorgesehenen Datenstream eintreffen. Dieser mit dem Internet-Draft konforme Verschlüsselungsansatz wird in dieser Arbeit als *QUIC-mit-TLS* bezeichnet, da TLS aufgrund der Abhängigkeit der QPP von den Ver-/Entschlüsselungsfunktionen tief mit[5] QUIC einbezogen ist und beide Komponenten damit eine Einheit bilden.

QUIC-mit-TLS ist ein Ansatz, zu dem bisher nur wenige Forschungsarbeiten hervorgebracht wurden (siehe Abschnitt 2.5). Dagegen finden gängige Implementierungen des TLS-1.3-Verschlüsselungsprotokolls (z.B. OpenSSL, BoringSSL, GnuTLS usw.) eine weite Verbreitung und sind direkt oder indirekt als Bestandteil einer Applikation Gegenstand zahlreicher Forschungsarbeiten wie z.B. [6]–[8] geworden. Es liegt daher der Gedanke nahe, die Daten von QUIC vollständig über[6] eine TLS-Bibliothek verschlüsseln zu lassen. Diese Lösung wird hier als *QUIC-über-TLS* bezeichnet und findet in der für QUIC vorgegebenen Spezifikation keine Berücksichtigung. In der nachfolgenden Abbildung 1.1 ist der Unterschied beider Ansätze zu sehen. Während die QPP beim QUIC-mit-TLS die Verwaltung der Schlüssel als auch die Ver- und Entschlüsselung auf Basis von TLS übernimmt, sieht das QUIC-über-TLS einen reinen Einsatz von TLS vor und besitzt daher keine Notwendigkeit für eine Zwischenkomponente wie der QPP.

[5] https://www.duden.de/rechtschreibung/mit_inklusive_auch_mitsamt#Bedeutung2b
[6] https://www.duden.de/rechtschreibung/ueber_auf_darueber_darauf#Bedeutung1j

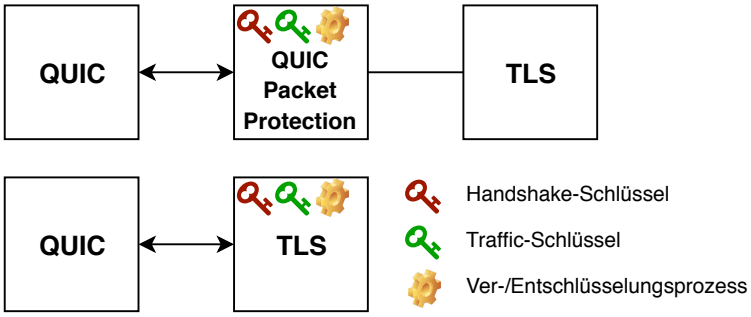


Abbildung 1.1: QUIC-mit-TLS (oben), QUIC-über-TLS (unten)

Da das QUIC-mit-TLS bisher kaum Gegenstand bisheriger Forschungsarbeiten gewesen ist und das QUIC-über-TLS eine Lösung darstellt, die möglicherweise in der Zukunft in Betracht gezogen werden könnte, bietet sich ein Vergleich beider Ansätze an. In Abhängigkeit von den ausgewählten Metriken wäre zu untersuchen, wie beide Ansätze gegeneinander abschneiden. Die Ergebnisse aus diesen Vergleich geben Aufschluss darüber, ob die tiefere Integration von TLS in QUIC-mit-TLS einen Mehrwert gegenüber QUIC-über-TLS bietet. Die in den Experimenten verwendeten Parameter weisen dabei auf mögliche Ausnahmefälle hin, in denen ein Ansatz dem anderen vorzuziehen ist. Des Weiteren bringt der Vergleich Aufschluss über das mögliche Potential von QUIC-über-TLS und somit eine Grundlage für die Beantwortung der Frage, ob die Existenz einer weiteren Verschlüsselungsalternative in QUIC berechtigt ist.

In der Vergleichsstudie dieser Masterarbeit werden die eben genannten Punkte genau untersucht und umfänglich beantwortet.

2 Grundlagen

In diesem Teil werden die Grundlagen vorgestellt, um ein Verständnis über die in dieser Arbeit vermittelten Inhalte zu schaffen. Hierfür wird zu Beginn eine Einführung in das Protokoll QUIC und dessen Features gegeben (Abschnitt 2.1). Im Anschluss werden in Abschnitt 2.2 die Grundlagen von TLS 1.3 erläutert sowie im Detail die Schlüsselerzeugung und die Funktion von AEAD. Abschnitt 2.3 beschreibt die Interaktion zwischen QUIC und TLS. Dabei wird hier die Schlüsselerzeugung von QUIC erläutert, die einige Unterschiede zur Schlüsselerzeugung von TLS aufweist. Die wesentlichen Merkmale des OMNeT++- und INET-Frameworks stellt Abschnitt 2.4 vor, die das Fundament für die Simulation legen, bevor im letzten Abschnitt 2.5 einige Forschungsarbeiten vorgestellt werden, welche im Zusammenhang mit der Verschlüsselung in QUIC stehen.

2.1 QUIC

Bei QUIC (ursprünglich ein Akronym für *Quick UDP Internet Connections* [9]) handelt es sich um ein Transportprotokoll, in dem viele Konzepte neuerer Protokolle wie z.B. SCTP [10] oder HTTP/2 [11] im Design eingeflossen sind. Im Folgendem werden die wesentlichen Funktionen von QUIC vorgestellt und kurz beschrieben.

- **Verschlüsselung und Authentifizierung**
  Die in QUIC versendeten Pakete werden mit TLS 1.3 verschlüsselt [12]. Durch die Verschlüsslung ist es Dritten nicht mehr möglich, die Informationen in den Daten einzusehen, womit die Vertraulichkeit gewährleistet wird. Des Weiteren bringt die Verwendung von TLS den Vorteil mit sich, dass der Server während des Verbindungsaufbaus durch sein Zertifikat authentifiziert wird und der Client optional auch authentifiziert werden kann.

- **Verringerte Latenz beim Verbindungsaufbau**
  Die Verwendung von TLS setzt die Nutzung eines Transportprotokolls voraus, das

Daten zuverlässig und in Reihenfolge übertragen kann. Für diesen Zweck wird in den meisten Fällen TCP verwendet. Die Kombination beider Protokolle hat den Nachteil, dass jeweils gesondert eine RTT für den Handshake von TCP und danach für TLS 1.3 anfällt. Die Integration von TLS 1.3 in QUIC verringert die RTT auf eins, wenn der Client und der Server zuvor noch nicht miteinander kommuniziert haben (*1-RTT handshake*). Treten der Client und der Server zu einem späteren Zeitpunkt erneut in Kontakt, so kann der Client zusätzlich zu einer Information aus der vergangenen Verbindung zum selben Server seine Daten anhängen, wodurch keine RTT auf den Handshake alleine entfällt (*0-RTT handshake*). Besonders bei vielen kurzlebigen Verbindungen, wie es bei HTTP der Fall ist, kommt diese Verringerung der Latenz sehr zum Tragen.

- **Stream Multiplexing**
  Anders als in TCP, das Daten in einen einzigen Bytestrom überträgt, ist die Datenübertragung in QUIC in logisch voneinander unabhängige Bytestreme (Streams) unterteilt. Daten von unterschiedlichen Streams können dabei in einem Paket kombiniert werden. Diese Form der Datenübertragung kam bereits im Transportprotokoll SCTP zum Einsatz [13] und wird auch in HTTP/2 verwendet. Bei QUIC bezieht sich die in Reihenfolge gesicherte Übertragung nur auf die Übertragung innerhalb eines Streams, weshalb ein Paketverlust in einem Stream die Übertragung der Daten auf den anderen Streams nicht verzögert. Dieses Problem tritt bei TCP auf und wird als *Head-of-Line Blocking* bezeichnet [14].

- **Implementierung im Userspace**
  Alle gängigen Netzwerkprotokolle ab der vierten Schicht des TCP/IP-Modells finden ihre Implementierung im Kernel des Betriebssystems (*kernel space*). QUIC setzt dagegen auf das verbindungslose UDP auf und erlaubt eine Implementierung des Protokolls in der User-Space-Verwaltungsstruktur des Betriebssystems. Die Vorteile dieser Vorgehensweise sind [15]:

  – Hohe Akzeptanz durch Kapselung der QUIC-Pakete als Nutzdaten in UDP-Pakete bei Middleboxen (Geräte oder Dienste, die häufig ihnen unbekannte Protokolle herausfiltern) und damit die Möglichkeit einer weiten Verbreitung.

  – Unabhängigkeit vom Anbieter des Betriebssystems, der Updates oder neue Releases nur in seinem eigenen Takt veröffentlicht.

- Einsatz moderner Softwareentwicklungsmethodiken mit Unit- und Ende-zu-Ende-Tests im *user space* möglich.
- Einsatz eines erweiterten Loggings durch das Wegfallen der Speicherbeschränkung im *user space*, wodurch ein ein Jahrzehnt alter Bug in der *Congestion Control* Cubic aufgedeckt werden konnte.

Diese Menge an Features zeigt bereits, dass QUIC ein großes Potential besitzt, die Datenkommunikation im Internet nachhaltig zu ändern. Es wurden Konzepte moderner Protokolle übernommen und dabei versucht, die Hindernisse der Protokolle, wie beispielsweise die geringe Verbreitung von SCTP, durch die Nutzung von UDP als Transportmedium zu vermeiden. Nach dem derzeitigen Stand wird der von der QUIC WG ausgearbeitete Internet-Draft zum Kern des Protokolls im Juli 2019 der IESG[1] zur Prüfung und Veröffentlichung als RFC[2] eingereicht [16]. Das Ziel ist die Ernennung von QUIC zum Internet Standard.

2.2 TLS 1.3

Im August 2018 wurde das Verschlüsselungsprotokoll TLS (*Transport Layer Security*) in der Version 1.3 als RFC 8446 [17] durch die IETF veröffentlicht. Auf Grundlage eines zuverlässigen und in Reihenfolge übertragenden Transportprotokolls stellt TLS für die Kommunikation zwischen zwei Hosts eine sichere Verbindung her. Dabei werden die Vertraulichkeit und Integrität der Verbindung sichergestellt, die Bestandteile der Schutzziele der Informationssicherheit darstellen [18]. Die Vertraulichkeit gewährleistet, dass der Inhalt der Datenübertragung nur für die kommunizierenden Hosts sichtbar ist, während die Integrität den Inhalt vor unberechtigter Modifikation schützt. Darüber hinaus setzt TLS bei einer Verbindung immer die Authentifizierung des Servers gegenüber dem Client voraus, womit der Client zweifelsfrei die Identität des Servers überprüfen kann.

Im Wesentlichen besteht eine TLS-Verbindung aus zwei Teilen: Dem *handshake protocol*, das vorbereitende Schritte zur Etablierung einer sicheren Datenübertragung durchführt und dem *record protocol*, das für die eigentliche Verschlüsselung der Datenübertragung zuständig ist. Nachfolgend wird der Ablauf beider Protokolle näher erläutert.

[1] Abkürzung für *Internet Engineering Steering Group*
[2] Abkürzung für *Request for Comments*

2.2.1 Handshake protocol

Bevor zwischen zwei Hosts eine sichere Verbindung zur Kommunikation stattfinden kann, müssen beide Teilnehmer auf einem sicheren Weg Schlüsselinformationen austauschen. Da zu Beginn keine Schlüssel zur Verschlüsselung existieren und damit der Vorgang des Austauschs für alle Teilnehmer des Netzwerks transparent einsehbar ist, liegt die Herausforderung darin, ein sicheres Geheimnis zwischen den Kommunikationsteilnehmern zu etablieren. In der Abbildung 2.1 ist der Nachrichtenverlauf eines TLS-Handshakes zu sehen. Hier werden die Schlüsselinformationen mithilfe des Diffie-Hellman-Protokolls (DH) ausgetauscht. Die Schlüsselinformation besteht aus einem *secret value*, der die Voraussetzung für die Berechnung eines *secret key* darstellt [19]. Mit diesem *secret key* erfolgt die symmetrische Ver- und Entschlüsselung der Kommunikation zwischen den Hosts. Für die Berechnung eines gemeinsamen *secret value* aus den Schlüsselinformationen kann entweder auf das übliche Diffie-Hellman-Verfahren [20] (*Finite Field DH*) oder auf ein Verfahren unter Einsatz von elliptischen Kurven (*Elliptic Curve DH*) zurückgegriffen werden. Unabhängig von der Wahl des Verfahrens wird das Protokoll mit (EC)DHE abgekürzt. Neben dem reinen Einsatz von (EC)DHE unterstützt TLS 1.3 den Austausch von Schlüsselinformationen über *Pre-Shared Keys* (PSK) und eine Kombination aus PSK und (EC)DHE. Diese Methoden setzen jedoch eine zuvor stattgefundene Verbindung voraus, in der beide Hosts einen PSK für eine spätere Verbindung teilen. Der Vorteil der Verwendung eines PSK ist, dass der Client bereits mit dem ersten Paket Applikationsdaten versenden kann, die mit dem PSK verschlüsselt sind. Daraus resultiert ein 0-RTT-Handshake, der die Latenz zu Beginn der Verbindung auf ein Minimum reduziert.

Zur Initiierung einer Verbindung zum Server sendet der Client ein ClientHello-Paket mit folgendem Inhalt:

- Zufällig generierter Nonce (`random`)
- Liste mit unterstützten symmetrischen Verschlüsselungen sowie Hashalgorithmen (`cipher_suites`; siehe Abschnitt 2.2.3)
- Legacy-Datenfelder, damit der TLS-1.3-Handshake wie ein TLS-1.2-Handshake erscheint (`legacy_version`, `legacy_session_id`, `legacy_compression_methods`)
- Erweiterungen

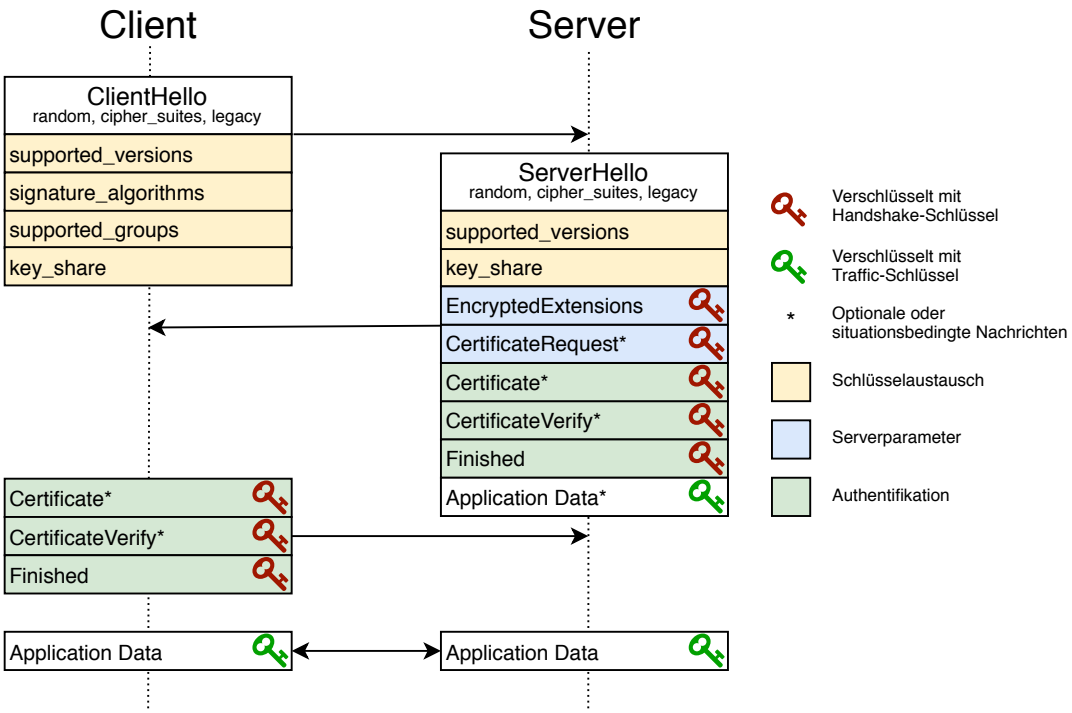


Abbildung 2.1: Versendete Nachrichten beim TLS-Handshake mit DH

In den Erweiterungen teilt der Client dem Server seine unterstützten TLS-Versionen, eine Liste von akzeptierten Signatur-Algorithmen und unterstützten *Diffie-Hellman Groups* mit. Zusätzlich wird eine Erweiterung `key_share` angegeben, die Parameter für einige oder alle angegebenen *Diffie-Hellman Groups* enthält.

Beim Empfang eines ClientHello generiert der Server einen eigenen Nonce und wählt eine vom Client unterstützte `cipher_suite`, *Diffie-Hellman Group* und TLS-Version aus. Daraus wird ein ServerHello-Paket mit derselben Struktur wie das ClientHello aufgebaut. Dem ServerHello-Paket werden überdies folgende Erweiterungen beigefügt:

- Ausgewählte TLS-Version
- `key_share` mit eigenem Parameter für ausgewählte *Diffie-Hellman Group*
- Ausgewählte Antworten auf die Erweiterungen im ClientHello in `EncryptedExtensions`
- Anforderung für Client-Zertifikat mit Angabe der unterstützten Signatur-Algorithmen, falls dessen Authentifizierung gewünscht wird (`CertificateRequest`)

- Ein Server-Zertifikat, das mit einem vom Client unterstützten Signatur-Algorithmus signiert wurde
- Eine Signatur des gesamten Handshakes mit dem vom Client unterstützten Signatur-Algorithmus und dem zum öffentlichen Schlüssel im Server-Zertifikat dazugehörigen privaten Schlüssel (`CertificateVerify`)
- *Message Authentication Code* (MAC) des gesamten Handshakes in Form einer Finished-Nachricht

Während der `key_share` für den Schlüsselaustausch noch im Klartext versendet wird, verschlüsselt der Server alle nachfolgenden Erweiterungen mit einem *secret key* (Handshake-Schlüssel). Dieser *secret key* wird auf der Basis eines gemeinsamen *secret value*, das aus den `key_share` des Clients und des Servers berechnet wurde, in mehreren Durchläufen abgeleitet. In weiteren Ableitungsdurchläufen wird ebenfalls auf der Grundlage eines anderen *secret value* der *secret key* für die Daten generiert (Traffic-Schlüssel). Mit diesem *secret key* darf der Server, auch wenn der Handshake zu diesem Zeitpunkt noch nicht abgeschlossen ist, erste Daten versenden. In Abschnitt 2.2.2 werden die Erzeugung aller *secret values* und *secret keys*, die für die Kommunikation benötigt werden näher erläutert.

Im letzten Schritt des *handshake protocols* leitet der Client nach Empfang des ServerHello seinen Handshake-Schlüssel ab und verifiziert anhand des `CertificateVerify`, ob der Server in Besitz des privaten Schlüssels ist, der zum Zertifikat passt. Wenn der Server mit einem `CertificateRequest` ein Zertifikat vom Client angefordert hat, so sendet dieser wie auch der Server zuvor sein eigenes Zertifikat und eine Signatur über den gesamten Handshake. Zuletzt schickt der Client auch eine Finished-Nachricht und leitet einen Traffic-Schlüssel zum verschlüsselten Datenversand ab.

2.2.2 Secret- und Schlüsselerzeugung in TLS

Sobald der Server ein ClientHello und der Client ein ServerHello empfangen hat, teilen beide Hosts ein *(EC)DHE Shared Secret*, das aus dem eigenen `key_share` und dem `key_share` des Kommunikationspartners berechnet wurde. Es liegt zunächst nahe, diesen *secret value* als *secret key* für die nachfolgende Kommunikation zu verwenden. Jedoch kann der *secret value* in einem nicht uniformen Wertebereich erzeugt worden sein, was einen möglichen Angreifer einen gewissen Vorteil in der Kompromittierung der Kommunikation verschaffen kann. Um dieses Risiko zu vermindern, werden *key derivation*

functions (KDF) verwendet [20]. Diese nehmen ein *secret value* als *input keying material* (IKM) entgegen und leiten daraus kryptographisch starke *secret keys* ab. TLS 1.3 verwendet für diesen Zweck eine auf HMAC basierende KDF (HKDF). Die Ableitung eines *secret keys* aus einem gegebenen *secret value* wird in HKDF in zwei Phasen unterteilt:

1. **HKDF-Extract(salt, IKM) = PRK**
   Vor dem Extract wird eine Hashfunktion für die HMAC-Berechnung festgelegt. Als Argumente nimmt die Funktion einen nicht notwendigerweise geheimen Wert (*salt*) und ein *secret value* als IKM an. Daraus wird ein *pseudorandom key* (PRK) mit der Länge der festgelegten Hashfunktion berechnet.

2. **HKDF-Expand(PRK, info, length) = OKM**
   Auch im Expand wird eine Hashfunktion für die HMAC-Berechnung festgelegt. Die Argumente bestehen aus einem PRK, der aus einem vorhergehenden Extract stammt, einer *info*, die einen applikationsspezifischen Wert darstellt und der gewünschten Länge des *output keying material* (OKM) in *length*.

In der Abbildung 2.2 ist der Ableitungsprozess der *Handshake* und *Traffic Secrets* und der *secret keys* abgebildet. Zu Beginn des Schlüsselaustauschs mit (EC)DHE steht kein PSK

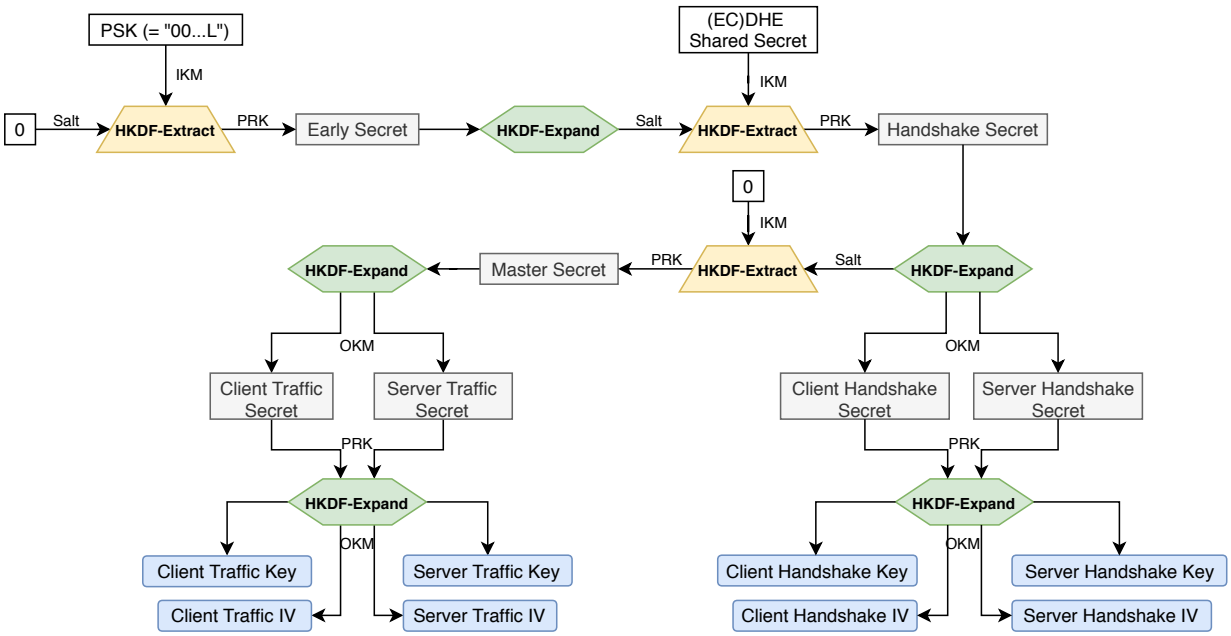


Abbildung 2.2: Ableitung von *secret values*, *keys* und IVs in TLS 1.3 mit (EC)DHE

zur Verfügung, weshalb als IKM ein PSK mit einem aus 0 bestehenden String der Länge

L (= Länge Hashfunktion) verwendet wird. Der *salt* ist hier auf 0 festgesetzt. Nach dem HKDF-Extract steht ein *Early Secret* bereit, das als PRK in ein Expand übergeben wird, um im nachfolgenden Extract als *salt* bereitzustehen. Zusammen mit dem ausgetauschten *(EC)DHE Shared Secret* wird aus dem *salt* ein *Handshake Secret* generiert. Im zweiten HKDF-Expand wird unter Verwendung unterschiedlicher vorher definierten *info*-Werten aus dem *Handshake Secret* ein *salt* und für jeweils eine Kommunikationsrichtung ein *Client Handshake Secret* und *Server Handshake Secret* erstellt. Die Erzeugung des *Master Secret* erfordert in der letzten Extract-Phase den vorher bestimmten *salt* mit einem IKM von 0. Dieses wird ebenfalls mit unterschiedlichen *info*-Werten in ein *Client Traffic Secret* und ein *Server Traffic Secret* abgeleitet. Zur Berechnung der endgültigen *secret keys* zur Verschlüsselung des Handshakes und des Datenverkehrs bedarf es noch für jeden *secret value* eines abschließenden HKDF-Expands, das die *secret keys* erzeugt. Zusätzlich werden Initialisierungsvektoren (IV) generiert, die der Verschlüsselung als Eingabeparameter dienen.

2.2.3 Authenticated Encryption with Associated Data (AEAD)

Das ClientHello und ServerHello beinhalten die Option, eine Liste von `cipher_suites` anzugeben. Bei dieser Liste handelt es sich im Detail um eine Angabe der unterstützten AEAD-Algorithmen des jeweiligen Hosts. Die Besonderheit an AEAD ist, dass es die Notwendigkeit von kryptografischen Anwendungen nach Vertraulichkeit und Nachrichtenauthentizität in einem Konzept vereint [21]. AEAD stellt Funktionen bereit, die einen symmetrischen Verschlüsselungsalgorithmus mit einem MAC kombinieren. Der Vorteil dieser festen Kombination ist, dass Sicherheitsrisiken durch eine falsche Auswahl oder Durchführungsreihenfolge der Verschlüsselungs- und Authentifikations-Algorithmen reduziert werden. Des Weiteren können die Entwickler von kryptografischen Algorithmen durch die Vereinigung spezielle Optimierungen vornehmen, die zuvor bei einer getrennten Auswahl nicht möglich gewesen wären. Ein Beispiel für eine AEAD-Funktion ist `AEAD_AES_128_GCM`. Diese Funktion verwendet den Galois/Counter Mode (GCM), der auf dem symmetrischen Verschlüsselungsalgorithmus Advanced Encryption Standard (AES) mit 128-Bit-Schlüssel basiert. Die Authentizität der Nachricht wird durch die Verwendung einer „universellen Hashfunktion“ im GCM gewährleistet [22].

Als Parameter nimmt die AEAD-Funktion einen *secret key*, Nonce, die zu verschlüsselnden/entschlüsselnden Daten und zuletzt *Associated Data* (AD) entgegen. Dabei ist gefor-

dert, dass der *secret key* aus einem uniformen Wertebereich stammt. Dies wird durch die Nutzung der HKDF erzielt, die im vorangegangenen Abschnitt 2.2.2 behandelt wurde.

Die AD stellen Informationen dar, die im Klartext übertragen werden und dabei authentifiziert sein sollen. Diese Daten sind häufig Header-Informationen von Paketen (z.B. Adressen, Ports), die für die Verarbeitung unverschlüsselt sein müssen, deren Authentizität beim Empfänger dennoch von Relevanz ist.

2.2.4 Record protocol

Sobald der Server gegenüber dem Client authentifiziert ist und die Traffic-Schlüssel durch das *handshake protocol* zwischen dem Client und dem Server bereitstehen, übernimmt das *record protocol* den weiteren Ablauf der Kommunikation. Die zu übertragenden Daten werden entgegengenommen und in sogenannte *records* unterteilt. Jeder *record* wird anschließend verschlüsselt und an den Kommunikationsteilnehmer gesendet. Wenn ein Host TLS-Daten empfängt, dann werden diese verifiziert, entschlüsselt, wieder zusammengesetzt und an die darüberliegende Applikation übergeben.

2.3 Verschlüsselung in QUIC

Für die Beschreibung der Funktionsweise der verschlüsselten Datenübertragung hat die QUIC WG einen zusätzlichen Internet-Draft bereitgestellt, der als Vorgabe für Implementierungen des QUIC-Protokolls dient [12]. In diesem Dokument wird ein Schutz von Paketen vorausgesetzt, welcher die Vertraulichkeit und Integrität der Pakete sicherstellt. Diese Anforderungen werden durch den Einsatz von TLS 1.3 erfüllt. Die Forderung des TLS-Protokolls nach einem zuverlässigen und in Reihenfolge übertragenden Transportprotokoll (siehe Abschnitt 2.2) kann QUIC erfüllen, wodurch zwischen den Protokollen eine wechselseitige Abhängigkeit herrscht.

Die grundlegenden Interaktionen zwischen QUIC und TLS 1.3 sind in der nachfolgenden Abbildung zu sehen:

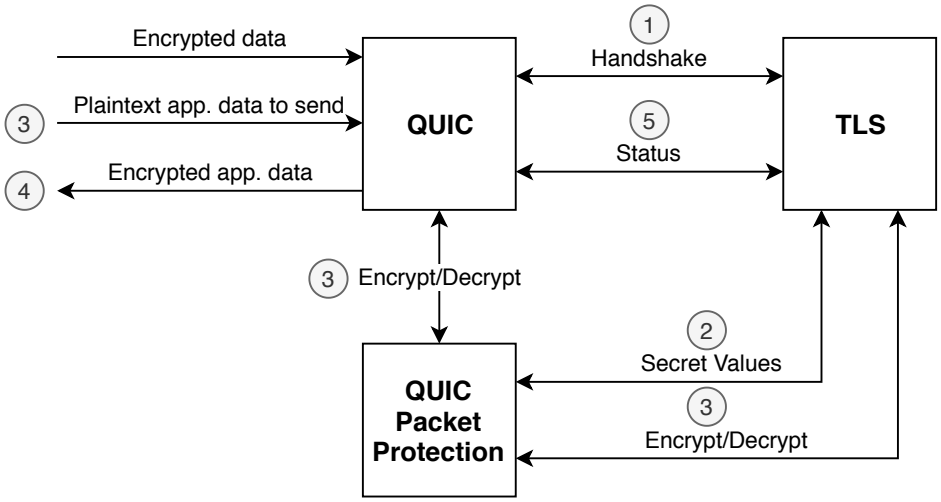


Abbildung 2.3: Interaktion zwischen QUIC, QPP und TLS

Während eines Verbindungsaufbaus werden TLS-Handshake-Pakete (ClientHello, ServerHello, Finished) von QUIC bei TLS angefordert oder bei Empfang an TLS ausgeliefert (1). Nach Abschluss des Handshakes stellt TLS zur Berechnung der Traffic-Schlüssel *secret values* bereit, die in die QUIC Packet Protection (QPP) exportiert werden (2). Diese Komponente führt auf der Grundlage der Verschlüsselungsfunktionen von TLS die Ver- und Entschlüsselung von Paketen durch und steht damit eng mit TLS in Beziehung. Sobald QUIC Daten im Klartext von der Applikation zum Versand erhält, werden diese zur Verschlüsselung an die QPP weitergeleitet, dort verschlüsselt (3) und letztlich durch QUIC zum Zielhost übertragen (4). Eingehende verschlüsselte Daten nehmen analog dazu den selben Weg über die QPP und stehen am Ende entschlüsselt zur weiteren Verarbeitung für QUIC bereit. Treten im Verbindungsverlauf Ereignisse über mögliche Fehler oder über das Vorhandensein von Schlüsselinformationen auf, so werden diese zwischen QUIC und TLS ausgetauscht (5).

2.3.1 Ablauf eines 1-RTT-Handshakes

Durch die Integration der TLS-Handshake-Nachrichten in den Handshake von QUIC kann eine Datenübertragung bereits nach einer RTT erfolgen. Dafür erstellt der Client im ersten Schritt ein *Initial Packet*, dass im Long-Header-Format von QUIC aufgebaut ist. In diesem wird der Typ des Pakets, eine Connection ID zur Identifizierung der Verbindung, die verwendete QUIC-Version und eine Paketnummer gesetzt. Danach erstellt der Client

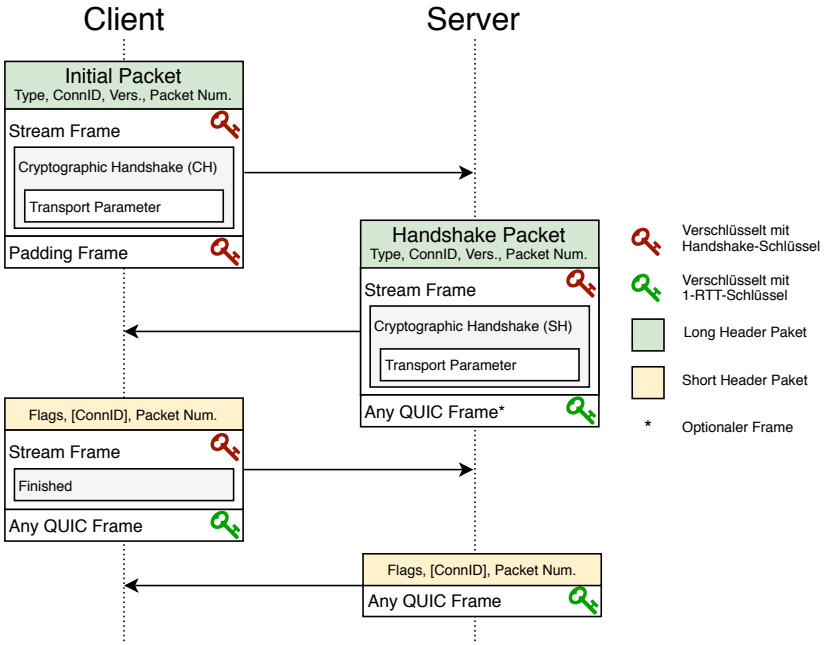


Abbildung 2.4: 1-RTT-Handshake in QUIC zwischen Client und Server

einen Stream Frame für den Stream 0. Stream Frames werden zum Versand von Daten verwendet und adressieren jeweils einen Stream. Wenn QUIC Stream Frames an den Stream 0 versendet, so sind diese für die TLS-Verbindung bestimmt und werden beim Empfang nicht an die Applikation weitergeleitet. In diesem Frame verpackt der Client das ClientHello (CH), das zur Initiierung einer TLS-Verbindung benötigt wird (siehe Abschnitt 2.2.1). Dem ClientHello wird zuvor eine QUIC-eigene Erweiterung hinzugefügt, die Parameter für die QUIC-Verbindung enthält (*Transport Parameter*). Abschließend wird das *Initial Packet* mit einem Padding Frame aufgefüllt, sodass das Paket eine Größe von mindestens 1200 Byte aufweist. Diese Frames dienen ausschließlich zum Auffüllen von Paketen und sind für die Verbindung semantisch irrelevant. Die Frames des fertiggestellten Pakets werden mit einem Handshake-Schlüssel verschlüsselt (dessen Erzeugung wird in Abschnitt 2.3.2 behandelt) und zum Server versendet. Die zugrundeliegende AEAD-Funktion ist in dieser Phase mit `AEAD_AES_128_GCM` fest vorgegeben.

Sobald ein *Initial Packet* beim Server mit an Stream 0 adressierten Stream Frames eintrifft, liefert QUIC die Daten nach der Entschlüsselung durch die QPP direkt an TLS weiter und fordert das dazugehörige ServerHello (SH) an. Die erhaltenen *Transport Parameter* des Clients werden validiert und gespeichert. Im Anschluss daran erzeugt der Server als Antwort ein *Handshake Packet*, das wie auch das *Initial Packet* im Long-

Header-Format steht. Das angeforderte ServerHello wird um die *Transport Parameter* des Servers erweitert und in ein Stream Frame für den Stream 0 eingefügt. Es erfolgt eine Verschlüsselung des Frames mit dem Handshake-Schlüssel und der Versand der Antwort an den Client. Zu diesem Zeitpunkt ist der Server in Besitz der Schlüsselinformationen des Clients und seiner eigenen, woraus der 1-RTT-Schlüssel zur Verschlüsselung von ersten Applikationsdaten abgeleitet werden kann (dessen Erzeugung wird in Abschnitt 2.3.2 behandelt). Es steht dem Server frei, bereits jetzt erste Daten an den Client zu senden. Die hierfür verwendete AEAD-Funktion ist dabei die selbe, die durch den TLS-Handshake ausgehandelt wurde.

Das ServerHello des *Handskake Packet* wird entschlüsselt an die TLS-Komponente des Clients ausgeliefert und die finale Finished-Nachricht angefordert. Auch der Client validiert die *Transport Parameter* des Servers und speichert diese zwischen. Der Stream Frame der Finished-Nachricht wird noch mit dem Handshake-Schlüssel verschlüsselt, während für alle nachfolgenden Frames der zwischen dem Client und Server vereinbarte 1-RTT-Schlüssel verwendet wird. Das Paket basiert ab diesem Zeitpunkt auf dem Short-Header-Format, das im Unterschied zum Long-Header-Format verschiedene Flags anstelle des Typs besitzt, die Version auslässt und optional die Connection ID weglassen kann, sofern es in den *Transport Parameter* von beiden Hosts angegeben wurde.

2.3.2 Secret- und Schlüsselerzeugung in QUIC

Die Erzeugung der *secret keys* für den Handshake und der nachfolgenden Datenübertragung basiert, wie auch in TLS 1.3, auf HKDF mit dem Extract-Expand-Verfahren (siehe Abschnitt 2.2.2). QUIC definiert eine eigene HKDF-Expand-Funktion (QHKDF-Expand), die eine andere Struktur des info-Arguments aufweist. Der Aufbau sieht wie folgt aus:

```
QHKDF-Expand(Secret, Label, Length) = HKDF-Expand(Secret, QhkdfExpandInfo,
    Length)
```

Beim `QhkdfExpandInfo` handelt es sich um ein *struct*, das über ein Attribut für ein Label und ein weiteres Attribut für dessen Länge enthält. Als zugrundeliegende Hashfunktion verwendet QUIC dieselbe wie TLS für die eigene Schlüsselerzeugung.

Zur Verschlüsselung der Handshake-Pakete von QUIC wird aus der Connection ID des *Initial Paket* und einem im Internet-Draft definierten *Handshake Salt* ein *Handshake Secret* gebildet, auf dessen Grundlage unter Verwendung unterschiedlicher Label *Handshake*

Secrets sowohl für die Client- als auch Server-Seite erzeugt werden (siehe Abbildung 2.5). Im letzten QHKDF-Expand-Schritt entstehen die *secret keys* und IVs für den Handshake.

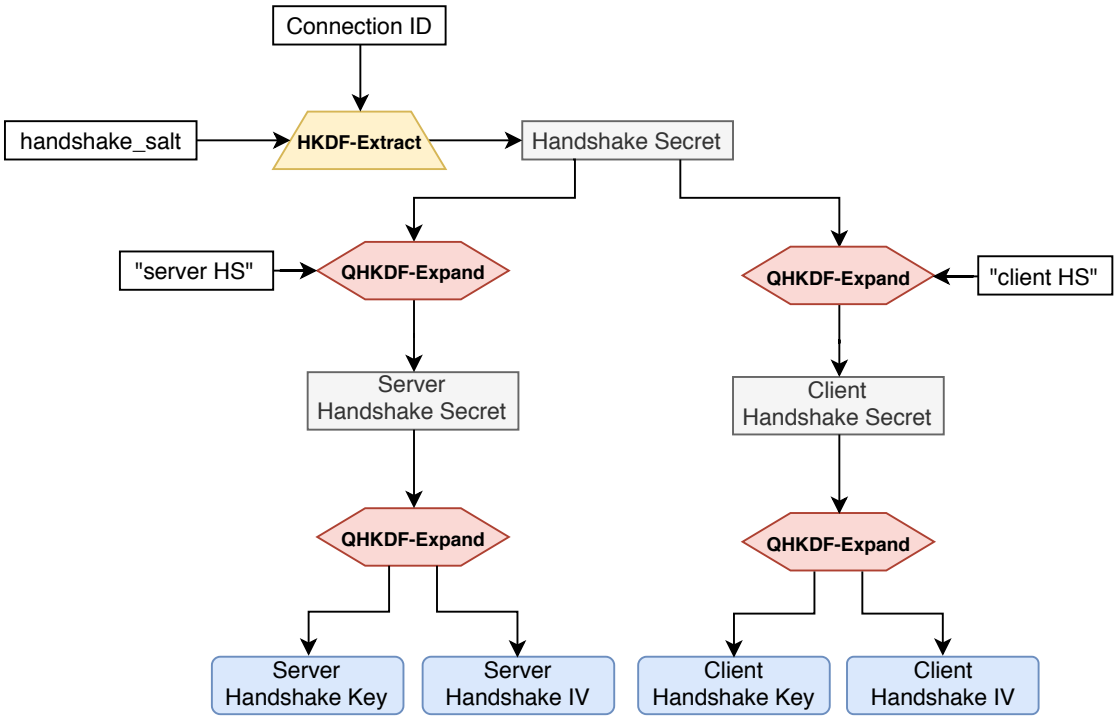


Abbildung 2.5: Ableitung von *secret values*, *keys* und IVs für den QUIC-Handshake

Nachdem ein Host die Schlüsselinformationen des anderen empfangen und der eigenen TLS-Komponente zur Verarbeitung weitergeleitet hat, stehen die für die Berechnung der 1-RTT-Schlüssel notwendigen *secret values* zur Verfügung. Mithilfe eines TLS-Exporters können die benötigten *secret values* unter Angabe des entsprechenden Labels zur QPP exportiert werden. Ein abschließendes QHKDF-Expand stellt die *secret keys* und IVs am Ende des 1-RTT-Handshakes her. Die Abbildung 2.6 veranschaulicht die Erzeugung.

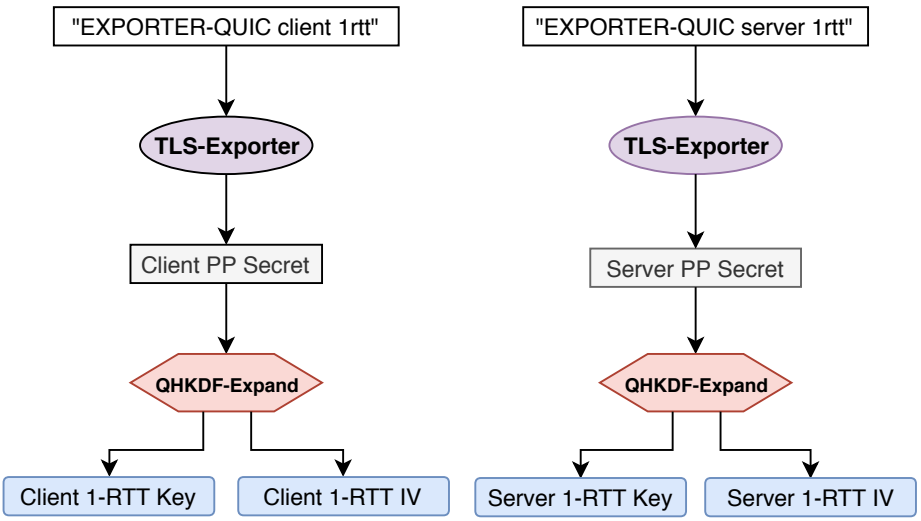


Abbildung 2.6: Abgeleitete *secret values*, *keys* und IVs für die QUIC-Datenübertragung

2.4 OMNeT++- und INET-Framework

Bei OMNeT++ [23] handelt es sich um ein ereignisorientiertes Simulationsframework für Netzwerke, das die Entwicklung eigener Simulationsmodelle in der Programmiersprache C++ ermöglicht. Es unterstützt die Modellierung von kabelgebundenen und ungebundenen Netzwerken, Queues, Protokollen und allen anderen Systemen, die sich mit einer ereignisorientierten Simulation darstellen lassen. Die Entwicklung erfolgt in einer IDE, die auf Eclipse basiert. Für die Entwicklung der Simulationsmodelle stellt OMNeT++ Strukturen bereit, die nachfolgend vorgestellt werden:

Module

In OMNeT++ erstellte Modelle bestehen aus mehreren zusammengesetzten Modulen. Bei den Modulen wird zwischen *simple modules* und *compound modules* unterschieden. *Simple modules* stellen eigenständige Einheiten dar, die miteinander kommunizieren können, während *compound modules* als Container dienen und mehrere *simple modules* zu einer Einheit gruppieren. Jedes *simple module* verfügt für die Nachrichtenkommunikation der Module untereinander über sogenannte *Gates*. Die *Gates* stellen die Schnittstelle der *simple modules* dar und können für den Empfang von Nachrichten (*Input Gate*) oder als Sendeausgang der eigenen Nachrichten (*Output Gate*) verwendet werden. Ebenfalls ist eine bidirektionale Nutzung möglich (*Inout Gate*). Neben der Schnittstellendefinition ist es auch erforderlich, die *Gates* der *simple modules* untereinander zu verbinden, wofür *Connections* zum Einsatz kommen. Dazu wird ein *Input/Inout Gate* eines *simple*

module über eine *Connection* mit dem *Output/Inout Gate* eines anderen verbunden. An dieser Stelle können jeder *Connection* bestimmte Eigenschaften wie z.B. eine maximale Durchsatzrate oder eine Verzögerung bei der Datenübertragung zugewiesen werden.

Nachrichten
Die Übermittlung von Nachrichten stellt im Internet als auch in OMNeT++ eine zentrale Methode zur Kommunikation dar. Für den Austausch von Daten werden Netzwerkpakete in Form von *cPackets* zwischen den *simple modules* verschickt. Zur besseren Identifizierung während der Simulation ist es möglich, den Paketen Namen zuzuweisen. Mit dem Setzen eines *message kinds* kann jedem Paket eine numerische Konstante zugewiesen werden, die bei Empfang den Zweck von diesem Paket beschreibt.

Die Kommunikation über ein für alle Beteiligten verständliches Protokoll erfordert die Einhaltung eines vordefinierten Formats. OMNeT++ stellt hierfür eine eigene Syntax bereit, die es dem Entwickler erlaubt, Paketformate für ein Protokoll zu definieren. Die Definition wird in Dateien mit der Endung *.msg* geschrieben und von einem *message compiler* in eine entsprechende C++-Definition umgewandelt.

Netzwerkbeschreibung
Die Beschreibung konkreter Modelle zur Simulation wird in der Spache NED (*Network Description*) von OMNeT++ realisiert. In dieser werden die zuvor auch in den NED-Dateien definierten Module deklariert und über *Connections* mit anderen Modulen zu einem Netzwerk verbunden. Über eine Konfigurationsdatei (*omnet.ini*) werden den Modulen Parameter zugewiesen, die das Verhalten des Modells anpassen.

Das INET-Framework [24] stellt eine Open-Source-Bibliothek für OMNeT++ zur Verfügung. In dieser ist eine Vielzahl an Netzwerkprotokollen wie IPv4, TCP, UDP usw. enthalten, auf deren Basis eigene Simulationsmodelle erstellt werden können. Eine Implementierung des QUIC-Protokolls ist bis zum Zeitpunkt dieser Ausarbeitung nicht vorhanden.

2.5 Verwandte Arbeiten

Bisher wurde in dieser Arbeit die vereinfachende Annahme getroffen, dass nur ein QUIC-Protokoll existiert. Jedoch teilt sich das Protokoll derzeit in zwei unterschiedliche Stränge auf. Zum einen existiert die ursprüngliche Version von QUIC, die durch Google entwickelt worden ist und auf zahlreichen Servern von Google bereits Anwendung findet (gQUIC).

Zum anderen hat sich mit Beginn der Standardisierung von QUIC bei der IETF eine Spezifikation herausgebildet (iQUIC), die einige Unterschiede zur Implementierung von Google aufweist. So verfolgt das iQUIC einen modularen Aufbau von QUIC, während das gQUIC monolithisch geprägt ist [15]. Bei der Verschlüsselung wurde beim gQUIC auf einen selbst entwickelten kryptografischen Unterbau gesetzt (*QUIC Crypto Protocol*), da zu diesem Zeitpunkt keine geeignete Lösung zur Verschlüsselung bereitstand, die die Features von QUIC unterstützt. Im Jahr 2016 wurde angekündigt, das eigene *QUIC Crypto Protocol* zugunsten des formal spezifizierten Verschlüsselungsprotokolls TLS 1.3 aufzugeben [25].

Sofern nicht direkt mit „gQUIC“ oder „iQUIC“ auf eine spezielle QUIC-Art Bezug genommen wird, soll mit „QUIC“ im weiteren Verlauf der Arbeit immer das iQUIC der IETF gemeint sein.

Durch das Chromium-Projekt[3] stellte Google bereits früh erste Implementierungen seines gQUIC öffentlich zur Verfügung. Deshalb beziehen sich viele Forschungsarbeiten auf diese frühen Entwicklungen, während das iQUIC zu diesen Zeitpunkten nicht vollständig spezifiziert war und keine reale Implementierung aufweisen konnte.

Die Forschungsarbeiten [26] und [27] zum gQUIC haben sich mit einer Performanceanalyse des Protokolls unter verschiedenen Rahmenbedingungen beschäftigt. Es hat sich in den Analysen gezeigt, dass das gQUIC in Testszenarien mit vielen HTTP-Ressourcen und ab einer Bandbreite von 50 Mbit/s gegenüber TCP schlechter abschneidet. Bei einer kleineren Anzahl an Ressourcen und einer geringeren Bandbreite stellt sich gQUIC als besser heraus. Die Experimente erfolgten in diesen Forschungsarbeiten im Internet. Der im Internet herrschende *Quality of Service* (QoS) ist *Best Effort*, was bedeutet, dass keine Garantien über die Übertragungsdauer, die gesicherte Paketreihenfolge oder überhaupt über die Ankunft von Paketen ohne Verluste gemacht werden können. Eine Reproduzierbarkeit derselben Ergebnisse ist damit nicht vollständig gegeben. In der Forschungsarbeit [28] aus dem Jahr 2018 wurde dagegen eine simulierte Performanceanalyse auf Basis des Caddy-Webservers durchgeführt, das quic-go[4] (iQUIC) als Grundlage verwendet. Hier ist in einer Gegenüberstellung mit TCP herausgekommen, dass TCP beim Aufruf der bereitgestellten Testwebseiten unter verschiedenen Parametern in 60 % aller Testfälle besser abschneidet als iQUIC.

[3] https://www.chromium.org/quic
[4] https://github.com/lucas-clemente/quic-go

Unter dem Gesichtspunkt der Sicherheit haben sich die Arbeiten [29]–[31] mit dem gQUIC beschäftigt. Jedoch beziehen sich die Arbeiten noch auf das *QUIC Crypto Protocol*, das durch TLS 1.3 ersetzt wird und zukünftig keine Relevanz mehr hat. Aus diesem Grund werden die Inhalte dieser Arbeiten hier nicht ausgeführt.

Im Rahmen der Konferenz ACM CoNEXT 2018 wurde die Verschlüsselungslösung nQUIC vorgestellt. Dabei handelt es sich um eine angepasste Variante des iQUIC, die auf Grundlage des *Noise Protocol Framework* die Verschlüsselung durchführt (Name nQUIC abgeleitet aus QUIC-Noise) [32]. Dieses Framework definiert mehrere *handshake pattern*, von denen das nQUIC das IK-Pattern als Grundlage verwendet. Die Verwendung dieses Patterns führt die Einschränkung mit sich, dass das Zertifikat des Servers nicht als Teil des Handshakes zum Client versendet wird, sondern diesem schon vorher bekannt sein muss (*Certificate Pinning*). Im Gegenzug entfällt die Notwendigkeit einer Public-Key-Infrastruktur (PKI). Der damit einhergehende Wegfall der *Certificate Authorities* (CA) hat den Vorteil, dass keine Kompromittierung einer CA stattfinden kann, die die ganze Zertifizierungs-Kette gefährden würde.
Die Autoren sehen das nQUIC in Situationen, in denen die Verwendung der spezifizierten iQUIC-Verschlüsselung zu kostenintensiv oder ungeeignet ist, als einen brauchbaren Ersatz. Im Umfeld des traditionellen Webs wird jedoch von der Nutzung von nQUIC abgeraten, da nQUIC nicht die notwendige Interoperabilität und kryptografische Agilität mitbringen würde. In einem Performancetest wurde die Zeit des Verbindungsaufbaus gemessen. Dazu wurden die iQUIC-Implementierungen Quinn[5] und quic-go[4] mit entsprechenden auf nQUIC angepassten Versionen dieser Implementierungen gegenübergestellt. Das Resultat war, dass zwischen den Quinn-Implementierungen die Version mit nQUIC eine marginale Verbesserung von 0,0006 ms erzielen konnte, während im Szenario mit quic-go die Verbesserung ≈0,019 ms betrug. Es muss hierbei angemerkt werden, dass die iQUIC-Implementierungen mit der Hashfunktion SHA-384 gearbeitet haben, während bei den auf nQUIC angepassten Versionen SHA-256 verwendet wurde. Dieser Unterschied kann einen Einfluss auf das Ergebnis haben, da die verschiedenen Hashfunktionen je nach Eingabegröße und verwendeter Hardware über unterschiedliche Berechnungsaufwände verfügen können [33]. Auch sind die genauen Parameter der Experimentdurchführung nicht bekannt, was ebenfalls das Aussagegewicht des ohnehin schon marginalen positiven Ergebnisses mindert.

Es bleibt festzuhalten, dass relativ zu der Anzahl an Forschungsarbeiten zum gQUIC noch nicht allzu viele Arbeiten zum iQUIC existieren. Jedoch ist mit dem Aufkommen

[5] https://github.com/djc/quinn

neuer iQUIC-Implementierungen [34] und der nahenden finalen Standardisierung damit zu rechnen, dass in Zukunft mehr Arbeiten Bezug zum iQUIC nehmen werden. Die hier vorgestellten Performanceanalysen zeigen, dass QUIC trotz der neuen Features und dem Fokus auf die Performance nicht in allen Situationen einen klaren Vorteil gegenüber TCP erzielen kann. Der Inhalt der Sicherheitsanalysen ist aus heutiger Sicht veraltet und nicht mehr relevant. Einzig [32] stellt eine erste Analyse der iQUIC-Verschlüsselung an und vergleicht diese mit der Eigenentwicklung nQUIC. Die dort hervorgebrachten Ergebnisse weisen jedoch aufgrund der Verwendung unterschiedlicher Hashfunktionen und den marginalen Unterschieden der hervorgebrachten Ergebnisse eine geringe Aussagekraft auf.

Mit den hier vorgestellten Forschungsarbeiten wird deutlich, dass derzeit eine Lücke bei der Analyse des iQUIC und dessen Verschlüsselung herrscht. Die vorliegende Masterarbeit nimmt sich diesem Umstand an und schließt mit der Durchführung einer Vergleichsstudie der eingangs erwähnten Verschlüsselungsansätze (QUIC-mit-TLS, QUIC-über-TLS; siehe Kapitel 1) auch eine Analyse der iQUIC-Verschlüsselung ein, die aussagekräftige Ergebnisse hervorbringen soll.

3 QUIC-Simulationsmodell

Die Entwicklung eines Simulationsmodells von QUIC wurde an der HAW Hamburg initiiert und maßgeblich durch den Autor dieser Arbeit vorangetrieben. Dieses Modell dient als Grundlage für die Durchführung der Vergleichsstudie. Zu Beginn werden die vorbereitenden Prozesse zur Entwicklung des Modells erläutert, die für die Realisierung eines Protokolls dieses Umfangs unerlässlich sind. An diesen Teil schließt eine Beschreibung des aktuellen Modells mit den bereits implementierten Features an. Der letzte Abschnitt befasst sich mit der Erweiterung des Simulationsmodells und geht dabei auf die Konzeption, Entwicklung und Verifikation der Verschlüsselungsansätze QUIC-mit-TLS und QUIC-über-TLS ein.

3.1 Definition eines Entwicklungsprozesses

Ein Transportprotokoll von Grund auf für eine dafür noch nicht zuvor berücksichtigte Umgebung zu entwickeln, stellt eine Herausforderung dar, die eine sorgfältige Planung erfordert. Mit dem Beginn der Entwicklung des Simulationsmodells von QUIC haben Realimplementierung des iQUIC wie das ngtcp2 oder mozQUIC bereits existiert, jedoch können diese nicht direkt in die Simulationsumgebung überführt werden, weshalb nur eine Neuentwicklung in Betracht kam. In Zusammenarbeit zwischen einigen Mitgliedern der CaDS-Projektgruppe[1] an der HAW Hamburg und Mitgliedern der Universität Essen-Duisburg wurde ein Modell von QUIC entwickelt, das bereits Gegenstand von Projektarbeiten des Masterstudiums und von Bachelorarbeiten war. Die vorbereitenden Schritte zur Initiierung der Entwicklung werden im Folgendem beschrieben:

Auswahl eines Vorgehensmodells
Um den Ablauf der Entwicklung des Modells (auch Workflow genannt [35]) in einer

[1] https://cads.informatik.haw-hamburg.de/

Arbeitsgruppe zu koordinieren, ist die Auswahl eines Vorgehensmodells notwendig, das der vorherrschenden Entwicklungssituation entspricht, die sich wie folgt darstellt:

- Die Arbeitsgruppe des Simulationsmodells ist über zwei Standorte verteilt, weshalb persönliche Treffen schwer zu realisieren sind.
- Eine stetige Fluktuation von Mitgliedern im Verlauf des Projekts ist nicht auszuschließen und darf dabei den Fortschritt der Entwicklung nicht wesentlich beeinträchtigen. Es sollte daher neuen Mitgliedern eine einfache Einarbeitung in das Vorgehensmodell ermöglicht werden, damit sie schnell an der Entwicklung teilnehmen können.

Unter der Berücksichtigung dieser Punkte wurde Kanban [36] als Vorgehensmodell ausgewählt. In einem Kanban-Board werden die bevorstehenden, in Arbeit befindlichen und beendeten Aufgaben dokumentiert. Die Vorteile von Kanban sind der geringe Verwaltungsaufwand nach dem erstmaligen Aufsetzen, die schnelle Einarbeitungszeit von neuen Mitgliedern und die Integration in Github[2]. Insbesondere der letzte Punkt vereinfacht die Verwaltung, da sowohl der Projektcode als auch der Entwicklungsprozess über Github verwaltet werden kann.

Bestimmung einer Entwicklungsstruktur
Da das QUIC-Protokoll bis zu seiner Standardisierung ständigen Änderungen unterliegt, wurde innerhalb der Arbeitsgruppe festgelegt, dass der Internet-Draft in der Version 10 [37] bis zur Fertigstellung als Implementierungsgrundlage dienen soll. Der Internet-Draft beschreibt QUIC als ein Konstrukt zahlreicher Features, die in sich abgeschlossene Einheiten bilden können. Dieser Aufbau legt die Verwendung einer feature-orientierten Entwicklung nahe, die ein Softwaresystem in seine Features zerlegt und anhand dieser Zerlegung der Entwicklung eine Struktur vorgibt [38].

Machbarkeits-, Notwendigkeitsanalyse
In einer Machbarkeits-, Notwendigkeitsanalyse wurde untersucht, welche Features von QUIC im Umfeld der Simulationsumgebung für die Kernfunktionalität erforderlich sind. Dabei stellte sich heraus, dass einige Funktionen wie das Error-Handling, die *Proof of Source Address Ownership* oder die *Stateless Retries* auf einen späteren Zeitpunkt der Entwicklung verschoben werden können, da sie auf die Kernfunktion des Protokolls keinen Einfluss haben. Die Implementierung einer einzigen Version von QUIC in OMNeT++

[2] https://help.github.com/articles/about-project-boards/ – Abgerufen am 14.02.2019

macht die Verhandlung zwischen zwei Hosts über eine gemeinsame QUIC-Version (*Version Negotiation*) überflüssig und wird deshalb nicht im Modell berücksichtigt.

Git-Workflow
Mit Git steht allen Entwicklern ein Werkzeug zur Verfügung, mit dem sich parallele Entwicklungen an QUIC koordinieren lassen. Die Verwendung der feature-orientieren Entwicklung unterstützt hier den Einsatz eines Git-Branching-Modells, das auf die Feature-Entwicklung ausgerichtet ist. Nach dem Vorbild des von Vincent Driessens vorgestellten Git-Branching-Modells [39] wurde ein einfaches Modell bestehend aus einem Master, Develop, Test und mehreren Feature Branchs ausgewählt. Ein Beispielablauf ist die Entwicklung eins Features auf einen eigenen Branch. Nach Fertigstellung des Features erfolgt die Zusammenführung mit dem Develop-Branch. Dies hat den Vorteil, dass durch den Merge-Commit alle Änderungen des Features auf einen Blick zu sehen sind. Hat der Develop-Branch einen stabilen Stand erreicht, so folgt die Überführung in den Test-Branch, der die neu implementieren Features auf die Erfüllung der gestellten Anforderungen überprüft. Wenn die Tests erfolgreich abgeschlossen wurden, dann wird im letzten Schritt der Code mit dem Master-Branch zusammengeführt. Dieses Modell ist einfach umzusetzen und stellt über den mehrstufigen Durchlaufprozess bis zum Master-Branch die Qualität der Entwicklung sicher.

Projektkonsens
Zusätzlich zum Git-Workflow müssen die Entwicklungen im Konsens der Arbeitsgruppe stehen und den implementierungsspezifischen Anforderungen genügen. Das bedeutet, dass die verwendete Methodik bei der Umsetzung der Features hinsichtlich ihrer Auswahl und Implementierung nachvollziehbar sein muss. Auch die Einhaltung der *INET Coding Conventions*[3] muss sichergestellt werden. Um diese Anforderungen zu erfüllen, folgt nach Fertigstellung von Arbeitspaketen ein *Code Review* durch den Projektleiter Prof. Dr. Martin Becke und im Anschluss daran findet ein *Code Review* mit den Mitgliedern statt, in dem die Entwicklung vorgestellt und diskutiert wird.

3.2 Aufbau

Basierend auf dem vorangegangen Entwicklungsprozess ist bisher ein Simulationsmodell des iQUIC entstanden, das folgende Features implementiert: Verbindungsauf- und abbau, Datenübertragung, Stream Multiplexing, Stream Scheduling, Paketverlusterkennung und

[3] https://inet.omnetpp.org/CodingConventions.html – Abgerufen am 14.02.2019

erneute Übertragung von verlorenen Paketen. Die Klassenstruktur für diese Version wurde im folgendem Klassendiagramm vereinfacht dargestellt.

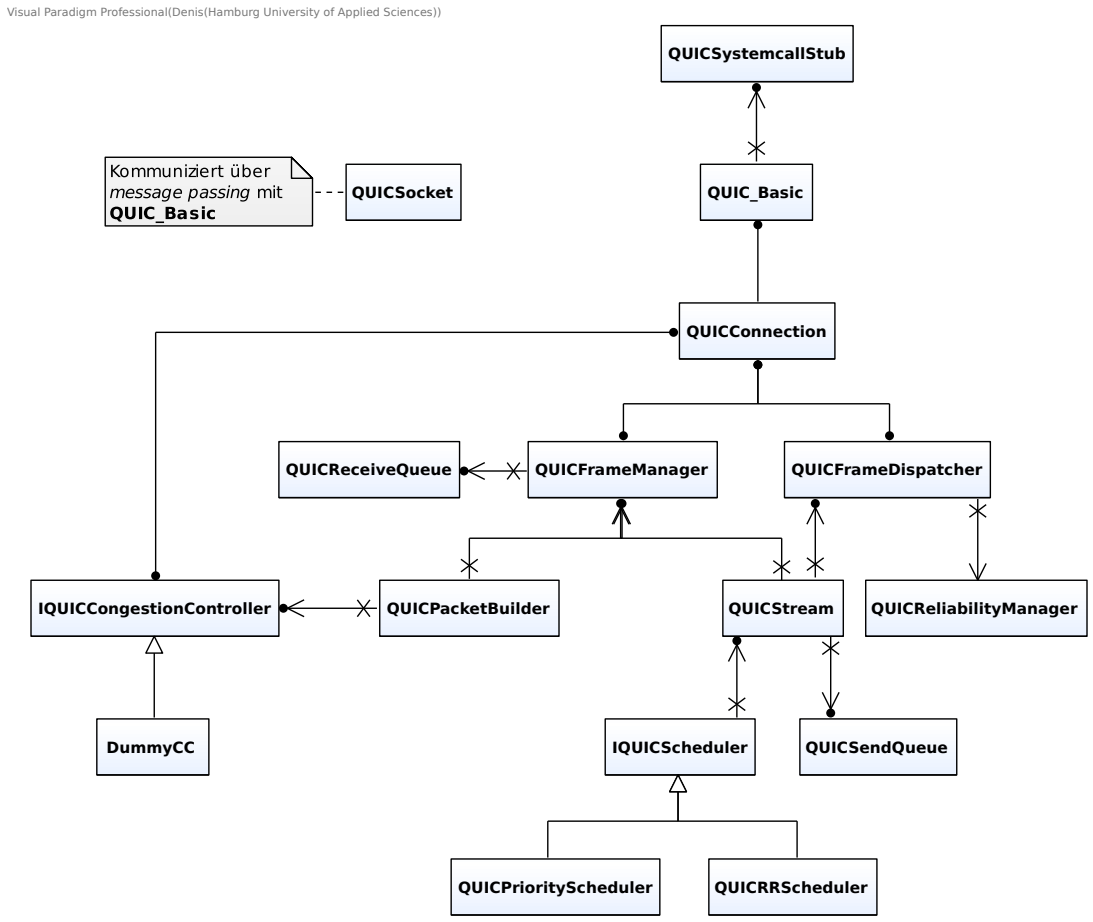


Abbildung 3.1: Vereinfachtes Klassendiagramm vom QUIC-Simulationsmodell

Die zentrale Klasse des Modells stellt `QUIC_Basic` dar. Bei dieser Klasse handelt es sich um ein OMNeT++-Modul, das über Gates zur Kommunikation mit der Applikation als auch über Gates zur darunterliegenden UDP-Schicht verfügt, auf welche QUIC aufbaut. Bei der Erstellung einer Simulation in einer NED-Datei repräsentiert diese Klasse das QUIC-Protokoll und wird über Connections mit einer Applikation und UDP verbunden. Der `QUICSystemCallStub` stellt die sendende Instanz des Modells dar, die auf die UDP-Gates von `QUIC_Basic` zurückgreift und damit die Daten an UDP weiterleitet. Die Verbindung zwischen zwei Hosts wird mit der `QUICConnection` abgebildet. Jede Verbindung wird über eine Connection ID identifiziert und verwaltet den Ein- und Ausgang

von Daten. Werden Daten empfangen, so leitet die `QUICConnection` diese zur weiteren Verarbeitung an den `QUICFrameDispatcher` weiter, der das Paket in seine einzelnen Frames zerlegt und diese durch den darin adressierten `QUICStream` verarbeiten lässt. Über den `QUICReliabilityManager` werden Paketverluste in der Datenübertragung erkannt und für die spätere Benachrichtigung des Kommunikationspartners gesichert.

Beim Versand von Daten führt die Applikation ein *send()* auf dem `QUICSocket` aus. Nach dem Message-Passing-Prinzip sendet der `QUICSocket` eine Nachricht mit den Daten an `QUIC_Basic`. Die `QUICConnection` erstellt mithilfe des `QUICFrameManagers` Frames aus den Daten und speichert diese in der `QUICSendQueue` des angegebenen Streams zwischen. Ein Scheduler (`QUICPriority-` oder `QUICRRScheduler`) wählt aus den Sendqueues der Streams Frames aus und übergibt diese dem `QUICPacketBuilder`, um ein Paket zusammenzustellen. Wenn ein Paket zum Versand bereitsteht, wird es der Congestion Control (`DummyCC`) übergeben, die es letztlich über den `QUICSystemCall-Stub` zum Host versendet.

3.3 Erweiterung

Nachdem im Abschnitt 3.2 der Aufbau des bestehenden Modells beschrieben wurde, geht es in diesem Abschnitt darum, das Modell soweit zu erweitern, dass eine Vergleichsstudie der Ansätze QUIC-mit-TLS und QUIC-über-TLS durchführbar wird. Dazu muss im ersten Schritt eine geeignete TLS-Bibliothek ausgewählt werden, die als Basis für die Verschlüsselungsansätze dienen soll. Danach folgt eine Beschreibung des Entwicklungsprozesses beider Verschlüsselungsansätze, der sich nach Befolgung des klassischen Software Engineerings in die Phasen Anforderungsanalyse, Entwurf, Implementierung und den abschließenden Test unterteilt [40].

3.3.1 Auswahl einer TLS-Bibliothek

Eine geeignete TLS-1.3-Bibliothek für beide Verschlüsselungsansätze auszuwählen, ist eine Entscheidung, in der mehrere Kriterien berücksichtigt werden müssen. In Form eines Kriterienkatalogs wurden insgesamt 9 TLS-Bibliotheken untersucht. Diese Anzahl an Implementierungen basiert auf einer Auflistung der *TLS Working Group* an der IETF [41], wovon nur die in C/C++ geschriebenen Implementierungen aufgenommen wurden. Zusätzlich wurde die Liste um die Bibliothek MatrixSSL ergänzt. Die Auswahl der Kriterien

Frage	Metrik
Steht der Quellcode frei zur Verfügung?	Lizenz
Wie aktiv wird die Bibliothek gepflegt?	Aktivität
Wie groß ist die Verbreitung?	Verbreitung
Existiert eine Dokumentation zur Verwendung?	Dokumentation
Wie groß ist der Implementierungsaufwand?	Aufwand

Tabelle 3.1: Fragen und abgeleitete Metriken für den Kriterienkatalog

erfolgt nach dem Goal-Question-Metric-Ansatz [42]. Dieser Ansatz leitet Metriken aus Zielen, Fragen und Problemen ab. Dazu wird im ersten Schritt ein übergeordnetes Ziel definiert, welches in diesem Fall die Bestimmung einer geeigneten TLS-1.3-Bibliothek für die Integration in das bestehende Simulationsmodell von QUIC ist. Aus diesem Ziel werden anschließend Fragen abgeleitet, die zur Überprüfung des Zielfortschritts beitragen. Jede Frage erhält im letzten Schritt eine Metrik, mit der die Frage beantwortet werden kann. Die einzelnen Fragen mit den daraus abgeleiteten Metriken sind in der Tabelle 3.1 abgebildet. Der resultierende Kriterienkatalog aus den Bibliotheken und ausgewählten Metriken ist auf der Seite 29 in der Tabelle 3.2 zu sehen.

Alle TLS-Bibliotheken haben gemein, dass sie über eine Lizenz verfügen, die einen freien Einsatz und damit eine Integration in das bestehende Simulationsmodell von QUIC erlaubt. Des Weiteren sind die Entwicklungen der Bibliotheken auf Github oder das letzte Veröffentlichungsdatum so aktuell, dass allen das Vorhandensein einer Projektaktivität zugeschrieben werden kann.

Hinsichtlich der Verbreitung liegen nicht zu allen Implementierungen Informationen vor. OpenSSL ist vermutlich die am weitesten verbreitete TLS-Bibliothek aufgrund der Vorinstallation auf zahlreichen Betriebssystemen für den Desktop- und Serverbereich (wie z.B. Ubuntu). Der auf OpenSSL basierende Fork BoringSSL findet seinen Einsatz im Chrome/Chromium Browser, Android und zahlreichen weiteren Produkten von Google. Ein allgemeiner Einsatz von BoringSSL wird jedoch nicht empfohlen[4], da die API jederzeit Änderungen unterliegen kann und damit keine Garantie über das Bestehen verwendeter Funktionssignaturen bei einem Update existiert. GnuTLS findet vor allem im Umfeld des *GNU Projects* Einsatz. Mit NSS besitzt Mozilla eine Bibliothek, die in den eigenen Produkten (Firefox, Thunderbird usw.) und von Unternehmen verwendet wird[5].

[4] https://boringssl.googlesource.com/boringssl – Aufgerufen am 18.02.2019
[5] https://developer.mozilla.org/en-US/docs/Mozilla/Projects/NSS/Overview

Bibliothek	Lizenz	Aktiv	Verbreitung	Doku.	Aufwand	Notiz
OpenSSL	OpenSSL License	Ja	Sehr weit	Vorhanden	Normal	
BoringSSL	ISC license	Ja	Google Produkte	Nicht vorhanden	Normal	Basiert auf OpenSSL; ist nicht für allgemeinen Einsatz gedacht
FIZZ	BSD License	Ja	Unbekannt	Nicht vorhanden	-	Basiert auf OpenSSL
PicoTLS	MIT	Ja	Unbekannt	Vorhanden	Normal	Basiert auf OpenSSL; Issue: https://git.io/fhdCe
wolfSSL	GPLv2	Ja	Unbekannt	Vorhanden	Normal	Bietet OpenSSL-Kompatibilität; Issue: https://git.io/fhdCv
GnuTLS	LGPLv2.1+	Ja	Zahlreiche GNU Applikationen	Vorhanden	Normal	Bietet OpenSSL-Kompatibilität (ohne BIO-Unterstützung)
CycloneSSL Open	GPLv2	Ja	Unbekannt	Nicht vorhanden	Normal	Für embedded systems
NSS	MPL 2	Ja	Mozilla Produkte, Red Hat, Apache, Oracle	Vorhanden	-	
MatrixSSL	GPLv2	Ja	Unbekannt	Vorhanden	Normal	Bietet OpenSSL-Kompatibilität; Für embedded systems

Tabelle 3.2: Kriterienkatalog für TLS-1.3-Bibliotheken

Nicht alle Bibliotheken verfügen über eine Dokumentation, die ihren Umgang beschreibt. Während BoringSSL und CycloneSSL noch über eine Beschreibung der API verfügen, existiert für FIZZ keinerlei Dokumentation.

Um die Umsetzbarkeit und den Aufwand der Integration der Bibliotheken in das Simulationsmodell von QUIC einschätzen zu können, wurde für die Bibliotheken (FIZZ und NSS ausgenommen) eine Testapplikation mit einem TLS-1.3-Verbindungsaufbau und dem anschließendem Transfer von Daten entwickelt (alle Testapplikationen sind auf der beigelegten CD enthalten). Dabei stellte sich heraus, dass die Erstellung einer Testapplikation für FIZZ ohne Dokumentation aufwändig ist und FIZZ daher keine weitere Berücksichtigung in der Auswahl finden konnte. Ebenfalls wird das NSS nicht weiter berücksichtigt, da die Erzeugung der notwendigen Zertifikate nach Befolgung der Dokumentation nicht erfolgreich gewesen ist. Bei PicoTLS und wolfSSL haben sich trotz Befolgung der Dokumentation Schwierigkeiten beim Kompilieren ergeben, die jedoch mit Unterstützung der Entwickler gelöst werden konnten (siehe Spalte Notiz in Tabelle 3.2). Auch die Kompilierung von GnuTLS wies Schwierigkeiten auf, da einige relevante Informationen nur auf der Github-Seite der Bibliothek aufzufinden waren. Insgesamt lässt sich festhalten, dass die TLS-Bibliotheken über ähnliche oder gar identische Funktionen verfügen und der Aufwand für die Erstellung der Testapplikationen sich nach erfolgreicher Kompilation bei allen als normal erwiesen hat.

Unter der Betrachtung aller Metriken der noch zur Auswahl stehenden Bibliotheken, kann der Schluss gezogen werden, dass OpenSSL die größte Eignung für das Simulationsmodell aufweist. Die weite Verbreitung und ausführliche Dokumentation sind Aspekte, die für die Bibliothek sprechen. Das Vorhandensein zahlreicher Beispielapplikationen dient bei der Erstellung einer eigenen Applikation als Orientierung und beschleunigt die Entwicklung. Außerdem deutet der Fakt, dass einige Bibliotheken auf OpenSSL basieren und sogar eine Kompatibilitätsschicht dafür besitzen (siehe Spalte Notiz in Tabelle 3.2) darauf hin, dass OpenSSL ein Maßstab unter den Bibliotheken darstellt. Aus diesem Grund bietet sich die Auswahl von OpenSSL umso mehr an.

3.3.2 Analyse wesentlicher Aspekte des Standards

Der Internet-Draft von QUIC zur TLS-Verschlüsselung [12] stellt eine Spezifikation dar, die zahlreiche Anforderungen an eine Implementierung definiert. Aufgrund des großen Umfangs der in der Spezifikation definierten Anforderungen kann in dieser Arbeit keine

absolut vollständige Anforderungsanalyse erfolgen. Deshalb beschränkt sich dieser Abschnitt auf die Vorstellung von Aspekten, die wesentlich für die Implementierung der TLS-Verschlüsselung sind.

Eine Besonderheit von RFCs und Internet-Drafts der IETF ist die Hervorhebung von Schlüsselwörtern wie z.B. *MUST*, *SHOULD* oder *MAY*. Diese Hervorhebungen definieren die Anforderungsstufe der jeweiligen Spezifikation und geben an, welche Anforderung zwingend erfüllt werden muss (*MUST*), erfüllt werden sollte (*SHOULD*) oder deren Erfüllung optional ist (*MAY*) [43]. Bei einer Analyse des Internet-Drafts ergeben sich insgesamt 69 mit Schlüsselwörtern versehene Anforderungen, die zusätzlich zur gewöhnlichen Funktionsbeschreibung der TLS-Verschlüsselung in QUIC hinzukommen.

Die in dieser Arbeit durchgeführte Vergleichsstudie soll unter der Annahme durchgeführt werden, dass eine mit geringem Verbindungsaufwand erstellte ordnungsgemäße fehlerfreie Verbindung zwischen zwei Hosts aufgebaut wird. Diese Einschränkung muss getroffen werden, um die Implementierung der Verschlüsselungsansätze mit der Durchführung der Vergleichsstudie in einem vertretbaren Zeitrahmen zu halten.

Damit werden all jene Fälle ausgeschlossen, in denen Fehler auftreten können oder zusätzliche Parameter zur Authentizität oder Validierung benötigt werden. Von dieser Einschränkung sind folgende Features betroffen, die in der Umsetzung von QUIC-mit-TLS und QUIC-über-TLS nicht berücksichtigt werden:

- Authentifikation mit Zertifikaten
- Fehlerbehandlung
- Source Address Validation
- Key Update
- Erneute Paketübertragung
- HelloRetryRequest
- Paketempfang vor Abschluss des Handshakes
- Security

Des Weiteren können aufgrund des aktuellen Entwicklungsstandes des Simulationsmodells und der in Abschnitt 3.1 beschriebenen Machbarkeits-, Notwendigkeitsanalyse die folgenden Anforderungen nicht berücksichtigt werden:

- Überprüfung auf ältere TLS-Version – Es ist nur eine Implementierung mit TLS 1.3 beabsichtigt
- Frame-Typen wie z.B. *MAX_DATA* oder *BLOCKED* dürfen nicht ungeschützt versendet werden – Bisher sind diese Frame-Typen nicht implementiert worden
- 0-RTT – Der Umfang dieses Features übersteigt den Rahmen dieser Masterarbeit und bietet sich daher für anschließende Arbeiten an
- Version Negotiation – Das Simulationsmodell unterstützt im Moment nur eine einzige Version von QUIC

Die resultierenden Auswirkungen des Wegfalls der oben genannten Anforderungen und Features sind unter der Annahme einer ordnungsgemäßen fehlerfreien Verbindung bei der Durchführung der Vergleichsstudie sind als gering bis nicht vorhanden anzusehen, da diese nur in Ausnahmefällen Anwendung finden. Es ergeben sich schließlich folgende Anforderungen zur Grundfunktionsweise der Verschlüsselung aus dem Internet-Draft, die in den Verschlüsselungsansätzen Umsetzung finden:

1. Stream 0 wird ausschließlich für TLS-Verbindung verwendet
2. TLS bleibt unmodifiziert
3. QUIC-Verbindungsaufbau nach Abbildung 2.4
   - Integration von TLS-Handshake-Paketen in QUIC
   - *Initial Paket* enthält ClientHello
   - *Handshake Paket* enthält ServerHello
4. Transportparameter
   - Vor Handshake-Start übergibt QUIC an TLS die Transportparameter
   - Transportparameter sind in ClientHello und `EncryptedExtensions` vorhanden
5. ClientHello darf nicht größer als 1171 Bytes sein
6. Schlüssel und IVs werden mit HKDF, QHKDF abgeleitet
7. 1-RTT-Schlüssel werden nach TLS-Handshake verwendet
8. 1-RTT-Schlüssel werden von exportierten *secret values* abgeleitet

9. Vor Etablierung eines *secret values* wird mit `AEAD_AES_128_GCM` verschlüsselt, danach erfolgt die Verschlüsselung mit einer von TLS ausgehandelten AEAD-Funktion

Die Anforderungen, die mit dem Verbindungsaufbau von QUIC in Zusammenhang stehen wurden unter der übergeordneten Anforderung 3 zusammengefasst. Ebenfalls sind die mit den Transportparametern in Zusammenhang stehenden Anforderungen in der Anforderung 4 gebündelt.
Als komplex stellen sich die Anforderungen 3 und 6 heraus. Die erstgenannte Anforderung erfordert eine fehlerfreie Interaktion zwischen QUIC und TLS und damit den Export der Daten von TLS zu QUIC als auch den umgekehrten Fall. Anforderung 6 bildet die Basis der gesamten Verschlüsselung mit der Erzeugung der Schlüssel und IVs. Besonders bei dieser Anforderung ist es wichtig eine Methode zu finden, mit der die Erzeugung auf Korrektheit überprüft werden kann (siehe Abschnitt 3.3.4)

3.3.3 Entwurf

Auf Grundlage des existierenden Simulationsmodells (siehe Abschnitt 3.2) muss ein Entwurf erstellt werden, der sich in die bestehende Struktur einfügt und der Erfüllung der Spezifikation nicht entgegensteht. In diesem Entwurf muss die Möglichkeit einer Einbindung der externen OpenSSL-Bibliothek gegeben sein. Darüber hinaus sollten die gängigen Prinzipien eines Architekturentwurfs Berücksichtigung finden. Die Prinzipien lauten wie folgt [42, Kap. 17.3]:

- **Modularisierung**: Der Entwurf ist in „sinnvolle und überschaubare“ Einheiten unterteilt
- **Lose Kopplung**: Zwischen den Einheit besteht nur eine schwache Beziehung
- **Information Hiding**: Es werden nur so viele Informationen preisgegeben, wie benötigt werden
- **Separation of concerns**: Jede Einheit ist nur für einen Aufgabenbereich verantwortlich
- **Hierarchische Gliederung**: Die Einheiten sind in einer hierarchiebildenden Beziehung angeordnet

Unter Einbeziehung dieser Aspekte und der Prinzipien ist das in Abbildung 3.2 abgebildete Klassendiagramm entstanden.

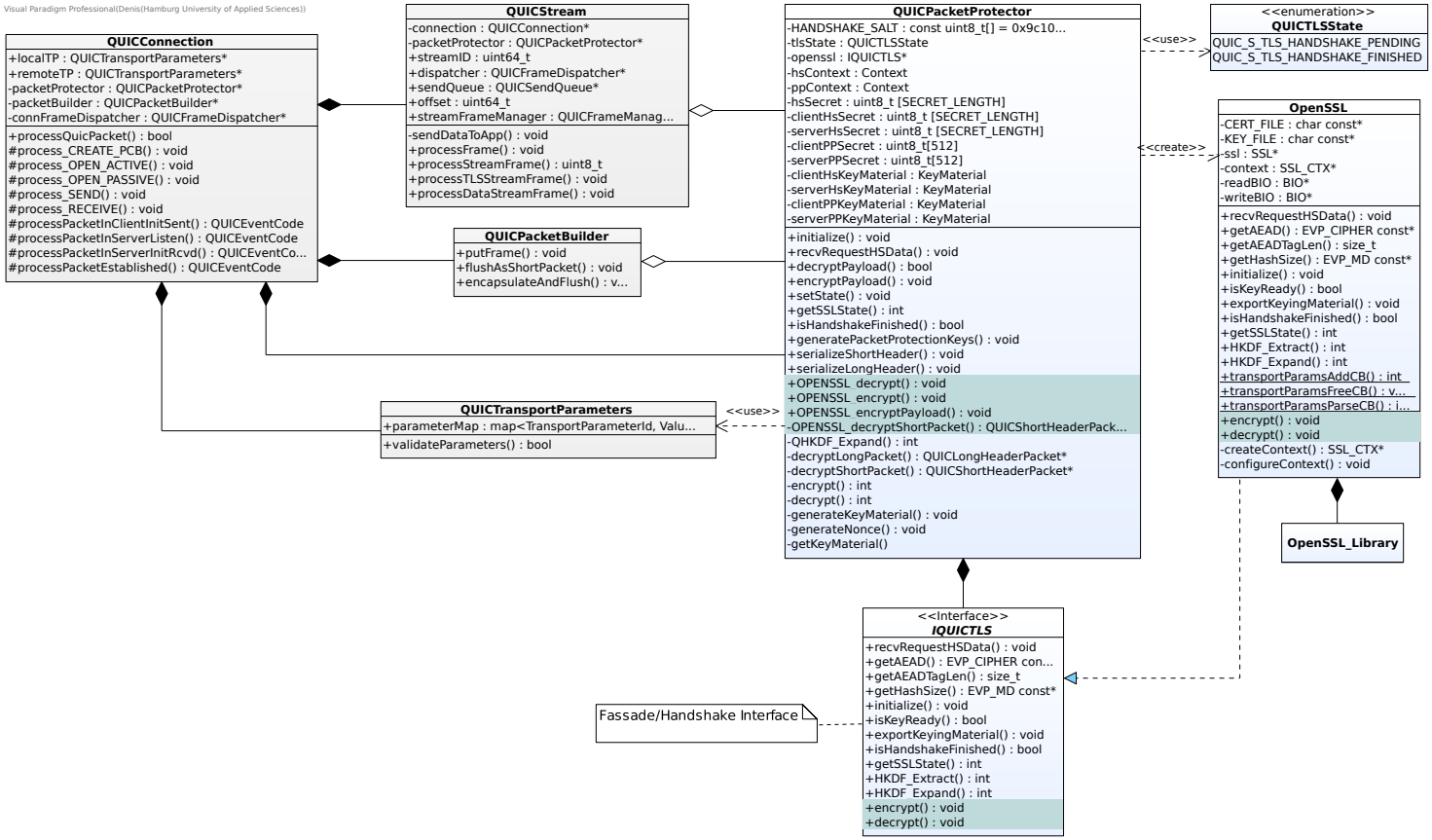


Abbildung 3.2: Klassendiagramm von QUIC-mit-TLS und QUIC-über-TLS

Das Diagramm enthält sowohl die Architektur für QUIC-mit-TLS als auch für QUIC-über-TLS. In blau auf der rechten Seite sind die an der Verschlüsselung beteiligten und neu hinzugekommenen Komponenten dargestellt. Beim `QUICPacketProtector` handelt es sich um die zentrale Komponente der QUIC-mit-TLS-Lösung. Diese Klasse stellt die einzige Schnittstelle für die anderen QUIC-Komponenten zur Ver- und Entschlüsselung von Paketen zur Verfügung und repräsentiert somit die im Internet-Draft vorgestellte *QUIC Packet Protection*. In der Variable *tlsState* wird der aktuelle Status der TLS-Verbindung vom Typ `QUICTLSState` gespeichert, um nach Abschluss des Handshakes die 1-RTT-Schlüssel zur Ver- und Entschlüsselung zu verwenden. Die Attribute zur Verwaltung der Schlüsselinformationen werden in den Variablen mit dem Suffix **Secret* und **KeyMaterial* gespeichert. In den Context-Variablen speichert die Klasse die verwendeten Verschlüsselungs- und Hashalgorithmen. Bei einer Betrachtung der Methoden-Signaturen fallen hier die mehrfach vorhandenen encrypt- und decrypt-Methoden auf. Die einfachen Methoden *encrypt()* und *decrypt()* verschlüsseln oder entschlüsseln konkrete Daten innerhalb des `QUICPacketProtector` und sind nach außen nicht sichtbar. Auf diese Methoden bauen die Payload-Methoden (*decrypt/encryptPayload()*) auf, die eine konkrete Liste von Frames übergeben bekommen und diese aufbereiten. Sie finden im `QUICPacketBuilder` und in der `QUICConnection` Einsatz. Bei den farblich hervorgehobenen Methoden handelt es sich um exklusive Methoden für die QUIC-über-TLS-Lösung. Da QUIC-über-TLS keine *Packet Protection* benötigt, aber weiterhin die Schnittstelle zur Verschlüsselung aufrecht erhalten werden soll, führen das *OPENSSL_decrypt()/OPENSSL_encrypt()* nur eine Weiterleitung zum Pendant ohne Präfix in der Klasse `OpenSSL` durch. Darüber hinaus sind beim `QUICPacketProtector` die Methode zum Empfang und Versand von TLS-Daten (*recvRequestHSData()*), die serialize-Methoden zur Erstellung der *additional data* für AEAD und das *QHKDF_Expand()* für die Ableitung der Handshake-Schlüssel und IVs erwähnenswert. Bis auf die hierarchische Gliederung, die in diesem Fall keinen Mehrwert bieten würde, finden alle Prinzipien eines Architekturentwurfs hier Anwendung.

Besonders bei der Integration der OpenSSL-Bibliothek ist die Befolgung der Entwurfsprinzipien wichtig, da aus dem Funktionsreichtum der Bibliothek mit ihren zahlreichen Verschlüsselungsprotokollen nur die Teilmenge mit TLS 1.3 von Interesse ist. Für diesen Zweck eignet sich das Fassaden-Design-Pattern [44, S. 212]. In diesem Pattern wird eine Fassadenklasse definiert, welche den Zugriff auf ein Subsystem (`OpenSSL_Library`) auf das Nötigste einschränkt. Um die Kopplung zwischen dem Subsystem und der Applikation weiter zu reduzieren, kann die Fassade als abstrakte Klasse realisiert werden (`IQUICTLS`),

von der eine Unterklasse die Repräsentation des Subsystems darstellt (`OpenSSL`). Das `IQUICTLS` enthält Methoden über den aktuell verwendeten AEAD-Algorithmus sowie zum Export der *1-RTT-Secrets* und die HKDF-Funktionen zum Ableiten von Schlüsseln. Die konkrete Unterklasse `OpenSSL` enthält zusätzlich zu den vorgegebenen Methoden Callback-Funktionen (Präfix *transportParams**), mit denen Transportparameter für die TLS-Erweiterungen mitgegeben oder ausgelesen werden können. Der Vorteil bei der Anwendung dieses Patterns ist, dass die Entwurfsprinzipien befolgt werden und damit eine hohe Übersichtlichkeit und Wartbarkeit erzielt wird.

In grau auf der linken Seite sind die mit der Verschlüsselung in Berührung kommenden Komponenten des Modells von QUIC zu finden, welche über Referenzen zur *Packet Protection* verfügen, um davon Gebrauch zu machen. Die process-Methoden der `QUICConnection` wurden dahingehend angepasst, dass die empfangenen Pakete vor Übergabe an den `QUICFrameDispatcher` durch die QPP entschlüsselt werden. Handelt es sich bei den empfangenen Paketen um Stream Frames, so werden diese in `QUICStream` entweder als TLS-Stream-Frame (Daten werden an TLS weitergereicht) oder als Data-Stream-Frame (Daten werden an Applikation weitergereicht) in den entsprechenden Methoden verarbeitet. Auf der Sendeseite verschlüsselt nun der `QUICPacketBuilder` die Pakete durch die QPP.

3.3.4 Verifikation

Nach erfolgter Implementierung der Verschlüsselungsansätze QUIC-mit-TLS und QUIC-über-TLS steht der Schritt der Verifikation aus. In diesem wird überprüft, ob die Implementierung konform mit der Spezifikation ist. Das hier genutzte Mittel zur Verifikation sind Programmtests [45, Kap. 2.6]. Im Prozess des Testens werden hierfür Testfälle ermittelt. Jeder Testfall besteht aus der Angabe von Rahmenbedingungen, der geeigneten Eingabeparameter, des erwarteten Ergebnisses nach Ablauf des Testfalls (Soll-Ergebnis) und der damit abgedeckten Anforderung. Nach der Ausführung des Testfalls gilt es im Rahmen der Auswertung, das aufgetretene Ergebnis mit dem Soll-Ergebnis zu vergleichen. Sofern beide Ergebnisse gleich sind, liegt eine Konformität mit der Spezifikation vor und der Testfall ist erfüllt.

Es zeigt sich bei den Anforderungen aus der Analyse der wesentlichen Aspekte (Abschnitt 3.3.2), dass nicht immer ein eigener Testfall sinnvoll und möglich ist. In diesen Fällen stellt eine Assertion im Programmcode oder eine Sichtprüfung ein ausreichendes

Testfall	Anfangszustand	Eingabe/Aktion	Erwartetes Verhalten	Anforderung
Stream 0	Simulation von Datenübertragung zwischen zwei Hosts	Datenübertragung	Daten werden nicht über Stream 0 versandt	1
TLS-Handshake-Nachrichten	Simulation von Datenübertragung zwischen zwei Hosts	Verbindungsaufbau	Übereinstimmung der ersten Bytes der TLS-Nachrichten mit den geforderten Nachrichtentypen	3
Transportparameter	Simulation von Datenübertragung zwischen zwei Hosts	Verbindungsaufbau	Transportparameter werden hinzugefügt und geparst	4
1-RTT-Schlüssel	Simulation von Datenübertragung zwischen zwei Hosts	Verbindungsaufbau	Ausgabe über die Generierung und Nutzung der 1-RTT-Schlüssel	7, 8
Secret-value-Erzeugung	Instanziierte QUIC-Connection	Connection-ID-Testvektor der QUIC WG	Übereinstimmung aller *handshake secret values* und *keys* mit vorgegebenen Werten der QUIC WG	-
Ver- und Entschlüsselung	Instanziierte QUIC-Connection mit fertigen *secret values* und *keys*; QPP ist empfangsbereit	Zu verschlüsselnder String „*ENCRYPTION_TEST*“	Entschlüsselung des verschlüsselten Strings sollte „*ENCRYPTION_TEST*“ ergeben	-

Tabelle 3.3: Testfälle von QUIC-mit-TLS und QUIC-über-TLS

Mittel dar, um die Konformität zum Internet-Draft zu überprüfen. Davon betroffen sind die Anforderungen 2, 5, 6 und 9.

Die Tabelle 3.3 zeigt alle Testfälle für die Implementierungen auf. Davon nehmen die ersten vier Fälle Bezug zu den in Abschnitt 3.3.2 definierten Anforderungen. Der Anfangszustand der ersten vier Testfälle besteht aus einem Simulationsaufbau, in dem zwei Hosts eine Verbindung aufbauen und miteinander Daten austauschen. Im ersten Testfall ist die Aktion der Datenübertragung von Relevanz. Hier wird überprüft, ob keine Daten über Stream 0 versendet werden und deckt somit die erste Anforderung ab. Der nachfolgende Testfall deckt die Menge an Unteranforderungen von Anforderung 3 ab und prüft auf das Vorhandensein und der richtigen Abfolge der TLS-Handshake-Nachrichtentypen. Eine Erfüllung des Testfalls ist gegeben, wenn die ersten Bytes jeder TLS-Handshake-Nachricht mit den erwarteten Bytes des jeweiligen Nachrichtentyps übereinstimmen. Testfall 3 verifiziert anhand von Ausgaben, ob die Übergabe und der Empfang der Transportparameter erfolgt ist und deckt hiermit die vierte Anforderung ab. Eine Ausgabe über die Generierung und Nutzung der 1-RTT-Schlüssel überprüft, ob die Anforderungen 7 und 8 erfüllt

werden.

Im Rahmen einer intuitiven Testfallermittlung [45, S. 165] wurden überdies zwei weitere Testfälle hinzugefügt: Ein Testfall für die Secret-Value-Erzeugung und einer für die allgemeine Ver- und Entschlüsselung. Der erste Testfall überprüft die Basisfunktionalität der QPP. Dafür wird eine `QUICConnection` erstellt, die als Eingabeparameter eine spezielle Connection ID erhält. Die QUIC WG hat auf Basis dieser Connection ID alle erwarteten *handshake secret values* und *keys* angegeben, anhand derer die korrekte Funktionsweise der HKDF-Extract/Expand- und QHKDF-Expand-Funktion verifiziert werden kann [46]. Eine Übereinstimmung dieser vordefinierten Werte mit den selbst berechneten Werten bestimmt für diesen Testfall das erwartete Verhalten. Der letzte Testfall soll den gesamten Verschlüsselungsprozess auf ihre Funktion überprüfen. Am Anfang wird hierfür eine Verbindung mit vorgegebenen *secret values* und *keys* instanziiert. Danach erhält die QPP als Eingabe einen zu verschlüsselnden String und gibt das verschlüsselte Ergebnis zurück, das direkt wieder entschlüsselt wird. Eine anschließende Prüfung gibt Aufschluss darüber, ob die Daten nach der Entschlüsselung noch die selben sind wie vor der Verschlüsselung.

Als Werkzeug zur Durchführung der Testfälle dient das *opp_test* von OMNeT++ [23, Kap. 15.2]. Mit *opp_test* können Unit-Tests definiert werden, die auf ein bestehendes Simulationsmodell zurückgreifen können. In einer Test-Datei wird eine Simulation erstellt und die erwarteten Ausgaben für die Erfüllung eines Anwendungsfalls spezifiziert. Dabei kann eine Test-Datei die Überprüfung mehrerer Testfälle beinhalten. Die Dateien für den Test sind auf der beigelegten CD enthalten. Eine Ausführung aller Testfälle zeigt, dass bei jedem Fall das erwartete Verhalten eingetreten ist. Damit gelten die Implementierungen QUIC-mit-TLS und QUIC-über-TLS für die definierten Anforderungen als verifiziert.

4 Vergleichsstudie

Nachdem in den letzten Kapiteln die Voraussetzungen zur Durchführung einer Vergleichsstudie geschaffen worden sind, gilt es die mit der Studie unmittelbar zusammenhängenden Rahmenbedingungen zu definieren und Maßnahmen zur Experimentdurchführung zu ergreifen. Dafür werden im ersten Schritt die Experimente vorgestellt, die als Vergleichsgrundlage dienen sollen (Abschnitt 4.1). Daran schließt in Abschnitt 4.2 die Bestimmung von Kriterien an, anhand derer die Verschlüsselungsansätze verglichen und Schlussfolgerungen gezogen werden können. Abschnitt 4.3 stellt das Design des Experiments vor, das die Voraussetzungen der Messung festlegt, bevor Abschnitt 4.4 die Hintergründe und die Entwicklung einer Messapplikation beschreibt. Eine Beschreibung des Aufbaus der Simulation in OMNeT++ wird in Abschnitt 4.5 gegeben. Den Abschluss bildet der Abschnitt 4.6 mit der Vorstellung der Messergebnisse und einer Diskussion, in der die Ergebnisse der Verschlüsselungsansätze verglichen und interpretiert werden.

4.1 Experimente

Ausgangspunkt der Experimente sind zwei Verschlüsselungsansätze für das QUIC-Protokoll, für die Anwendungsfälle entwickelt werden müssen, unter denen die Auswirkungen der Verschlüsselung am stärksten hervortreten können. Das Simulationsmodell erlaubt derzeit nur eine 1:1-Verbindung zwischen einem Host und einem Server. Aus diesem Grund kann der Fall, in dem mehrere Clients sich mit einem Server verbinden, nicht abgebildet werden. Da sich TLS aus dem *handshake* und dem *record protocol* zusammensetzt, sind Experimente erforderlich, die den Verbindungsaufbau als auch die Datenübertragung abdecken. Für den Verbindungsaufbau wurde daher der folgende Anwendungsfall festgelegt:

- *Client initiiert mehrere Verbindungen*
  Für diesen Fall existiert kein praxisnaher Bezug, jedoch lässt sich anhand der Viel-

zahl an Verbindungsinitiierungen der Schwerpunkt auf die Verschlüsselungskomponente von QUIC beim Verbindungsaufbau setzen. Die Auslastung des Clients gibt Aufschluss darüber, welcher Verschlüsselungsansatz in dieser Situation besser abschneidet. Der Parameter besteht hier aus der Frequenz, mit der der Client neue Verbindungen initiiert.

Bei der Datenübertragung kann die Paketgröße der Daten angepasst werden und findet hier Berücksichtigung. Es ergeben sich folgende zwei Anwendungsfälle:

- *Verschlüsselung vieler kleiner Datenpakete*
  Durch die Erzeugung vieler Pakete wird vor allem die Serialisierung der Header-Daten für die *associated data* in Anspruch genommen. Während diese bei QUIC-mit-TLS aus dem QUIC-Header bestehen, ist es bei QUIC-über-TLS der Record-Header von TLS. Die Datenübertragungsrate stellt hier die veränderliche Variable dar. Mit 100 Byte ist die Größe des Datenpakets festgelegt, womit es um einen Faktor von über 12 kleiner ist als im nachfolgendem Fall.

- *Verschlüsselung vieler großer Datenpakete*
  Im Gegensatz zum vorherigen Anwendungsfall wird hier die Datenübertragung größerer Pakete betrachtet. Es bildet die häufig in der Praxis anzutreffende Situation ab, in der größere Dateien versendet und dadurch die Pakete bis an das Maximum befüllt werden. Hierbei ist von Interesse, inwiefern sich dieser Fall von dem vorherigen Fall unterscheidet. Die Parameter sind dabei die selben wie im vorherigen Fall und mit 1245 Byte wird das Paket bis zum Maximum befüllt. Diese Größe ergibt sich aus der folgenden Berechnung:

  $$\text{Max. Paketgröße QUIC mit IP (1280 B) - IPv4-Header (27 B) - UDP-Header (8 B)} = 1245\,\text{B}$$

Bei dem Computersystem, auf dem die Experimente durchgeführt werden sollen, handelt es sich um ein Notebook, das über einen Intel i5-6200U mit 2 x 2,30 GHz und 8 GB Arbeitsspeicher verfügt. Der Prozessor ist der Mittelklasse von Intel zuzuordnen und stammt aus dem Jahr 2015. Auf dem Gerät läuft das Betriebssystem Manjaro Linux 18.0.4 mit einem Linux Kernel in der Version 4.14.105 (Schutz gegen Spectre Variante 1 und 2 sowie Meltdown vorhanden).
Die Festlegung konkreter Parameter für die Verbindungen/Minute und die Datenübertragungsrate sind von der Leistungsfähigkeit des Computersystems abhängig. Da für das verwendete Computersystem kein Maßstab für die Parameter zur Orientierung existiert,

Fall	Verb./Min.	Übertragungsrate (Mbit/s)	Paketgröße (Byte)
1	1000 - 10000	-	-
2	-	200 - 1000	100
3	-	200 - 1000	1245

Tabelle 4.1: Übersicht der Parameter der durchzuführenden Experimente

müssen im ersten Schritt selbst Parametergrenzen und Schrittweiten definiert werden, die als Ausgangspunkt für die Experimente dienen. Im Fall der Datenübertragungsrate wird die Grenze auf 1 Gbit/s festgesetzt, der sich mit einer Schrittweite von 200 Mbit/s angenähert wird. Die Verbindungsrate ist mit 10 000 Verb./min nach oben fixiert und wird in 1000er-Schritten angenähert. Sollten sich am Ende die Ergebnisunterschiede als zu gering herausstellen, werden die Wertgrenzen noch erweitert, um aussagekräftigere Ergebnisse hervorzubringen.

In der Tabelle 4.1 sind alle Experimente mit dem Wertebereich der Parameter aufgelistet. Diese drei Experimente werden jeweils an einem Verschlüsselungsansatz durchgeführt. Daraus folgt, dass für jeden Verschlüsselungsansatz 20 Einzelmessungen durchgeführt werden müssen. In der Tabelle A.1 im Anhang sind alle durchzuführenden Messungen ausführlich dargestellt.

4.2 Vergleichskriterien

Dieser Abschnitt beschäftigt sich mit der Bestimmung von Kriterien, die im Rahmen der Experimentdurchführung erfasst werden sollen. Wie bereits in Abschnitt 3.3.1 vorgestellt, werden die Kriterien mithilfe des Goal-Question-Metric-Verfahrens ermittelt. Das übergeordnete Ziel hierbei ist herauszufinden, welcher der beiden Verschlüsselungsansätze besser für QUIC geeignet ist. Aus diesem Ziel wurden folgende Fragestellungen mit der damit adressierten Metrik abgeleitet:

Frage	Metrik
Wie hoch ist die CPU-Auslastung?	CPU-Auslastung
Wieviel Speicher wird in Anspruch genommen?	Speicherverbrauch
Wie lange dauert die Verschlüsselung?	Zeit
Wie hoch ist der Entwicklungsaufwand?	Entwicklungsaufwand
Wie hoch ist der Wartungsaufwand?	Wartungsaufwand

Tabelle 4.2: Fragen und abgeleitete Metriken für die Vergleichsstudie

Die ersten drei Fragestellungen und Metriken sind häufig Bestandteil von Performanceanalysen und decken durch die resultierenden eindeutigen Ergebnisse den quantitativen Anteil zur Beantwortung der Zielfragestellung ab. Dagegen behandeln die letzten beiden Fragestellungen die implementierungsspezifische Seite der Verschlüsselungsansätze und erfordern somit eine qualitative Beantwortung. Die Beantwortung dieser Fragestellungen kann nur auf Grundlage des eigenen subjektiven Eindrucks geschehen. Das bedeutet, dass die Antwort keine Allgemeingültigkeit besitzt und je nach Entwickler unterschiedlich ausfallen könnte.

4.3 Experimentdesign

Die Durchführung eines einzigen Experimentdurchlaufs auf einem gewöhnlichen Computersystem bringt zum einen das Problem, dass Messfehler keine Berücksichtigung finden und zum anderen, dass der gemessene Wert keine Einschätzung über die Richtigkeit liefern kann. Diesen Problemen wird mit der Befolgung der Basisprinzipien der Replikation und des Blockings eines Experimentdesigns entgegengewirkt [47]. Unter dem Begriff der Replikation wird die Wiederholung eines Experiments verstanden, um eine Einschätzung über die Messfehler und die Lage des wahren Werts zu erhalten. Das Blocking sieht die Minimierung oder Beseitigung von Störeinflüssen vor, die Auswirkungen auf das Ergebnis des Experiments haben. Zuerst wird die Befolgung des Blocking-Prinzips beschrieben, bevor im Anschluss auf die Replikation eingegangen wird.

Die mehrfache Ausführung eines Programms auf einem Computersystem unter den selben Versuchsbedingungen kann dazu führen, dass sich die Ergebnisse einer Messung zwischen den Durchläufen unterscheiden. Diese werden durch hardwareseitige als auch softwareseitige Faktoren verursacht. Die Arbeiten [48] und [49] haben sich mit der Variation

von Messergebnissen bei der Analyse von Programmausführungszeiten beschäftigt und dabei folgende Einflussfaktoren festgestellt, die hier neu kategorisiert, stellenweise zusammengefasst und mit Faktoren aus dem Buch *Modern Operating Systems* von Andrew S. Tanenbaum [50] ergänzt wurden:

Softwareseitige Faktoren

Die Grundlage jeder Applikation stellt das darunter laufende Betriebssystem dar. Auf diesem werden gewöhnlicherweise mehrere Programme gleichzeitig ausgeführt, was dazu führen kann, dass zu einem Zeitpunkt, an dem eine Applikation auf die Hardware zugreifen möchte, dieser Zugriff erst verzögert gewährt werden kann. An dieser Verzögerung kann das Auftreten oder die Abarbeitung eines Interrupts verantwortlich sein. Ein Interrupt signalisiert das Auftreten eines besonderen Ereignisses (z.B. Betätigung einer Taste, Ablauf eines Timers usw.), das einer Bearbeitung bedarf. Die CPU bricht dafür ihre aktuelle Tätigkeit ab, nimmt sich der Bearbeitung des Ereignisses an und kehrt nach Abschluss zur vorherigen Tätigkeit zurück.
Weiter ist durch die Möglichkeit, mehrere Threads abwechselnd auf einer CPU laufen zu lassen (Multitasking) die Herausforderung entstanden, entscheiden zu müssen, in welcher Reihenfolge die Threads Zugriff auf die CPU erhalten sollen (*scheduling*). Mit dem Aufkommen von Multiprozessorsystemen kommt erschwerend das Problem hinzu, auf welchen Prozessorkern der Thread ausgeführt werden soll. Diese Ausführungsreihenfolgen werden vom *Scheduler* festgelegt und können nur in einem eingeschränkten Rahmen beeinflusst werden.
Die dynamische Taktfrequenzanpassung der CPU und andere CPU-Funktionen können durch das Betriebssystem eingestellt werden und somit Einfluss auf die Ausführungsgeschwindigkeit nehmen.
Ebenfalls ist neben der Ausführung eines Programms auf der CPU auch eine schnelle Bereitstellung der auszuführenden Operationen von Relevanz. Durch das Vorhandensein unterschiedlicher Speicherhierarchien können große Verzögerungen zustandekommen, da diese sich in den Zugriffszeiten um mehrere Größenordnungen unterscheiden können. Ist ein Großteil der Daten in den schnellen Caches der CPU vorhanden, so fällt die Ausführungszeit entsprechend gering aus, da der CPU die benötigten Daten zur Ausführung schnell zur Verfügung stehen. Liegen die Daten dagegen im Arbeitsspeicher oder gar auf der Festplatte, folgt daraus eine erhebliche Erhöhung der Ausführungszeit. Für die Verwaltung der Daten über die Speicherhierarchien hinweg ist der *memory manager* zuständig, auf dessen Verhalten nicht direkt Einfluss genommen werden kann.
Zuletzt müssen die Metriken der zu messenden Applikation durch eine Messapplikati-

on erfasst und dokumentiert werden. Die Messapplikation tritt dabei wie jede andere Applikation in Konkurrenz um die Hardwareressourcen mit der zu messenden Anwendung, weshalb eine geringe Einflussnahme auf die Ausführungsumgebung von Vorteil ist. Neben einer geringen Einflussnahme ist die Genauigkeit, mit der die Metriken der gemessenen Applikation erfasst werden, von großer Wichtigkeit. Jede Vermeidung eines anderen Faktors ist wirkungslos, wenn die Messergebnisse die notwendige Genauigkeit vermissen lassen.

Hardwareseitge Faktoren
Unter den hardwareseitigen Faktoren werden all jene zusammengefasst, auf die mit Software nicht direkt Einfluss genommen werden kann. Darunter fallen insbesondere Mechanismen zur Beschleunigung der Ausführungsgeschwindigkeit auf dem Prozessor. Beispiele für solche Mechanismen sind *out-of-order execution*, *automatic data prefetching*, *speculative execution*, *branch prediction* usw.

Nachdem eine Übersicht über mögliche Einflussfaktoren einer Messung gegeben worden ist, werden Maßnahmen für die Durchführung der Experimente definiert, um die Auswirkungen dieser Faktoren soweit wie möglich zu minimieren. Diese Maßnahmen bestehen aus Vorbedingungen, die erfüllt werden müssen, um eine bessere Reproduzierbarkeit zu erzielen. Aufgrund der Schwierigkeit, die Einflüsse der hardwareseitigen Faktoren zu minimieren, wird im Nachfolgendem nur auf die softwareseitigen Faktoren eingegangen.

Um die Auswirkungen des Schedulings auf die Experimente zu beseitigen, wird dem Kernel beim Start der Parameter `isolcups=1` mitgegeben. Das Setzen des Parameters bewirkt, dass der angegebene CPU-Kern (hier der zweite) vom Scheduling-Prozess ausgenommen wird und für andere Prozesse nicht mehr zur Ausführung von Operationen zur Verfügung steht.
Damit der zu messenden Applikation dieser isolierte CPU-Kern zugewiesen werden kann, muss die CPU-Affinität des Prozesses gesetzt werden. Die CPU-Affinität ist eine Scheduler-Eigenschaft, mit der ein Prozess an einem CPU-Kern gebunden werden kann. Der Einsatz des Werzeugs *taskset* bewirkt durch die Angabe einer Prozess-ID und einer Menge an CPU-Kernen die ausschließliche Ausführung auf diesen CPU-Kernen [51].
Im nächsten Schritt gilt es den Einfluss von auftretenden Interrupts zu verringern. Mithilfe einer Einstellung an der SMP-IRQ-Affinität lässt sich auf Betriebssystemen mit Linux bestimmen, auf welchen CPU-Kernen die Interrrupts behandelt werden sollen [52]. Im Verzeichnis `/proc/irq` steht für jede mögliche Interrupt-Quelle ein Verzeichnis in dem die Datei *smp_affinity* enthalten ist. In dieser Datei ist in einem hexadezimalen Wert

kodiert, welche CPU-Kerne für die Behandlung des Interrupts zulässig sind ('f' erlaubt Behandlung auf 16 CPU-Kernen, '1' beschränkt Behandlung auf ersten Kern). Da der zweite CPU-Kern bereits für die zu messende Applikation reserviert ist, wird bei allen Interrupt-Quellen die Behandlung des Interrupts auf den ersten CPU-Kern festgelegt. Mit dieser Vorgehensweise wird ein Großteil der Interrupt-Last auf den ersten CPU-Kern übertragen. Jedoch bleibt der zweite Kern nicht absolut von Interrupts befreit, da einige Interrupt-Quellen wie z.B. der *local timer interrupt* sich nicht einstellen lassen.
Die dynamische Anpassung der CPU-Taktfrequenz ist ein Mechanismus, mit dem Energie eingespart werden kann, wenn die maximale Rechenkapazität durch die aktuelle Ausführung der Instruktionen nicht ausgeschöpft werden kann. Es wird permanent versucht, die größtmögliche Effizienz zu erreichen, indem die Taktfrequenz soweit angehoben wird, um die aktuellen Instruktionen optimal bearbeiten zu können, ohne dabei Rechenkapazität ungenutzt zu lassen [53]. Über das sys-Dateisystem in Linux kann auf Geräte, Kernelmodule, Dateisysteme und andere Kernelkomponenten zugegriffen werden [54]. Darüber lässt sich auch eine Fixierung der Taktfrequenz erzielen, um mögliche Nebeneffekte der dynamischen Taktfrequenzanpassung zu vermeiden. In der Datei `/sys/devices/system/cpu/cpu1/cpufreq/scaling_setspeed` wird für diesen Zweck die Frequenz auf den angegebenen Maximalwert der CPU (2,3 GHz) festgesetzt. Dadurch wird ebenfalls der Effekt des Turbo-Boosts[1] unterbunden, der die CPU-Frequenz für kurze Zeit über die maximale Taktfrequenz hinaus erhöhen kann. Voraussetzung für diese Einstellung ist jedoch die Deaktivierung des Treibers *intel_pstate* [55] in Verbindung mit dem Setzen des CPU-Governors auf „*userspace*“ [56].
Wie auch in [48] beschrieben, wird vor jedem Start eines neuen Experiments das Computersystem neu gestartet. Der Grund dafür sind eventuell vorbehaltene Speicherrückstände eines vorherigen Experiments, die den neuen Experimentdurchlauf beeinflussen könnten. Weitere Faktoren, die bisher keine Erwähnung gefunden haben, aber bei der Ausführung der Experimente von Relevanz sind, sind der Betrieb des Notebooks im Akku- oder im Strom-Modus und die Kompilierung der Applikation für den Produktiveinsatz. Während des Akkubetriebs können zahlreiche Energiesparmechanismen eingreifen und die Messungen beeinträchtigen, weshalb bei allen Experimenten das Gerät am Stromnetz angeschlossen betrieben wird.
Für die Entwicklung wurde sowohl die TLS-Bibliothek OpenSSL als auch das Simulationsmodell von QUIC in einem debug-fähigen Format kompiliert, was die Entwicklung erleichtert, aber auch negative Auswirkungen auf die Ausführungszeit hat und damit

[1] https://www.intel.de/content/www/de/de/architecture-and-technology/turbo-boost/turbo-boost-technology.html – Abgerufen am 18.03.2019

nicht die beste Leistung der Applikationen widerspiegelt. Deshalb werden beide Applikationen erneut für den Produktiveinsatz kompiliert.
Zuletzt werden nicht benötigte Benutzerprozesse beendet, um mögliche Effekte beim Teilen der selben Speicherstrukturen mit der zu messenden Applikation zu unterbinden.

Die Vermeidung möglicher Störeinflüsse alleine trägt nicht zu einer Sicherheit der Messung bei. Aus diesem Grund ist es erforderlich, dass die Experimente zusammen mit den Messungen nach dem oben beschriebenen Prinzip der Replikation mehrfach durchgeführt werden. Die Arbeit [48] empfiehlt 30 Wiederholungen, wovon der Median anstelle des Durchschnitts für die Auswertung verwendet werden soll. [49] setzt in seinen Experimenten ebenfalls auf 30 Wiederholungen. Jedoch beziehen sich beide Arbeiten ausschließlich auf die gemessene Ausführungsgeschwindigkeit ihrer Applikation. Da die Übertragung eines *Initial Packets* bei der Verbindungsinitiierung oder eines Datenpakets ein Vorgang ist, der sehr wenig Zeit in Anspruch nimmt und damit schwer für die Messapplikation zu erfassen ist, wird anstelle einer Anzahl an Wiederholungen mit einer Dauer gearbeitet, innerhalb derer diese Pakete fortwährend gesendet werden.

Die gemessenen Zeiten der Verschlüsselung spiegeln einen vollständigen Verschlüsselungsablauf wider, weshalb die Auswahl des Medians angebracht ist, um den Einfluss von Spitzenwerten auszuschließen, die womöglich durch den Einfluss größerer Störfaktoren zustande gekommen sind. Dagegen können die Messungen der CPU-Last und Speicherauslastung nur in Intervallen gemessen werden, die unabhängig von der Verbindungsinitiierungs- und Datenübertragungsrate sind. Daher kann jeder Messpunkt einen anderen Zeitpunkt der Verschlüsselung abbilden, weshalb alle Werte in Form eines Mittelwerts hier Berücksichtigung finden. Der Nachteil dieser Vorgehensweise ist, dass auch Spitzenwerte berücksichtigt werden, die durch größere Störfaktoren zustande gekommen sind. Die Dauer eines Experiments wird auf 15 Sekunden festgesetzt, womit für die CPU- und Speicherlast 30 Messwerte bei einem Messintervall von 0,5 s zur Verfügung stehen. Im selben Zeitraum erzeugt die Zeitmessung je nach Senderate mehrere tausend Werte, da diese direkt im Code eingebunden wird und daher in Abhängigkeit zur eingestellten Verbindungsinitiierungs- und Datenübertragungsrate steht.

Jeder Anwendungsfall wird, wie auch in den oben erwähnten Arbeiten, mit den selben Parametern 30 Mal wiederholt. Daraus ergibt sich bei einer Anzahl von 40 durchzuführenden Messungen (siehe Tabelle A.1 im Anhang) eine Gesamtzahl von 1200 Durchläufen. Da die Messwerte durch die oben beschriebenen Faktoren Abweichungen voneinander aufweisen, ist es nicht möglich, den absolut richtigen Wert zu bestimmen. Um einen

Eindruck über die Zuverlässigkeit der Messstichproben zu erhalten, wird daher für jede Probe ein Konfidenzintervall berechnet. Mit der Wahrscheinlichkeit eines zuvor festgelegten Konfidenzniveaus (hier 95 %) enthält dieses angegebene Intervall den tatsächlichen Wert der Messung. Liegt ein Messwert außerhalb des Intervalls, so folgt daraus, dass entweder das Konfidenzintervall das falsche ist und damit zu den 5 % gehört, die den exakten Messwert nicht enthalten oder – was eher wahrscheinlich ist –, kann der Messwert nicht der exakte Wert sein. Dieses statistische Mittel erlaubt damit eine bessere Einordnung der Messergebnisse.

Sobald alle Messergebnisse vorliegen, erfolgt zur besseren Veranschaulichung eine grafische Aufbereitung in Form von Diagrammen. Anhand der verwendeten Eingabeparameter, die die Problemgröße (n) darstellen und den resultierenden Messwerten ($T(n)$) kann das Wachstumsverhalten der Anwendungsfälle betrachtet und in eine Komplexitätsklasse ($g(n)$) eingeordnet werden, sodass $T(n) = O(g(n))$ gilt. Die Anzahl an Elementaroperationen stellt jedoch nicht die Grundlage der Komplexitätsbestimmung dar, da sich die durchgeführten Operationen der Verschlüsselungsansätze nicht in Beziehung setzen lassen. Stattdessen wird die Zeitkomplexität der Verschlüsselungsdauer, die Speicherkomplexität des in Anspruch genommenen Speichers und aufgrund des geringen Wertebereichs eingeschränkt die Komplexität der CPU-Auslastung festgestellt.

4.4 Entwicklung einer Messapplikation

Die Erfassung der in Abschnitt 4.2 definierten Metriken erfordert die Verwendung von Methodiken und Werkzeugen, die ein möglichst genaues Bild über die Auslastung des Computersystems verschaffen. OMNeT++ bietet für diesen Zweck keine Möglichkeiten, weshalb entweder auf fremde Werkzeuge (z.B. *sysstat*, *ps*, *top*) oder eine Eigenentwicklung zurückgegriffen werden muss. Um ein Verständnis über die Auswahl der benötigten Parameter aufzubauen, wurde die Entwicklung einer eigenen Applikation bevorzugt (Quellcode auf beiliegender CD). Damit gehen die Vorteile einher, dass die Parameter, Berechnungen und Messintervalle frei festgelegt werden können und transparent zur Verfügung stehen, was im Falle einer Verwendung eines fremden Werkzeugs nicht einfach möglich ist. Ebenfalls ist das Ein- und Ausgabeformat frei wählbar, was im Zuge der Messdurchführung und der Ergebnisaufbereitung hilfreich ist. Die Implementierung der Applikation erfolgt in der Programmiersprache C, die aufgrund ihrer Nähe zur Hardware die Entwicklung von effizienten Applikationen ermöglicht [57]. Diese Wahl wurde vor

dem Hintergrund getroffen, dass das Werkzeug zur Messung die zu messende Applikation während der Ausführung so wenig wie möglich beeinträchtigen sollte. Auf Basis des proc-Dateisystems in Linux liest die Messapplikation die benötigten Werte aus und verarbeitet diese entsprechend. Dieses Dateisystem enthält Informationen zu aktuell laufenden Prozessen und zum Gesamtzustand des Computersystems [58].

In Zeitintervallen von einer halben Sekunde zeichnet die C-Applikation die Verarbeitungszeit eines Prozesses auf der CPU und den verwendeten Speicher auf. Zur Berechnung der Prozessorauslastung eines Prozesses wird die Ausführungszeit im *kernel space*, *user space* sowie die Zeit möglicher Kinder-Prozesse zusammen addiert und mit der totalen Zeit der CPU in diesem Zeitraum in Relation gesetzt. Daraus resultiert ein Prozentsatz, der den Ausführungsanteil des Prozesses auf der CPU in einem Messintervall darstellt.

Für die Bestimmung des verwendeten Speichers eines Prozesses wird die *Resident set size* (verwendeter Arbeitsspeicher) mit dem verwendeten Swap-Speicher (ausgelagerter Speicher außerhalb des Arbeitsspeichers) addiert. Das Endergebnis beinhaltet auch die Größe des verwendeten *shared memory*, das zwischen mehreren Prozessen gemeinsam geteilte Daten, wie z.B. Bibliotheken, enthält. Diese Vorgehensweise zur Ermittlung des genutzten Speichers wird auch vom Kommandozeilenwerkzeug *top* [59] und von der *Passenger Library* (ohne Berücksichtigung des *shared memory*) [60] angewendet. In einer Gegenüberstellung mit dem Taskmanager der Linux-Desktopumgebung XFCE [61] und *htop* [62] konnte verifiziert werden, dass die ausgegebenen Ergebnisse der Messapplikation mit denen der Taskmanager ungefähr übereinstimmen. Die aufgetretenen minimalen Abweichungen sind womöglich auf die unterschiedlichen Zugriffszeitpunkte der Parameter zurückzuführen.

Im letzten Schritt muss die Dauer der Verschlüsselung gemessen werden. Dafür wird auf die C++-Bibliothek *chrono* [63] zurückgegriffen, mit der die Messung der Zeit in verschiedenen Auflösungen ermöglicht wird. Die Bibliothek stellt verschiedene *clocks* bereit, die sich hinsichtlich ihres Startpunkts unterscheiden und eine unterschiedliche Tickgenauigkeiten aufweisen können. Mithilfe des Quellcodes einer Applikation zur Bestimmung der Auflösung der einzelnen *clocks* [63, S. 150] (Quellcode auf beiliegender CD) wurde festgestellt, dass alle *clocks* des Computersystems für die Experimentdurchführung mit der selben Auflösung von einer Nanosekunde laufen.

Da die *steady_ clock* als einzige garantieren kann, dass sie während ihrer Laufzeit nicht angepasst wird, kann die Differenz von nacheinander aufgenommenen Zeitpunkten niemals

negativ sein. Aus diesem Grund wird diese *clock* zur Messung der betreffenden Codestellen verwendet. Die zu messenden Codestellen beschränken sich hierbei ausschließlich auf die Initialisierung der QPP mit der Generierung der *secret keys* und der Verschlüsselung von Paketen, um nur die Zeiten der Verschlüsselungsansätze isoliert vom Rest des Transportprotokolls gegenüberstellen zu können.

4.5 Simulationsaufbau

Wie in Abschnitt 2.4 beschrieben, werden Modelle zur Simulation in OMNeT++ in NED-Dateien beschrieben. Der Aufbau des hier verwendeten Modells besteht aus einem Netzwerk `QUICNet` (stellt ein *compound module* dar), das das gesamte Modell kapselt. In diesem sind ein Channel-Typ (*connection* mit festgelegten Parametern), die verwendeten *simple* und *compound modules* (auch als *submodules* zusammengefasst) und die *connections* zwischen den *submodules* definiert. Die *submodules* des Modells bestehen aus zwei Hosts, einem Router, der die Pakete zwischen den Hosts weiterleitet und einem IPv4-NetworkConfigurator. Letzterer weist jedem Host eine eindeutige IPv4-Adresse zu und konfiguriert die Routingtabelle des Routers so, dass die Hosts untereinander erreichbar werden. Bei den beiden Hosts handelt es sich um NetPerfMeterHosts, auf denen eine OMNeT++-Version der Applikation NetPerfMeter[2] läuft. Mit NetPerfMeter können über die gängigen Transportprotokolle wie TCP, SCTP und UDP Testdaten versendet und dabei die Performance gemessen werden. Für die Entwicklung des Simulationsmodells von QUIC wurde NetPerfMeter dahingehend erweitert, dass es auch mit dem entwickelten QUIC kommunizieren kann. Die Abbildung 4.1 bildet die Beschreibung der NED-Datei ab.

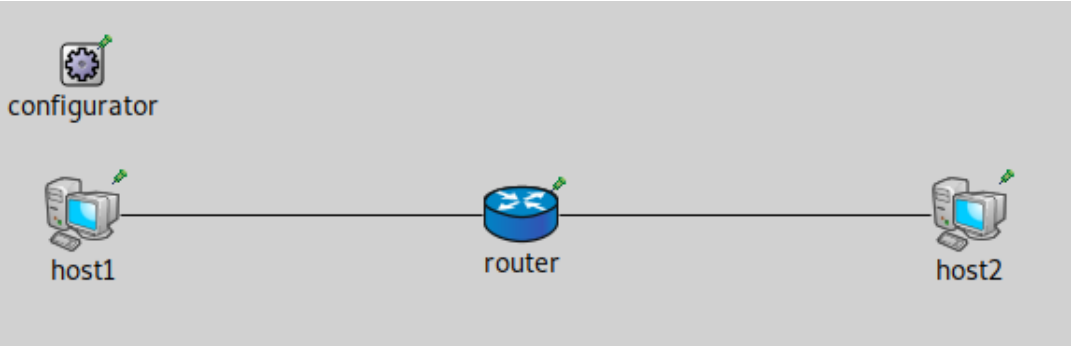


Abbildung 4.1: Grafische Übersicht über den Aufbau der Simulation

[2]https://www.uni-due.de/~be0001/netperfmeter/ – Abgerufen am 18.03.2019

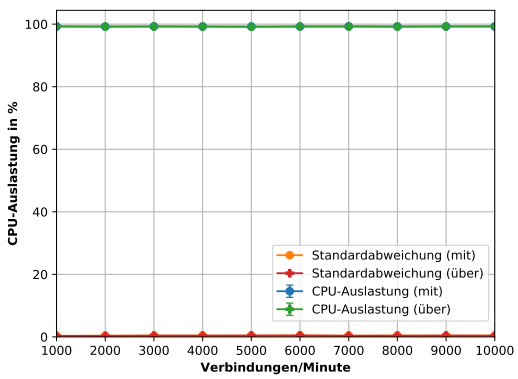
Im Standardablauf von NetPerfMeter initiiert der Client eine Verbindung zum Server über ein vorher in der Konfigurationsdatei *omnet.ini* spezifiziertes Transportprotokoll und sendet Testdaten zum Server. Kurz danach sendet auch der Server Daten, wodurch beide Hosts sich gegenseitig Daten senden und den Empfang bestätigen. In den Experimenten ist das Senden des Servers nicht erforderlich, weshalb dieses Verhalten in NetPerfMeter unterbunden wird. Zur Realisierung des Experiments mit den mehrfachen Verbindungsinitiierungen wurde der Code von NetPerfMeter dahingehend angepasst, dass der Timer zum Starten einer neuen Verbindung nach jedem Ablauf neu gestartet und nicht sofort gelöscht wird. Über die Angabe des nächsten Ablaufzeitpunkts des Timers kann die Frequenz der Verbindungsinitiierungen eingestellt werden. Ebenfalls wurde der serverseitige Paketempfang außer Kraft gesetzt, um unnötige Zustandsübergänge und Antworten auf die Verbindungsinitiierungen zu vermeiden.

4.6 Ergebnisse

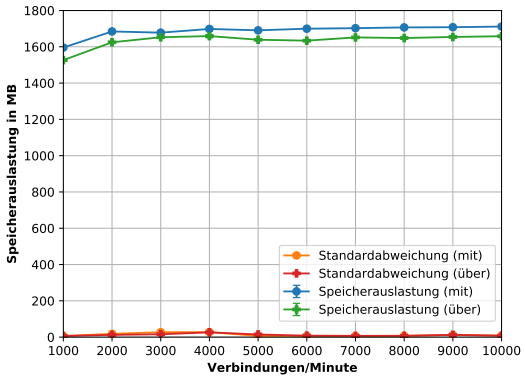
Nach erfolgter Experimentdurchführung wurden für jeden der drei Anwendungsfälle drei Diagramme für die CPU- und Speicherauslastung sowie die Verschlüsselungszeit erstellt. Im Folgenden werden die Ergebnisse vorgestellt und diskutiert.

Experiment 1: Mehrfache Verbindungsinitiierungen durch Client
Der erste Anwendungsfall legt den Schwerpunkt auf die Verbindungsinitiierung und soll die Auslastung des Clients während dieser Phase abbilden. Die Abbildung 4.2 zeigt für jede gemessene Metrik ein Diagramm. Die Y-Achse steht für den Wert der jeweiligen Metrik, während die X-Achse die stetige Erhöhung der initiierten Verbindungen/Minute anzeigt. Jeder Messpunkt verfügt über ein Konfidenzintervall und über eine zusätzlich abgebildete Standardabweichung (roter, orangener Graph), die das Streuungsmaß aller in Verbindung mit dem jeweiligen Messpunkt gemessenen Werte angibt.

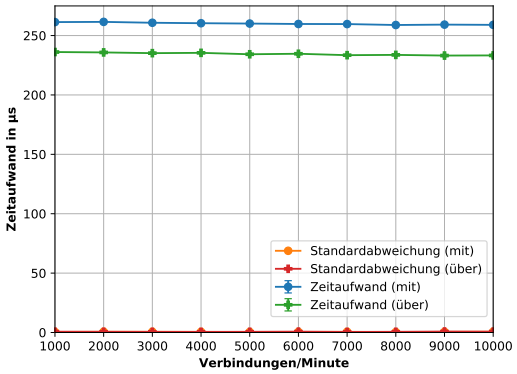
In der Abbildung 4.2a ist die CPU-Auslastung beider Verschlüsselungsansätze zu erkennen. Es zeigt sich, dass die Auslastung sowohl bei QUIC-mit-TLS als auch bei QUIC-über-TLS über die gesamte Laufzeit konstant bei etwas unter 100 % verbleibt und damit keine Unterschiede auszumachen sind. Abbildung 4.2b stellt die Speicherbelegung in Megabyte dar. Hier ist auffällig, dass bei beiden Verschlüsselungsansätzen schon bei einer geringen Rate von 1000 Verb./min eine hohe Speicherbelegung von 1,6 GB (QUIC-mit-TLS) beziehungsweise 1,5 GB (QUIC-über-TLS) festzustellen ist. Diese erhöht sich leicht bei 2000 Verb./min und bleibt bis zum Ende nahezu konstant. QUIC-über-TLS weist



(a) CPU-Auslastung



(b) Speicherbelegung



(c) Zeitaufwand

Abbildung 4.2: Erster Anwendungsfall mit vielen Verbindungsinitiierungen

dabei eine insgesamt etwas geringere Belegung auf. Die letzte Abbildung 4.2c zeigt die benötigte Dauer zur Initialisierung der QPP und der Anforderung des *Initial Packet* an. In diesem Fall verbleibt die Verschlüsselungszeit von QUIC-mit-TLS bei ca. 260 µs, während QUIC-über-TLS mit 230 µs etwas darunter liegt.

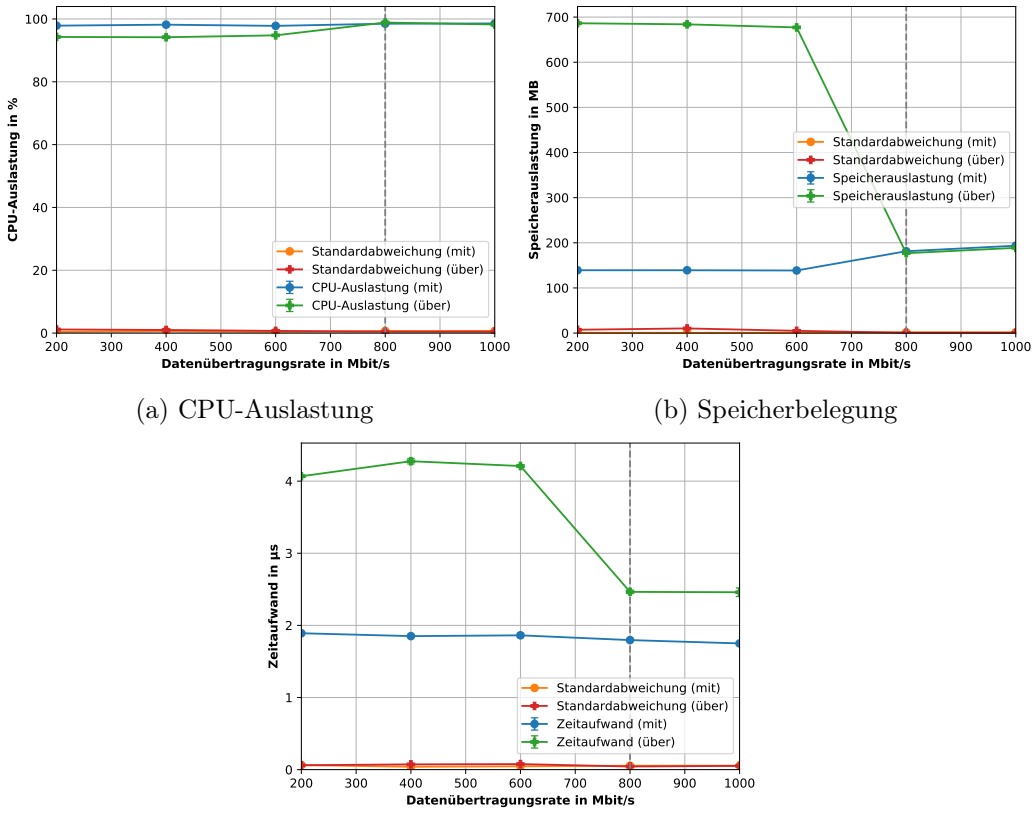
Das Konfidenzintervall für die Mittelwerte der CPU-, Speicherauslastung und den Median der Zeit ist hierbei so klein, dass es auf den Diagrammen nicht erkennbar ist. Dies liegt in der geringen Varianz der Messwerte begründet, welche über den gesamten Anwendungsfall hinweg zu einer geringen Standardabweichung führen.

Es lässt sich festhalten, dass die Abbildung 4.2a keine Aussage zu den Verschlüsselungsansätzen erlaubt. Die Ursache hierfür ist zum einen die geringe Performance der CPU, da bei beiden Ansätzen die CPU-Last annähernd 100 % beträgt. Die Verwendung eines Prozessors mit einer höheren Taktfrequenz oder der Möglichkeit, mehr Arbeitsschritte pro Takt durchzuführen, könnte diesen Umstand womöglich beheben.
Zum anderen besteht ein Problem bei der Messung der CPU-Last. Diese ist nicht nur auf die Initialisierung der QPP und der Erstellung des *Initial Packet* beschränkt, sondern umfasst das gesamte Simulationsmodell mit allen anderen Features, die die Messung beeinflussen, auch wenn der Schwerpunkt auf die Verbindungsinitiierung gesetzt worden ist.
Die Messung der Speicherbelegung ist ebenfalls von dem Einfluss anderer Features, die nicht direkt etwas mit der Verschlüsselung zu tun haben, betroffen. Jedoch steht auf dem Messgerät genug Arbeitsspeicher zur Verfügung, um diesen Mehreinfluss aufzufangen, der bei beiden Ansätzen zu gleichen Teilen während des Experiments vorhanden ist. Aus dem Ergebnis dieser Messung zeigt sich, dass QUIC-über-TLS in der Phase der Verbindungsinitiierung insgesamt weniger Speicher benötigt und damit besser als QUIC-mit-TLS abschneidet.
Beim benötigten Zeitaufwand ist QUIC-über-TLS über die gesamte Messdauer um ca. 30 µs schneller. Auffällig ist, dass bei beiden Ansätzen die benötigte Zeit konstant bleibt. Dies spricht dafür, dass die gemessene Zeit nicht mit der Anzahl der Verbindungen/-Minute korreliert und sich diese Form der Darstellung als nicht geeignet erweist. Die Ursache für das bessere Abschneiden von QUIC-über-TLS beim Speicherverbrauch und der Zeit liegt vermutlich darin begründet, dass die Handshake-Pakete im Internet-Draft-konformen QUIC-mit-TLS mit dem eigenen Handshake-Schlüssel verschlüsselt werden und somit einen zusätzlichen Bedarf an Ressourcen erzeugen. Demgegenüber verursacht QUIC-über-TLS diesen Mehraufwand nicht und benötigt deshalb weniger Ressourcen.

Das Wachstumsverhalten lässt sich aufgrund der vorhin beschriebenen Probleme bei der CPU-Auslastung und dem Zeitaufwand nicht bestimmen. Die Speicherbelegung weist bei 2000 Verb./min eine leichte Erhöhung auf, bleibt aber danach konstant. Dieses Verhalten lässt den Schluss zu, dass der belegte Speicher während der Verbindungsinitiierung so schnell wieder freigegeben wird, dass bei der Messung mit der Erhöhung der Verbindungsinitiierungsrate keine Erhöhung des verbrauchten Speichers festzustellen ist.

Experiment 2: Verschlüsselung von kleinen Datenpaketen

Die Ergebnisse beider Verschlüsselungsansätze im zweiten Anwendungsfall bei einer Datenübertragung mit einer Datengröße von 100 B sind der Abbildung 4.3 zu entnehmen. Zu beachten ist hierbei die gestrichelte Linie bei 800 Mbit/s. Ab dieser Rate trat bei beiden Verschlüsselungsansätzen während der Entschlüsselung ein vorzeitig terminierendes Verhalten auf, das eine Deaktivierung des serverseitigen Paketempfangs bis zum Abschluss des Experiments erforderlich machte, um die Messungen fortsetzen zu können.



(a) CPU-Auslastung

(b) Speicherbelegung

(c) Zeitaufwand

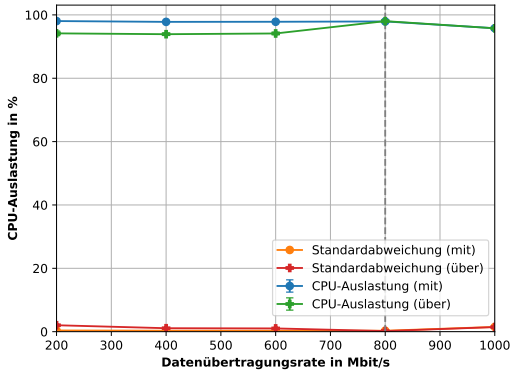
Abbildung 4.3: Zweiter Anwendungsfall mit Datenübertragung (100 B)

Verglichen mit der CPU-Auslastung im ersten Anwendungsfall (Abbildung 4.2a) ergibt sich für QUIC-über-TLS in Abbildung 4.3a bis 600 Mbit/s eine leicht geringere CPU-Auslastung. Ab dem Zeitpunkt der Deaktivierung des serverseitigen Paketempfangs steigt die CPU-Last von QUIC-über-TLS auf das Niveau von QUIC-mit-TLS und verbleibt dort mit zum Experimentende. Beim Speicher weist QUIC-über-TLS eine um 560 MB größere Belegung als QUIC-mit-TLS auf, bevor diese ab einer Datenübertragungsrate von 800 Mbit/s auf den nahezu denselben Wert wie QUIC-mit-TLS fällt (Abbildung 4.3b). Danach steigt die Belegung bei beiden Ansätzen mit einer Erhöhung der Übertragungsrate auf 1 Gbit/s auf knapp 200 MB an. Der Zeitaufwand in Abbildung 4.3c zeigt einen ähnlichen Verlauf wie die Speicherbelegung zuvor, dies allerdings mit dem Unterschied, dass ab 800 Mbit/s das QUIC-über-TLS weiterhin ca. 0,7 µs länger für die Verschlüsselung der Datenpakete benötigt. QUIC-mit-TLS liegt dagegen bei einem konstanten Zeitaufwand von ca. 1,8 µs.

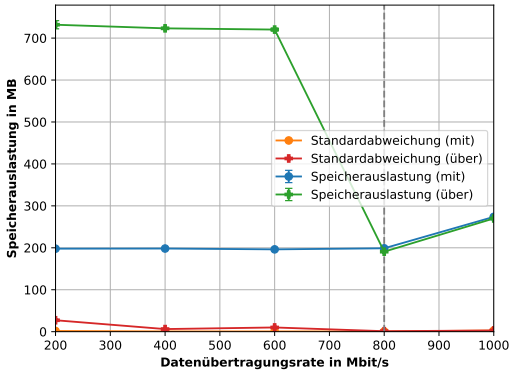
In diesem Anwendungsfall ist die Auslastung der CPU nicht so hoch wie im Fall zuvor, weshalb das QUIC-über-TLS im Verhältnis zu QUIC-mit-TLS eine leicht geringere Auslastung aufweist, welches unverändert bei fast 100 % verbleibt. Trotz der Deaktivierung der serverseitigen Paketverarbeitung bei 800 Mbit/s ist bei QUIC-über-TLS eine Erhöhung der Last auf das Niveau von QUIC-mit-TLS festzustellen. Das deutet darauf hin, dass der rechnerische Mehraufwand durch die Bandbreitenerhöhung größer ist als die Erleichterung durch den Wegfall der serverseitigen Paketverarbeitung. Da das QUIC-mit-TLS bereits an der Grenze der maximalen CPU-Last ist, ist keine weitere Erhöhung möglich.
Besonders fallen die Unterschiede beim Speicher und der Zeit zwischen den Ansätzen auf, die erst mit der Deaktivierung der serverseitigen Paketverarbeitung wegfallen. Während QUIC-über-TLS einen starken Rückgang des Speicherverbrauchs und der Verschlüsselungszeit aufweist, steigt der Speicherverbrauch beim QUIC-mit-TLS leicht an und die Zeit bleibt nahezu konstant. Es liegt hier die Möglichkeit eines implementierungsspezifischen Fehlers beim QUIC-über-TLS nahe, der im zeitlichen Rahmen dieser Arbeit nicht auf eine Ursache zurückzuführen war. Aus diesem Grund finden diese Ergebnisse keine weitere Berücksichtigung in den nachfolgenden Schlussfolgerungen.

Experiment 3: Verschlüsselung von großen Datenpaketen
Im letzten Anwendungsfall in der Abbildung 4.4 wurde eine Datenübertragung mit der maximal zulässigen Menge an Daten pro Paket durchgeführt. Die Verläufe der Graphen spiegeln sich nahezu mit den Verläufen aus der Abbildung 4.3, weshalb auf eine nähere Beschreibung verzichtet wird. Bei 1 Gbit/s ist das Problem aufgetreten, dass der

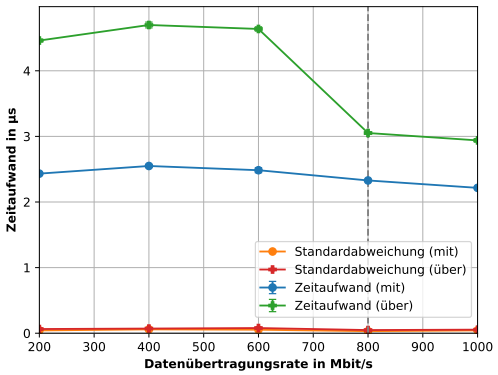
Draining-Timer zum Verbindungsabbau fälschlicherweise bei beiden Verschlüsselungsansätzen abgelaufen ist und damit die Datenübertragung vorzeitig beendet hat. Um die Messung für diesen Messpunkt abschließen zu können, wurde dieser Timer ausgesetzt.



(a) CPU-Auslastung



(b) Speicherbelegung



(c) Zeitaufwand

Abbildung 4.4: Dritter Anwendungsfall mit Datenübertragung (1245 B)

Zum Erreichen der selben Datenübertragungsrate wie in Fall 2 müssen in diesem Fall nicht so viele Pakete gesendet werden (QUIC-mit-TLS 3,8 Mio. mit 1245 B und QUIC-mit-TLS 5,6 Mio. OMNeT++-Nachrichten mit 100 B bei 1 Gbit/s). Das Senden größerer Pakete bewirkt im Vergleich zum zweiten Fall, dass beide Ansätze mehr Speicher (QUIC-über-TLS ca. 40 MB mehr, QUIC-mit-TLS ca. 60 MB mehr) und Zeit (QUIC-über-TLS ca. 0,2 µs länger, QUIC-mit-TLS ca. 0,55 µs länger) benötigen. Der Mehraufwand durch die Verschlüsselung größerer Pakete stellt sich dabei als größer heraus, als die weniger häufig stattfindende Serialisierung der Header-Daten einsparen kann.

Insgesamt lassen sich die Erkenntnisse aus den Experimenten wie folgt zusammenfassen: Die Messung der CPU-Auslastung hat sich in einem begrenzten Umfang als aussagekräf-

tig erwiesen. Bei den Datenübertragungen wies QUIC-über-TLS über weite Teile eine etwas geringere Auslastung auf und war ansonsten wie QUIC-mit-TLS am Limit der CPU-Ressourcen. Diese Mehrbelastung bei QUIC-mit-TLS deutet darauf hin, dass entweder die Serialisierung der Header-Daten nach der Vorgabe des Internet-Drafts einen größeren Rechenaufwand benötigt oder die Implementierung der Verschlüsselung in QUIC-mit-TLS weist eine Abweichung vom erwarteten Implementierungszustand auf, die einen unnötigen Mehraufwand erzeugt.
In der Phase der Verbindungsinitiierung kann QUIC-über-TLS eine konstant geringere Speicherbelegung und Verschlüsselungszeit vorweisen, die vermutlich, wie oben beschrieben, auf den Wegfall der Verschlüsselung zurückzuführen ist. Auch wenn der Unterschied gering ausfällt, so folgt daraus, dass QUIC-über-TLS in Situationen, in denen die anfängliche Latenz von großer Bedeutung ist, dem QUIC-mit-TLS vorzuziehen ist.
Der belegte Speicher und die Verschlüsselungszeit war bei QUIC-mit-TLS über alle Experimente hinweg relativ konstant und zeigte auch bei der Deaktivierung der serverseitigen Paketverarbeitung keine auffälligen Veränderungen. Im Gegensatz dazu nahm QUIC-über-TLS ungewöhnlich viel Speicher während der Datenübertragung ein und benötigte deutlich mehr Zeit zur Verschlüsselung. Bei 800 Mbit/s gab der Ansatz so viel Speicher frei, bis es das Niveau von QUIC-mit-TLS erreicht hat und reduzierte die benötigte Verschlüsselungszeit erheblich. Trotz dessen lag die Verschlüsselungszeit beim QUIC-über-TLS weiterhin über der von QUIC-mit-TLS, was gegen eine Verwendung in latenzsensitiven Datenübertragungen spricht. Es liegt der Verdacht eines implementierungsspezifischen Fehlers bei der Entschlüsselung im QUIC-über-TLS nahe, weshalb ein Vergleich beider Ansätze hier keinen Erkenntnisgewinn bringt.
Das Konfidenzintervall und die Standardabweichung fallen über alle Experimente hinweg sehr gering aus, was dafür spricht, dass zwischen den Messungen nur wenige Unterschiede auszumachen sind.

Entwicklungs- und Wartungsaufwand
Neben der Quantifizierung der Verschlüsselungsansätze ist auch die qualitative Beurteilung ein nicht zu vernachlässigender Aspekt, der im Rahmen der Vergleichsstudie Berücksichtigung finden soll. Wie in Abschnitt 4.2 erläutert, erfolgt diese Beurteilung auf Basis der eigenen Erfahrungen über den Aufwand zur Entwicklung der Verschlüsselungsansätze und einer Einschätzung über den Aufwand zur Wartung.

Das QUIC-mit-TLS stellt den Internet-Draft-konformen Verschlüsselungsansatz dar, der sich durch das Vorhandensein der QPP zu Ver- und Entschlüsselung von QUIC-über-TLS unterscheidet. Das bedeutet, dass der Codeumfang von QUIC-mit-TLS den von QUIC-

über-TLS übersteigt und somit einen größeren Aufwand bei der Entwicklung erfordert. Erschwerend kommt die Tatsache hinzu, dass QUIC sich in aktiver Entwicklung befindet und daher öfter neue Versionen des Internet-Drafts erscheinen, die eine Anpassung des bestehenden Codes erforderlich machen. Dagegen hat QUIC-über-TLS den Vorteil, unabhängig vom Internet-Draft zu sein und stattdessen auf die bereitgestellten Funktionen der TLS-Bibliothek zurückgreifen zu können. Die Verwendung einer weitverbreiteten Bibliothek, wie das hier benutzte OpenSSL, bietet zudem Zugriff auf zahlreiche vollständige Codebeispiele und Best Practices im Umgang damit, die für die QPP nicht bereitstehen. Ebenfalls ist zu berücksichtigen, dass die Funktionen von OpenSSL ausführlich getestet wurden und im breiten Einsatz in der Praxis sind. Über diese Möglichkeiten verfügt QUIC-mit-TLS nicht, weshalb nicht auszuschließen ist, dass Fehler vorhanden sein können, die die Funktion oder gar die Sicherheit des Protokolls gefährden.

Es bleibt festzuhalten, dass QUIC-über-TLS derzeit die entwicklungs- und wartungsärmere Variante darstellt. Zum Zeitpunkt der Standardisierung des Ansatzes in QUIC-mit-TLS und des Vorhandenseins eines finalen RFC fällt der Aspekt der Gefahr einer Änderung der Grundfunktionsweise weg, womit nach dem anfänglichen Aufwand zur Entwicklung der QPP beide Verschlüsselungsansätze in Betracht gezogen werden können.

5 Fazit

Das Ziel dieser Arbeit bestand darin, den Internet-Draft-konformen Ansatz zur Verschlüsselung von Paketen in QUIC, QUIC-mit-TLS, und einen alternativen Ansatz, der vollkommen über eine TLS-Bibliothek läuft, QUIC-über-TLS, zu entwickeln und im Rahmen einer Vergleichsstudie gegenüberzustellen. Die durchgeführten Messungen konnten aufzeigen, dass QUIC-über-TLS im Fall der mehrfachen Verbindungsinitiierungen dem konformen Ansatz leicht überlegen ist und minimal weniger Last auf der CPU während einer Datenübertragung erzeugt. Es eignet sich daher mehr für Anwendungen, bei denen eine geringe anfängliche Latenz von Bedeutung ist. Dafür weist QUIC-mit-TLS während der Datenübertragung eine konstant niedrige Speicherbelegung und Verschlüsselungszeit auf, weshalb es in latenzsensitiven Datenübertragungen die geeignetere Wahl darstellt. Hinsichtlich des Entwicklungs- und Wartungsaufwands kann der Ansatz QUIC-über-TLS Vorteile aufweisen, die jedoch nach erfolgter Standardisierung des Ansatzes QUIC-mit-TLS nicht mehr so stark ins Gewicht fallen.

Die vorzeitige Terminierung der Simulation bei der Messung mit größeren Bandbreiten (serverseitige Paketverarbeitung, *Draining Timer*) und mögliche Fehler in der Implementierung in QUIC-über-TLS hatten zur Folge, dass einige Ergebnisse der Messungen nicht die erwünschte Aussagekraft haben und damit keine weitere Berücksichtigung finden konnten. Es ist deshalb von Interesse, ob die gewonnenen Ergebnisse und Tendenzen bei einer Behebung der Probleme reproduziert und damit eine insgesamt aussagekräftigere Aussage über die Eignung der Ansätze getroffen werden könnte. Ferner wäre eine Fertigstellung aller Features des Simulationsmodells von QUIC für die Aussagekraft einer erneuten Messung förderlich.

Der alternative Verschlüsselungsansatz QUIC-über-TLS war in gewissen Situationen dem QUIC-mit-TLS überlegen und hat somit das Potenzial, als ein alternativer Ansatz zur Verschlüsslung mit QUIC-mit-TLS in Erwägung gezogen zu werden.

A Anhang

Nr.	Ansatz	Fall	Verb./Min	Übertragungsrate (Mbit/s)	Paketgröße (Byte)
1	Über-TLS	1	1000	-	-
2	Über-TLS	1	2000	-	-
3	Über-TLS	1	3000	-	-
4	Über-TLS	1	4000	-	-
5	Über-TLS	1	5000	-	-
6	Über-TLS	1	6000	-	-
7	Über-TLS	1	7000	-	-
8	Über-TLS	1	8000	-	-
9	Über-TLS	1	9000	-	-
10	Über-TLS	1	10000	-	-
11	Mit-TLS	1	1000	-	-
12	Mit-TLS	1	2000	-	-
13	Mit-TLS	1	3000	-	-
14	Mit-TLS	1	4000	-	-
15	Mit-TLS	1	5000	-	-
16	Mit-TLS	1	6000	-	-
17	Mit-TLS	1	7000	-	-
18	Mit-TLS	1	8000	-	-
19	Mit-TLS	1	9000	-	-
20	Mit-TLS	1	10000	-	-
21	Über-TLS	2	-	200	100
22	Über-TLS	2	-	400	100
23	Über-TLS	2	-	600	100
24	Über-TLS	2	-	800	100
25	Über-TLS	2	-	1000	100
26	Mit-TLS	2	-	200	100
27	Mit-TLS	2	-	400	100
28	Mit-TLS	2	-	600	100
29	Mit-TLS	2	-	800	100
30	Mit-TLS	2	-	1000	100
31	Über-TLS	3	-	200	1245
32	Über-TLS	3	-	400	1245
33	Über-TLS	3	-	600	1245
34	Über-TLS	3	-	800	1245
35	Über-TLS	3	-	1000	1245
36	Mit-TLS	3	-	200	1245
37	Mit-TLS	3	-	400	1245
38	Mit-TLS	3	-	600	1245
39	Mit-TLS	3	-	800	1245
40	Mit-TLS	3	-	1000	1245

Tabelle A.1: Auflistung über die Einzelmessungen der Experimente

Literatur

[1] D. Clark, „The Design Philosophy of the DARPA Internet Protocols“, *SIGCOMM Comput. Commun. Rev.*, Jg. 18, Nr. 4, S. 106–114, Aug. 1988, ISSN: 0146-4833. DOI: 10.1145/52325.52336. Adresse: http://doi.acm.org/10.1145/52325.52336.

[2] J. Postel, „Transmission Control Protocol“, RFC Editor, STD 7, Sep. 1981, http://www.rfc-editor.org/rfc/rfc793.txt. Adresse: http://www.rfc-editor.org/rfc/rfc793.txt.

[3] J. Iyengar und M. Thomson, „QUIC: A UDP-Based Multiplexed and Secure Transport“, IETF Secretariat, Internet-Draft draft-ietf-quic-transport-13, Juni 2018, http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-13.txt. Adresse: http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-13.txt.

[4] R. Jim. (27. Juni 2013). Experimenting with QUIC, Adresse: https://blog.chromium.org/2013/06/experimenting-with-quic.html (besucht am 02. 01. 2019).

[5] IETF. (22. Aug. 2016). Charter IETF QUIC, Adresse: https://datatracker.ietf.org/doc/charter-ietf-quic/00-00/ (besucht am 02. 01. 2019).

[6] K. Bhargavan, B. Blanchet und N. Kobeissi, „Verified Models and Reference Implementations for the TLS 1.3 Standard Candidate“, in *2017 IEEE Symposium on Security and Privacy (SP)*, Mai 2017, S. 483–502. DOI: 10.1109/SP.2017.26.

[7] A. Delignat-Lavaud, C. Fournet, M. Kohlweiss, J. Protzenko, A. Rastogi, N. Swamy, S. Zanella-Beguelin, K. Bhargavan, J. Pan und J. K. Zinzindohoue, „Implementing and Proving the TLS 1.3 Record Layer“, in *2017 IEEE Symposium on Security and Privacy (SP)*, Mai 2017, S. 463–482. DOI: 10.1109/SP.2017.58.

[8] J. van Thoor, J. de Ruiter und E. Poll, „Learning state machines of TLS 1.3 implementations“, 2018.

[9] (10. Apr. 2018). GitHub Issue: QUIC is QUIC, and does not stand for anything., Adresse: https://github.com/quicwg/base-drafts/pull/1282 (besucht am 03. 01. 2019).

[10] R. Stewart, „Stream Control Transmission Protocol“, RFC Editor, RFC 4960, Sep. 2007, http://www.rfc-editor.org/rfc/rfc4960.txt. Adresse: http://www.rfc-editor.org/rfc/rfc4960.txt.

[11] M. Belshe, R. Peon und M. Thomson, „Hypertext Transfer Protocol Version 2 (HTTP/2)“, RFC Editor, RFC 7540, Mai 2015, http://www.rfc-editor.org/rfc/rfc7540.txt. Adresse: http://www.rfc-editor.org/rfc/rfc7540.txt.

[12] M. Thomson und S. Turner, „Using Transport Layer Security (TLS) to Secure QUIC“, IETF Secretariat, Internet-Draft draft-ietf-quic-tls-10, März 2018, http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-10.txt. Adresse: http://www.ietf.org/internet-drafts/draft-ietf-quic-tls-10.txt.

[13] L. Ong und J. Yoakum, „An Introduction to the Stream Control Transmission Protocol (SCTP)“, RFC Editor, RFC 3286, Mai 2002.

[14] K. .-.-. G. Grinnem, T. Andersson und A. Brunstrom, „Performance Benefits of Avoiding Head-of-Line Blocking in SCTP“, in *Joint International Conference on Autonomic and Autonomous Systems and International Conference on Networking and Services - (icas-isns'05)*, Okt. 2005, S. 44–44. DOI: 10.1109/ICAS-ICNS.2005.73.

[15] A. Langley, A. Riddoch, A. Wilk, A. Vicente, C. Krasic, D. Zhang, F. Yang, F. Kouranov, I. Swett, J. Iyengar, J. Bailey, J. Dorfman, J. Roskind, J. Kulik, P. Westin, R. Tenneti, R. Shade, R. Hamilton, V. Vasiliev, W.-T. Chang und Z. Shi, „The QUIC Transport Protocol: Design and Internet-Scale Deployment“, in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, Ser. SIGCOMM '17, Los Angeles, CA, USA: ACM, 2017, S. 183–196, ISBN: 978-1-4503-4653-5. DOI: 10.1145/3098822.3098842. Adresse: http://doi.acm.org/10.1145/3098822.3098842.

[16] L. Eggert. (13. Nov. 2018). QUIC WG About Page, Adresse: https://datatracker.ietf.org/wg/quic/about/ (besucht am 04. 01. 2019).

[17] E. Rescorla, „The Transport Layer Security (TLS) Protocol Version 1.3“, RFC Editor, RFC 8446, Aug. 2018.

[18] J. Andress, *The Basics of Information Security: Understanding the Fundamentals of InfoSec in Theory and Practice*. Syngress, 2014, ISBN: 9780128008126.

[19] IEEE, „IEEE Standard Specifications for Public-Key Cryptography“, *IEEE Std 1363-2000*, S. 1–228, Aug. 2000. DOI: 10.1109/IEEESTD.2000.92292.

[20] H. Krawczyk und P. Eronen, „HMAC-based Extract-and-Expand Key Derivation Function (HKDF)“, RFC Editor, RFC 5869, Mai 2010, http://www.rfc-editor.org/rfc/rfc5869.txt. Adresse: http://www.rfc-editor.org/rfc/rfc5869.txt.

[21] D. McGrew, „An Interface and Algorithms for Authenticated Encryption“, RFC Editor, RFC 5116, Jan. 2008, http://www.rfc-editor.org/rfc/rfc5116.txt. Adresse: http://www.rfc-editor.org/rfc/rfc5116.txt.

[22] M. J. Dworkin, „SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC“, Gaithersburg, MD, United States, Techn. Ber., 2007.

[23] A. Varga. (). Simulation Manual OMNeT++, Adresse: https://www.omnetpp.org/doc/omnetpp/manual/ (besucht am 18. 10. 2018).

[24] Opensim Ltd. (). What Is INET Framework?, Adresse: https://inet.omnetpp.org/Introduction.html (besucht am 24. 07. 2018).

[25] W.-T. C. Adam Langley. (6. Dez. 2016). QUIC Crypto, Adresse: https://docs.google.com/document/d/1g5nIXAIkN_Y-7XJW5K45IblHd_L2f5LTaDUDwvZ5L6g/ (besucht am 11. 02. 2019).

[26] A. M. Kakhki, S. Jero, D. Choffnes, C. Nita-Rotaru und A. Mislove, „Taking a Long Look at QUIC: An Approach for Rigorous Evaluation of Rapidly Evolving Transport Protocols“, in *Proceedings of the 2017 Internet Measurement Conference*, Ser. IMC '17, London, United Kingdom: ACM, 2017, S. 290–303, ISBN: 978-1-4503-5118-8. DOI: 10.1145/3131365.3131368. Adresse: http://doi.acm.org/10.1145/3131365.3131368.

[27] P. Megyesi, Z. Krämer und S. Molnár, „How quick is QUIC?“, in *2016 IEEE International Conference on Communications (ICC)*, Mai 2016, S. 1–6. DOI: 10.1109/ICC.2016.7510788.

[28] K. Nepomuceno, I. N. d. Oliveira, R. R. Aschoff, D. Bezerra, M. S. Ito, W. Melo, D. Sadok und G. Szabó, „QUIC and TCP: A Performance Evaluation“, in *2018 IEEE Symposium on Computers and Communications (ISCC)*, Juni 2018, S. 00 045–00 051. DOI: 10.1109/ISCC.2018.8538687.

[29] R. Lychev, S. Jero, A. Boldyreva und C. Nita-Rotaru, „How Secure and Quick is QUIC? Provable Security and Performance Analyses“, in *2015 IEEE Symposium on Security and Privacy*, Mai 2015, S. 214–231. DOI: 10.1109/SP.2015.21.

[30] M. Fischlin und F. Günther, „Multi-Stage Key Exchange and the Case of Google's QUIC Protocol“, in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '14, Scottsdale, Arizona, USA: ACM, 2014, S. 1193–1204, ISBN: 978-1-4503-2957-6. DOI: 10.1145/2660267.2660308. Adresse: http://doi.acm.org/10.1145/2660267.2660308.

[31] T. Jager, J. Schwenk und J. Somorovsky, „On the Security of TLS 1.3 and QUIC Against Weaknesses in PKCS#1 V1.5 Encryption“, in *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, Ser. CCS '15, Denver, Colorado, USA: ACM, 2015, S. 1185–1196, ISBN: 978-1-4503-3832-5. DOI: 10.1145/2810103.2813657. Adresse: http://doi.acm.org/10.1145/2810103.2813657.

[32] M. Hall-Andersen, D. Wong, N. Sullivan und A. Chator, „nQUIC: Noise-Based QUIC Packet Protection“, in *Proceedings of the Workshop on the Evolution, Performance, and Interoperability of QUIC*, Ser. EPIQ'18, Heraklion, Greece: ACM, 2018, S. 22–28, ISBN: 978-1-4503-6082-1. DOI: 10.1145/3284850.3284854. Adresse: http://doi.acm.org/10.1145/3284850.3284854.

[33] VAMPIRE - Virtual Applications and Implementations Research Lab. (25. Jan. 2019). eBACS: ECRYPT Benchmarking of Cryptographic Systems, Adresse: http://bench.cr.yp.to/results-hash.htm (besucht am 12. 02. 2019).

[34] QUIC WG. (2018). QUIC Implementierungen, Adresse: https://github.com/quicwg/base-drafts/wiki/Implementations (besucht am 24. 07. 2018).

[35] K. W. Hennig, „Der Arbeitsablauf“, in *Betriebswirtschaftliche Organisationslehre*. Wiesbaden: Gabler Verlag, 1971, S. 79–109, ISBN: 978-3-663-13747-4. DOI: 10.1007/978-3-663-13747-4_3. Adresse: https://doi.org/10.1007/978-3-663-13747-4_3.

[36] T. Epping, *Kanban für die Softwareentwicklung*. Springer Berlin Heidelberg, 2011. DOI: https://doi.org/10.1007/978-3-642-22595-6.

[37] J. Iyengar und M. Thomson, „QUIC: A UDP-Based Multiplexed and Secure Transport“, IETF Secretariat, Internet-Draft draft-ietf-quic-transport-10, März 2018, http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-

10.txt. Adresse: http://www.ietf.org/internet-drafts/draft-ietf-quic-transport-10.txt.

[38] S. Apel und C. Kästner, „An Overview of Feature-Oriented Software Development“, *Journal of Object Technology*, Jg. 8, Nr. 5, S. 49–84, Juli 2009, (column), ISSN: 1660-1769. DOI: 10.5381/jot.2009.8.5.c5. Adresse: http://www.jot.fm/contents/issue_2009_07/column5.html.

[39] V. Driessen. (5. Jan. 2010). A successful Git branching model, Adresse: https://nvie.com/posts/a-successful-git-branching-model/ (besucht am 21. 06. 2018).

[40] I. Sommerville, *Software Engineering*, Ser. International Computer Science Series. Pearson, 2011, ISBN: 9780137053469.

[41] (1. Dez. 2018). TLS 1.3 Implementations, Adresse: https://github.com/tlswg/tls13-spec/wiki/Implementations (besucht am 02. 01. 2019).

[42] H. L. Jochen Ludewig, *Software Engineering*. dpunkt.verlag, 17. Mai 2013, 688 S., ISBN: 978-3-86491-299-3.

[43] S. Bradner, „Key words for use in RFCs to Indicate Requirement Levels“, RFC Editor, BCP 14, März 1997, http://www.rfc-editor.org/rfc/rfc2119.txt. Adresse: http://www.rfc-editor.org/rfc/rfc2119.txt.

[44] E. Gamma, R. Johnson, R. Helm und J. Vlissides, *Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software*, Ser. Programmer's choice. Pearson Deutschland, 2011, ISBN: 9783827330437.

[45] T. L. Andreas Spillner, *Basiswissen Softwaretest*. Dpunkt.Verlag GmbH, 13. Sep. 2012, ISBN: 978-3-86490-024-2.

[46] Q. WG. (25. Dez. 2018). Test Vector for the Clear Text AEAD key derivation, Adresse: https://github.com/quicwg/base-drafts/wiki/Test-Vector-for-the-Clear-Text-AEAD-key-derivation#draft-10-test-vectors (besucht am 22. 02. 2019).

[47] D. C. Montgomery, *Design and Analysis of Experiments*, 9. Aufl. Wiley, 2017, ISBN: 9781119113478.

[48] P. E. Nogueira, R. Matias Jr. und E. Vicente, „An Experimental Study on Execution Time Variation in Computer Experiments“, in *Proceedings of the 29th Annual ACM Symposium on Applied Computing*, Ser. SAC '14, Gyeongju, Republic of Korea: ACM, 2014, S. 1529–1534, ISBN: 978-1-4503-2469-4. DOI: 10.1145/2554850.2555022. Adresse: http://doi.acm.org/10.1145/2554850.2555022.

[49] S.-A.-A. Touati, J. Worms und S. Briais, „The Speedup-Test: a statistical methodology for programme speedup analysis and computation“, *Concurrency and Computation: Practice and Experience*, Jg. 25, Nr. 10, S. 1410–1426, 2013. DOI: 10.1002/cpe.2939. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/cpe.2939. Adresse: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.2939.

[50] A. S. Tanenbaum und H. Bos, *Modern Operating Systems*, 4. Aufl. Upper Saddle River, NJ, USA: Prentice Hall Press, 2014, ISBN: 9780133591620.

[51] R. M. Love. (2004). taskset(1) - Linux man page, Adresse: https://linux.die.net/man/1/taskset (besucht am 23. 03. 2019).

[52] T. Brecht. (2002). SMP IRQ Affinity, Adresse: https://cs.uwaterloo.ca/~brecht/servers/apic/SMP-affinity.txt (besucht am 23. 03. 2019).

[53] R. J. Wysocki. (2017). CPU Performance Scaling, Adresse: https://www.kernel.org/doc/html/v4.14/admin-guide/pm/cpufreq.html (besucht am 24. 03. 2019).

[54] M. Kerrisk. (30. Apr. 2018). Linux Programmer's Manual - sysfs, Adresse: http://man7.org/linux/man-pages/man5/sysfs.5.html (besucht am 24. 03. 2019).

[55] R. J. Wysocki. (2017). intel_pstate CPU Performance Scaling Driver, Adresse: https://www.kernel.org/doc/html/v4.13/admin-guide/pm/intel_pstate.html (besucht am 24. 03. 2019).

[56] D. Brodowski, N. Golde, R. J. Wysocki und V. Kumar. (2017). Linux CPUFreq Governors, Adresse: https://www.kernel.org/doc/Documentation/cpu-freq/governors.txt (besucht am 24. 03. 2019).

[57] S. Prata, *C++ Primer Plus*, Ser. Developer's Library. Pearson Education, 2011, ISBN: 9780321776402.

[58] M. Kerrisk. (6. März 2019). Linux Programmer's Manual - proc, Adresse: http://man7.org/linux/man-pages/man5/proc.5.html (besucht am 13. 03. 2019).

[59] M. Kerrisk. (2019). User Commands - top, Adresse: http://man7.org/linux/man-pages/man1/top.1.html (besucht am 13. 03. 2019).

[60] Phusion Holding B.V. and contributors to the Passenger Library. (). Accurately measuring memory usage, Adresse: https://www.phusionpassenger.com/library/indepth/accurately_measuring_memory_usage.html (besucht am 13. 03. 2019).

[61] Xfce Development Team. (3. Juni 2018). xfce4-taskmanager, Adresse: https://goodies.xfce.org/projects/applications/xfce4-taskmanager (besucht am 13. 03. 2019).

[62] (). htop - an interactive process viewer for Unix, Adresse: https://hisham.hm/htop/ (besucht am 25. 03. 2019).

[63] N. M. Josuttis, *The C++ Standard Library: A Tutorial and Reference*, 2. Aufl. Addison-Wesley Professional, 2012, ISBN: 9780321623218.

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 Mastertarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Eine Vergleichsstudie zu TLS-1.3-Bibliotheken in einer QUIC-Simulationsumgebung

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