HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

TLS 1.3 for strongSwan:

A Client-Side Prototype

Technical Documentation

Study Project, Spring Term 2020

Departement of Computer Science
University of Applied Sciences Rapperswil (HSR)
www.hsr.ch

Authors:
Pascal Knecht (`pascal.knecht@hsr.ch`, @ryru_foo)
Méline Sieber (`meline.sieber@hsr.ch`, @MelineSieber)
Advisors:
Prof. Dr. Andreas Steffen; Tobias Brunner, HSR

Date: 29th May 2020

Contents

List of Figures

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

FHO Fachhochschule Ostschweiz

Semester Project 2020

TLS 1.3 for strongSwan

Students: Pascal Knecht, Méline Sieber

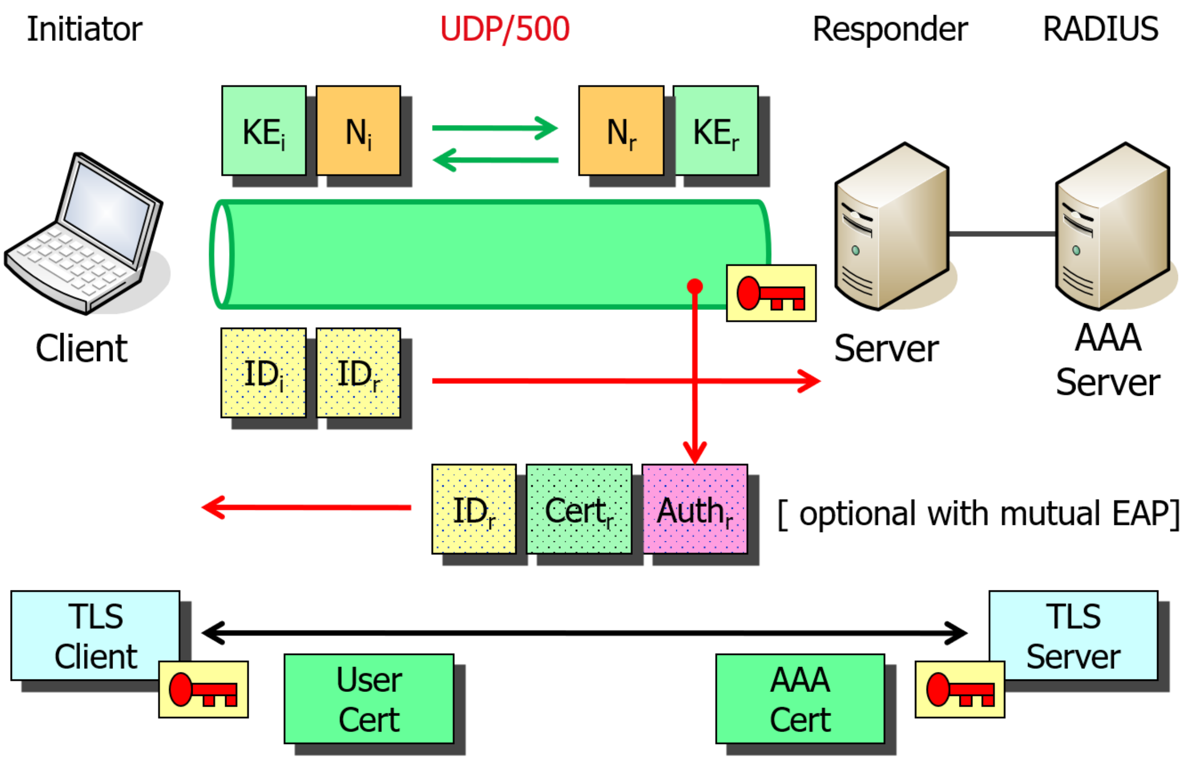
Advisor: Prof. Dr. Andreas Steffen

Issue Date: Monday, February 17th 2020

Submission Date: Friday, May 29th 2020

Introduction

The strongSwan open source VPN solution has a Transport Layer Security (TLS) stack of its own implemented by the strongSwan `libtls` library [1]. TLS is used by the strongSwan IKEv2 daemon for EAP-based authentication (EAP-TLS, EAP-TTLS, EAP-PEAP) as shown in the figure below but `libtls` could potentially be employed by any stand-alone TLS application.



Due to several deficiencies the TLS 1.2 version supported by strongSwan has been deprecated and therefore an upgrade to the latest TLS 1.3 standard defined by RFC 8446 [2] is urgently required.

The task for the semester project defined by this document is to extend the strongSwan `libtls` library with a minimal viable TLS 1.3 implementation.

Objectives

Mandatory:

- Getting acquainted with the existing TLS source code written in the C programming language and using some strongSwan-specific object-oriented macros as implemented by the strongSwan `libtls` library [1].
- Understanding the TLS 1.3 standard as defined by RFC 8446 [2] and identifying the basic differences and extensions relative to TLS 1.2.
- Implementation of a minimal viable TLS 1.3 client without client-side authentication.
- Negotiation of fallback to legacy TLS 1.x versions.
- Testing of minimal viable TLS 1.3 client against TLS 1.3 OpenSSL server.
- Interoperability of minimal viable TLS 1.3 client with TLS 1.3-enabled web sites (Google, Facebook, etc.).

Optional:

- Implementation of a minimal viable TLS 1.3 server
- Implementation of X.509 certificate-based TLS 1.3 client authentication

Links

[1] strongSwan `libtls` library, github source code repository
https://github.com/strongswan/strongswan/tree/master/src/libtls

[2] RFC 8446 "The Transport Layer Security (TLS) Protocol Version 1.3", August 2018
https://tools.ietf.org/html/rfc8446

Rapperswil, February 17th 2020

a. steffen

Prof. Dr. Andreas Steffen

1. Abstract

Introduction: The Transport Layer Security protocol (TLS) secures network connections between a client and a server. It encrypts and authenticates data from higher-level protocols such as the Hypertext Transfer Protocol (HTTP), and guarantees that the information transmitted remains confidential and unmodified. The most widely used TLS version today is still version 1.2 from 2008 (RFC 5246), even though 1.3 has been released in 2018 (RFC 8446). The strongSwan project, which is maintained by the University of Applied Sciences Rapperswil (HSR), is an open-source IPsec implementation written in C. strongSwan features its own TLS stack in the library libtls. It enables client-authentication via various EAP authentication methods (TLS, TTLS, PEAP), which is used to establish an IKEv2 connection. However, libtls supports only TLS up to version 1.2.

Objective: This project aims to implement the client-side of TLS 1.3 in libtls and to update the TLS stack. The concrete goal is to successfully perform a minimal handshake and then exchange application data between a strongSwan client and a server running TLS version 1.3. As part of this minimal handshake, it is necessary to integrate new or adapt existing messages that are exchanged between client and server. In addition, TLS 1.3 requires fundamental changes to the cryptographic mechanisms that enable a secure and authenticated encryption.

Until a connection is established, the handshake passes through various states in a state machine. This has considerably changed in the new version, which also implies that the handshake flow and state machine must be adapted too. Our scope includes three optional features: The server-side in a TLS handshake, client-side authentication and remaining non-mandatory extensions.

Result: The updated client implementation in libtls can successfully perform a minimal handshake with a server running TLS 1.3. It has been successfully and extensively tested with external TLS 1.3 servers such as those from Google or Facebook, but also with a local OpenSSL server. During implementation the new cryptographic mechanisms proved to be more difficult that originally anticipated. Especially the HKDF (HMAC-based Extract-and-Expand Key Derivation Function, RFC 5869) was an unexpected major challenge, especially since it is only marginally described in the TLS specification. Due to these difficulties, only the minimal handshake was implemented, the client-side authentication and the server-side was omitted.

For future work, the HKDF implementation needs attention: It is functional, but the code needs to be refactored. The features defined as optional, i.e. the server-side and client-side authentication, were not implemented due to time constraints. Still, they are relevant to the strongSwan project and will hopefully be completed as a Bachelor thesis.

2. Introduction

2.1. Overview

Rome wasn't built in a day, and the Transport Layer Security (TLS) in version 1.3 neither. TLS is the standard that secures connections between a server and a client and is just above layer four of the OSI model, the transport layer (see figure 2.1).[1] The most popular TLS version 1.2 was defined in 2008 (RFC 5246), but only found broad adoption around 2014.[2] In that same year, work on TLS 1.3 started. After several revisions, it was written down in RFC 8446 (2018), with significant departures from its predecessor. It took thus roughly ten years to introduce a new version of the TLS protocol. Rome took longer to build, but to be fair, TLS is no city. The strongSwan project spans a similar time frame and secures communications on networks as well. Its IPSec implementation authenticates and encrypts network traffic on layer three, the network layer. Originally, strongSwan was a 2005 fork of the disconintued FreeS/WAN project by John Gilmore.[3] The project features its own TLS stack, yet only up to TLS version 1.2. The present work aims to implement RFC 8446 in strongSwan on the client-side and thus provides an update to the latest TLS version 1.3.

In the remaining sections of this chapter we shortly introduce the main features of TLS 1.3 and its major differences to its predecessor. We also describe the strongSwan project in greater detail and why there is a need for a TLS stack in an IPsec solution. Lastly, to comply with the time frame given, we must narrow down our goals to a manageable scope.

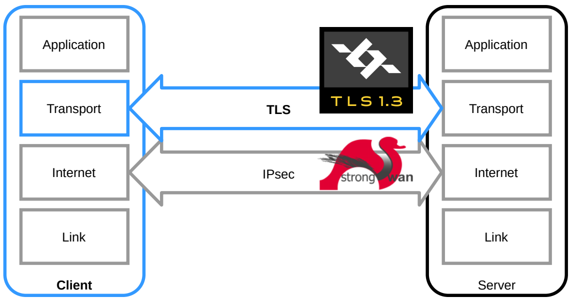


Figure 2.1.: The location of TLS 1.3 and strongSwan in the OSI model.

After the introduction, we turn to our implementation of RFC 8446. The strongSwan code is well-documented and follows its own coding style. However, architecture information is lacking, especially in the form of diagrams and illustrations. This encouraged us to document the existing architecture of the TLS stack code. We then justify our design decision that had a greater impact on the overall code. The next sections describe how new cryptographic processes are now part of the TLS stack in strongSwan and other notable changes to the existing code.

Lastly, we present our final product in the remaining chapter and how our design decisions and the overall construction process have influenced the outcome.

[1] If not indicated otherwise, all illustrations are our own.
[2] Sch17.
[3] Ste05.

2.2. The Transport Layer Security Protocol (TLS)

The Transport Layer Security protocol (TLS) secures communication between client and server and is one of the fundamental elements in today's internet. It encrypts and authenticates data from higher-level protocols such as the Hypertext Transfer Protocol (HTTP), and guarantees that the information transmitted from one endpoint to another remains confidential and unmodified.[4]

Communication between client and server is achieved by exchanging messages of the Record Protocol, which in itself encapsulates the Handshake protocols, the Alert protocol, the ChangeCipherSpec protocol and the Application Data protocol.[5] In order to establish a secure connection and transmit encrypted data, the protocol passes through three stages. In a first stage, a handshake establishes how client and server want to communicate with each other. This is the task of the Handshake protocol of TLS, which "authenticates the communicating parties, negotiates cryptographic modes and parameters, and establishes shared keying material."[6] During the second stage, the actual data is sent, for example the content of a website. This, on the other hand, is the task of the Application Data protocol, which uses the parameters established during the handshake to protect the traffic.[7] This third stage is marked by the termination of the connection by either one of the peers.

The project presented here focuses on the handshake process. Figure 2.2 shows a handshake in TLS version 1.2:

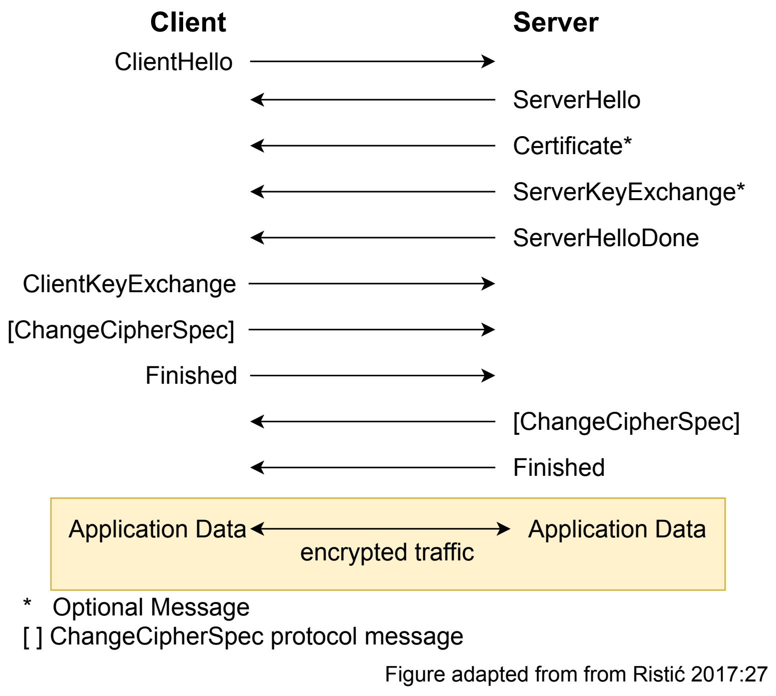


Figure 2.2.: Full handshake in TLS 1.2 with server authentication.

In short, the client initiates a connection with a `ClientHello` message and sends its cryptographic capabilities to the server, notably the protocol version and cipher suites it supports.

[4] Cf. [Aum18, p. 236] and [Fal12, p. 876].
[5] Fal12, p. 877.
[6] Res18, p. 6.
[7] Res18, p. 6.

The server sends its own information back in several messages and signals the end of the negotiation with a `ServerHelloDone`. Then the client sends its cryptographic information in a `ClientKeyExchange` message. Note that the messages exchanged up to this point are still sent in plaintext.

If the handshake was successful so far, enough cryptographic material is available to encrypt the ensuing traffic. The client signals this state with a message from the `ChangeCipherSpec` protocol. A `Finished` message concludes the handshake. After the server has responded with the same messages, encrypted application data can flow between the two endpoints, sent using the Record protocol.[8]

TLS 1.2 was an important step towards securing communication over the Internet, yet has its flaws and disadvantages. The handshake flow above is only one out of many and the complexity of TLS 1.2 made its implementation prone to errors, bugs and security issues.[9] The current version of TLS 1.3 presents a major overhaul of the previous TLS version 1.2. The two most notable changes relate to the handshake process, and the encryption and authentication schemes to be used.

The handshake, which establishes a connection between client and server, has been greatly simplified in TLS 1.3. This becomes most apparent when we compare the state machines of the two TLS versions: The official specification does not list any state machine at all, with the various possible state machines leading to some messy graphics and faulty implementations.[10] On the other hand, TLS 1.3 devotes a whole appendix chapter that lists the various possible state machines.[11]

In comparison to figure 2.2, server and client exchange fewer messages:

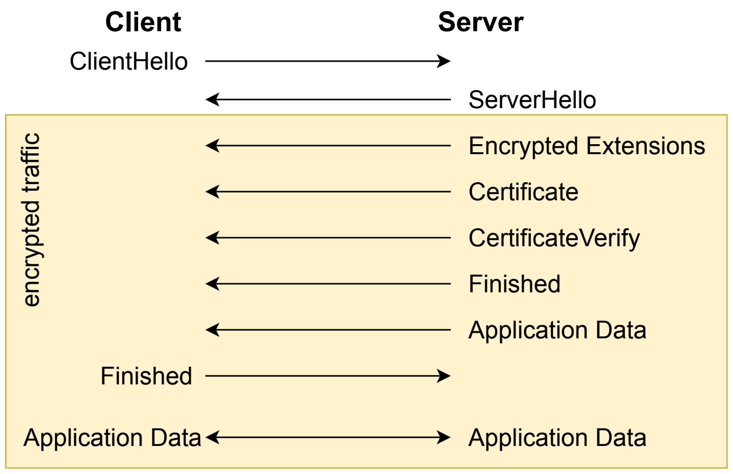


Figure 2.3.: Full handshake in TLS 1.3 with server authentication.

[8]We presume a basic knowledge of TLS here and note only the most important aspects, and major differences and improvements in TLS 1.3 that are also relevant to our work. More detailed changes that became apparent during implementation are described in the following chapter. A good introduction to TLS 1.2 can be found in [Ris17].

[9]A cursory search lists 655 security vulnerabilities in Mitre's CVE database, including Crime (2012), the notorious Heartbleed (2014), Poodle (2015) and other attacks. `https://cve.mitre.org/cgi-bin/cvekey.cgi?keyword=TLS`, visited 2020-04-23.

[10]Beu17.

[11]Res18, p. 120.

The client basically announces "Hello, I want to establish a connection" and provides the necessary key material, the server answers with its own key material and other information needed, and the handshake is over. Essentially, this is a major speed-up, since it eliminates a full round-trip between client and server. The `ChangeCipherSpec` message, which formerly announced the transition to encrypted application data, is now eliminated. It has only been preserved to enable TLS 1.3 connections over middleboxes such as firewalls and intrusion detection systems and is not processed further.[12] In addition to this speed-up, TLS 1.3 even allows for a zero-round-trip (0-RTT) if the client has previously connected to the server.

The other important improvement in figure 2.3: Data is encrypted much earlier in the whole client-server message exchange. The encryption happens right after the `ServerHello`, since the key material is earlier ready than in TLS 1.2. All following information such as the server certificate are now encrypted, while being sent in plaintext in previous TLS versions.

What is not visible in the figure above is the expanded use of extensions. They already existed in previous TLS versions, but with the latest one a lot of information has been moved to new or pre-existing extensions, notably also the supported TLS versions of client and server.

The major change in TLS 1.3 is only visible under the hood, however: Encryption algorithms, hashing and signature algorithms have been pruned and re-organized. This change is most visible in the list of cipher suites of which a client can offer a selection to the server. The page-long cipher suite combinations[13] have been reduced to just five recommended cipher suites.[14] Cipher suites in TLS 1.2 are of the following format, with an example just below:

```
TLS_[key exchange]_[authentication]_with_[cipher]_[mac or prf]
```

```
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
```

This is thus a cipher suite that uses ECDHE for the key exchange, RSA for authentication, 128bit-AES in GCM mode as bulk cipher to decrypt the application data, and SHA-256 as hash function. In TLS 1.3, there remain only five cipher suites using Authenticated Encryption with Associated Data[15] (AEAD):

```
TLS_[AEAD]_[hash]
```

```
TLS_AES_128_GCM_SHA256
```

This is the 128bit AES-GCM authenticated encryption algorithm together with a SHA256-based pseudo-random function (PRF).[16] The key exchange and signature algorithm field have been moved to their own respective extension. In addition, the recommended ciphers for key exchange all support forward secrecy. If an attacker gains the private key of a server or client, she is not able to decrypt past TLS sessions.

Another notable change is the key derivation: TLS 1.3 uses a new HMAC-based Key Derivation Function (HKDF) that has been defined in RFC 5869 in 2010.[17] We discuss the consequences of the new AEAD cipher suites and the new HDKF in the chapter 3.3.

[12] Res18, pp. 77, 140.

[13] The IANA lists over more than 300 cipher suites: `https://www.iana.org/assignments/tls-parameters/tls-parameters.xhtml`, visited on 2020-04-23.

[14] Res18, p. 133.

[15] For a detailed description of AEAD see [Aum18, p. 148]

[16] To be more precise: The hashing algorithm is used to build an HMAC (Keyed-Hash Message Authentication Code) within the HKDF (Hash-based Key Derivation Function). Both concepts are explained in the implementation chapter.

[17] Kra10.

TLS 1.3 brings many new advantages and implies a quick adoption. Holz et al. note that a couple of providers and big internet corporations were able to quickly implement TLS 1.3, e.g. Cloudflare, Google and Facebook.[18] Yet smaller providers have been rather slow in adopting the new TLS version as the authors note. We hope to speed up the process and at least lay the foundation for the strongSwan project for TLS 1.3 support.

[18]Hol19.

2.3. strongSwan and TLS

The strongSwan project is an open-source IPsec implementation. Originally, it was based on FreeS/WAN, but was completely rewritten in an object-oriented coding style in the programming language C.[19] IPsec features so-called "Security Associations" (SA), with which two peers establish security attributes, for example encryption algorithm and keys. In strongSwan, IKE and IKEv2 (Internet Key Exchange) are the protocols to create these Security Associations. In strongSwan, the `charon` library implements the IKEv2 protocol and acts as a keying daemon.[20] Since strongSwan version 4.5.0, `charon` supports EAP-TLS[21] to mutually authenticate client and server with certificates.[22]

The Extensible Authentication Protocol (EAP)[23] is a framework that allows a broad set of methods to authenticate peers. In terms of TLS, the methods EAP-TLS, EAP-TTLS and EAP-PEAP are relevant:

- **EAP Transport Layer Security (EAP-TLS)** as defined in RFC 5216 uses the handshake protocol of TLS to authenticate both peers with certificates, while the TLS encryption functionality itself is not used.[24] A Public Key Infrastructure (PKI) is required and must be able to provide a client certificate.[25]
- **EAP Tunneled Transport Layer Security (EAP-TTLS)** as defined in RFC 5281 first establishes a TLS connection between client and server and then authenticates each other in a second step over this encrypted channel. A certificate is only mandatory for the server, but optional for the client. After the secured tunnel is established, other authentication mechanisms can be used such as EAP.[26]
- **Protected EAP (PEAP)** as developed by Cisco Systems, Microsoft and RSA Security[27], is similar to EAP-TTLS and only requires a server-side certificate and optionally client-side certificate to establish a secure TLS connection. Afterwards EAP messages are sent encrypted over the connection.[28]

strongSwan itself implements a range of EAP methods and currently supports all TLS methods that are offered by EAP and listed above. The TLS library of strongSwan is further used for the "Posture Transport Protocol over TLS" (PT-TLS).[29] These use cases are the main reason why a TLS stack is present in strongSwan.

The TLS stack has been explicitly written from scratch for strongSwan as the library `libtls`. TLS version 1.0, 1.1 and 1.2 and most of the associated cipher suites and cryptographic primitives are currently supported.

Figure 2.4 shows which components rely on this TLS stack:

[19] str18.
[20] str17.
[21] Sim08.
[22] str15.
[23] Wik.
[24] Sta17, p. 165.
[25] Fal12, p. 838, Note that EAP-TTLS and PEAP do not need client certificates for authentication.
[26] Sta17, p. 165.
[27] `https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol#PEAP`, visited 2020-05-23.
[28] `https://security.stackexchange.com/questions/147344/eap-tls-vs-eap-ttls-vs-eap-peap/149643`, visited on 2020-05-23.
[29] San15.

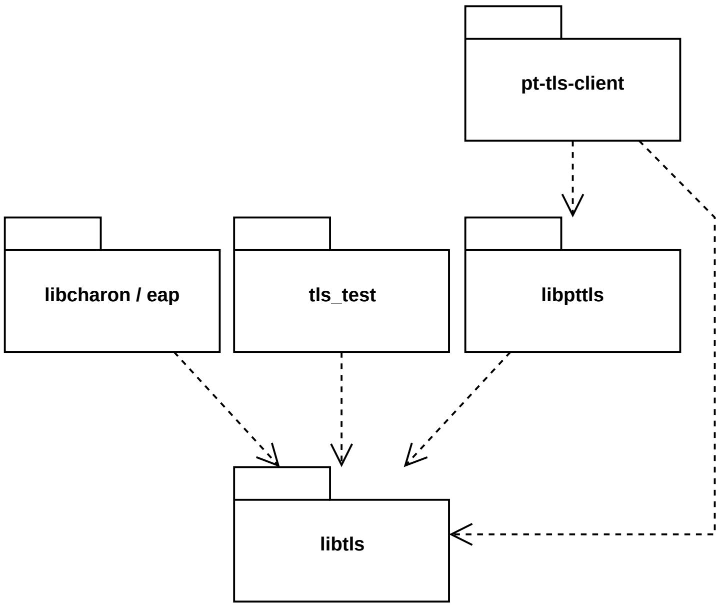


Figure 2.4.: Package diagram of the components `libcharon`, `libpttls` and `tls_test` which rely on `libtls`.

- The `eap` plug-in within the library `libcharon` uses TLS to implement mutual authentication over TLS (EAP-TLS, EAP-TTLS and PEAP).
- `tls_test` is a simple command-line executable that allows to set up a TLS connection as client or server. Once a TLS channel is set up, it allows each side to transfer data to each other similar to a Telnet session.
- `libpttls` is the library that implements the Posture Transport Protocol over TLS (PT-TLS).
- `pt-tls-client` is a concrete client that uses `libpttls` and `libttls`.

The last three components use TLS in a "regular" way, which means a client builds an encrypted connection to a server and then communicates over this secured channel.

2.4. Project Scope

TLS 1.3 is defined in an extensive RFC standard[30] to achieve a minimal viable product (MVP), we mostly follow the compliance requirements as detailed in chapter 9 of the RFC.

2.4.1. Minimal Viable Product

The goal of this work is to implement the client-side components of a minimal TLS 1.3 handshake to a TLS 1.3-compliant server such as OpenSSL 1.1.1c.

This includes the mandatory new cryptographic computations, state machine, messages and extensions. The code to be written resides mainly in the strongSwan TLS library `libtls`. The code is written for strongSwan version 5.8.4.

Messages

Our minimal implementation uses the following messages:

- `ClientHello`
- `HelloRetryRequest`
- `ServerHello`
- `EncryptedExtensions`
- `Certificate`
- `CertificateVerify`
- `Finished`

Extensions

The RFC standard defines these mandatory extensions which must be implemented for a valid TLS 1.3 client:[31]

- `server_name(0)`: Defined in RFC 6066 and already implemented in strongSwan
- `supported_versions(43)`: New in TLS 1.3, chapter 4.2.1
- `cookie(44)`: New in TLS 1.3, chapter 4.2.2
- `signature_algorithms(13)` and `signature_algorithms_cert(50)`: New in TLS 1.3, chapter 4.2.3
- `supported_groups(10)`, chapter 4.2.7
- `key_share(51)`: New in TLS 1.3, chapter 4.2.8

[30] Res18.
[31] Res18, p. 103.

State Machine

The state machine below shows which part we implement to achieve a minimal viable product, drawn in green colour.[32]

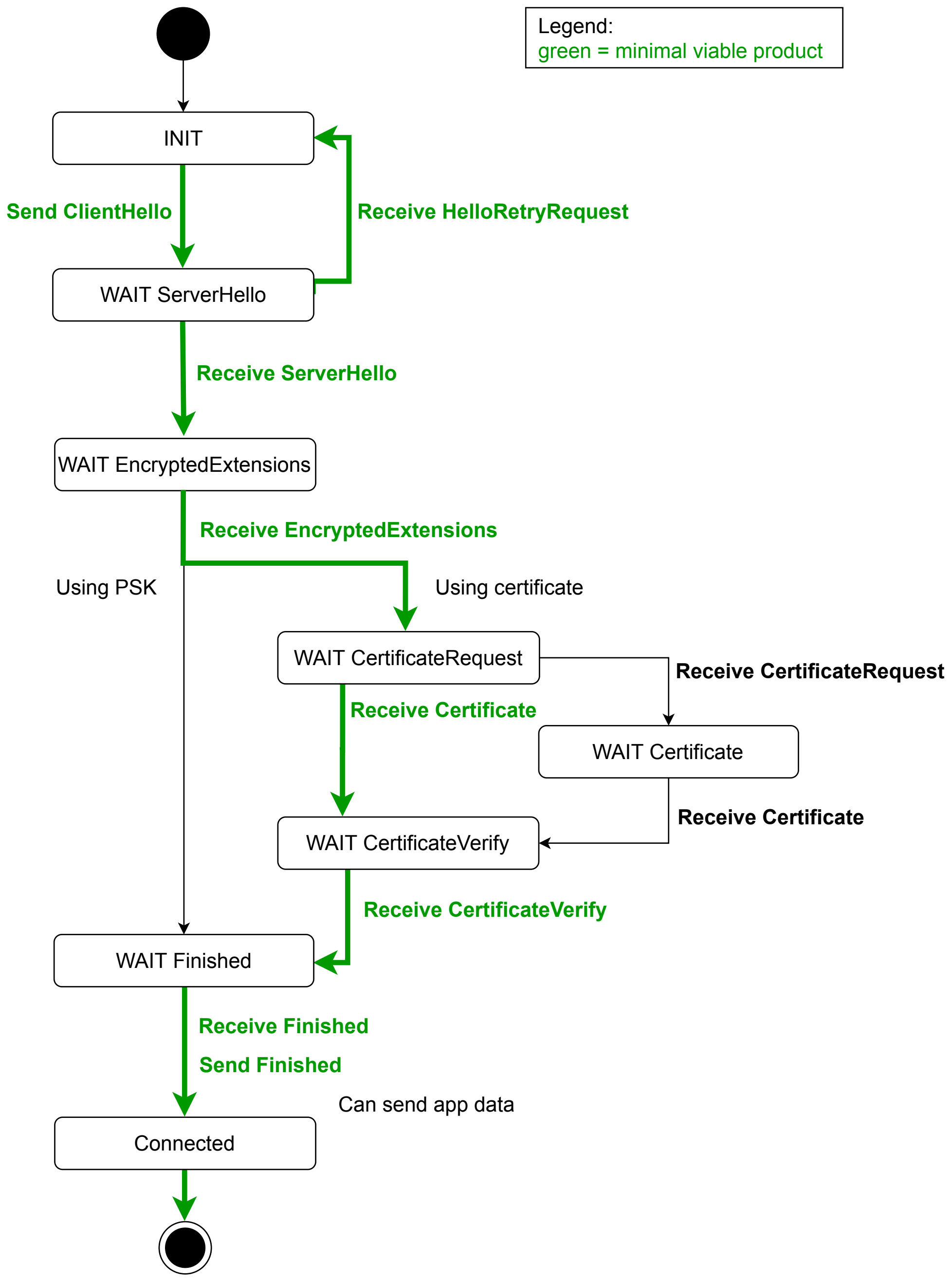


Figure 2.5.: Client-side state machine for a minimal viable product (green).

[32]Res18, 120ff, The state machines shown are slightly simplified to improve readability.

2.4.2. Optional Features

There are three optional features that can be added to the MVP, listed in order of priority: server-side implementation, client-side authentication, and the remaining non-mandatory TLS 1.3 extensions.

Option 1: Server-Side Implementation

The existing TLS library in strongSwan also provides the server role of TLS 1.2 and plays a vital part in how strongSwan is used. Figure 2.6 shows the server-side state machine to our client-side MVP in orange colour.

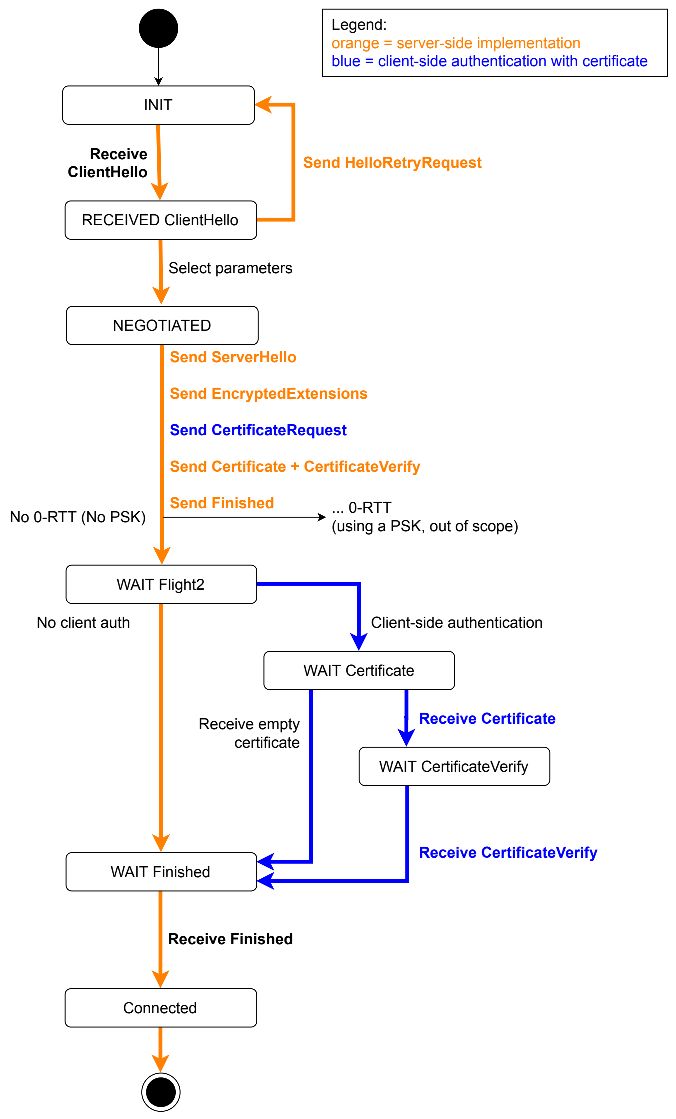


Figure 2.6.: Server-side implementation (orange color).

Option 2: Client-Side Authentication with Certificate

EAP entails the concept of mutual authentication, where EAP-TLS provides the client-side authentication via a X.509 certificate that is used in a TLS connection. Figures 2.7 and 2.6 show the path through the client and server state machines in blue colour.

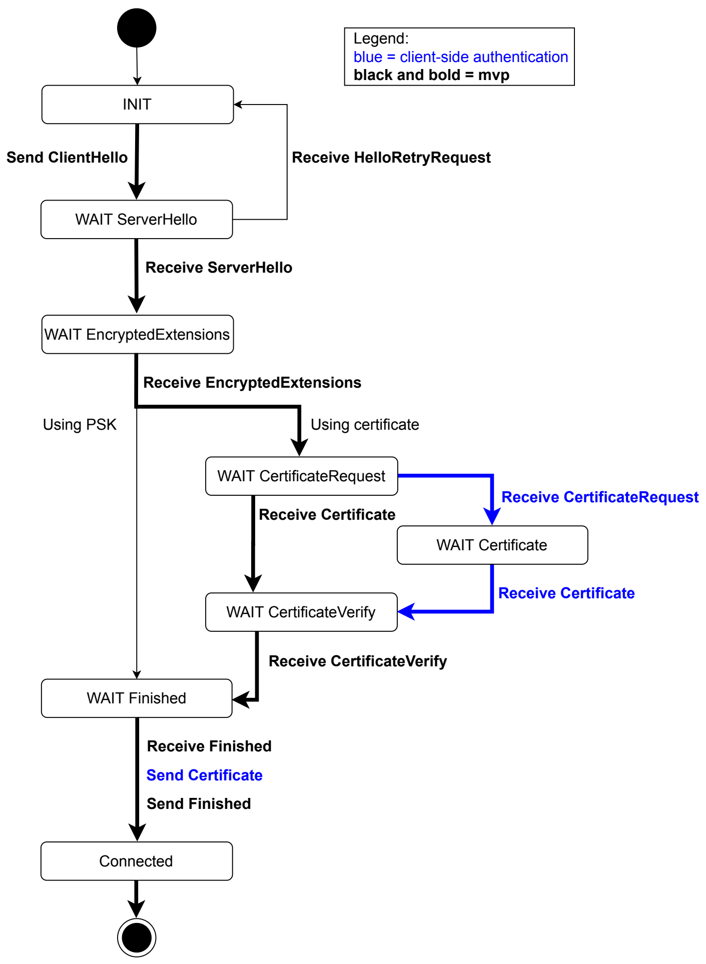


Figure 2.7.: Client-side authentication with certificate (blue color).

Option 3: Further TLS 1.3 Extensions

The following extensions are new in TLS 1.3, but not mandatory to be compliant.

- `early_data(42)`
- `certificate_authorities(47)`
- `oid_filters(48)`
- `post_handshake_auth(49)`

2.4.3. Out of Scope

In order to keep the scope small, we defined these points to be out of scope:

- Middleboxes (for example reverse and forward proxies, firewalls, load balancers etc.)
- Pre-Shared Key (PSK) that allows a zero-roundtrip (0-RTT)

3. Implementation

In the following sections we describe how the minimal viable product was incorporated in the existing TLS stack of the strongSwan project. We start with a short description of the existing code base in `libtls` and explain our main design choice. The remaining sections show our major additions to and adaptations of the code base: The new HMAC-based key derivation function (HKDF) and the Authenticated Encryption with Associated Data (AEAD), as well as the new handshake and state machine. The chapter closes with a section on sequence numbers.

3.1. Architecture Overview of libtls

Our implementation extends the `libtls`-library of the strongSwan project. The library encompasses roughly 11‘200 lines of code, spread over 17 .c- and 19 h-files.[1] In order to make informed decisions about the implementation and to seamlessly integrate the new TLS stack, it was necessary to map and visualise the existing code.

3.1.1. Class Diagram

The existing code follows a special C-syntax that allows a pseudo-object-oriented coding style.[2] The object-oriented code is achieved in the following way: Every .h- and corresponding .c-file is a class, e.g. the Crypto class corresponds to the files `tls_crypto.c` and `tls_crypto.h`. Header-files without a corresponding .c-file are treated as interfaces, e.g. the interface Handshake (`tls_handshake.h`) is implemented by the classes Peer and Server. Every class follows the same structure, using the Crypto-class as an example:

Private data of a class is indicated by the prefix `private` as in `struct private_tls_crypto_t`. An object is instantiated using the `create`-function and a special macro called `INIT`, e.g. `tls_crypto_t *tls_crypto_create(tls_t *tls, tls_cache_t *cache)`. Since it is created on the heap, `destroy()` takes care of the memory management when the object is no longer needed.

The public methods of the Crypto class are of type `struct tls_crypto_t` and implemented using a macro called `METHOD`. It has the following schema:[3]

```
METHOD([class name], [method name], [return value], [private this],
       [list of arguments]...)
```

[1]Measured using the Statistic plugin for IntelliJ CLion.

[2]The project's own coding guidelines have more detailed descriptions about the chosen style. The explanations here only describe what is necessary to understand the diagrams [Hut05, p. 56].

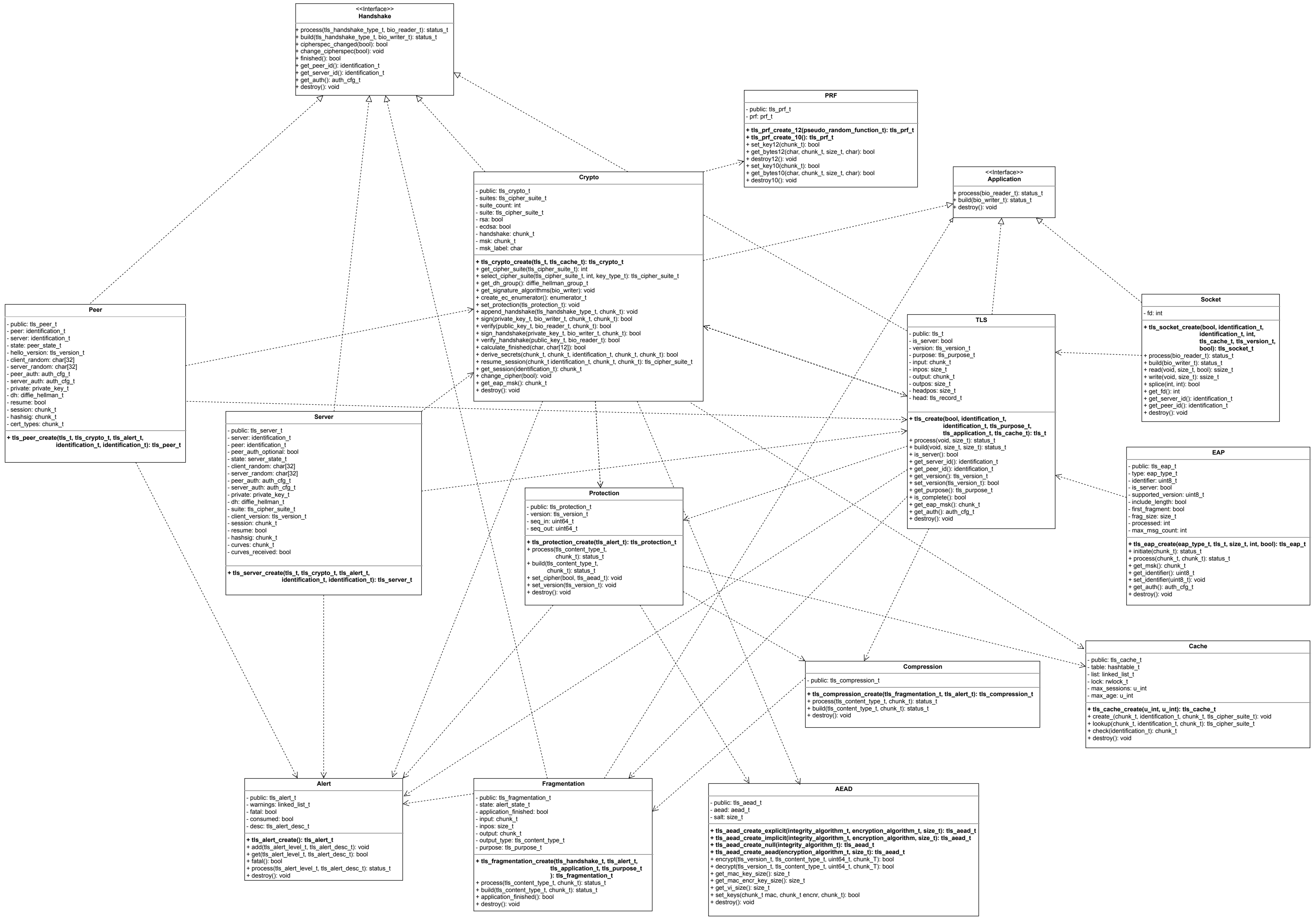
[3]For the sake of simplicity we chose to only list the relevant parts of those methods in the diagram, i.e. name of the method, arguments and return value.

Diagram 3.1 shows how `libtls` was organised at the start of the project. Relationships between classes are indicated by an arrow. If an arrow points from class X to class Y, it means that class X has a reference of type Y in its private part. Thus the line `tls_aead_t *aead_in` in `struct private_tls_crypto_t`, which is the AEAD transformation for inbound traffic, uses something from the AEAD class, i.e. from `tls_aead.c`.

3.1.2. Sequence Diagram

The UML sequence diagram on page 22 illustrates the call order and object lifetime in a simple client-side TLS handshake situation.

The first call made by `tls_test` represents the `libtls` executable, which calls the function `tls_socket_create` of class Socket. Note this is a simplified diagram that emphasises the call order regarding the `libtls` library, removing the return arrows usually found in sequence diagrams. The vertical white beam represents the object life-time, which ends when the object's destroy function is called, as explained in the previous section.



<<Interface>>
Handshake
+ process(tls_handshake_type_t, bio_reader_t): status_t
+ build(tls_handshake_type_t, bio_writer_t): status_t
+ cipherspec_changed(bool): bool
+ change_cipherspec(bool): void
+ finished(): bool
+ get_peer_id(): identification_t
+ get_server_id(): identification_t
+ get_auth(): auth_cfg_t
+ destroy(): void
PRF
- public: tls_prf_t
- prf: prf_t
+ tls_prf_create_12(pseudo_random_function_t): tls_prf_t
+ tls_prf_create_10(): tls_prf_t
+ set_key12(chunk_t): bool
+ get_bytes12(char, chunk_t, size_t, char): bool
+ destroy12(): void
+ set_key10(chunk_t): bool
+ get_bytes10(char, chunk_t, size_t, char): bool
+ destroy10(): void
<<Interface>>
Application
+ process(bio_reader_t): status_t
+ build(bio_writer_t): status_t
+ destroy(): void
Crypto
- public: tls_crypto_t
- suites: tls_cipher_suite_t
- suite_count: int
- suite: tls_cipher_suite_t
- rsa: bool
- ecdsa: bool
- handshake: chunk_t
- msk: chunk_t
- msk_label: char
+ tls_crypto_create(tls_t, tls_cache_t): tls_crypto_t
+ get_cipher_suite(tls_cipher_suite_t): int
+ select_cipher_suite(tls_cipher_suite_t, int, key_type_t): tls_cipher_suite_t
+ get_dh_group(): diffie_hellman_group_t
+ get_signature_algorithms(bio_writer): void
+ create_ec_enumerator(): enumerator_t
+ set_protection(tls_protection_t): void
+ append_handshake(tls_handshake_type_t, chunk_t): void
+ sign(private_key_t, bio_writer_t, chunk_t, chunk_t): bool
+ verify(public_key_t, bio_reader_t, chunk_t): bool
+ sign_handshake(private_key_t, bio_writer_t, chunk_t): bool
+ verify_handshake(public_key_t, bio_reader_t): bool
+ calculate_finished(char, char[12]): bool
+ derive_secrets(chunk_t, chunk_t, identification_t, chunk_t, chunk_t): bool
+ resume_session(chunk_t identification_t, chunk_t, chunk_t): tls_cipher_suite_t
+ get_session(identification_t): chunk_t
+ change_cipher(bool): void
+ get_eap_msk(): chunk_t
+ destroy(): void
Socket
- fd: int
+ tls_socket_create(bool, identification_t, identification_t, int, tls_cache_t, tls_version_t, bool): tls_socket_t
+ process(bio_reader_t): status_t
+ build(bio_writer_t): status_t
+ read(void, size_t, bool): ssize_t
+ write(void, size_t): ssize_t
+ splice(int, int): bool
+ get_fd(): int
+ get_server_id(): identification_t
+ get_peer_id(): identification_t
+ destroy(): void
TLS
- public: tls_t
- is_server: bool
- version: tls_version_t
- purpose: tls_purpose_t
- input: chunk_t
- inpos: size_t
- output: chunk_t
- outpos: size_t
- headpos: size_t
- head: tls_record_t
+ tls_create(bool, identification_t, identification_t, tls_purpose_t, tls_application_t, tls_cache_t): tls_t
+ process(void, size_t): status_t
+ build(void, size_t, size_t): status_t
+ is_server(): bool
+ get_server_id(): identification_t
+ get_peer_id(): identification_t
+ get_version(): tls_version_t
+ set_version(tls_version_t): bool
+ get_purpose(): tls_purpose_t
+ is_complete(): bool
+ get_eap_msk(): chunk_t
+ get_auth(): auth_cfg_t
+ destroy(): void
Peer
- public: tls_peer_t
- peer: identification_t
- server: identification_t
- state: peer_state_t
- hello_version: tls_version_t
- client_random: char[32]
- server_random: char[32]
- peer_auth: auth_cfg_t
- server_auth: auth_cfg_t
- private: private_key_t
- dh: diffie_hellman_t
- resume: bool
- session: chunk_t
- hashsig: chunk_t
- cert_types: chunk_t
+ tls_peer_create(tls_t, tls_crypto_t, tls_alert_t, identification_t, identification_t): tls_peer_t
Server
- public: tls_server_t
- server: identification_t
- peer: identification_t
- peer_auth_optional: bool
- state: server_state_t
- client_random: char[32]
- server_random: char[32]
- peer_auth: auth_cfg_t
- server_auth: auth_cfg_t
- private: private_key_t
- dh: diffie_hellman_t
- suite: tls_cipher_suite_t
- client_version: tls_version_t
- session: chunk_t
- resume: bool
- hashsig: chunk_t
- curves: chunk_t
- curves_received: bool
+ tls_server_create(tls_t, tls_crypto_t, tls_alert_t, identification_t, identification_t): tls_server_t
Protection
- public: tls_protection_t
- version: tls_version_t
- seq_in: uint64_t
- seq_out: uint64_t
+ tls_protection_create(tls_alert_t): tls_protection_t
+ process(tls_content_type_t, chunk_t): status_t
+ build(tls_content_type_t, chunk_t): status_t
+ set_cipher(bool, tls_aead_t): void
+ set_version(tls_version_t): void
+ destroy(): void
EAP
- public: tls_eap_t
- type: eap_type_t
- identifier: uint8_t
- is_server: bool
- supported_version: uint8_t
- include_length: bool
- first_fragment: bool
- frag_size: size_t
- processed: int
- max_msg_count: int
+ tls_eap_create(eap_type_t, tls_t, size_t, int, bool): tls_eap_t
+ initiate(chunk_t): status_t
+ process(chunk_t, chunk_t): status_t
+ get_msk(): chunk_t
+ get_identifier(): uint8_t
+ set_identifier(uint8_t): void
+ get_auth(): auth_cfg_t
+ destroy(): void
Cache
- public: tls_cache_t
- table: hashtable_t
- list: linked_list_t
- lock: rwlock_t
- max_sessions: u_int
- max_age: u_int
+ tls_cache_create(u_int, u_int): tls_cache_t
+ create_(chunk_t, identification_t, chunk_t, tls_cipher_suite_t): void
+ lookup(chunk_t, identification_t, chunk_t): tls_cipher_suite_t
+ check(identification_t): chunk_t
+ destroy(): void
Compression
- public: tls_compression_t
+ tls_compression_create(tls_fragmentation_t, tls_alert_t): tls_compression_t
+ process(tls_content_type_t, chunk_t): status_t
+ build(tls_content_type_t, chunk_t): status_t
+ destroy(): void
Alert
- public: tls_alert_t
- warnings: linked_list_t
- fatal: bool
- consumed: bool
- desc: tls_alert_desc_t
+ tls_alert_create(): tls_alert_t
+ add(tls_alert_level_t, tls_alert_desc_t): void
+ get(tls_alert_level_t, tls_alert_desc_t): bool
+ fatal(): bool
+ process(tls_alert_level_t, tls_alert_desc_t): status_t
+ destroy(): void
Fragmentation
- public: tls_fragmentation_t
- state: alert_state_t
- application_finished: bool
- input: chunk_t
- inpos: size_t
- output: chunk_t
- output_type: tls_content_type_t
- purpose: tls_purpose_t
+ tls_fragmentation_create(tls_handshake_t, tls_alert_t, tls_application_t, tls_purpose_t): tls_fragmentation_t
+ process(tls_content_type_t, chunk_t): status_t
+ build(tls_content_type_t, chunk_t): status_t
+ application_finished(): bool
+ destroy(): void
AEAD
- public: tls_aead_t
- aead: aead_t
- salt: size_t
+ tls_aead_create_explicit(integrity_algorithm_t, encryption_algorithm_t, size_t): tls_aead_t
+ tls_aead_create_implicit(integrity_algorithm_t, encryption_algorithm, size_t): tls_aead_t
+ tls_aead_create_null(integrity_algorithm_t): tls_aead_t
+ tls_aead_create_aead(encryption_algorithm_t, size_t): tls_aead_t
+ encrypt(tls_version_t, tls_content_type_t, uint64_t, chunk_T): bool
+ decrypt(tls_version_t, tls_content_type_t, uint64_t, chunk_T): bool
+ get_mac_key_size(): size_t
+ get_mac_encr_key_size(): size_t
+ get_vi_size(): size_t
+ set_keys(chunk_t mac, chunk_t encnr, chunk_t): bool
+ destroy(): void

Figure 3.1.: Class diagram of libtls at the start of the project.

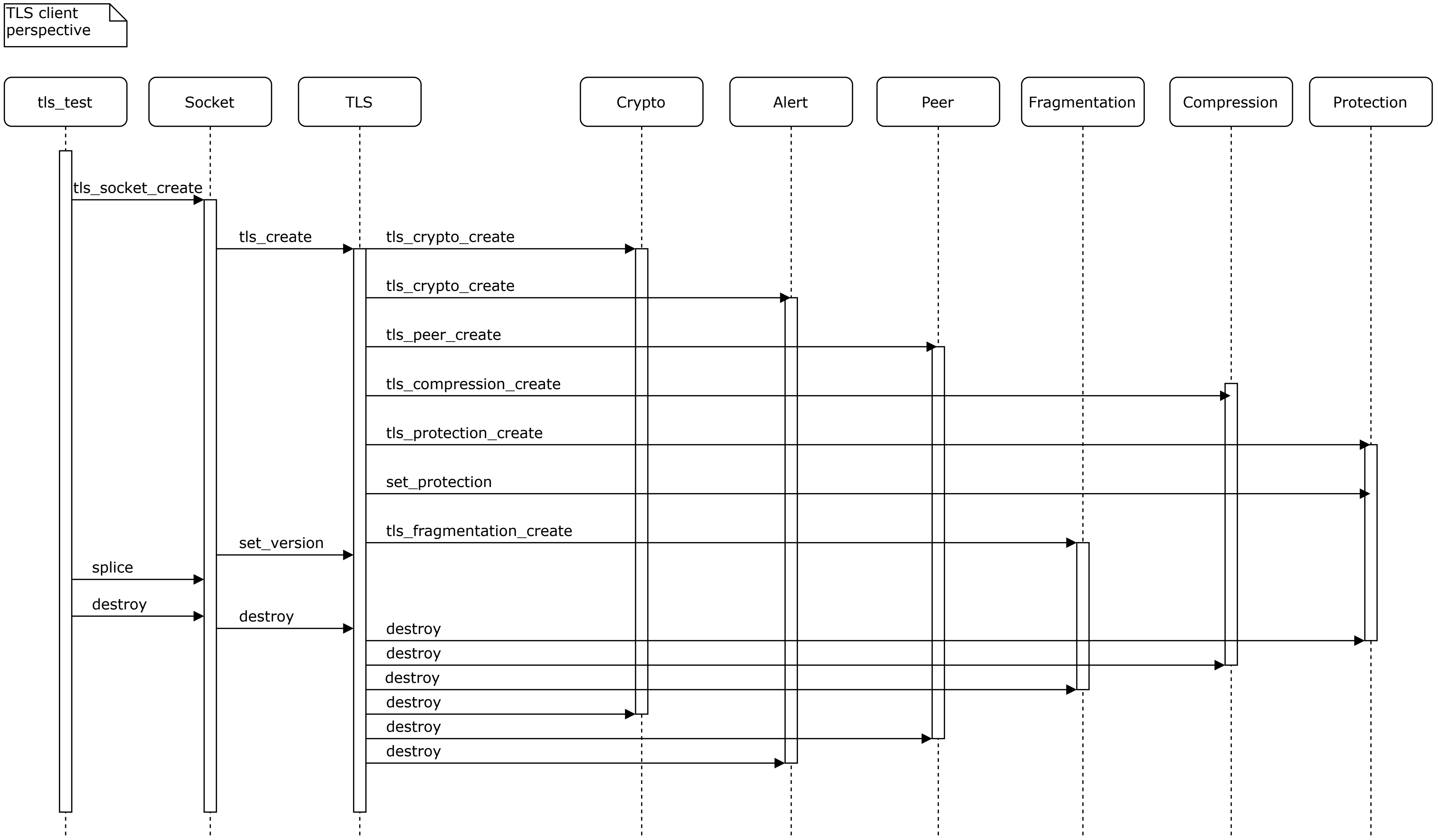


Figure 3.2.: Sequence diagram of libtls when the project started.

3.2. Design Choices

Before starting out with our implementation, we faced two possible paths:

1. Write the new TLS stack from scratch.
2. Extend the current TLS stack and distinguish the version where needed.

As described in the introduction, the handshake in TLS 1.3 has been greatly simplified. TLS 1.3 does away with a lot of old baggage from previous versions, so a fresh implementation in new, separates files makes a lot of sense. In addition, the existing code is already highly complex, and an integration into the existing stack means adding even more complexity. On the other hand, this path implies that the team needs to thoroughly understand the current implementation also in the context of the whole strongSwan project, even though both team members have not worked with the code base before.

The second path reduces the need for the team to fully understand all the inner workings of the `libtls` library and how it connects to the rest of the code. It does not change the existing architecture and tries to apply changes and checks where needed. However, this alternative introduces many new version distinctions in the shape of `if-else` branches. Also, more complexity is the foe of good security, and a change in one place might introduce vulnerabilities in another or just break the code.

After careful deliberation, we chose the second path, in view of the new-to-us code and the limited time frame given. We therefore tried to work with the existing code where possible. This also has the advantage that it respects existing structures. New functions or files were only introduced where the new RFC deviated form the previous versions. In the code, this is mostly visible as this check:

```
if (this->tls->get_version_max(this->tls) < TLS_1_3)
{
        /* existing code, TLS 1.2 */
}
else
{
        /* code for TLS 1.3 */
}
```

The maximum version is the highest version supported by the client *and* the version on which client and server have agreed.

Our initial decision led to many subtle and more obvious code changes scattered throughout the code base, mostly in `tls_peer.c` and `tls_crypto.c`. The most apparent change is visible in the list of cipher suites, which used to fill pages in TLS 1.2 and has been boiled down to just five in TLS 1.3.[4] The list with the old and new cipher suites is located in the file `tls_crypto.c` as `static suite_algs_t suite_algs[]`. Initially, we added the five new cipher suites to this list, but they were all filtered out by the private method `filter_unsupported_suites`. In TLS 1.3, the cipher suite structure stays the same, but the key exchange and authentication fields have been turned into extensions. If we set them to default values only, they were filtered out. This is why we had to distinguish the new and old TLS versions in the cipher suite list and enhanced the existing `suite_algs_t struct` with a TLS version field.

[4]See section 2.2.

By using this new field we were able to apply only the filters relevant to the desired TLS version:

```
typedef struct {
        tls_cipher_suite_t suite;
        key_type_t key;
        diffie_hellman_group_t dh;
        hash_algorithm_t hash;
        pseudo_random_function_t prf;
        integrity_algorithm_t mac;
        encryption_algorithm_t encr;
        size_t encr_size;
        tls_version_t tls_version; /* new field */
} suite_algs_t;
```

Further consequences of our design choices became apparent when we had to intertwine the new TLS 1.3 state machine with the existing one and establish a connection to send application data.[5]

[5]See section 3.5.

3.3. HMAC-based Key Derivation Function

In TLS it is necessary to generate keys which encrypt the traffic exchanged by the peers. However, the algorithm differs between TLS versions. TLS 1.2 uses a pseudo-random function (PRF) to generate keys from the Master-Key.[6] TLS 1.3 on the other hand uses an HMAC-based Key Derivation Function (HKDF) to generate traffic keys.[7] strongSwan does not bring a ready-to-use HKDF implementation the way TLS 1.3 requires. We had to add this functionality based on provided HKDF boilerplate code described in appendix D.0.2. This section describes the theoretical details of the HKDF in TLS 1.3 and our concrete implementation.

3.3.1. Theory

An HKDF has two fundamental functions: `HKDF-Extract` and `HKDF-Expand`.[8] However, TLS 1.3 introduces two new additional functions, `HKDF-Expand-Label` and `Derive-Secret`.[9] This results in these four functions:

- `HKDF-Extract` extracts a pseudo-random key (PRK) from the source key[10]. The extraction function is based on an HMAC, hence the name HKDF. This function takes two parameters: An input key material `IKM` and a `salt`.
- `HKDF-Expand` is a second step in which the generated pseudo-random key is fed to an HMAC. The HMAC in turn acts as a pseudo-random function to extract the required amount of bits. This function takes three parameters: A `Secret`, a `HkdfLabel` and the desired output key material (OKM) `Length`.
- `HKDF-Expand-Label` transforms `Label` and `Context` into a `HkdfLabel` structure[11] and calls `HKDF-Expand` to derive an OKM. This function takes the four arguments `Secret`, `Label`, `Context` and `Length`.
  - `Context` contains the handshake messages of the current state or an empty string as a hash.
  - `Length` specifies the desired output length in bytes.
- `Derive-Secret` hashes the raw handshake message bytes and calls `HKDF-Extract-Label` to derive an OKM. This function takes the three arguments `Secret`, `Label` and `Messages`.
  - `Secret`: The PRK secret from the `HKDF-Extract` function.
  - `Label`: Relevant to our implementation are these labels.[12]
    - "`tls13 c hs traffic`"

[6] Res08, p. 26.
[7] Res18, p. 95.
[8] Kra10.
[9] The basic concepts of an HKDF are described in chapter 7.1, "Key Schedule", and chapter 7.3, "Traffic Key Calculation" [Res18, 91ff].
[10] Dan Boneh has an excellent video explaining HKDF in detail. See [Bon20].
[11] Res18, p. 91.
[12] For all eleven labels see the overview on page 93 [Res18].

    * "`tls13 s hs traffic`"
    * "`tls13 c ap traffic`"
    * "`tls13 s ap traffic`"
    * "`tls13 derived`"
  - `Messages` specifies the unencrypted handshake bytes, without record headers, of client and server up to the current state of the TLS handshake.

The hashing algorithm used for the HKDF is specified in the negotiated TLS cipher suite. Since the number of cipher suites in TLS 1.3 are reduced to five, only two possible hashing algorithms currently remain: SHA-256 or SHA-384.

Figure 3.3 illustrates how the HKDF is used in TLS, and is directly drawn from the illustration in RFC 8446.[13] It suggests that the HKDF in TLS 1.3 can also be interpreted as a state machine: Starting from a phase 0, every call to `HKDF-Extract` is a one-way transition into the next phase. This results in a state machine with four phases.[14]

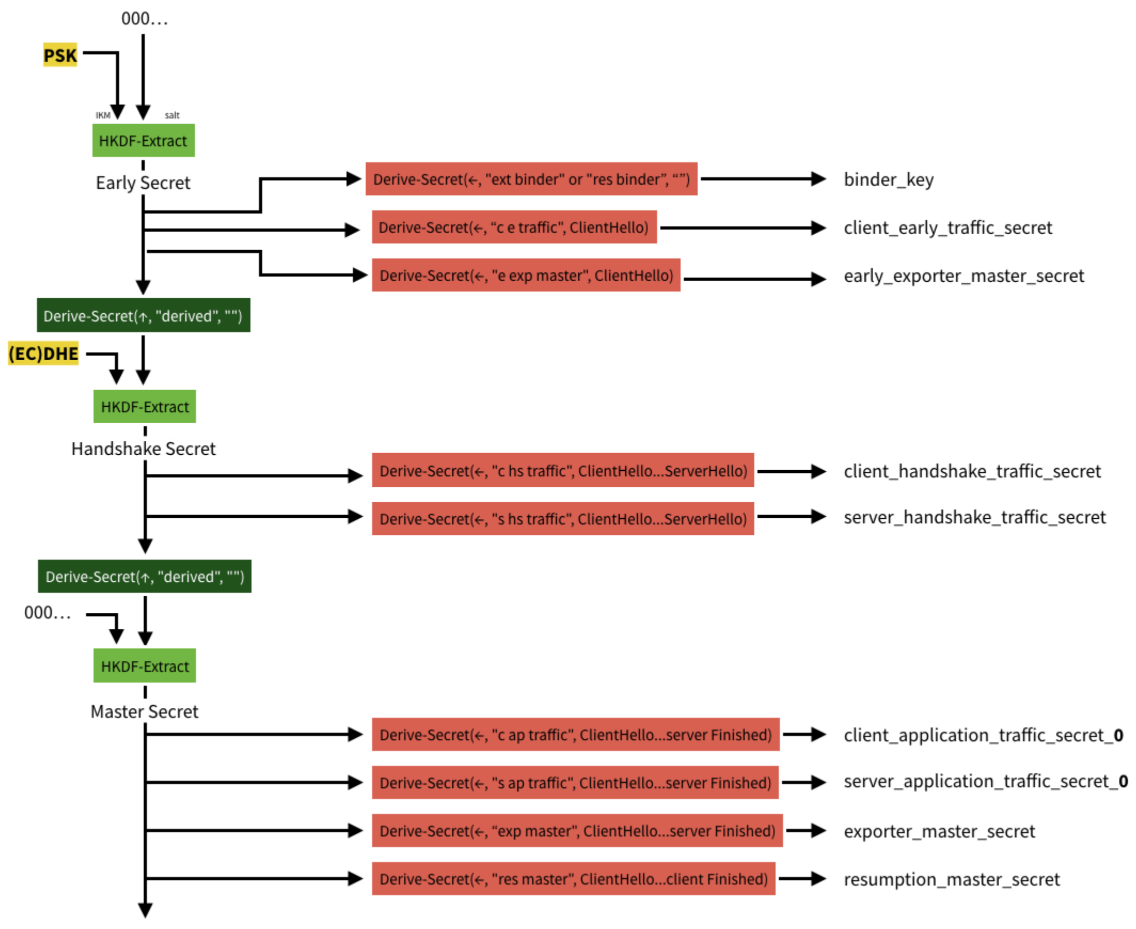

000...
PSK
IKM
salt
HKDF-Extract
Early Secret
Derive-Secret(←, "ext binder" or "res binder", "")
binder_key
Derive-Secret(←, "c e traffic", ClientHello)
client_early_traffic_secret
Derive-Secret(←, "e exp master", ClientHello)
early_exporter_master_secret
Derive-Secret(↑, "derived", "")
(EC)DHE
HKDF-Extract
Handshake Secret
Derive-Secret(←, "c hs traffic", ClientHello...ServerHello)
client_handshake_traffic_secret
Derive-Secret(←, "s hs traffic", ClientHello...ServerHello)
server_handshake_traffic_secret
Derive-Secret(↑, "derived", "")
000...
HKDF-Extract
Master Secret
Derive-Secret(←, "c ap traffic", ClientHello...server Finished)
client_application_traffic_secret_0
Derive-Secret(←, "s ap traffic", ClientHello...server Finished)
server_application_traffic_secret_0
Derive-Secret(←, "exp master", ClientHello...server Finished)
exporter_master_secret
Derive-Secret(←, "res master", ClientHello...client Finished)
resumption_master_secret

Figure 3.3.: Each `HKDF-Extract` signals a one-way state transition.

[13] Res18, p. 93.

[14] The illustration in figure 3.3 is courtesy of David Wong, `https://www.davidwong.fr/tls13/#section-7.1`, visited on 2020-05-23.

The above-mentioned labels play an important role in the HKDF. Unfortunately, RFC 8446 does not provide a single overview and description of all possible labels. It is therefore important to mention these other labels and where they can be used as well.

As mentioned, eleven labels are defined in section 7.1 of the RFC and associated to different stages in the HKDF state machine.[15] The label "`tls13 derived`" is only used during state transitions. These labels are all passed as arguments to the `Derive-Secret` function:

- Four PSK keys are derived from the `Early Secret`:
  - "`tls13 ext binder`"
  - "`tls13 res binder`"
  - "`tls13 c e traffic`"
  - "`tls13 e exp master`"
- Two handshake traffic secrets are derived from the `Handshake Secret`:
  - "`tls13 c hs traffic`"
  - "`tls13 s hs traffic`"
- The last four secrets are used for application traffic secrets and other use cases of TLS 1.3. They are derived from the `Traffic Secret`:
  - "`tls13 c ap traffic`"
  - "`tls13 s ap traffic`"
  - "`tls13 exp master`"
  - "`tls13 res master`"

There are a couple of labels that are used directly on `HKDF-Expand-Label` and can be called on all states of the HKDF state machine except phase 0:

- Provide actual keying material[16]:
  - "`tls13 key`"
  - "`tls13 iv`"
- Generate the key to authenticate the handshake `Finished` messages[17]:
  - "`tls13 finished`"

Key and initialisation vector (IV) are used in the AEAD ciphers to actually encrypt and decrypt traffic, be it for the handshake or application data. The label "`tls13 finished`" is only called once on each side, after a `Finished` message is received.

[15]Res18, p. 93.
[16]Section 7.3 of [Res18, p. 95].
[17]Section 4.4.4 of [Res18, p. 71].

3.3.2. Implementation

The HKDF is used in different stages of the handshake process and functions like a state machine itself. We therefore called these internal states of the HKDF "phases":

- Phase 0: the initial state
- Phase 1: generate pre-shared keys (not in our scope)
- Phase 2: generate handshake keys
- Phase 3: generate application data keys

Figure 3.4 illustrates the link between the HKDF in figure 3.3 and our phases.

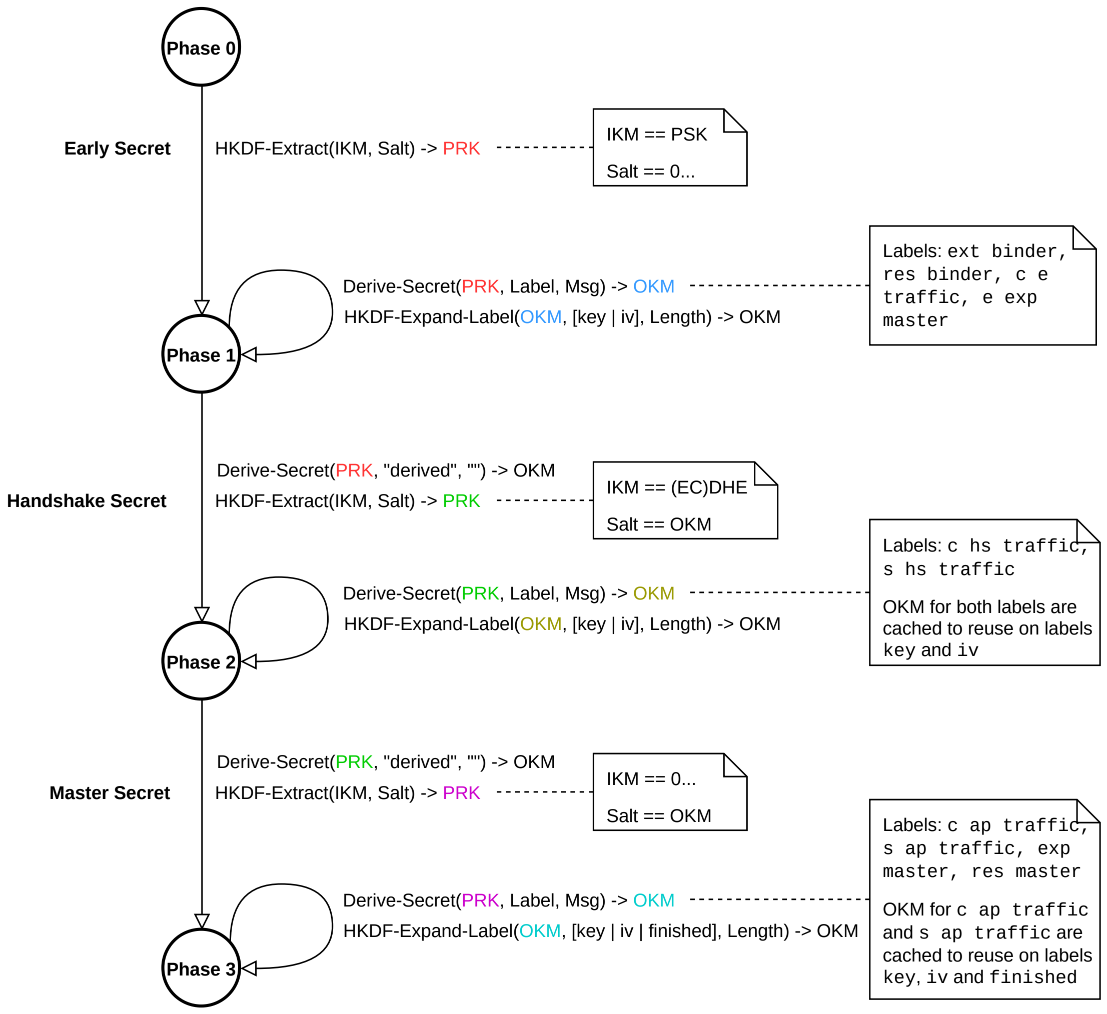
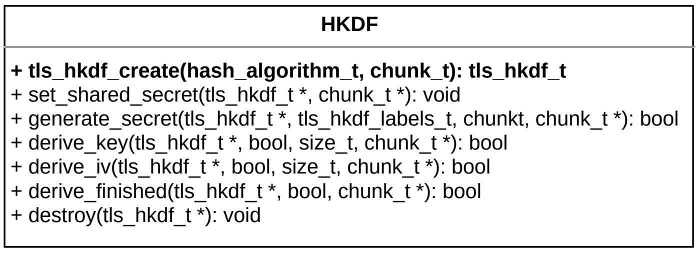

Phase 0
Early Secret
HKDF-Extract(IKM, Salt) -> PRK
IKM == PSK
Salt == 0...
Derive-Secret(PRK, Label, Msg) -> OKM
HKDF-Expand-Label(OKM, [key | iv], Length) -> OKM
Labels: ext binder, res binder, c e traffic, e exp master
Phase 1
Handshake Secret
Derive-Secret(PRK, "derived", "") -> OKM
HKDF-Extract(IKM, Salt) -> PRK
IKM == (EC)DHE
Salt == OKM
Derive-Secret(PRK, Label, Msg) -> OKM
HKDF-Expand-Label(OKM, [key | iv], Length) -> OKM
Labels: c hs traffic, s hs traffic
OKM for both labels are cached to reuse on labels key and iv
Phase 2
Master Secret
Derive-Secret(PRK, "derived", "") -> OKM
HKDF-Extract(IKM, Salt) -> PRK
IKM == 0...
Salt == OKM
Derive-Secret(PRK, Label, Msg) -> OKM
HKDF-Expand-Label(OKM, [key | iv | finished], Length) -> OKM
Labels: c ap traffic, s ap traffic, exp master, res master
OKM for c ap traffic and s ap traffic are cached to reuse on labels key, iv and finished
Phase 3

Figure 3.4.: The state machine for our HKDF implementation with the four phases.

These two aspects – repeated use and internal states – encouraged us to encapsulate our whole implementation in an HKDF class. We designed the public interface of the HKDF class as visualised in figure 3.5:



```
HKDF
+ tls_hkdf_create(hash_algorithm_t, chunk_t): tls_hkdf_t
+ set_shared_secret(tls_hkdf_t *, chunk_t *): void
+ generate_secret(tls_hkdf_t *, tls_hkdf_labels_t, chunkt, chunk_t *): bool
+ derive_key(tls_hkdf_t *, bool, size_t, chunk_t *): bool
+ derive_iv(tls_hkdf_t *, bool, size_t, chunk_t *): bool
+ derive_finished(tls_hkdf_t *, bool, chunk_t *): bool
+ destroy(tls_hkdf_t *): void
```

Figure 3.5.: UML class diagram of our current HKDF implementation.

The class is implemented in `tls_hkdf.h` and `tls_hkdf.c`. We verify its functionality with unit tests in `test_hkdf.c` based on static data and keying material.[18] The constructor `tls_hkdf_create` as well as all methods are documented using Doxygen, in accordance with the rest of the documentation.

The main idea of this design is to encapsulate all the HKDF functionality and internal state logic and provide an easier-to-use interface. A user may set a PSK secret when initialising a concrete HKDF object or leave it blank when not using secrets from the first phase. By providing one of the ten labels defined in `enum tls_hkdf_labels_t`, a user sets the HKDF state machine into the appropriate state using the function `generate_secrets`.

As one may notice from figure 3.4, a state transition always consists of the two steps `Derive-Secret` with the `derived` label and `HKDF-Extract` using the OKM from the former function. In our implementation we considered this fact and encapsulated these steps into the function `generate_secret`. Therefore, the label `derived` is not exposed to the caller and is solely used as internal label.

The states, as described earlier, provide different encryption keys and IVs derived with the corresponding public methods `derive_key` and `derive_iv`. To verify handshake authentication, the public method `derive_finished` is used. While `derive_key` and `derive_iv` can be called in all three non-zero states, `derive_finished` is only called in phase 3 of the internal HKDF state machine.

3.3.3. HKDF and AEAD

The idea behind Authenticated Encryption with Associated Data (AEAD) is to encrypt the payload of a message and additionally authenticate the plaintext packet headers used to route the packet.[19] This allows the receiver to discover if the whole packet has been secretly modified in transit. All five cipher suites in TLS 1.3 are AEAD ciphers.[20] The key and initialisation vector (IV) is provided by the HKDF.

strongSwan has already built-in support for AEAD, since TLS 1.2 standardised it.[21] However, TLS 1.3 uses AEAD somewhat differently, and one of the maintainers adapted the code to

[18]We use the handshake bytes and keying material provided by *The New Illustrated TLS Connection* ([Dri]). The unit tests also give a good idea on how to the class is used in the code.

[19]See also illustration 3.6. It is heavily inspired by Dan Bonehs video explaining authenticated encryption.[Bon15]

[20]This can also be recognised by the block cipher mode CCM and GCM respectively ChaCha20-Poly1305 for the stream cipher.

[21]Res08, p. 24.

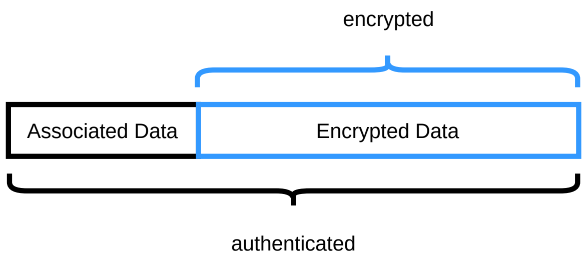


Figure 3.6.: Basic concept of authenticated encryption with associated data (AEAD). .

allow the current AEAD implementation to work with TLS 1.3. A new class was added, `tls_aead_seq.c`, and other changes happened in:

- The public interface `tls_aead.h`
- All AEAD implementations of the interface:
  - `tls_aead.c`
  - `tls_aead.c`
  - `tls_aead_expl.c`
  - `tls_aead_impl.c`
  - `tls_aead_null.c`
- `tls_protection.c` with implements `tls_aead.h`

Because TLS 1.3 encrypts the actual content type within the `TLSInnerPlaintext` structure and always sets the `TLSCiphertext` structure content type to `Application Data`, the interface had to be modified.[22] This led to changes in all current existing AEAD implementations. Because `tls_protection.c` also uses the concrete AEAD object, it also required minimal changes there.

The new `tls_aead_seq.c` class is to be used with TLS 1.3 and its constructor has been added to the public interface in `tls_aead.h`. The class mainly differs in the the way how key material is handled. According to RFC 8446, the nonce and initialisation vector (IV) is calculated differently than in the prior TLS version.[23]

[22]Res18, p. 81.
[23]Res18, p. 83.

3.4. Range-Based Version Handling

Prior to TLS 1.3, each peer sent its supported TLS version in the handshake version field. This was a single value: A client offered its highest possible version as a two-byte value in its `ClientHello` message, for example `0x0303` for TLS 1.2. The server responded with its highest supported version that was equal or smaller to the client, also a two-byte value in the `ServerHello` message. If both peers supported this, the handshake continued with this agreed-on TLS version.

TLS 1.3 deprecates this mechanism in favour of a more extensible approach. The new protocol version negotiation is outsourced to the `supported_versions` TLS handshake extension. This allows to specify a range of supported versions instead of a single value. If a client supports TLS 1.0 up to TLS 1.3, it sends the four corresponding hex values with the most preferred listed first.

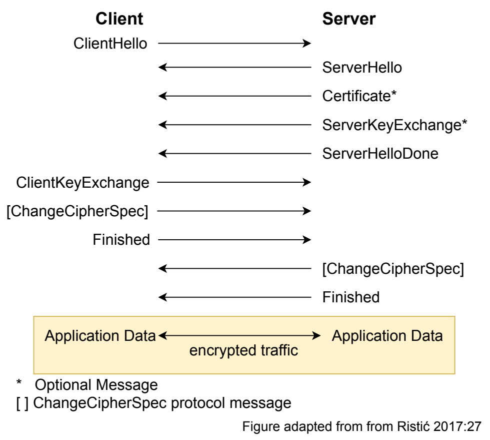
The original value field from earlier TLS versions has been preserved, however: A TLS-1.3-capable client sets the `ClientHello.legacy_version` field to `0x0303` (TLS 1.2) due to backward compatibility, but always sends the `supported_versions` extension along.[24] This means also that the hex value of TLS 1.3 (0x0304) is *never* set in a `ClientHello.legacy_version` or `ServerHello.legacy_version` field.

In addition, the TLS Record header, which contains `legacy_record_version`, is set to the value of TLS 1.2 for all TLS 1.3 connections. The RFC makes one exception due to compatibility reasons and requires an implementation to set the record that contains a TLS 1.3 `ClientHello` message to the value of TLS 1.0.[25]

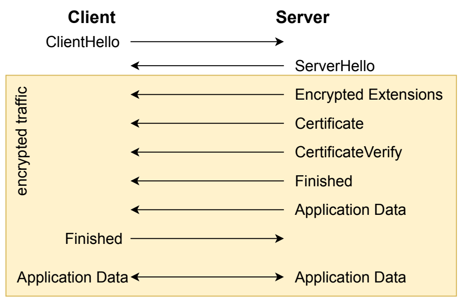
To address this new negotiation mechanism, we replaced the current TLS version interface `get_version` in `tls.h` with the two methods `get_version_min` and `get_version_max`. The latter has a dual function: It announces the highest supported TLS version of a peer, but later in the handshake returns the TLS version on which the peers agreed. This new version range capability is then used to create the `ClientHello` message within `tls_peer.c`.

Due to the fact that we omitted the whole compatibility topic, our current implementation does not quite conform to the RFC. Our implementation sets the `ClientHello.legacy_version` correctly to `0x0303` but also uses this value for the `legacy_record_version` in the record value, where `0x0301` would be the correct value.

[24] Res18, p. 139.
[25] Res18, p. 79.



(a) TLS 1.2



(b) TLS 1.3

Figure 3.7.: A full TLS handshake with server authentication.

3.5. Handshake and State Machine: Notable Changes

Until client and server can exchange encrypted application data, they go through a handshake to agree on the specifics of the connection. As mentioned in the introduction, the RFC for the new TLS version lists state machines for server and client, information that was missing in the previous RFC for TLS 1.2. This section details how we incorporated these changes in handshake and state machine into the existing code.

In comparison to its earlier version, TLS 1.3 simplifies the handshake, removes a round-trip and some old messages:[26]

- Obsolete: `ServerHelloDone`, `ServerKeyExchange`, `ClientKeyExchange`
- New: `HelloRetryRequest`, `EncryptedExtensions`

The other, less notable change is the transition to encrypted data. In TLS 1.2, the transition from plaintext traffic to encrypted application data is announced by the `ChangeCipherSpec` protocol message. It signals the peers that they are now both ready to transition to encrypted traffic. In the end, there are only two states: unencrypted and encrypted traffic.

On the other hand, TLS 1.3 has three states and no dedicated messages that signal a transition (see figure 3.8). The transition from state 0 to state 1 happens after the client receives the `ServerHello` message: At that point, the peers have agreed on the TLS version and the cipher suite for all the remaining handshake messages. All further messages are now encrypted with the handshake traffic secret – a notable change to TLS 1.2, which sent nearly all handshake messages in plaintext.[27]

	State 0	**State 1**	**State 2**
TLS 1.2	plaintext handshake	encrypted application data	not applicable
TLS 1.3	plaintext handshake	encrypted handshake	encrypted application data

Figure 3.8.: Transitions from unencrypted to encrypted traffic in TLS 1.2 and 1.3.

[26]See also figure 3.7

[27]The `ServerFinished` message is encrypted.

In our implementation, the handshake proceeds as follows:

Client sends ClientHello: The client generates the necessary keys specific to TLS 1.3 via the command `lib->crypto->create_dh(lib->crypto, CURVE_25519)`.[28] It also sends a range of extensions including the supported TLS versions.[29]

Client receives ServerHello: After the Client has processed the message, it initialises the agreed-on cipher suite and generates the key material needed for the rest of the handshake. This happens by way of the new HKDF which is described in the previous section.

Client receives ChangeCipherSpec: This message still exists in TLS 1.3, but it is only there for middlebox compatibility, devoid of the original purpose. It can be sent or received nevertheless at any point after the `ClientHello` messages.[30] If the strongSwan TLS client receives a `ChangeCipherSpec` message, the state `STATE_CIPHERSPEC_RECEIVED` is set but nothing happens whatsoever.

Client receives EncryptedExtension: This is the first message encrypted with the handshake traffic secret generated by the HKDF. None of the extensions pertaining to this message are within our scope, so a skeleton was just implemented in a new method called `process_encrypted_extensions` in `tls_peer.c`.

Client receives Certificate: The message[31] has two additions in comparison to TLS 1.2: A certificate request context is prepended to the list of certificates, and each certificate entry ends with extensions. Since we did not include the latter in our minimal viable product, the data is just read, but not further processed.

Client receives CertificateVerify: In TLS 1.2, it is used for client-side authentication only[32] and the TLS 1.2 client in strongSwan is only able to send such a message. In TLS 1.3, this message is now mandatory for server-side authentication via certificate: "To prove that the server owns the server certificate (giving the certificate validity in this TLS session), it signs a hash of the handshake messages using the certificate's private key. The signature can be proven valid by the client by using the certificate's public key."[33] The new method `process_cert_verify` processes now such a message.[34]

Client receives Finished: The new `Finished` differs greatly from TLS 1.2. It uses the new HKDF and its output is of variable length, whereas in TLS 1.2 it had a fixed length of 12 bytes. A new public method `calculate_finished_tls13` that calculates the data to be verified by the client or sent to the server for verification, both by way of the HKDF.

Client sends Finished: This concludes the handshake[35] and marks the transition to application data. The HKDF is now used to generate the application traffic keys to de- and encrypt

[28]For an explanation of the chosen curve see the results chapter in 4.

[29]Our implementation sends `server_name`, `supported_versions`, `signature_algorithms`, `supported_groups` (`elliptic_curve` extension in TLS 1.2), and `key_share`.

[30]Res18, p. 77.

[31]Res18, p. 64.

[32]Res08, p. 62.

[33]Annotation to "Server Certificate Verify" in [Dri].

[34]On a side note, the RFC for TLS 1.3 does not comply to its own description of the transcript hash as defined earlier in section 4.4.1 of the RFC. For `CertificateVerify`, the transcript hash is `Transcript-Hash(Handshake Context, Certificate)`[Res18, p. 69]. `Certificate` is ambiguous, since it is listed separately from `Handshake Context`: Is it material from the server certificate (its public key) or is it part of the transcript hash, i.e. the transcript hash including the `Certificate` message? It turns out that it is the latter.

[35]Res18, p. 71.

application data. Note that in TLS 1.3, the `Finished` messages are exchanged in the exact opposite order as in TLS 1.2 (see also figure 3.7): The server sends the `Finished` message first, then the client sends its own. Again the HKDF is used with the “`finished`” label to compute the message.

Client sends and receives Post-Handshake Messages: After the actual handshake, client and server can still exchange Handshake messages. They are also encrypted with the application traffic key.[36] Since our OpenSSL test server sent a `NewSessionTicket` message, we simply parsed it but did not further process its contents.

The messages above transition through a state machine that is implemented in the public methods `build` (outgoing traffic) and `process` (incoming traffic) in `tls_peer.c` As detailed in section 3.2, we made the conscious decision to distinguish between the TLS version within the code and not re-implement everything from scratch.

Initially, we did not distinguish the state machines, since so many states are similar to TLS 1.2. The more we processed in our implementation, the more we faced problems due to the rising complexity. Only when we clearly split the state machines by version did we achieve a bug-free implementation.

[36]Res18, p. 73.

3.6. Sequence Numbers

Client and server separately maintain a sequence number of reading and writing records.[37] If the sequence numbers are not initialised and incremented properly, the messages sent back and forth cannot be decrypted. In contrast to TLS 1.2, the nonces are initialised twice in TLS 1.3: Once when the unencrypted traffic switches to encrypted handshake traffic, and once when the encrypted handshake traffic switches to encrypted application data traffic.[38]

In our implementation, the sequence number is first set to zero when the `aead` object is created after the `ServerHello`. Also note `tls_aead_create_seq`, which actually creates the newly implemented `aead` object[39]:

```c
/* File: tls_crypto.c*/
static bool create_aead(private_tls_crypto_t *this, suite_algs_t *algs)
{
if (this->tls->get_version_max(this->tls) < TLS_1_3)
{
this->aead_in = tls_aead_create_aead(algs->encr, algs->encr_size);
this->aead_out = tls_aead_create_aead(algs->encr, algs->encr_size);
}
else
{
this->aead_in = tls_aead_create_seq(algs->encr, algs->encr_size);
this->aead_out = tls_aead_create_seq(algs->encr, algs->encr_size);

/* call resets sequence numbers: */
this->protection->set_cipher(this->protection, TRUE, this->aead_in);
this->protection->set_cipher(this->protection, FALSE, this->aead_out);

}
/* ... [error messages]  */
return TRUE;
}

/* File tls_protection.c */
METHOD(tls_protection_t, set_cipher, void,
private_tls_protection_t *this, bool inbound, tls_aead_t *aead)
{
if (inbound)
{
this->aead_in = aead;
this->seq_in = 0; /* sequence number reset */
}
else
{
this->aead_out = aead;
this->seq_out = 0; /* sequence number reset */
}
}
```

The sequence number is reset to zero when the `Finished` message is sent by the client to conclude the handshake: As soon as the state `STATE_FINISHED_SENT` is set, a call to `this->crypto->change_cipher` resets the sequence numbers.

[37] Res18, p. 82.

[38] "Each sequence number is set to zero at the beginning of a connection and whenever the key is changed" [Res18, p. 82]. Since TLS is built on top of TCP, packet loss or disorder is not a problem.

[39] See section 3.3.3.

3.7. Tests

3.7.1. Test Environment

The TLS library in strongSwan comes with a set of basic unit test suites. `test_socket.c` specifies six client-server tests, which test both peers for the TLS protocol version 1.0 to TLS 1.2. `test_suites.c` tests the cipher suite names. We left both test suites untouched as discussed with our advisors since `test_socket.c` is unable to complete because of our changes on the client-side TLS implementation. Further we added `test_hkdf.c` to verify the functionality of our newly-implemented HKDF feature.[40] Except for the HKDF implementation, we excluded memory leak detection due to time constraints.

It has to be noted that we only implemented the client-side TLS 1.3 stack, so we were not able to write unit tests similar to `test_socket.c` to automatically verify the functionality of our implementation. We had to test everything semi-automated with self-written shell scripts. The basic test setup consists of an OpenSSL server as described in appendix C and the freshly compiled `tls_test` binary.

We also manually tested the functionality of our TLS 1.3 client with servers on the Internet, e.g. `www.google.com` on port 443 as described in appendix E. It has to be mentioned that this approach is very error-prone, because the server certificate validation is non-deterministic. `tls_test` requires the `--cert` parameter to point to the server's public key to verify authenticity of the connection. Since Google runs many servers with different server certificates we were not able to establish a connection with every attempt.[41]

In table 3.9 we provide an overview of our semi-automated test cases for successful TLS version negotiation of various combinations. Test data of all our test cases can be found in appendix E.

Client max TLS version	**Server max TLS version**	**Negotiated TLS version**	**Test Result**
TLS 1.3	TLS 1.3	TLS 1.3	✓
TLS 1.2	TLS 1.3	TLS 1.2	✓
TLS 1.2	TLS 1.2	TLS 1.2	✓
TLS 1.3	TLS 1.2	TLS 1.2	✓

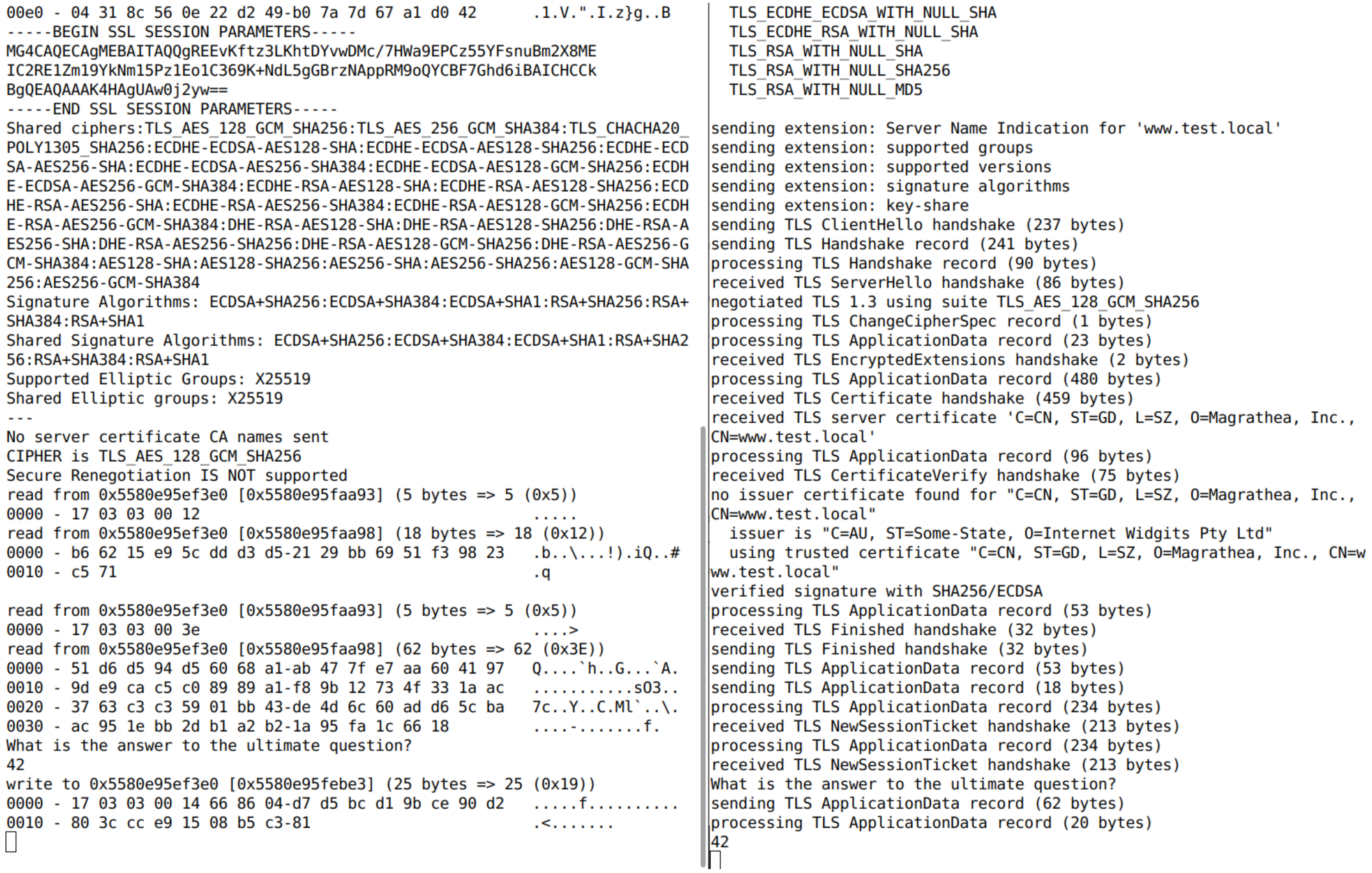
Figure 3.9.: Successful TLS negotiation test cases.

3.7.2. Successful Handshake

On the next page we see a successful TLS 1.3 handshake between an OpenSSL server (left) and a strongSwan client (right). On the client side, the list of offered ciphers suites is visible at the top, albeit cut-off for brevity.

[40]The HKDF implementation passes all tests: `https://travis-ci.org/github/bytinbit/strongswan/builds/689415018`, visited on 2020-05-23.

[41]This happens because on a first connection, the server certificate from `www.google.com` is downloaded via a browser, and in a second step our client uses the certificate to connect to `www.google.com`. It is not guaranteed that the client then connects to the same server with the same certificate as in the first step.



```
00e0 - 04 31 8c 56 0e 22 d2 49-b0 7a 7d 67 a1 d0 42      .1.V.".I.z}g..B
-----BEGIN SSL SESSION PARAMETERS-----
MG4CAQECAgMEBAITAQQgREEvKftz3LKhtDYvwDMc/7HWa9EPCz55YFsnuBm2X8ME
IC2RE1Zm19YkNm15Pz1Eo1C369K+NdL5gGBrzNAppRM9oQYCBF7Ghd6iBAICHCCk
BgQEAQAAAK4HAgUAw0j2yw==
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:TLS_CHACHA20_
POLY1305_SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA256:ECDHE-ECD
SA-AES256-SHA:ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:ECDH
E-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES128-SHA256:ECD
HE-RSA-AES256-SHA:ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-GCM-SHA256:ECDH
E-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA256:DHE-RSA-A
ES256-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-G
CM-SHA384:AES128-SHA:AES128-SHA256:AES256-SHA:AES256-SHA256:AES128-GCM-SHA
256:AES256-GCM-SHA384
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:RSA+SHA256:RSA+
SHA384:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:RSA+SHA2
56:RSA+SHA384:RSA+SHA1
Supported Elliptic Groups: X25519
Shared Elliptic groups: X25519
---
No server certificate CA names sent
CIPHER is TLS_AES_128_GCM_SHA256
Secure Renegotiation IS NOT supported
read from 0x5580e95ef3e0 [0x5580e95faa93] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 12                                    .....
read from 0x5580e95ef3e0 [0x5580e95faa98] (18 bytes => 18 (0x12))
0000 - b6 62 15 e9 5c dd d3 d5-21 29 bb 69 51 f3 98 23   .b..\...!).iQ..#
0010 - c5 71                                             .q

read from 0x5580e95ef3e0 [0x5580e95faa93] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 3e                                    ....>
read from 0x5580e95ef3e0 [0x5580e95faa98] (62 bytes => 62 (0x3E))
0000 - 51 d6 d5 94 d5 60 68 a1-ab 47 7f e7 aa 60 41 97   Q....`h..G...`A.
0010 - 9d e9 ca c5 c0 89 89 a1-f8 9b 12 73 4f 33 1a ac   ...........sO3..
0020 - 37 63 c3 c3 59 01 bb 43-de 4d 6c 60 ad d6 5c ba   7c..Y..C.Ml`..\.
0030 - ac 95 1e bb 2d b1 a2 b2-1a 95 fa 1c 66 18         ....-.......f.
What is the answer to the ultimate question?
42
write to 0x5580e95ef3e0 [0x5580e95febe3] (25 bytes => 25 (0x19))
0000 - 17 03 03 00 14 66 86 04-d7 d5 bc d1 9b ce 90 d2   .....f..........
0010 - 80 3c cc e9 15 08 b5 c3-81                        .<.......
```

```
  TLS_ECDHE_ECDSA_WITH_NULL_SHA
  TLS_ECDHE_RSA_WITH_NULL_SHA
  TLS_RSA_WITH_NULL_SHA
  TLS_RSA_WITH_NULL_SHA256
  TLS_RSA_WITH_NULL_MD5

sending extension: Server Name Indication for 'www.test.local'
sending extension: supported groups
sending extension: supported versions
sending extension: signature algorithms
sending extension: key-share
sending TLS ClientHello handshake (237 bytes)
sending TLS Handshake record (241 bytes)
processing TLS Handshake record (90 bytes)
received TLS ServerHello handshake (86 bytes)
negotiated TLS 1.3 using suite TLS_AES_128_GCM_SHA256
processing TLS ChangeCipherSpec record (1 bytes)
processing TLS ApplicationData record (23 bytes)
received TLS EncryptedExtensions handshake (2 bytes)
processing TLS ApplicationData record (480 bytes)
received TLS Certificate handshake (459 bytes)
received TLS server certificate 'C=CN, ST=GD, L=SZ, O=Magrathea, Inc.,
CN=www.test.local'
processing TLS ApplicationData record (96 bytes)
received TLS CertificateVerify handshake (75 bytes)
no issuer certificate found for "C=CN, ST=GD, L=SZ, O=Magrathea, Inc.,
CN=www.test.local"
  issuer is "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd"
  using trusted certificate "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=w
ww.test.local"
verified signature with SHA256/ECDSA
processing TLS ApplicationData record (53 bytes)
received TLS Finished handshake (32 bytes)
sending TLS Finished handshake (32 bytes)
sending TLS ApplicationData record (53 bytes)
sending TLS ApplicationData record (18 bytes)
processing TLS ApplicationData record (234 bytes)
received TLS NewSessionTicket handshake (213 bytes)
processing TLS ApplicationData record (234 bytes)
received TLS NewSessionTicket handshake (213 bytes)
What is the answer to the ultimate question?
sending TLS ApplicationData record (62 bytes)
processing TLS ApplicationData record (20 bytes)
42
```

Figure 3.10.: Screenshot of a handshake between a TLS 1.3 server (left) and a strongSwan client (right, 2020-05-21).

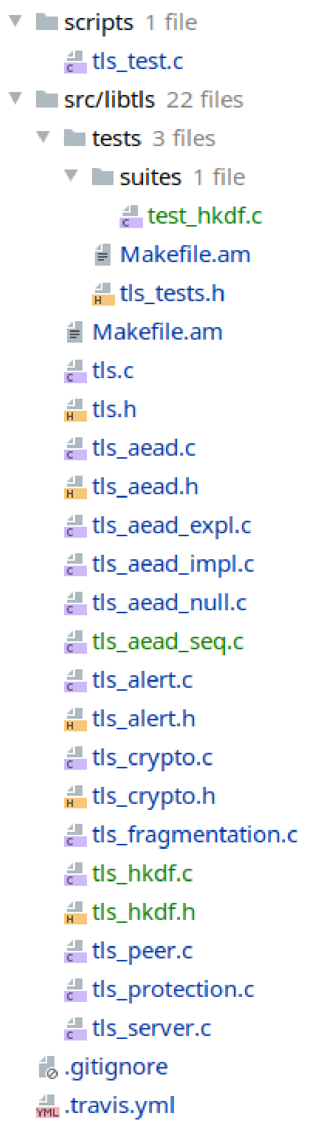
4. Results

The project entailed complex code, the terseness of an RFC and cryptographic operations which had to be understood *and* implemented. It was a wise decision to define modest goals and everything else as optional. In the end, we were able to implement the minimal viable product: We are able to successfully establish a TLS 1.3 connection between a client and a server and thus wrote the client-side of a TLS 1.3 connection. Figure 4.1 shows all files that were affected by our project, where green color indicates new files. Even if `tls_aead_seq.c` was added, its code affected all other `aead` files.[1]

TLS 1.3 uses the new HKDF scheme to derive cryptographic keys. We had to implement this functionality while drawing on existing cryptographic primitives in strongSwan. The newly created implementation works fine from a functional perspective, unit tests with known traffic data and keying material are successful. Since RFC 8446 scatters explanations of the HKDF's application throughout the text, our implementation of the HKDF grew organically along as we worked our way through the handshake. It must be left to future work to refactor the existing code and introduce a more simple and complete way to generate key material via the HKDF.

During the integration of our HKDF into the client-side handshake, it became clear that our initial concept of an HKDF implementation was not ideal and has some flaws. We therefore suggest the following implementation that is more robust and easier-to-use.

The public method `generate_secret` controls the flow of the internal state machine by the label it is called with. The method returns the secrets `Early Secret`, `Handshake Secret` and `Master Secret`. However, they are never used directly to derive concrete secrets. The code can therefore be improved as follows:



```
▼ scripts 1 file
    tls_test.c
▼ src/libtls 22 files
  ▼ tests 3 files
    ▼ suites 1 file
        test_hkdf.c
      Makefile.am
      tls_tests.h
    Makefile.am
    tls.c
    tls.h
    tls_aead.c
    tls_aead.h
    tls_aead_expl.c
    tls_aead_impl.c
    tls_aead_null.c
    tls_aead_seq.c
    tls_alert.c
    tls_alert.h
    tls_crypto.c
    tls_crypto.h
    tls_fragmentation.c
    tls_hkdf.c
    tls_hkdf.h
    tls_peer.c
    tls_protection.c
    tls_server.c
  .gitignore
  .travis.yml
```

Figure 4.1.: Affected files

- Remove return value parameter in `generate_secret`, since the secrets are never used. This would simplify the API to callers.
- Remove the whole function and control the internal state machine flow directly with the other methods. This would reduce the amount of

[1]See also section 3.3.3.

public methods, but would lead to additional parameters for the functions `derive_key`, `derive_iv` and `derive_finished`.

The public method `derive_key` and `derive_iv` may be refactored to one method to simplify the interface even more. The public method `derive_finished` uses the same private function to compute its secret as `derive_key` and `derive_iv` and has no validity check if it is in phase 3. This could be improved by guaranteeing that the function only provides a secret when it is called in the correct state.

The above list is a set of (incomplete) suggestions which may not work well with each other and must be reviewed with the maintainers in more detail before they are implemented.

We did not implement the server side (option 1) and neither the client-side authentication with a certificate (option 2).[2] What prolonged our implementation process were the HKDF itself, the complexities of extending the existing state machine and the final transition to an established connection. Future work must implement the two missing options that are relevant to strongSwan – since the server-side is still missing, our implementation only works with an (external) TLS 1.3-server.[3]

In section 2.4.1 we defined all mandatory extensions that our client should support. In the end, we implemented all except one, the `cookie` extension, after the maintainers agreed that it was unneeded in the strongSwan project. The `HelloRetryRequest` message, which is also listed in our scope, was not implemented.

Regarding the cryptographic primitives, we added support for all new five cipher suites defined in RFC 8446. Our HKDF implementation supports both hashing algorithms.[4] As key exchange method we currently only support Elliptic Curve Diffie-Hellman (ECDHE) with curve X25519.[5] The reason for this is how key exchange mechanisms differ between TLS versions. TLS 1.2 and prior used information from the `ClientKeyExchange` message to calculate key material, TLS 1.3 calculates the client key material at the beginning and then sends it in the `ClientHello` message, thus in the first message of the client to the server. In addition, older TLS implementations only calculated one key pair using a negotiated algorithm. A TLS 1.3 client initially sends its key material generated with different algorithms and simply hopes that the server supports one of these. Therefore, other TLS 1.3 implementations usually calculate two key pairs.[6] The existing code base currently does not offer a uniform way to calculate and store key exchange material from multiple algorithms. Because of this, we decided to support curve X25519 only.[7]

As signature hash algorithms, we send these six in the `signature_algorithms` extension:

- `ecdsa_secp256r1_sha256` (0x0403)
- `ecdsa_secp384r1_sha384` (0x0503)

[2]See also section 2.4.2

[3]See appendix C on how to generate the key material and setup an OpenSSL server to see our code in action.

[4]RFC 8446 specifies SHA-256 and SHA-384

[5]See also RFC 7748, "Elliptic Curves for Security": `https://tools.ietf.org/html/rfc7748`, visited on 2020-05-24.

[6]Mozilla Firefox for example calculates and offers key exchange for secp256r1 and X25519, and the server picks its preference of the two.

[7]According to the RFC, a TLS 1.3-compliant application must support secp256r1 and should support X25519. We chose the latter due to personal preferences although secp256r1 is also supported by strongSwan and should work as well, though we never actually tested this.

- `ecdsa_sha1` (0x0203)
- `rsa_pkcs1_sha256` (0x0401)
- `rsa_pkcs1_sha384` (0x0501)
- `rsa_pkcs1_sha1` (0x0201)

This is a subset of all the definitions on page 42 in RFC 8446. The implementation of the extension is dynamic, which means that the amount of supported signature hashes varies depending on how the compilation flags have been set.

In section 3.2 we decided to do an in-place TLS version distinction rather than write everything from scratch. As expected, it added complexity to the current code, and it is not easily to distinguish which code accounts for which TLS version. It may be another task to refactor the code to the point that files distinguish the TLS versions and not `if-else` branches.

Finally, this study project was a project like no other: Even though we faced a global pandemic thanks to COVID-19 with drastic changes to our daily lives, we are proud to have achieved our main goal nevertheless. Even though we did not build Rome in a day, we at least built a client-side TLS 1.3 stack.

A. List of Abbreviations

AEAD Authenticated Encryption with Associated Data

CA Certificate Authority

CSR Certificate Signing Request

EAP Extensible Authentication Protocol

EAP-PEAP EAP with Protected Extensible Authentication Protocol

EAP-TLS EAP with Transport Layer Security

EAP-TTLS EAP with Tunneled Transport Layer Security

GCM Galois/Counter Mode

HKDF HMAC-based Extract-and-Expand Key Derivation Function (RFC 5869)

ECDHE Elliptic-Curve Diffie-Hellman Ephemeral

HMAC Keyed-Hash Message Authentication Code

IKE Internet Key Exchange

IKM Input Key Material

IV Initialisation Vector

MVP Minimal Viable Product

OKM Output Key Material

OSI Open Systems Interconnection Model

PKI Public Key Infrastructure

PRF Pseudo-Random Function

PSK Pre-Shared Key

PT-TLS Posture Transport Protocol over TLS

RFC Request For Comment

RSA Rivest-Shamir-Adleman

RTT Round-Trip Time

SA Security Association

TLS Transport Layer Security

B. Bibliography

[Aum18] Aumasson, Jean-Philippe. *Serious Cryptography*. No Starch Press, 2018.

[Beu17] Beurdouche, Benjamin and Bhargavan, Karthikeyan et al. "A messy state of the union: taming the composite state machines of TLS". In: *Commun. ACM* 60.2 (2017), pp. 99–107. DOI: 10.1145/3023357. URL: https://doi.org/10.1145/3023357.

[Bon15] Boneh, Dan. *Cryptography: Authenticated Encryption*. 2015. URL: https://www.youtube.com/watch?v=40m3gcdGDu0 (visited on 22/05/2020).

[Bon20] Boneh, Dan. *Key Derivation*. 2020. URL: https://www.coursera.org/lecture/crypto/key-derivation-A1ETP (visited on 12/05/2020).

[Dri] Driscoll, Michael. *The New Illustrated TLS Connection*. URL: https://tls13.ulfheim.net/ (visited on 21/03/2020).

[Fal12] Fall, Kevin R. and Stevens, Richard W. *TCP/IP illustrated, Volume 1. The Protocols*. 2nd ed. Addison-Wesley, 2012.

[Hol19] Holz, Ralph et al. *The Era of TLS 1.3: Measuring Deployment and Use with Active and Passive Methods*. 2019. arXiv: 1907.12762 [cs.CR].

[Hut05] Hutter, Jan and Willi, Martin. *strongSwan II. Eine IKEv2-Implementierung für Linux*. Diplomarbeit. Hochschule für Technik Rapperswil, Dec. 2005, pp. 1–186. URL: http://security.hsr.ch/theses/DA_2005_IKEv2.pdf.

[Kra10] Krawczyk, Hugo. *HMAC-based Extract-and-Expand Key Derivation Function (HKDF)*. RFC 5869. IETF Tools, May 2010, pp. 1–14. URL: https://tools.ietf.org/html/rfc5869.

[Res18] Rescorla, Eric. *The Transport Layer Security (TLS) Protocol Version 1.3*. RFC 8446. IETF Tools, Aug. 2018, pp. 1–160. URL: https://tools.ietf.org/html/rfc8446.

[Res08] Rescorla, Eric and Dierks, Tim. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5216. IETF Tools, Aug. 2008, pp. 1–104. URL: `https://tools.ietf.org/html/rfc5246`.

[Ris17] Ristić, Ivan. *Bulletproof SSL and TLS. Understanding and Deploying SSL/TLS and PKI to Secure Servers and Web Applications*. Feisty Duck, 2017.

[San15] Sangster, Paul. *A Posture Transport Protocol over TLS (PT-TLS)*. RFC 6876. IETF Tools, Oct. 2015, pp. 1–44. URL: `https://tools.ietf.org/html/rfc6876`.

[Sch17] Schmidt, Jürgen. "Weniger ist mehr. Was die anstehende Version TLS 1.3 bringt". In: *c't* 4 (2017), pp. 172–174.

[Sim08] Simon, Dan et al. *The EAP-TLS Authentication Protocol*. RFC 5216. IETF Tools, Mar. 2008, pp. 1–34. URL: `https://tools.ietf.org/html/rfc5216`.

[Sta17] Stallings, William. *Network Security Essentials. Applications and Standards*. Pearson Education Limited, 2017.

[Ste05] Steffen, Andreas. *Advanced Features of Linux strongSwan*. Tech. rep. 2005. URL: `https://strongswan.org/docs/LinuxTag2005-strongSwan.pdf` (visited on 25/05/2020).

[str18] strongSwan. *About*. 2018. URL: `https://wiki.strongswan.org/about.html` (visited on 13/03/2020).

[str15] strongSwan. *EAP-TLS*. 2015. URL: `https://wiki.strongswan.org/projects/strongswan/wiki/EapTls` (visited on 23/02/2020).

[str17] strongSwan. *IKE keying daemon charon*. 2017. URL: `https://wiki.strongswan.org/projects/strongswan/wiki/Charon` (visited on 13/03/2020).

[Wik] Wikipedia. *Extensible Authentication Protocol*. URL: `https://en.wikipedia.org/wiki/Extensible_Authentication_Protocol` (visited on 13/03/2020).

C. Compile and Execution Instructions

C.1. Set up Test Infrastructure with OpenSSL

We assume an OpenSSL version is installed, at least version 1.1.1 (September 2018).

(1) Generate a self-signed certificate authority (CA) with password 1337 (required). The certificate authority can be created with a certificate signing request (CSR, step 2.1) or without (step 2.2).

```
# 1. create key
$ openssl ecparam -genkey -name secp256r1 | openssl ec -out ca.key -aes128

# 2.1a create csr
$ openssl req -new -key ca.key -out ca.csr

# 2.1b sign ca certificate
$ openssl x509 -req -days 365 -in ca.csr -signkey fd.key -out ca.crt

# 2.2 self-signed ca without a csr
$ openssl req -new -x509 -days 365 -key ca.key -out ca.crt
```

(2) Server: generate a strong private key with password 7331:

```
$ openssl ecparam -genkey -name secp256r1 | openssl ec -out server.key
 ↪  -aes128
```

(3) Create a Certificate Signing Request (CSR) and send it to a CA, which is the one created in (1):

```
$ openssl req -new -sha256 -key server.key -out server.csr
```

(4) Sign the certificate:

```
$ openssl x509 -req -CAcreateserial -in server.csr -sha256 -CA ca.crt
 ↪  -CAkey ca.key -out server.crt
```

In the end, the the following files exist in order of creation:

- Client side: `ca.key, (ca.csr,) ca.crt`
- Server side: `server.key, server.csr, server.crt`

Run the OpenSSL-server with password 7331:

```
$ openssl s_server -accept localhost:8443 -key server.key -cert server.crt
 ↪  -debug -pass pass:7331 -keylogfile keylogs
```

The flag -keylogfile keylogs writes all the keys to a file called keylogs. It allows Wireshark to decrypt sniffed traffic.[1]

Run the strongSwan client:

```
$ ../strongswan_root/scripts/tls_test --connect localhost --port 8443
↪  --cert server.crt --debug 2
```

C.2. Set up strongSwan

The most simple steps to set up strongSwan for our purpose is described in the following steps. Note that EAP-TLS and EAP-TTLS have to be explicitly enabled. Also the OpenSSL back-end to support most of the cryptographic primitives has to be configured explicitly.

```
$ git clone https://git.strongswan.org/strongswan.git
$ cd strongswan/
$ ./autogen.sh
$ make clean
$ ./configure --enable-eap-tls --enable-eap-ttls --enable-openssl
$ make
$ make install
```

Verify functionality by running all unit tests:

```
$ make check
```

The client can then be started by the following command:

```
$ ./scripts/tls_test
```

If only required plugins should be enabled, one can use the following configuration options:

```
$ ./configure --disable-aes --disable-des --disable-rc2 --disable-sha2
↪  --disable-sha1 --disable-md5 --disable-pgp  --disable-dnskey
↪  --disable-sshkey --disable-gmp --disable-xcbc --disable-cmac
↪  --disable-hmac --disable-random --disable-pkcs1 --disable-pkcs7
↪  --disable-pkcs8 --disable-pkcs12 --disable-attr
↪  --disable-kernel-netlink --disable-stroke --disable-vici
↪  --disable-updown --disable-xauth-generic --disable-counters
↪  --disable-resolve --disable-socket-default --disable-nonce
↪  --disable-x509 --disable-revocation --disable-fips-prf --disable-drbg
↪  --disable-constraints --disable-pubkey --disable-pem
↪  --disable-curve25519 --enable-eap-tls --enable-eap-ttls
↪  --enable-openssl
```

Initially we tried to use valgrind to manually find memory leaks. Our advisors pointed out the configuration flag --enable-leak-detective which provides the same validation constraints also used by the automated unit tests. We strongly recommend to always set this option while working on the strongSwan project. Now, when running unit tests by make check, memory leaks are checked as well.

[1]Via the menu points Edit -> Preferences -> (Pre)-Master-Secret log filename.

Temporarily disabling memory leak detection can be achieved by setting the flag LEAK_DETECTIVE_DISABLE. The following command also sets the flag TESTS_VERBOSE to print more verbose output to stdout.

```
$ make check TESTS_VERBOSE=2 LEAK_DETECTIVE_DISABLE=0
```

D. Source Code

D.0.1. Code-Repository

The complete source code of our implementation can be found online here:

`https://github.com/bytinbit/strongswan`

Interesting branches are:

- `sa-dev-libtls`: the final code
- `feature-mvp`: complete commit history until code freeze
- `feature-hkdf`: complete commit history of the HKDF implementation until code freeze

D.0.2. HKDF Boilerplate Code

The strongSwan maintainer Tobias Brunner[1] provided the boilerplate code for an HKDF, implemented in strongSwan's code base. The implementation conforms to RFC 5869.[2]

```
#include <crypto/prf_plus.h>

prf_t *prf;
prf_plus_t *prf_plus;
chunk_t salt, IKM, PRK, info, OKM;
size_t L;

prf = lib->crypto->create_prf(lib->crypto, PRF_HMAC_SHA2_256);

/* HKDF-Extract(salt, IKM) -> PRK */
salt = chunk_from_chars(
    0x00, 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x07,
    0x08, 0x09, 0x0a, 0x0b, 0x0c,
);
IKM = chunk_from_chars(
    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
    0x0b, 0x0b, 0x0b, 0x0b, 0x0b, 0x0b,
);
prf->set_key(prf, salt);
prf->allocate_bytes(prf, IKM, &PRK);
```

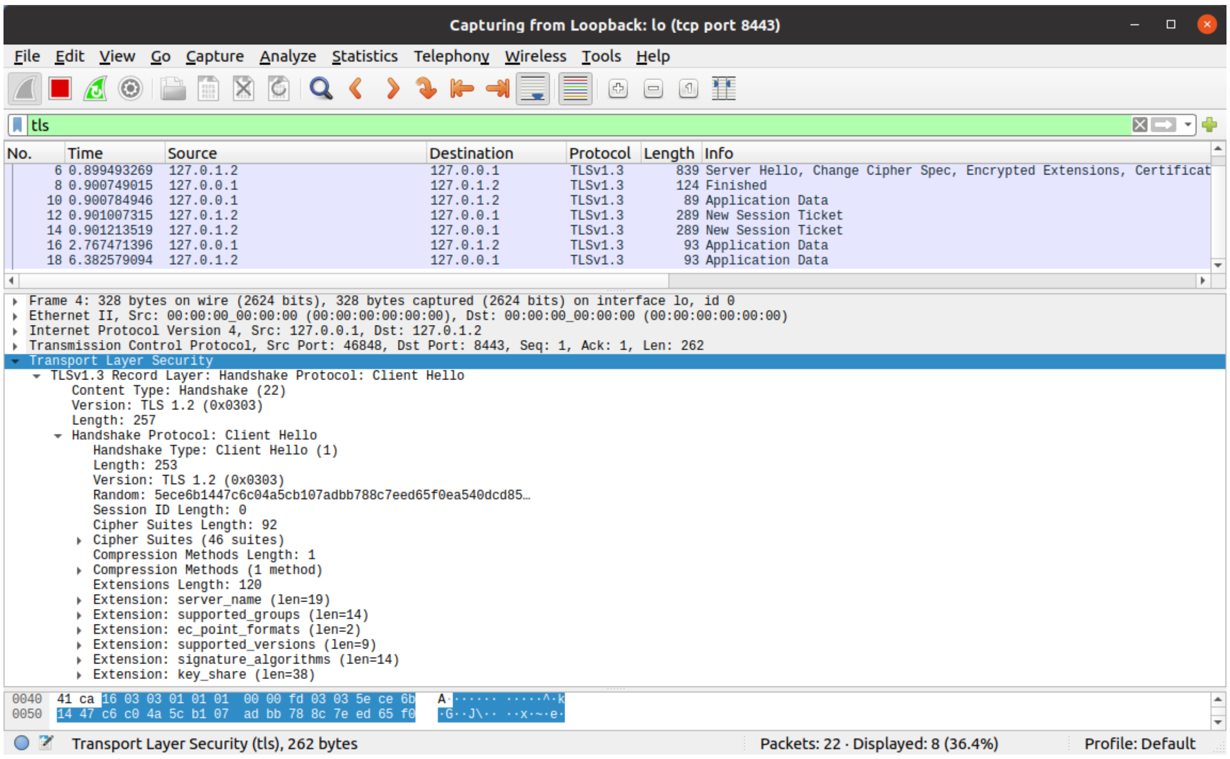
[1]Email from 26.3.2020

[2]For the implementation in TLS 1.3, see section 3.3.

```
DBG1(DBG_APP, "=== %B", &PRK);

/* HKDF-Expand(PRK, info, L) -> OKM */
info = chunk_from_chars(
    0xf0, 0xf1, 0xf2, 0xf3, 0xf4, 0xf5, 0xf6, 0xf7,
    0xf8, 0xf9,
);
L = 42;
prf->set_key(prf, PRK);
prf_plus = prf_plus_create(prf, TRUE, info);
prf_plus->allocate_bytes(prf_plus, L, &OKM);
DBG1(DBG_APP, "=== %B", &OKM);

chunk_clear(&PRK);
chunk_clear(&OKM);
prf_plus->destroy(prf_plus);
prf->destroy(prf);
```


Capturing from Loopback: lo (tcp port 8443)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tls

```
No. Time         Source     Destination Protocol Length Info
  6 0.899493269  127.0.1.2  127.0.0.1   TLSv1.3   839 Server Hello, Change Cipher Spec, Encrypted Extensions, Certificat
  8 0.900749015  127.0.0.1  127.0.1.2   TLSv1.3   124 Finished
 10 0.900784946  127.0.0.1  127.0.1.2   TLSv1.3    89 Application Data
 12 0.901007315  127.0.1.2  127.0.0.1   TLSv1.3   289 New Session Ticket
 14 0.901213519  127.0.1.2  127.0.0.1   TLSv1.3   289 New Session Ticket
 16 2.767471396  127.0.0.1  127.0.1.2   TLSv1.3    93 Application Data
 18 6.382579094  127.0.1.2  127.0.0.1   TLSv1.3    93 Application Data

Frame 4: 328 bytes on wire (2624 bits), 328 bytes captured (2624 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.2
Transmission Control Protocol, Src Port: 46848, Dst Port: 8443, Seq: 1, Ack: 1, Len: 262
Transport Layer Security
  TLSv1.3 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 257
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 253
      Version: TLS 1.2 (0x0303)
      Random: 5ece6b1447c6c04a5cb107adbb788c7eed65f0ea540dcd85…
      Session ID Length: 0
      Cipher Suites Length: 92
      Cipher Suites (46 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 120
      Extension: server_name (len=19)
      Extension: supported_groups (len=14)
      Extension: ec_point_formats (len=2)
      Extension: supported_versions (len=9)
      Extension: signature_algorithms (len=14)
      Extension: key_share (len=38)

0040  41 ca 16 03 03 01 01 01  00 00 fd 03 03 5e ce 6b   A·······  ·····^·k
0050  14 47 c6 c0 4a 5c b1 07  ad bb 78 8c 7e ed 65 f0   ·G··J\··  ··x·~·e·

Transport Layer Security (tls), 262 bytes        Packets: 22 · Displayed: 8 (36.4%)        Profile: Default
```

Figure E.1.: Screenshot of a handshake between a strongSwan TLS 1.3 client and an OpenSSL server supporting TLS 1.3 in Wireshark (2020-05-27).

E. Test Data

E.1. Local Connection: TLS 1.3 Client to TLS 1.3 Server

A successful TLS 1.3 handshake, server side

```
openssl s_server -accept www.test.local:8443 -key tls_ecdsa/server.key -cert
↪  tls_ecdsa/server.crt -debug -pass pass:7331 -keylogfile output/keylogs
Using default temp DH parameters
ACCEPT
read from 0x5562d4480b40 [0x5562d448c753] (5 bytes => 5 (0x5))
0000 - 16 03 03 01 01                                    .....
read from 0x5562d4480b40 [0x5562d448c758] (257 bytes => 257 (0x101))
0000 - 01 00 00 fd 03 03 5e ce-6b 14 47 c6 c0 4a 5c b1   ......^.k.G..J\.
0010 - 07 ad bb 78 8c 7e ed 65-f0 ea 54 0d cd 85 f0 fa   ...x.~.e..T.....
0020 - 3f ae 7d 56 02 91 00 00-5c 13 01 13 02 13 03 13   ?.}V....\.......
0030 - 04 13 05 c0 09 c0 23 c0-0a c0 24 c0 2b c0 2c c0   ......#...$.+.,.
0040 - 13 c0 27 c0 14 c0 28 c0-2f c0 30 00 33 00 67 00   ..'...(./.0.3.g.
0050 - 39 00 6b 00 9e 00 9f 00-45 00 be 00 88 00 c4 00   9.k.....E.......
0060 - 16 00 2f 00 3c 00 35 00-3d 00 9c 00 9d 00 41 00   ../.<.5.=.....A.
0070 - ba 00 84 00 c0 c0 08 c0-12 00 0a c0 06 c0 10 00   ................
```

```
0080 - 02 00 3b 00 01 01 00 00-78 00 00 00 13 00 11 00   ..;.....x.......
0090 - 00 0e 77 77 77 2e 74 65-73 74 2e 6c 6f 63 61 6c   ..www.test.local
00a0 - 00 0a 00 0e 00 0c 00 17-00 18 00 19 00 15 00 13   ................
00b0 - 00 1d 00 0b 00 02 01 00-00 2b 00 09 08 03 04 03   .........+......
00c0 - 03 03 02 03 01 00 0d 00-0e 00 0c 04 03 05 03 02   ................
00d0 - 03 04 01 05 01 02 01 00-33 00 26 00 24 00 1d 00   ........3.&.$...
00e0 - 20 eb ed 56 16 fd 54 6c-b8 bb 12 aa d7 43 40 5d    ..V..Tl.....C@]
00f0 - ac 30 72 61 41 42 ef 8a-8e f2 f7 77 06 59 70 32   .0raAB.....w.Yp2
0100 - 43                                                C
write to 0x5562d4480b40 [0x5562d4495a20] (773 bytes => 773 (0x305))
0000 - 16 03 03 00 5a 02 00 00-56 03 03 df 77 b6 93 b9   ....Z...V...w...
0010 - bc f1 89 4c d1 37 ab ec-e6 e5 30 ed aa 19 0f 60   ...L.7....0....`
0020 - 56 74 b2 b1 bb 0d 7a e2-e8 d8 31 00 13 01 00 00   Vt....z...1.....
0030 - 2e 00 2b 00 02 03 04 00-33 00 24 00 1d 00 20 19   ..+.....3.$... .
0040 - 24 02 ed e7 e5 ee 5f 9d-94 15 fb b2 bf 0e e1 b0   $....._.........
0050 - f8 11 70 e6 18 f9 8e c7-2b 3c 97 5f b2 b6 21 14   ..p.....+<._..!.
0060 - 03 03 00 01 01 17 03 03-00 17 21 33 9d a5 00 8f   ..........!3....
0070 - a6 f3 5a 4a f0 92 9f b1-5f 14 1c f5 b0 45 4c 73   ..ZJ...._....ELs
0080 - 14 17 03 03 01 e0 fc 8d-39 24 5d 68 e7 53 ac 27   ........9$]h.S.'
0090 - 11 75 18 e6 6e 3a f8 0f-3f 4d 2f 0f 0b 17 d4 be   .u..n:..?M/.....
00a0 - 46 38 26 3f 5b b8 12 08-bb ae 8b 7a 48 e0 f8 4a   F8&?[......zH..J
00b0 - e2 cf b9 ac cc 45 27 60-d6 e8 9f 82 9d a6 e5 62   .....E'`.......b
00c0 - 3a 13 7a bf e1 60 f9 19-3e 5c c3 dd 1a 0c 85 1b   :.z..`..>\......
00d0 - 1c 25 ba 03 5b b1 42 4a-6f 3a 04 b7 24 d2 62 21   .%..[.BJo:..$.b!
00e0 - 1f 9b 34 fc ab 16 f1 96-2e 28 7a 53 7c 3f 21 13   ..4......(zS|?!.
00f0 - b0 62 9e f8 56 1b 66 d3-a4 8b 1e c1 9d 0e 06 5e   .b..V.f........^
0100 - ef 71 cb d6 25 85 1c 89-9f 0a 54 77 8c 4c 74 33   .q..%.....Tw.Lt3
0110 - d8 df 34 e0 b8 f1 62 4d-d3 f2 64 88 59 74 c5 77   ..4...bM..d.Yt.w
0120 - 72 53 f6 18 60 d2 ae 4c-0f 6e db c9 18 8f 8f 13   rS..`..L.n......
0130 - 0e 60 2f fc 41 05 97 be-92 13 29 a6 74 2b 82 93   .`/.A.....).t+..
0140 - 84 4b 7f e6 3a 46 64 f5-1f a8 f0 c0 45 d6 c7 b6   .K..:Fd.....E...
0150 - ae 8c c6 55 66 98 b6 03-25 e2 36 4a 68 de 8d fe   ...Uf...%.6Jh...
0160 - 7b 94 a1 69 88 3a 20 f9-03 17 53 34 1a b3 24 53   {..i.: ...S4..$S
0170 - 51 9a 0d 15 9d 68 c9 b5-38 6a 80 d5 30 50 86 ac   Q....h..8j..0P..
0180 - 6c 69 bf 9d bb 6b fe b1-06 c0 e2 9e 6e 61 f2 b3   li...k......na..
0190 - 38 20 75 8c 97 7e 6d 9e-9b e2 2b 5c fa 03 60 54   8 u..~m...+\..`T
01a0 - fa 2d 60 d7 8f cc f8 1d-39 b8 71 ea 93 73 65 4e   .-`.....9.q..seN
01b0 - 54 62 74 4d fd 48 c5 d2-13 22 b9 26 47 ad 72 e3   TbtM.H...".&G.r.
01c0 - a7 04 8f 9d 33 3c 87 a8-f8 d5 a7 c3 82 cb 33 94   ....3<........3.
01d0 - 28 3f dd 11 db 3e 67 17-0d d7 69 0d 47 e8 0a 40   (?...>g...i.G..@
01e0 - 00 aa 2e 93 3d c8 ee 96-a8 08 94 2a 36 ad 88 26   ....=......*6..&
01f0 - 40 25 b8 47 5a 16 6f 61-53 2f f1 ac 16 4e d3 b4   @%.GZ.oaS/...N..
0200 - 38 d9 56 06 d1 a2 95 2d-ce ec 95 be e1 d9 d2 d8   8.V....-........
0210 - 2a c0 11 ae 82 79 67 bc-99 f2 04 76 0c 4a 5b 64   *....yg....v.J[d
0220 - fb f1 89 3d 73 98 dc 08-c9 8d a7 50 42 ee ca 03   ...=s......PB...
0230 - 56 7c 59 54 20 75 28 b9-85 4e 1e eb 0d 70 ac 62   V|YT u(..N...p.b
0240 - 10 a4 73 d0 82 bc 88 80-33 55 28 86 6b 83 01 57   ..s.....3U(.k..W
0250 - 37 f3 f6 a0 7d f8 73 28-af 72 cc 95 e6 c8 14 6b   7...}.s(.r.....k
0260 - 2e 42 a6 b1 36 c0 17 03-03 00 60 fe 36 13 64 79   .B..6.....`.6.dy
0270 - d4 5a 05 4a 44 40 ba 41-5b 75 05 76 be bb 82 64   .Z.JD@.A[u.v...d
0280 - 08 96 21 94 e1 33 bc a5-58 96 36 8d 15 24 ba 1c   ..!..3..X.6..$..
0290 - 62 4e 0b 44 f1 6f e4 93-72 26 19 21 0d 63 09 fe   bN.D.o..r&.!.c..
02a0 - ea 86 19 3a be 4a 55 cd-ab 7f 62 2b bb a9 0f b3   ...:.JU...b+....
02b0 - aa 80 54 fd df 8a 44 77-30 39 2d 59 08 03 d3 72   ..T...Dw09-Y...r
02c0 - 4e 77 d8 6f 8c 81 97 63-93 73 ed 17 03 03 00 35   Nw.o...c.s.....5
02d0 - fe 4e 27 96 f1 e9 38 b9-5a f5 a3 ca 50 81 7d f0   .N'...8.Z...P.}.
02e0 - ef 89 98 90 8f 0b 52 c5-82 66 53 1f 1b 79 30 52   ......R..fS..y0R
02f0 - b9 fe ee 74 9d 95 68 c7-6d 56 49 1c a2 cd 02 90   ...t..h.mVI.....
0300 - 83 75 07 0e f5                                    .u...
```

```
read from 0x5562d4480b40 [0x5562d448c753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 35                                    ....5
read from 0x5562d4480b40 [0x5562d448c758] (53 bytes => 53 (0x35))
0000 - 24 d4 05 8c 40 3f f8 72-b1 4c 5d 91 a6 6f bf be   $...@?.r.L]..o..
0010 - ca 2c e0 90 1b f6 24 60-a1 bc d7 34 d0 e9 98 51   .,....$`...4...Q
0020 - 8b 5e 83 25 08 ad 9e 18-7a 91 a3 d8 e6 ec d6 78   .^.%....z......x
0030 - ec cb 12 22 41                                    ..."A
write to 0x5562d4480b40 [0x5562d4495a20] (223 bytes => 223 (0xDF))
0000 - 17 03 03 00 da db b6 12-a5 9c 64 c8 8d 48 66 53   ..........d..HfS
0010 - fe b2 47 fd 17 46 7a 09-67 44 26 6c cf 26 54 e3   ..G..Fz.gD&l.&T.
0020 - 54 b4 99 80 ed be c7 29-ce b6 93 8e 2a f7 b5 1b   T......)....*...
0030 - f0 2f a8 d8 c1 74 5f dc-5b d7 89 e4 aa e8 36 cd   ./...t_.[.....6.
0040 - 41 10 99 44 43 18 b7 4f-89 38 d8 ad af 6c db 5b   A..DC..O.8...l.[
0050 - c8 bf f9 bd 24 d2 fe a1-63 5b 73 1f 78 f5 b6 e9   ....$...c[s.x...
0060 - dd df 1a be c8 95 54 d6-6c 07 56 69 8a c7 f4 ba   ......T.l.Vi....
0070 - d9 a6 cd e1 1b f1 75 3d-dc f8 ad e2 fc d8 1c 57   ......u=.......W
0080 - 5d fe 48 20 86 0a 02 68-21 0e 23 75 ef 18 c7 91   ].H ...h!.#u....
0090 - 76 0d fd bc b1 39 09 8e-9d d1 ac 21 cc ca b0 15   v....9.....!....
00a0 - e6 c4 7b 40 0e 2d 8a 76-4d e1 42 6d 83 c5 25 4e   ..{@.-.vM.Bm..%N
00b0 - 13 bf 85 08 c2 b0 37 1d-28 34 c9 a4 22 62 45 dc   ......7.(4.."bE.
00c0 - e1 a9 a8 17 cf 8f 9e 0c-e4 85 b5 61 3f 97 6e e2   ...........a?.n.
00d0 - b1 7f 47 e0 ef 95 fe 3a-a2 07 ff f3 59 6d b0      ..G....:....Ym.
write to 0x5562d4480b40 [0x5562d4495a20] (223 bytes => 223 (0xDF))
0000 - 17 03 03 00 da 8e 3f e9-e5 73 8c 3c 7f 8f 82 03   ......?..s.<....
0010 - 1f 1b 49 9e 80 0d 15 c8-f8 5a 88 4f 46 08 56 cf   ..I......Z.OF.V.
0020 - 40 b9 d0 06 70 36 38 25-10 4a 2b cc a3 cf 33 31   @...p68%.J+...31
0030 - ec 06 28 fd ab a9 7b 9e-f5 2c e9 2a d5 8c e7 3b   ..(...{..,.*...;
0040 - 97 9b 56 3b 00 f7 3b e4-ab c7 5e 74 a0 e9 50 8e   ..V;..;...^t..P.
0050 - 6a 42 d7 0a ae 66 a9 f7-d7 e1 45 d1 7d 51 25 ff   jB...f....E.}Q%.
0060 - fe ed 85 8c 0d 3d 2f ef-96 5a 4b 01 92 22 ff 74   .....=/..ZK..".t
0070 - aa 61 95 9d ba 3d 50 d6-01 ee b6 17 4b dc a6 b6   .a...=P.....K...
0080 - 43 ab 44 36 8b 4e cd 4c-18 95 c4 64 33 61 a3 57   C.D6.N.L...d3a.W
0090 - 46 2e 61 ee b9 ad f3 a6-4c e9 47 31 fd f6 ad 50   F.a.....L.G1...P
00a0 - 39 f1 10 84 0c af 63 92-7e 73 13 94 cb 8b bd fc   9.....c.~s......
00b0 - a8 a9 dd 77 96 28 5d 65-31 d3 ae 73 f5 49 dc 55   ...w.(]e1..s.I.U
00c0 - 39 17 f8 e3 cc bc 42 d4-fa 0b 95 53 8a 66 78 1b   9.....B....S.fx.
00d0 - 47 f8 89 e2 3e aa a1 3b-8c 2f e0 06 d4 dd be      G...>..;./.....
-----BEGIN SSL SESSION PARAMETERS-----
MGOCAQECAgMEBAITAQQgsSDqsplaEIxW5CaQn+eATPg+Ii5zTWltXUSvl4hDEVAE
IHn1DkWBsz3g47ncOsQbjmJtwFGmN0eTMVRtgfypWLpRoQYCBF7OaxSiBAICHCCk
BgQEAQAAAK4GAgRKAKVP
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
TLS_CHACHA20_POLY1305_SHA256:ECDHE-ECDSA-AES128-SHA:ECDHE-ECDSA-AES128-SHA256:
ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA384:
ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:
ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-GCM-SHA256:
ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA256:
DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-GCM-SHA256:
DHE-RSA-AES256-GCM-SHA384:AES128-SHA:AES128-SHA256:
AES256-SHA:AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:
RSA+SHA256:RSA+SHA384:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:
RSA+SHA256:RSA+SHA384
Supported Elliptic Groups: P-256:P-384:P-521:P-224:P-192:X25519
Shared Elliptic groups: P-256:P-384:P-521:X25519
CIPHER is TLS_AES_128_GCM_SHA256
```

```
Secure Renegotiation IS NOT supported
read from 0x5562d4480b40 [0x5562d448c753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 12                                    .....
read from 0x5562d4480b40 [0x5562d448c758] (18 bytes => 18 (0x12))
0000 - 52 55 ec 7f bb 6a 9e 0b-ec 5b ee ca 27 e6 b8 3c   RU...j...[..'..<
0010 - ee 3c                                             .<

read from 0x5562d4480b40 [0x5562d448c753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 16                                    .....
read from 0x5562d4480b40 [0x5562d448c758] (22 bytes => 22 (0x16))
0000 - e8 e3 a3 0e 9f 81 af 95-cf b0 95 dd 1e ba 41 86   ..............A.
0010 - 95 8f 8e 96 bf e2                                 ......
ping
pong
write to 0x5562d4480b40 [0x5562d44908a3] (27 bytes => 27 (0x1B))
0000 - 17 03 03 00 16 b9 fd 96-57 c6 54 f3 c4 e0 a2 90   ........W.T.....
0010 - 11 da f3 33 f2 d0 32 7e-c4 a5 f2                  ...3..2~...
read from 0x5562d4480b40 [0x5562d448c753] (5 bytes => 0 (0x0))
ERROR
shutting down SSL
CONNECTION CLOSED
```

A successful TLS 1.3 handshake, client side

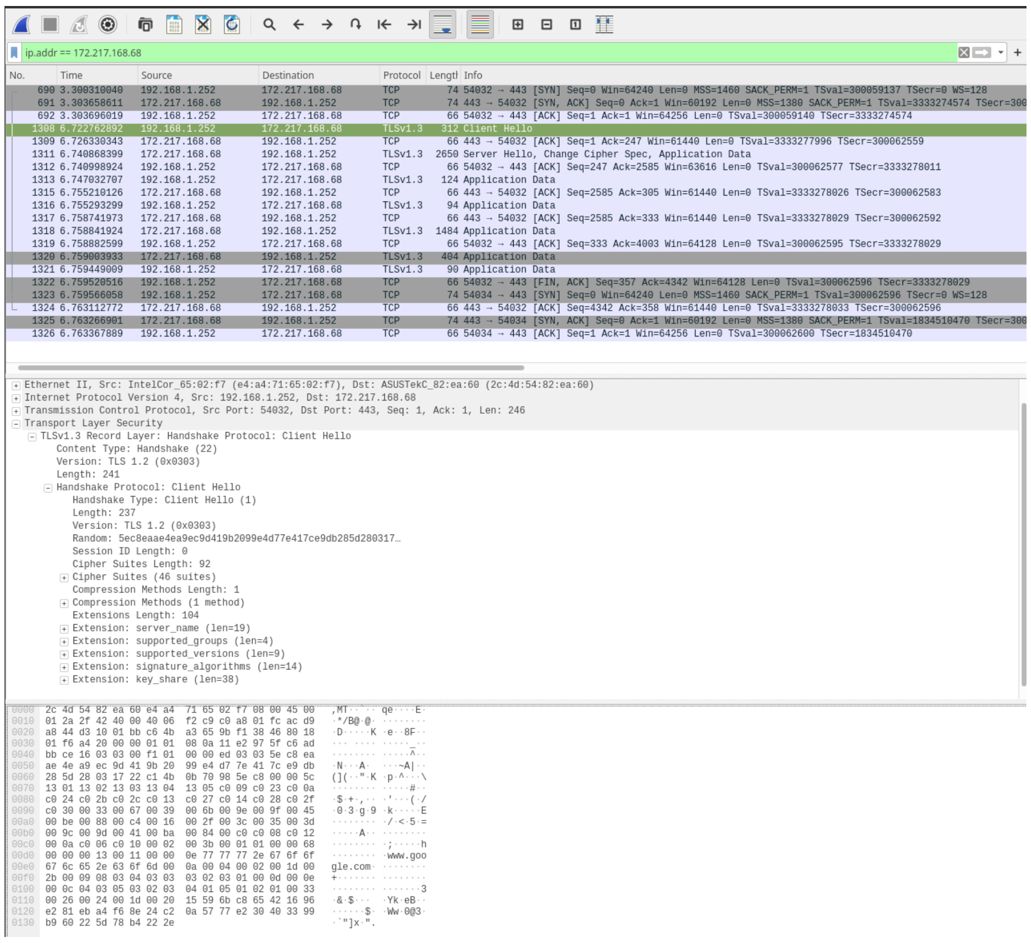
```
scripts/tls_test --connect www.test.local --port 8443 --cert
↪  /home/pascal/Documents/Bildung/fh/hsr/sem6/SA/sa-strongswan-doku/
↪  scripts/server-client/tls_ecdsa/server.crt

negotiated TLS 1.3 using suite TLS_AES_128_GCM_SHA256
received TLS server certificate 'C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local'
no issuer certificate found for "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
  issuer is "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd"
  using trusted certificate "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
ping
pong
^C
```

E.2. External Connection: TLS 1.3 Client to TLS 1.3 Server

A successful TLS 1.3 handshake with a Google server

```
$ ../../../strongswan_code/scripts/tls_test --connect www.google.com --port 443 --debug 2
↪  --cert 20200523_www-google-com.pem
46 supported TLS cipher suites:
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
TLS_AES_128_CCM_SHA256
TLS_AES_128_CCM_8_SHA256
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
```



```
No.   Time          Source          Destination     Protocol Length Info
  690 3.300310040   192.168.1.252   172.217.168.68  TCP        74 54032 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=300059137 TSecr=0 WS=128
  691 3.303658611   172.217.168.68  192.168.1.252   TCP        74 443 → 54032 [SYN, ACK] Seq=0 Ack=1 Win=60192 Len=0 MSS=1380 SACK_PERM=1 TSval=3333274574 TSecr=300
  692 3.303696019   192.168.1.252   172.217.168.68  TCP        66 54032 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=300059140 TSecr=3333274574
 1308 6.722762892   192.168.1.252   172.217.168.68  TLSv1.3   312 Client Hello
 1309 6.726330343   172.217.168.68  192.168.1.252   TCP        66 443 → 54032 [ACK] Seq=1 Ack=247 Win=61440 Len=0 TSval=3333277996 TSecr=300062559
 1311 6.740868399   172.217.168.68  192.168.1.252   TLSv1.3  2650 Server Hello, Change Cipher Spec, Application Data
 1312 6.740998924   192.168.1.252   172.217.168.68  TCP        66 54032 → 443 [ACK] Seq=247 Ack=2585 Win=63616 Len=0 TSval=300062577 TSecr=3333278011
 1313 6.747032707   192.168.1.252   172.217.168.68  TLSv1.3   124 Application Data
 1315 6.755210126   172.217.168.68  192.168.1.252   TCP        66 443 → 54032 [ACK] Seq=2585 Ack=305 Win=61440 Len=0 TSval=3333278026 TSecr=300062583
 1316 6.755293299   192.168.1.252   172.217.168.68  TLSv1.3    94 Application Data
 1317 6.758741973   172.217.168.68  192.168.1.252   TCP        66 443 → 54032 [ACK] Seq=2585 Ack=333 Win=61440 Len=0 TSval=3333278029 TSecr=300062592
 1318 6.758841924   172.217.168.68  192.168.1.252   TLSv1.3  1484 Application Data
 1319 6.758882599   192.168.1.252   172.217.168.68  TCP        66 54032 → 443 [ACK] Seq=333 Ack=4003 Win=64128 Len=0 TSval=300062595 TSecr=3333278029
 1320 6.759003933   172.217.168.68  192.168.1.252   TLSv1.3   404 Application Data
 1321 6.759449009   192.168.1.252   172.217.168.68  TLSv1.3    90 Application Data
 1322 6.759520516   192.168.1.252   172.217.168.68  TCP        66 54032 → 443 [FIN, ACK] Seq=357 Ack=4342 Win=64128 Len=0 TSval=300062596 TSecr=3333278029
 1323 6.759566058   192.168.1.252   172.217.168.68  TCP        74 54034 → 443 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 SACK_PERM=1 TSval=300062596 TSecr=0 WS=128
 1324 6.763112772   172.217.168.68  192.168.1.252   TCP        66 443 → 54032 [ACK] Seq=4342 Ack=358 Win=61440 Len=0 TSval=3333278033 TSecr=300062596
 1325 6.763266901   172.217.168.68  192.168.1.252   TCP        74 443 → 54034 [SYN, ACK] Seq=0 Ack=1 Win=60192 Len=0 MSS=1380 SACK_PERM=1 TSval=1834510470 TSecr=300
 1326 6.763367889   192.168.1.252   172.217.168.68  TCP        66 54034 → 443 [ACK] Seq=1 Ack=1 Win=64256 Len=0 TSval=300062600 TSecr=1834510470
```

```
+ Ethernet II, Src: IntelCor_65:02:f7 (e4:a4:71:65:02:f7), Dst: ASUSTekC_82:ea:60 (2c:4d:54:82:ea:60)
+ Internet Protocol Version 4, Src: 192.168.1.252, Dst: 172.217.168.68
+ Transmission Control Protocol, Src Port: 54032, Dst Port: 443, Seq: 1, Ack: 1, Len: 246
- Transport Layer Security
  - TLSv1.3 Record Layer: Handshake Protocol: Client Hello
      Content Type: Handshake (22)
      Version: TLS 1.2 (0x0303)
      Length: 241
    - Handshake Protocol: Client Hello
        Handshake Type: Client Hello (1)
        Length: 237
        Version: TLS 1.2 (0x0303)
        Random: 5ec8eaae4ea9ec9d419b2099e4d77e417ce9db285d280317…
        Session ID Length: 0
        Cipher Suites Length: 92
      + Cipher Suites (46 suites)
        Compression Methods Length: 1
      + Compression Methods (1 method)
        Extensions Length: 104
      + Extension: server_name (len=19)
      + Extension: supported_groups (len=4)
      + Extension: supported_versions (len=9)
      + Extension: signature_algorithms (len=14)
      + Extension: key_share (len=38)
```

```
0000  2c 4d 54 82 ea 60 e4 a4  71 65 02 f7 08 00 45 00   ,MT·····qe····E·
0010  01 2a 2f 42 40 00 40 06  f2 c9 c0 a8 01 fc ac d9   ·*/B@·@·········
0020  a8 44 d3 10 01 bb c6 4b  a3 65 9b f1 38 46 80 18   ·D·····K·e··8F··
0030  01 f6 a4 20 00 00 01 01  08 0a 11 e2 97 5f c6 ad   ·············_··
0040  bb ce 16 03 03 00 f1 01  00 00 ed 03 03 5e c8 ea   ·············^··
0050  ae 4e a9 ec 9d 41 9b 20  99 e4 d7 7e 41 7c e9 db   ·N···A·····~A|··
0060  28 5d 28 03 17 22 c1 4b  0b 70 98 5e c8 00 00 5c   (](··"·K·p·^···\
0070  13 01 13 02 13 03 13 04  13 05 c0 09 c0 23 c0 0a   ·············#··
0080  c0 24 c0 2b c0 2c c0 13  c0 27 c0 14 c0 28 c0 2f   ·$·+·,···'···(·/
0090  c0 30 00 33 00 67 00 39  00 6b 00 9e 00 9f 00 45   ·0·3·g·9·k·····E
00a0  00 be 00 88 00 c4 00 16  00 2f 00 3c 00 35 00 3d   ········/·<·5·=
00b0  00 9c 00 9d 00 41 00 ba  00 84 00 c0 c0 08 c0 12   ·····A··········
00c0  00 0a c0 06 c0 10 00 02  00 3b 00 01 01 00 00 68   ·········;·····h
00d0  00 00 00 13 00 11 00 00  0e 77 77 77 2e 67 6f 6f   ·········www.goo
00e0  67 6c 65 2e 63 6f 6d 00  0a 00 04 00 02 00 1d 00   gle.com·········
00f0  2b 00 09 08 03 04 03 03  03 02 03 01 00 0d 00 0e   +···············
0100  00 0c 04 03 05 03 02 03  04 01 05 01 02 01 00 33   ···············3
0110  00 26 00 24 00 1d 00 20  15 59 6b c8 65 42 16 96   ·&·$·····Yk·eB··
0120  e2 81 eb a4 f6 8e 24 c2  0a 57 77 e2 30 40 33 99   ······$··Ww·0@3·
0130  b9 60 22 5d 78 b4 22 2e                            ·`"]x·".
```

Figure E.2.: Screenshot of a handshake between a TLS 1.3 server and an external Google server as shown in Wireshark (2020-05-23).

```
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA384
TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_AES_128_CBC_SHA
TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
TLS_DHE_RSA_WITH_AES_256_CBC_SHA
TLS_DHE_RSA_WITH_AES_256_CBC_SHA256
TLS_DHE_RSA_WITH_AES_128_GCM_SHA256
TLS_DHE_RSA_WITH_AES_256_GCM_SHA384
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_128_CBC_SHA256
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_DHE_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_DHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA
TLS_RSA_WITH_AES_128_CBC_SHA256
TLS_RSA_WITH_AES_256_CBC_SHA
TLS_RSA_WITH_AES_256_CBC_SHA256
TLS_RSA_WITH_AES_128_GCM_SHA256
TLS_RSA_WITH_AES_256_GCM_SHA384
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA
TLS_RSA_WITH_CAMELLIA_128_CBC_SHA256
```

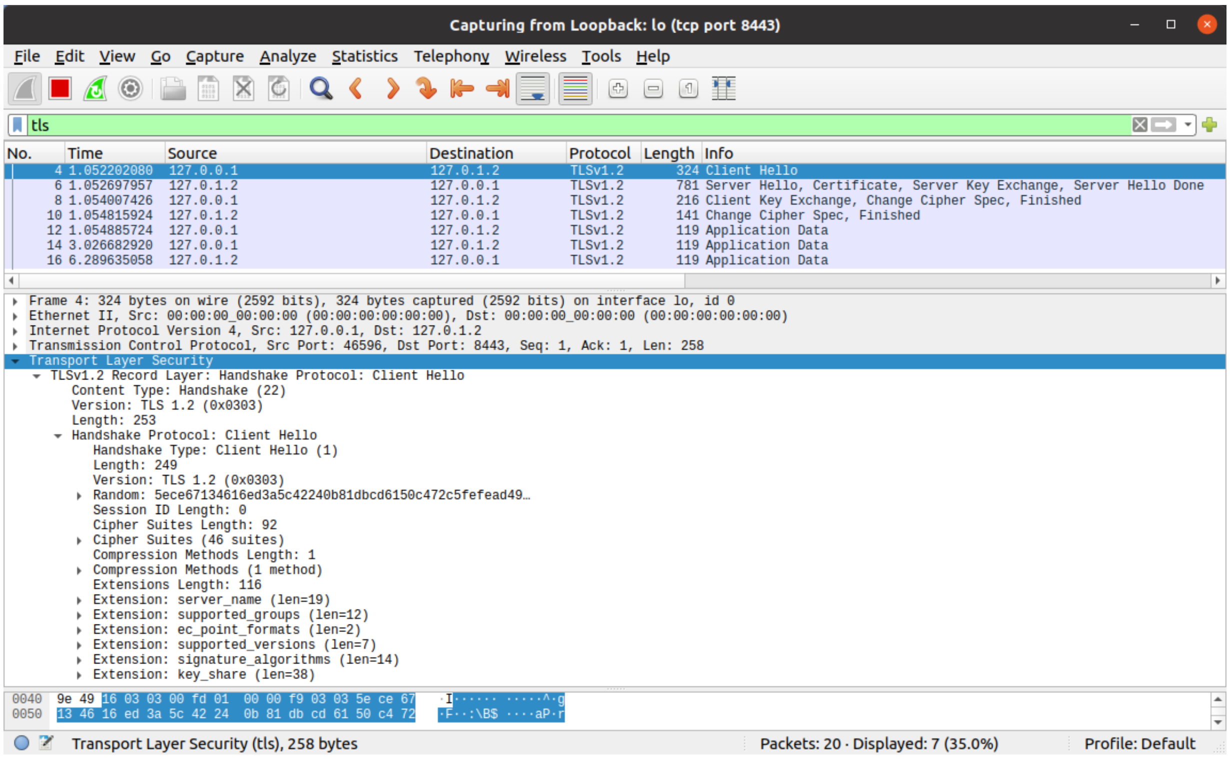
```
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA
TLS_RSA_WITH_CAMELLIA_256_CBC_SHA256
TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
TLS_RSA_WITH_3DES_EDE_CBC_SHA
TLS_ECDHE_ECDSA_WITH_NULL_SHA
TLS_ECDHE_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_SHA
TLS_RSA_WITH_NULL_SHA256
TLS_RSA_WITH_NULL_MD5
hello
sending extension: Server Name Indication for 'www.google.com'
sending extension: supported groups
sending extension: supported versions
sending extension: signature algorithms
sending extension: key-share
sending TLS ClientHello handshake (237 bytes)
sending TLS Handshake record (241 bytes)
processing TLS Handshake record (90 bytes)
received TLS ServerHello handshake (86 bytes)
negotiated TLS 1.3 using suite TLS_AES_128_GCM_SHA256
processing TLS ChangeCipherSpec record (1 bytes)
processing TLS ApplicationData record (2478 bytes)
received TLS EncryptedExtensions handshake (2 bytes)
received TLS Certificate handshake (2336 bytes)
received TLS server certificate 'C=US, ST=California, L=Mountain View, O=Google LLC,
↪  CN=www.google.com'
received TLS CertificateVerify handshake (75 bytes)
no issuer certificate found for "C=US, ST=California, L=Mountain View, O=Google LLC,
↪  CN=www.google.com"
issuer is "C=US, O=Google Trust Services, CN=GTS CA 1O1"
using trusted certificate "C=US, ST=California, L=Mountain View, O=Google LLC,
↪  CN=www.google.com"
verified signature with SHA256/ECDSA
received TLS Finished handshake (32 bytes)
sending TLS Finished handshake (32 bytes)
sending TLS ApplicationData record (53 bytes)
sending TLS ApplicationData record (23 bytes)
processing TLS ApplicationData record (1413 bytes)
processing TLS ApplicationData record (333 bytes)
HTTP/1.0 400 Bad Request
Content-Type: text/html; charset=UTF-8
Referrer-Policy: no-referrer
Content-Length: 1555
Date: Sat, 23 May 2020 09:19:42 GMT

<!DOCTYPE html>
<html lang=en>
... more html code follows
```

E.3. Local Connection: TLS 1.2 Client to TLS 1.3 Server

A successful TLS 1.2 handshake, server side

```
openssl s_server -accept www.test.local:8443 -key tls_ecdsa/server.key -cert
↪  tls_ecdsa/server.crt -debug -pass pass:7331 -keylogfile output/keylogs
Using default temp DH parameters
```


Capturing from Loopback: lo (tcp port 8443)
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
tls

```
No.  Time         Source      Destination  Protocol  Length  Info
   4 1.052202080  127.0.0.1   127.0.1.2    TLSv1.2      324  Client Hello
   6 1.052697957  127.0.1.2   127.0.0.1    TLSv1.2      781  Server Hello, Certificate, Server Key Exchange, Server Hello Done
   8 1.054007426  127.0.0.1   127.0.1.2    TLSv1.2      216  Client Key Exchange, Change Cipher Spec, Finished
  10 1.054815924  127.0.1.2   127.0.0.1    TLSv1.2      141  Change Cipher Spec, Finished
  12 1.054885724  127.0.0.1   127.0.1.2    TLSv1.2      119  Application Data
  14 3.026682920  127.0.0.1   127.0.1.2    TLSv1.2      119  Application Data
  16 6.289635058  127.0.1.2   127.0.0.1    TLSv1.2      119  Application Data
```

```
Frame 4: 324 bytes on wire (2592 bits), 324 bytes captured (2592 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.2
Transmission Control Protocol, Src Port: 46596, Dst Port: 8443, Seq: 1, Ack: 1, Len: 258
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 253
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 249
      Version: TLS 1.2 (0x0303)
      Random: 5ece67134616ed3a5c42240b81dbcd6150c472c5fefead49…
      Session ID Length: 0
      Cipher Suites Length: 92
      Cipher Suites (46 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 116
      Extension: server_name (len=19)
      Extension: supported_groups (len=12)
      Extension: ec_point_formats (len=2)
      Extension: supported_versions (len=7)
      Extension: signature_algorithms (len=14)
      Extension: key_share (len=38)
```

```
0040  9e 49 16 03 03 00 fd 01  00 00 f9 03 03 5e ce 67   ·I······ ·····^·g
0050  13 46 16 ed 3a 5c 42 24  0b 81 db cd 61 50 c4 72   ·F··:\B$ ····aP·r
```

```
Transport Layer Security (tls), 258 bytes     Packets: 20 · Displayed: 7 (35.0%)     Profile: Default
```

Figure E.3.: Screenshot of a handshake between a strongSwan TLS 1.2 client and an OpenSSL server supporting TLS 1.3 in Wireshark (2020-05-27).

```
ACCEPT
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 fd                                    .....
read from 0x55f62dbb5b40 [0x55f62dbc1758] (253 bytes => 253 (0xFD))
0000 - 01 00 00 f9 03 03 5e ce-67 13 46 16 ed 3a 5c 42   ......^.g.F..:\B
0010 - 24 0b 81 db cd 61 50 c4-72 c5 fe fe ad 49 19 a9   $....aP.r....I..
0020 - a1 b7 b3 3b 6b 6e 00 00-5c 13 01 13 02 13 03 13   ...;kn..\.......
0030 - 04 13 05 c0 09 c0 23 c0-0a c0 24 c0 2b c0 2c c0   ......#...$.+.,.
0040 - 13 c0 27 c0 14 c0 28 c0-2f c0 30 00 33 00 67 00   ..'...(./.0.3.g.
0050 - 39 00 6b 00 9e 00 9f 00-45 00 be 00 88 00 c4 00   9.k.....E.......
0060 - 16 00 2f 00 3c 00 35 00-3d 00 9c 00 9d 00 41 00   ../.<.5.=.....A.
0070 - ba 00 84 00 c0 c0 08 c0-12 00 0a c0 06 c0 10 00   ................
0080 - 02 00 3b 00 01 01 00 00-74 00 00 00 13 00 11 00   ..;.....t.......
0090 - 00 0e 77 77 77 2e 74 65-73 74 2e 6c 6f 63 61 6c   ..www.test.local
00a0 - 00 0a 00 0c 00 0a 00 17-00 18 00 19 00 15 00 13   ................
00b0 - 00 0b 00 02 01 00 00 2b-00 07 06 03 03 03 02 03   .......+........
00c0 - 01 00 0d 00 0e 00 0c 04-03 05 03 02 03 04 01 05   ................
00d0 - 01 02 01 00 33 00 26 00-24 00 1d 00 20 86 c5 47   ....3.&.$... ..G
00e0 - 11 1c f5 c4 0e 5b 73 2a-fd d3 b8 db 1f 4c 86 1d   .....[s*.....L..
00f0 - 71 36 54 7f 24 38 0f 96-de 48 b6 42 7d           q6T.$8...H.B}
write to 0x55f62dbb5b40 [0x55f62dbcaa20] (715 bytes => 715 (0x2CB))
0000 - 16 03 03 00 54 02 00 00-50 03 03 1b c6 b5 89 4d   ....T...P......M
0010 - dd b8 8d fc 48 fb f9 27-84 b3 c1 ae 30 52 45 da   ....H..'....0RE.
0020 - 5c f7 28 44 4f 57 4e 47-52 44 01 20 db be 2b 9f   \.(DOWNGRD. ..+.
0030 - 3e bf 6c ad 3a 87 fe c3-ee 23 bd 59 63 e4 18 e5   >.l.:....#.Yc...
0040 - 40 ba 96 0e cf 06 65 98-22 1b 5b 78 c0 09 00 00   @.....e.".[x....
0050 - 08 00 0b 00 04 03 00 01-02 16 03 03 01 cc 0b 00   ................
0060 - 01 c8 00 01 c5 00 01 c2-30 82 01 be 30 82 01 64   ........0...0..d
0070 - a0 03 02 01 02 02 14 43-e4 0e 1a c2 cb 53 e7 8a   .......C.....S..
0080 - df 05 d5 94 68 bd e3 43-c3 13 6e 30 0a 06 08 2a   ....h..C..n0...*
0090 - 86 48 ce 3d 04 03 02 30-45 31 0b 30 09 06 03 55   .H.=...0E1.0...U
00a0 - 04 06 13 02 41 55 31 13-30 11 06 03 55 04 08 0c   ....AU1.0...U...
00b0 - 0a 53 6f 6d 65 2d 53 74-61 74 65 31 21 30 1f 06   .Some-State1!0..
00c0 - 03 55 04 0a 0c 18 49 6e-74 65 72 6e 65 74 20 57   .U....Internet W
00d0 - 69 64 67 69 74 73 20 50-74 79 20 4c 74 64 30 1e   idgits Pty Ltd0.
00e0 - 17 0d 32 30 30 34 32 39-31 32 33 37 35 37 5a 17   ..200429123757Z.
00f0 - 0d 32 30 30 35 32 39 31-32 33 37 35 37 5a 30 5a   .200529123757Z0Z
0100 - 31 0b 30 09 06 03 55 04-06 13 02 43 4e 31 0b 30   1.0...U....CN1.0
0110 - 09 06 03 55 04 08 0c 02-47 44 31 0b 30 09 06 03   ...U....GD1.0...
0120 - 55 04 07 0c 02 53 5a 31-18 30 16 06 03 55 04 0a   U....SZ1.0...U..
0130 - 0c 0f 4d 61 67 72 61 74-68 65 61 2c 20 49 6e 63   ..Magrathea, Inc
0140 - 2e 31 17 30 15 06 03 55-04 03 0c 0e 77 77 77 2e   .1.0...U....www.
0150 - 74 65 73 74 2e 6c 6f 63-61 6c 30 59 30 13 06 07   test.local0Y0...
0160 - 2a 86 48 ce 3d 02 01 06-08 2a 86 48 ce 3d 03 01   *.H.=....*.H.=..
0170 - 07 03 42 00 04 83 94 4a-8c 3c 2c a6 2e eb b1 34   ..B....J.<,....4
0180 - 6e 56 37 43 47 20 5c e5-35 21 b2 9a 69 e5 42 0a   nV7CG \.5!..i.B.
0190 - 4a 1a 73 ed cc b8 3a 61-a3 4a a9 ec 04 c5 3c 0d   J.s...:a.J....<.
01a0 - 03 3d 62 c7 2b f0 c0 68-3b 9a 2c 90 da d0 7f a9   .=b.+..h;.,.....
01b0 - 7e c4 8a 99 25 a3 1d 30-1b 30 19 06 03 55 1d 11   ~...%..0.0...U..
01c0 - 04 12 30 10 82 0e 77 77-77 2e 74 65 73 74 2e 6c   ..0...www.test.l
01d0 - 6f 63 61 6c 30 0a 06 08-2a 86 48 ce 3d 04 03 02   ocal0...*.H.=...
01e0 - 03 48 00 30 45 02 21 00-dd 3f 21 03 7c a7 9e d8   .H.0E.!..?!.|...
01f0 - 4b 07 65 a5 d8 4a c4 cc-d3 d5 3b a2 07 6d 98 d1   K.e..J....;..m..
0200 - bd 97 c3 9f 56 94 94 e2-02 20 34 07 8c 9e 14 1c   ....V.... 4.....
0210 - 3e ec a9 0b 38 51 3c 23-3f b4 08 03 1b b9 e3 66   >...8Q<#?......f
0220 - f3 0e 9d f2 d8 39 69 a7-a9 23 16 03 03 00 93 0c   .....9i..#......
0230 - 00 00 8f 03 00 17 41 04-12 bf 46 7d bd bb cc 4e   ......A...F}...N
0240 - b2 97 84 6f f2 80 58 b9-aa 9c 8d 2b 71 48 6a d1   ...o..X....+qHj.
0250 - 2e 59 0b 5b 6d 51 12 4b-89 fe e5 37 d9 bb a8 b6   .Y.[mQ.K...7....
0260 - ad 75 44 4b 52 42 ab d7-a6 e5 3b 53 7f 41 2f 62   .uDKRB....;S.A/b
0270 - b8 ac b8 32 f4 a0 80 dd-04 03 00 46 30 44 02 20   ...2.......F0D.
0280 - 7f ca c7 81 a6 84 a7 7f-d4 36 7d 75 03 2e 1c e4   .........6}u....
0290 - 21 44 36 2d 25 2b c0 2c-2c 23 ae 71 3f 26 33 10   !D6-%+.,,#.q?&3.
02a0 - 02 20 14 2e 6e 75 f4 11-45 66 3c 13 57 d7 57 77   . ..nu..Ef<.W.Ww
02b0 - fc a8 98 0c 4e f0 66 29-c0 81 4f df e5 49 1d 32   ....N.f)..O..I.2
02c0 - fe 57 16 03 03 00 04 0e-00 00 00                  .W.........
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 46                                    ....F
read from 0x55f62dbb5b40 [0x55f62dbc1758] (70 bytes => 70 (0x46))
0000 - 10 00 00 42 41 04 01 a0-b5 e8 94 1f aa 8d 92 9b   ...BA...........
0010 - 13 12 27 cc ea 1d 3c f6-d8 41 f9 da f9 70 a1 e8   ..'...<..A...p..
0020 - 92 ec 44 3a 56 9d 0e 72-2c 5b bc 3c f6 50 c0 0b   ..D:V..r,[.<.P..
0030 - 2d 24 b9 3f 1e 40 90 20-3f 8c 65 1d 1e 07 8f 9a   -$.?.@. ?.e.....
0040 - 33 4f 69 88 d2 94                                 3Oi...
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 14 03 03 00 01                                    .....
read from 0x55f62dbb5b40 [0x55f62dbc1758] (1 bytes => 1 (0x1))
0000 - 01                                                .
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 40                                    ....@
read from 0x55f62dbb5b40 [0x55f62dbc1758] (64 bytes => 64 (0x40))
0000 - 1f bb 4b 78 16 1d 46 0b-52 19 72 0d a3 6c 0d cc   ..Kx..F.R.r..l..
0010 - 9a 94 8a 1e 03 a0 33 a5-e6 00 8b 15 f2 a4 82 ad   ......3.........
0020 - e6 39 1d 75 29 6d d9 c7-b6 20 3c b0 89 3f 0e 79   .9.u)m... <..?.y
0030 - f6 a4 72 1d 04 05 a8 e2-47 1a 73 ee 61 f4 c7 db   ..r.....G.s.a...
write to 0x55f62dbb5b40 [0x55f62dbcaa20] (75 bytes => 75 (0x4B))
0000 - 14 03 03 00 01 01 16 03-03 00 40 4a e4 af d3 75   .........@J...u
0010 - 1a d4 74 cf af 76 68 cb-71 3f f9 20 8a e0 3f 9a   ..t..vh.q?. ..?.
```

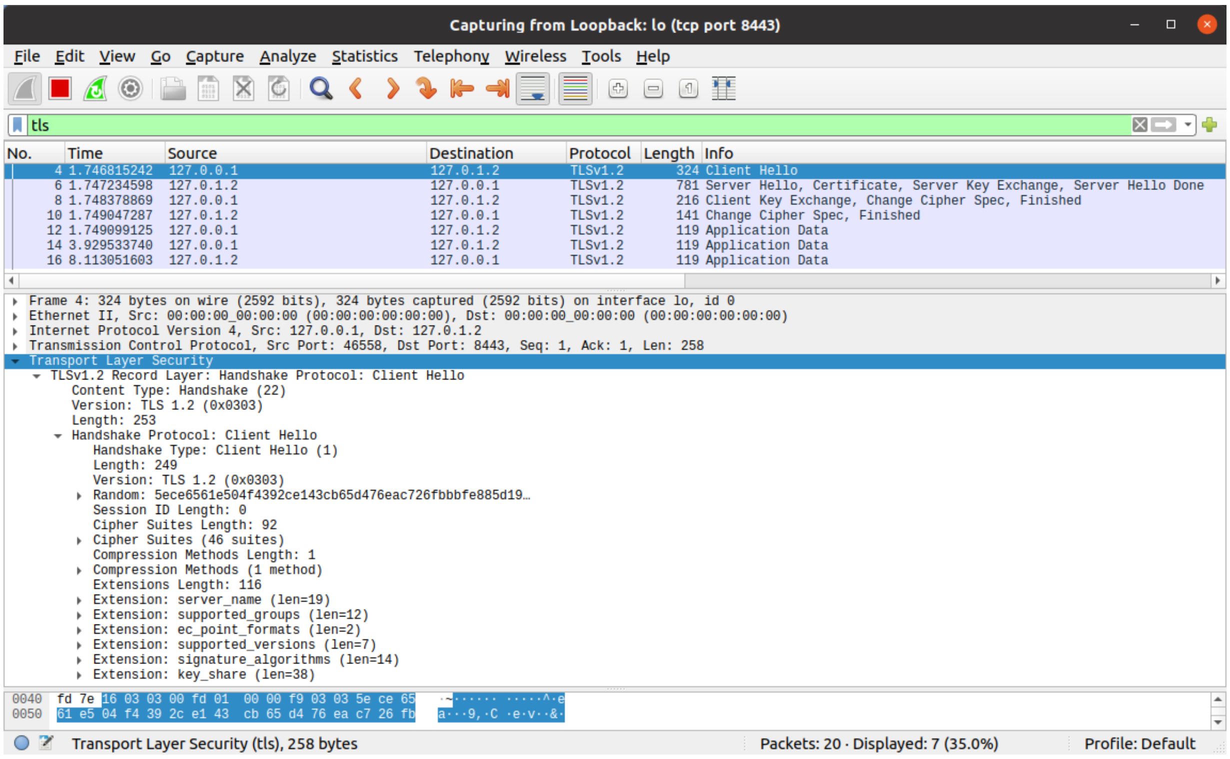
```
0020 - a8 cf fa aa f9 00 95 c7-5e 07 5f 79 5f 04 c3 26   ........^._y_..&
0030 - 05 cf c1 99 20 d2 84 18-03 eb ed 15 ee 76 76 48   .... ........vvH
0040 - de 7a 8f e4 84 03 85 a9-4c 2d 4a                  .z......L-J
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMDBALACQQg274rnz6/bK06h/7D7iO9WWPkGOVAupYOzwZlmCIbW3gE
MGe0uOl+0dTQ6fh0+rG/Bt0JJLopsIUYP6iq/Pr33tCN0DWPvQkhGW1gLP0fVQ5e
K6EGAgRezmcTogQCAhwgpAYEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
TLS_CHACHA20_POLY1305_SHA256:ECDHE-ECDSA-AES128-SHA:
ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:
ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:
ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA:
ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES256-SHA384:
ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-SHA:
DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA256:
DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:AES128-SHA:
AES128-SHA256:AES256-SHA:AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:
RSA+SHA256:RSA+SHA384:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:RSA+SHA256:RSA+SHA384
Supported Elliptic Curve Point Formats: uncompressed
Supported Elliptic Groups: P-256:P-384:P-521:P-224:P-192
Shared Elliptic groups: P-256:P-384:P-521
CIPHER is ECDHE-ECDSA-AES128-SHA
Secure Renegotiation IS NOT supported
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                    ....0
read from 0x55f62dbb5b40 [0x55f62dbc1758] (48 bytes => 48 (0x30))
0000 - ba b4 b2 f2 4f b8 5b 1d-e4 90 57 bf 40 12 51 1e   ....O.[...W.@.Q.
0010 - 6b 26 cf 69 a3 ac e1 5e-70 27 9d 67 d1 c0 f7 8d   k&.i...^p'.g....
0020 - 24 18 60 b3 47 0c 7b 71-98 ee 8a 79 46 f9 4b a6   $.`.G.{q...yF.K.

read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                    ....0
read from 0x55f62dbb5b40 [0x55f62dbc1758] (48 bytes => 48 (0x30))
0000 - 38 8a 58 54 3b b3 b3 c2-cd 5f bf 74 25 9e f3 00   8.XT;...._.t%...
0010 - 64 e6 d5 77 f9 f6 97 5d-24 fc a6 cf a9 fe 44 4d   d..w...]$.....DM
0020 - 64 cc 38 a3 8b 93 ee 4e-d6 8c 24 64 01 c4 22 e6   d.8....N..$d..".
ping
pong
write to 0x55f62dbb5b40 [0x55f62dbc58a3] (53 bytes => 53 (0x35))
0000 - 17 03 03 00 30 2c ef 8a-f3 a4 59 b9 1b 67 ce 6e   ....0,....Y..g.n
0010 - 67 c5 93 13 99 f6 27 e8-9e d6 24 ec 9f 9c 5e 4c   g.....'...$...^L
0020 - 8a 64 9c 54 33 19 16 dc-17 e7 d6 55 29 fe f5 7f   .d.T3......U)...
0030 - ce 03 f0 d0 d8                                    .....
read from 0x55f62dbb5b40 [0x55f62dbc1753] (5 bytes => 0 (0x0))
ERROR
shutting down SSL
CONNECTION CLOSED
```

A successful TLS 1.2 handshake, client side

```
scripts/tls_test --connect www.test.local --port 8443 --cert
↪ /home/pascal/Documents/Bildung/fh/hsr/sem6/SA/sa-strongswan-doku/scripts/
↪ server-client/tls_ecdsa/server.crt

negotiated TLS 1.2 using suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
received TLS server certificate 'C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local'
no issuer certificate found for "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
```



```
Capturing from Loopback: lo (tcp port 8443)

tls

No.  Time         Source      Destination  Protocol  Length Info
   4 1.746815242  127.0.0.1   127.0.1.2    TLSv1.2   324 Client Hello
   6 1.747234598  127.0.1.2   127.0.0.1    TLSv1.2   781 Server Hello, Certificate, Server Key Exchange, Server Hello Done
   8 1.748378869  127.0.0.1   127.0.1.2    TLSv1.2   216 Client Key Exchange, Change Cipher Spec, Finished
  10 1.749047287  127.0.1.2   127.0.0.1    TLSv1.2   141 Change Cipher Spec, Finished
  12 1.749099125  127.0.0.1   127.0.1.2    TLSv1.2   119 Application Data
  14 3.929533740  127.0.0.1   127.0.1.2    TLSv1.2   119 Application Data
  16 8.113051603  127.0.1.2   127.0.0.1    TLSv1.2   119 Application Data

Frame 4: 324 bytes on wire (2592 bits), 324 bytes captured (2592 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.2
Transmission Control Protocol, Src Port: 46558, Dst Port: 8443, Seq: 1, Ack: 1, Len: 258
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 253
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 249
      Version: TLS 1.2 (0x0303)
      Random: 5ece6561e504f4392ce143cb65d476eac726fbbbfe885d19…
      Session ID Length: 0
      Cipher Suites Length: 92
      Cipher Suites (46 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 116
      Extension: server_name (len=19)
      Extension: supported_groups (len=12)
      Extension: ec_point_formats (len=2)
      Extension: supported_versions (len=7)
      Extension: signature_algorithms (len=14)
      Extension: key_share (len=38)

0040  fd 7e 16 03 03 00 fd 01  00 00 f9 03 03 5e ce 65   ·~······ ·····^·e
0050  61 e5 04 f4 39 2c e1 43  cb 65 d4 76 ea c7 26 fb   a···9,·C ·e·v··&·

Transport Layer Security (tls), 258 bytes     Packets: 20 · Displayed: 7 (35.0%)     Profile: Default
```

Figure E.4.: Screenshot of a handshake between a strongSwan TLS 1.2 client and an OpenSSL server supporting only TLS 1.2 in Wireshark (2020-05-27).

```
  issuer is "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd"
  using trusted certificate "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
ping
pong
^C
```

E.4. Local Connection: TLS 1.2 Client to TLS 1.2 Server

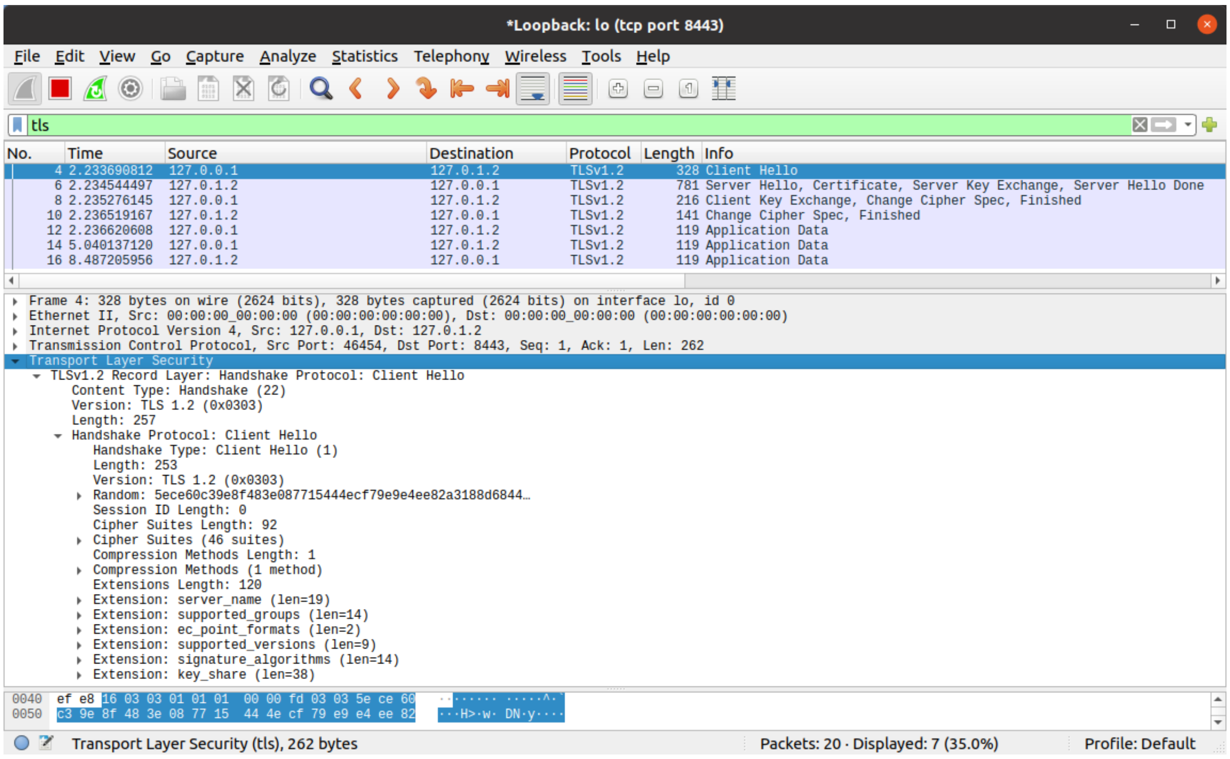
A successful TLS 1.2 handshake, server side

```
openssl s_server -accept www.test.local:8443 -key tls_ecdsa/server.key -cert
↪  tls_ecdsa/server.crt -debug -pass pass:7331 -keylogfile output/keylogs -tls1_2
Using default temp DH parameters
ACCEPT
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 fd                                    .....
read from 0x55a44e664b40 [0x55a44e670758] (253 bytes => 253 (0xFD))
0000 - 01 00 00 f9 03 03 5e ce-65 61 e5 04 f4 39 2c e1   ......^.ea...9,.
0010 - 43 cb 65 d4 76 ea c7 26-fb bb fe 88 5d 19 47 35   C.e.v..&....].G5
0020 - e1 ef 5b 04 75 e6 00 00-5c 13 01 13 02 13 03 13   ..[.u...\.......
0030 - 04 13 05 c0 09 c0 23 c0-0a c0 24 c0 2b c0 2c c0   ......#...$.+.,.
0040 - 13 c0 27 c0 14 c0 28 c0-2f c0 30 00 33 00 67 00   ..'...(./.0.3.g.
0050 - 39 00 6b 00 9e 00 9f 00-45 00 be 00 88 00 c4 00   9.k.....E.......
0060 - 16 00 2f 00 3c 00 35 00-3d 00 9c 00 9d 00 41 00   ../.<.5.=.....A.
0070 - ba 00 84 00 c0 c0 08 c0-12 00 0a c0 06 c0 10 00   ................
0080 - 02 00 3b 00 01 01 00 00-74 00 00 00 13 00 11 00   ..;.....t.......
0090 - 00 0e 77 77 77 2e 74 65-73 74 2e 6c 6f 63 61 6c   ..www.test.local
```

```
00a0 - 00 0a 00 0c 00 0a 00 17-00 18 00 19 00 15 00 13   ................
00b0 - 00 0b 00 02 01 00 00 2b-00 07 06 03 03 03 02 03   .......+........
00c0 - 01 00 0d 00 0e 00 0c 04-03 05 03 02 03 04 01 05   ................
00d0 - 01 02 01 00 33 00 26 00-24 00 1d 00 20 cd 2a 38   ....3.&.$... .*8
00e0 - 56 47 01 67 c3 aa ff 98-17 b6 8b c5 a9 cf 70 92   VG.g..........p.
00f0 - 79 63 b7 3d 5f d7 27 92-39 e5 fa 52 2f            yc.=_.'.9..R/
write to 0x55a44e664b40 [0x55a44e679a20] (715 bytes => 715 (0x2CB))
0000 - 16 03 03 00 54 02 00 00-50 03 03 25 99 9b 24 95   ....T...P..%..$.
0010 - 2d 50 d1 3f cc ec d8 b7-54 c5 f0 6e 1a e3 bb 27   -P.?....T..n...'
0020 - 4a 89 40 35 9d 8e 9b 2b-66 7c e1 20 51 a4 3f c3   J.@5...+f|. Q.?.
0030 - f9 8f c1 ad 78 c9 d1 ed-6c 80 31 22 8d 68 07 1a   ....x...l.1".h..
0040 - ea 0e bb 73 56 d4 85 67-95 e5 3d 3c c0 09 00 00   ...sV..g..=<....
0050 - 08 00 0b 00 04 03 00 01-02 16 03 03 01 cc 0b 00   ................
0060 - 01 c8 00 01 c5 00 01 c2-30 82 01 be 30 82 01 64   ........0...0..d
0070 - a0 03 02 01 02 02 14 43-e4 0e 1a c2 cb 53 e7 8a   .......C.....S..
0080 - df 05 d5 94 68 bd e3 43-c3 13 6e 30 0a 06 08 2a   ....h..C..n0...*
0090 - 86 48 ce 3d 04 03 02 30-45 31 0b 30 09 06 03 55   .H.=...0E1.0...U
00a0 - 04 06 13 02 41 55 31 13-30 11 06 03 55 04 08 0c   ....AU1.0...U...
00b0 - 0a 53 6f 6d 65 2d 53 74-61 74 65 31 21 30 1f 06   .Some-State1!0..
00c0 - 03 55 04 0a 0c 18 49 6e-74 65 72 6e 65 74 20 57   .U....Internet W
00d0 - 69 64 67 69 74 73 20 50-74 79 20 4c 74 64 30 1e   idgits Pty Ltd0.
00e0 - 17 0d 32 30 30 34 32 39-31 32 33 37 35 37 5a 17   ..200429123757Z.
00f0 - 0d 32 30 30 35 32 39 31-32 33 37 35 37 5a 30 5a   .200529123757Z0Z
0100 - 31 0b 30 09 06 03 55 04-06 13 02 43 4e 31 0b 30   1.0...U....CN1.0
0110 - 09 06 03 55 04 08 0c 02-47 44 31 0b 30 09 06 03   ...U....GD1.0...
0120 - 55 04 07 0c 02 53 5a 31-18 30 16 06 03 55 04 0a   U....SZ1.0...U..
0130 - 0c 0f 4d 61 67 72 61 74-68 65 61 2c 20 49 6e 63   ..Magrathea, Inc
0140 - 2e 31 17 30 15 06 03 55-04 03 0c 0e 77 77 77 2e   .1.0...U....www.
0150 - 74 65 73 74 2e 6c 6f 63-61 6c 30 59 30 13 06 07   test.local0Y0...
0160 - 2a 86 48 ce 3d 02 01 06-08 2a 86 48 ce 3d 03 01   *.H.=....*.H.=..
0170 - 07 03 42 00 04 83 94 4a-8c 3c 2c a6 2e eb b1 34   ..B....J.<,....4
0180 - 6e 56 37 43 47 20 5c e5-35 21 b2 9a 69 e5 42 0a   nV7CG \.5!..i.B.
0190 - 4a 1a 73 ed cc b8 3a 61-a3 4a a9 ec 04 c5 3c 0d   J.s...:a.J....<.
01a0 - 03 3d 62 c7 2b f0 c0 68-3b 9a 2c 90 da d0 7f a9   .=b.+..h;.,.....
01b0 - 7e c4 8a 99 25 a3 1d 30-1b 30 19 06 03 55 1d 11   ~...%..0.0...U..
01c0 - 04 12 30 10 82 0e 77 77-77 2e 74 65 73 74 2e 6c   ..0...www.test.l
01d0 - 6f 63 61 6c 30 0a 06 08-2a 86 48 ce 3d 04 03 02   ocal0...*.H.=...
01e0 - 03 48 00 30 45 02 21 00-dd 3f 21 03 7c a7 9e d8   .H.0E.!..?!.|...
01f0 - 4b 07 65 a5 d8 4a c4 cc-d3 d5 3b a2 07 6d 98 d1   K.e..J....;..m..
0200 - bd 97 c3 9f 56 94 94 e2-02 20 34 07 8c 9e 14 1c   ....V.... 4.....
0210 - 3e ec a9 0b 38 51 3c 23-3f b4 08 03 1b b9 e3 66   >...8Q<#?......f
0220 - f3 0e 9d f2 d8 39 69 a7-a9 23 16 03 03 00 93 0c   .....9i..#......
0230 - 00 00 8f 03 00 17 41 04-69 61 0b 02 df ce 4a b9   ......A.ia....J.
0240 - 9e 67 ed 8d fe ff ff 55-db f2 81 6b 9b b1 ce 21   .g.....U...k...!
0250 - f2 38 5f e6 34 45 f0 ca-d2 55 89 bc 25 d7 e6 7e   .8_.4E...U..%..~
0260 - b7 c5 09 69 da 86 94 12-12 94 a9 22 49 4b a5 87   ...i......."IK..
0270 - 58 41 9a 19 87 de a1 4c-04 03 00 46 30 44 02 20   XA.....L...F0D.
0280 - 69 d7 7b bd 7d ca 85 82-53 71 ae ea 85 c1 f4 97   i.{.}...Sq......
0290 - 19 6b e6 20 7a 3f cd c4-e2 8d ce 48 17 3b c7 a6   .k. z?.....H.;..
02a0 - 02 20 5e 99 36 a1 a4 ff-0d f3 11 13 f5 31 9a f1   . ^.6........1..
02b0 - de 2d 9a f7 8f a7 7c 04-e1 96 e7 80 d6 d2 d2 84   .-....|.........
02c0 - cb ba 16 03 03 00 04 0e-00 00 00                  ...........
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 46                                    ....F
read from 0x55a44e664b40 [0x55a44e670758] (70 bytes => 70 (0x46))
0000 - 10 00 00 42 41 04 b7 e8-83 df b2 58 bb ef 6a b7   ...BA......X..j.
0010 - 13 e0 9b 65 3e 37 1a f1-75 18 41 6b 60 f1 88 18   ...e>7..u.Ak`...
0020 - b1 5a 03 fd cd bd b5 a7-0b 52 ef 6a 09 7f 72 24   .Z.......R.j..r$
0030 - fc b5 77 59 57 fc 0a a8-bd 27 fd b9 17 ef d2 19   ..wYW....'......
```

```
0040 - 63 63 84 66 03 65                                  cc.f.e
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 14 03 03 00 01                                     .....
read from 0x55a44e664b40 [0x55a44e670758] (1 bytes => 1 (0x1))
0000 - 01                                                 .
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 40                                     ....@
read from 0x55a44e664b40 [0x55a44e670758] (64 bytes => 64 (0x40))
0000 - 17 9f 7d 4d 77 a6 93 86-56 87 a0 5c 21 0b fe a8   ..}Mw...V..\!...
0010 - 3c 7a 9e 7b 02 8e 64 4a-01 3a c1 c2 02 ba 5e 85   <z.{..dJ.:....^.
0020 - f7 27 58 70 ec e2 3c 94-98 cd 4c 62 95 9a 2f 5d   .'Xp..<...Lb../]
0030 - 01 7c d6 4d 5c e1 0b b3-dd bc 74 c9 e7 a1 f3 bf   .|.M\.....t.....
write to 0x55a44e664b40 [0x55a44e679a20] (75 bytes => 75 (0x4B))
0000 - 14 03 03 00 01 01 16 03-03 00 40 ed b1 87 09 4a   ..........@....J
0010 - 91 7e 8d f9 73 3a 8f 04-fd 47 6e 88 3b f4 a1 75   .~..s:...Gn.;..u
0020 - d7 86 1f 81 b8 b7 ca ca-28 3f 86 74 eb df d2 3a   ........(?.t...:
0030 - 3c 08 f9 df fd 80 98 c5-fa 5e d7 14 99 a7 25 b3   <........^....%.
0040 - d0 ee ab 64 44 3d 6f 8f-a4 4e 47                  ...dD=o..NG
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMDBALACQQgUaQ/w/mPwa14ydHtbIAxIo1oBxrqDrtzVtSFZ5XlPTwE
MIAj7G/inefJ3nb4N5AnWIjna95GdSx3esSg7Aqu3sF+TtGzg4odI/SWp5EMRycW
KKEGAgRezmVhogQCAhwgpAYEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
TLS_CHACHA20_POLY1305_SHA256:ECDHE-ECDSA-AES128-SHA:
ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:ECDHE-ECDSA-AES256-SHA384:
ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:
ECDHE-RSA-AES128-SHA:ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:
ECDHE-RSA-AES256-SHA384:ECDHE-RSA-AES128-GCM-SHA256:
ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-SHA:DHE-RSA-AES128-SHA256:
DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA256:DHE-RSA-AES128-GCM-SHA256:
DHE-RSA-AES256-GCM-SHA384:AES128-SHA:AES128-SHA256:AES256-SHA:
AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:
RSA+SHA256:RSA+SHA384:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:RSA+SHA256:RSA+SHA384
Supported Elliptic Curve Point Formats: uncompressed
Supported Elliptic Groups: P-256:P-384:P-521:P-224:P-192
Shared Elliptic groups: P-256:P-384:P-521
CIPHER is ECDHE-ECDSA-AES128-SHA
Secure Renegotiation IS NOT supported
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                     ....0
read from 0x55a44e664b40 [0x55a44e670758] (48 bytes => 48 (0x30))
0000 - c6 91 87 83 2a d0 d5 31-30 1e 4b a3 67 ac d9 85   ....*..10.K.g...
0010 - 60 68 29 18 21 4a aa b5-a0 e9 23 65 d2 37 4e df   `h).!J....#e.7N.
0020 - 08 51 1f 49 84 46 5b 6d-f1 52 e9 04 91 dc ed 69   .Q.I.F[m.R.....i

read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                     ....0
read from 0x55a44e664b40 [0x55a44e670758] (48 bytes => 48 (0x30))
0000 - 2b ed bd 74 00 71 07 ad-38 76 44 9e e8 50 0f 04   +..t.q..8vD..P..
0010 - 27 5a e3 e6 61 20 cc b7-06 52 40 55 22 1d 4a c9   'Z..a ...R@U".J.
0020 - 90 8d 8a 9e 28 33 23 f8-3d f6 01 8d 66 02 4a 86   ....(3#.=...f.J.
ping
pong
write to 0x55a44e664b40 [0x55a44e6748a3] (53 bytes => 53 (0x35))
0000 - 17 03 03 00 30 f6 01 98-0c 0d 64 85 8d 03 69 d6   ....0.....d...i.
0010 - fc e6 cd 45 fd ba 89 74-c8 7f 81 29 66 38 cb 49   ...E...t...)f8.I
```



```
*Loopback: lo (tcp port 8443)

tls

No.  Time         Source      Destination  Protocol Length Info
   4 2.233690812  127.0.0.1   127.0.1.2    TLSv1.2     328 Client Hello
   6 2.234544497  127.0.1.2   127.0.0.1    TLSv1.2     781 Server Hello, Certificate, Server Key Exchange, Server Hello Done
   8 2.235276145  127.0.0.1   127.0.1.2    TLSv1.2     216 Client Key Exchange, Change Cipher Spec, Finished
  10 2.236519167  127.0.1.2   127.0.0.1    TLSv1.2     141 Change Cipher Spec, Finished
  12 2.236620608  127.0.0.1   127.0.1.2    TLSv1.2     119 Application Data
  14 5.040137120  127.0.0.1   127.0.1.2    TLSv1.2     119 Application Data
  16 8.487205956  127.0.1.2   127.0.0.1    TLSv1.2     119 Application Data

Frame 4: 328 bytes on wire (2624 bits), 328 bytes captured (2624 bits) on interface lo, id 0
Ethernet II, Src: 00:00:00_00:00:00 (00:00:00:00:00:00), Dst: 00:00:00_00:00:00 (00:00:00:00:00:00)
Internet Protocol Version 4, Src: 127.0.0.1, Dst: 127.0.1.2
Transmission Control Protocol, Src Port: 46454, Dst Port: 8443, Seq: 1, Ack: 1, Len: 262
Transport Layer Security
  TLSv1.2 Record Layer: Handshake Protocol: Client Hello
    Content Type: Handshake (22)
    Version: TLS 1.2 (0x0303)
    Length: 257
    Handshake Protocol: Client Hello
      Handshake Type: Client Hello (1)
      Length: 253
      Version: TLS 1.2 (0x0303)
      Random: 5ece60c39e8f483e087715444ecf79e9e4ee82a3188d6844…
      Session ID Length: 0
      Cipher Suites Length: 92
      Cipher Suites (46 suites)
      Compression Methods Length: 1
      Compression Methods (1 method)
      Extensions Length: 120
      Extension: server_name (len=19)
      Extension: supported_groups (len=14)
      Extension: ec_point_formats (len=2)
      Extension: supported_versions (len=9)
      Extension: signature_algorithms (len=14)
      Extension: key_share (len=38)

0040  ef e8 16 03 03 01 01 01  00 00 fd 03 03 5e ce 60   ········ ·····^·`
0050  c3 9e 8f 48 3e 08 77 15  44 4e cf 79 e9 e4 ee 82   ···H>·w· DN·y····

Transport Layer Security (tls), 262 bytes     Packets: 20 · Displayed: 7 (35.0%)     Profile: Default
```

Figure E.5.: Screenshot of a handshake between a strongSwan TLS 1.3 client and an OpenSSL server supporting only TLS 1.2 in Wireshark (2020-05-27).

```
0020 - 60 73 ca 5a 70 68 13 9c-78 61 b3 a2 13 04 a7 b9   `s.Zph..xa......
0030 - 4f cb 4b ac 5d                                    O.K.]
read from 0x55a44e664b40 [0x55a44e670753] (5 bytes => 0 (0x0))
ERROR
shutting down SSL
CONNECTION CLOSED
```

A successful TLS 1.2 handshake, client side

```
scripts/tls_test --connect www.test.local --port 8443 --cert
↪  /home/pascal/Documents/Bildung/fh/hsr/sem6/SA/sa-strongswan-doku/scripts/
↪  server-client/tls_ecdsa/server.crt

negotiated TLS 1.2 using suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
received TLS server certificate 'C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local'
no issuer certificate found for "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
  issuer is "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd"
  using trusted certificate "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
ping
pong
^C
```

E.5. Local Connection: TLS 1.3 Client to TLS 1.2 Server

A successful TLS 1.2 handshake, server side

```
openssl s_server -accept www.test.local:8443 -key tls_ecdsa/server.key -cert
↪  tls_ecdsa/server.crt -debug -pass pass:7331 -keylogfile output/keylogs -tls1_2
Using default temp DH parameters
ACCEPT
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 16 03 03 01 01                                     .....
read from 0x564024a33b40 [0x564024a3f758] (257 bytes => 257 (0x101))
0000 - 01 00 00 fd 03 03 5e ce-60 c3 9e 8f 48 3e 08 77   ......^.`...H>.w
0010 - 15 44 4e cf 79 e9 e4 ee-82 a3 18 8d 68 44 ca 5b   .DN.y.......hD.[
0020 - 7a 2f 6c fb 58 21 00 00-5c 13 01 13 02 13 03 13   z/l.X!..\.......
0030 - 04 13 05 c0 09 c0 23 c0-0a c0 24 c0 2b c0 2c c0   ......#...$.+.,.
0040 - 13 c0 27 c0 14 c0 28 c0-2f c0 30 00 33 00 67 00   ..'...(./.0.3.g.
0050 - 39 00 6b 00 9e 00 9f 00-45 00 be 00 88 00 c4 00   9.k.....E.......
0060 - 16 00 2f 00 3c 00 35 00-3d 00 9c 00 9d 00 41 00   ../.<.5.=.....A.
0070 - ba 00 84 00 c0 c0 08 c0-12 00 0a c0 06 c0 10 00   ................
0080 - 02 00 3b 00 01 01 00 00-78 00 00 00 13 00 11 00   ..;.....x.......
0090 - 00 0e 77 77 77 2e 74 65-73 74 2e 6c 6f 63 61 6c   ..www.test.local
00a0 - 00 0a 00 0e 00 0c 00 17-00 18 00 19 00 15 00 13   ................
00b0 - 00 1d 00 0b 00 02 01 00-00 2b 00 09 08 03 04 03   .........+......
00c0 - 03 03 02 03 01 00 0d 00-0e 00 0c 04 03 05 03 02   ................
00d0 - 03 04 01 05 01 02 01 00-33 00 26 00 24 00 1d 00   ........3.&.$...
00e0 - 20 a6 6d b8 2e f7 71 d0-b3 f4 cb fc 4e d9 85 0d    .m...q.....N...
00f0 - 50 20 62 be a8 f3 33 a7-1c 69 53 2b 1a 0d af a1   P b...3..iS+....
0100 - 1b                                                .
write to 0x564024a33b40 [0x564024a48a20] (715 bytes => 715 (0x2CB))
0000 - 16 03 03 00 54 02 00 00-50 03 03 e2 20 32 e3 89   ....T...P... 2..
0010 - ab dd be a7 b2 43 c5 5c-63 6b 01 72 93 66 3b a9   .....C.\ck.r.f;.
0020 - 02 8f c6 e6 01 89 77 bd-62 73 8f 20 a6 05 94 6b   ......w.bs. ...k
0030 - a5 00 f2 f7 5e 05 df a7-db 38 32 1f 4f 06 98 6e   ....^....82.O..n
0040 - a6 08 b2 16 d0 ba 91 31-21 ad 5d 13 c0 09 00 00   .......1!.].....
0050 - 08 00 0b 00 04 03 00 01-02 16 03 03 01 cc 0b 00   ................
0060 - 01 c8 00 01 c5 00 01 c2-30 82 01 be 30 82 01 64   ........0...0..d
0070 - a0 03 02 01 02 02 14 43-e4 0e 1a c2 cb 53 e7 8a   .......C.....S..
0080 - df 05 d5 94 68 bd e3 43-c3 13 6e 30 0a 06 08 2a   ....h..C..n0...*
0090 - 86 48 ce 3d 04 03 02 30-45 31 0b 30 09 06 03 55   .H.=...0E1.0...U
00a0 - 04 06 13 02 41 55 31 13-30 11 06 03 55 04 08 0c   ....AU1.0...U...
00b0 - 0a 53 6f 6d 65 2d 53 74-61 74 65 31 21 30 1f 06   .Some-State1!0..
00c0 - 03 55 04 0a 0c 18 49 6e-74 65 72 6e 65 74 20 57   .U....Internet W
00d0 - 69 64 67 69 74 73 20 50-74 79 20 4c 74 64 30 1e   idgits Pty Ltd0.
00e0 - 17 0d 32 30 30 34 32 39-31 32 33 37 35 37 5a 17   ..200429123757Z.
00f0 - 0d 32 30 30 35 32 39 31-32 33 37 35 37 5a 30 5a   .200529123757Z0Z
0100 - 31 0b 30 09 06 03 55 04-06 13 02 43 4e 31 0b 30   1.0...U....CN1.0
0110 - 09 06 03 55 04 08 0c 02-47 44 31 0b 30 09 06 03   ...U....GD1.0...
0120 - 55 04 07 0c 02 53 5a 31-18 30 16 06 03 55 04 0a   U....SZ1.0...U..
0130 - 0c 0f 4d 61 67 72 61 74-68 65 61 2c 20 49 6e 63   ..Magrathea, Inc
0140 - 2e 31 17 30 15 06 03 55-04 03 0c 0e 77 77 77 2e   .1.0...U....www.
0150 - 74 65 73 74 2e 6c 6f 63-61 6c 30 59 30 13 06 07   test.local0Y0...
0160 - 2a 86 48 ce 3d 02 01 06-08 2a 86 48 ce 3d 03 01   *.H.=....*.H.=..
0170 - 07 03 42 00 04 83 94 4a-8c 3c 2c a6 2e eb b1 34   ..B....J.<,....4
0180 - 6e 56 37 43 47 20 5c e5-35 21 b2 9a 69 e5 42 0a   nV7CG \.5!..i.B.
0190 - 4a 1a 73 ed cc b8 3a 61-a3 4a a9 ec 04 c5 3c 0d   J.s...:a.J....<.
01a0 - 03 3d 62 c7 2b f0 c0 68-3b 9a 2c 90 da d0 7f a9   .=b.+..h;.,.....
01b0 - 7e c4 8a 99 25 a3 1d 30-1b 30 19 06 03 55 1d 11   ~...%..0.0...U..
01c0 - 04 12 30 10 82 0e 77 77-77 2e 74 65 73 74 2e 6c   ..0...www.test.l
01d0 - 6f 63 61 6c 30 0a 06 08-2a 86 48 ce 3d 04 03 02   ocal0...*.H.=...
01e0 - 03 48 00 30 45 02 21 00-dd 3f 21 03 7c a7 9e d8   .H.0E.!..?!.|...
01f0 - 4b 07 65 a5 d8 4a c4 cc-d3 d5 3b a2 07 6d 98 d1   K.e..J....;..m..
0200 - bd 97 c3 9f 56 94 94 e2-02 20 34 07 8c 9e 14 1c   ....V.... 4.....
0210 - 3e ec a9 0b 38 51 3c 23-3f b4 08 03 1b b9 e3 66   >...8Q<#?......f
```

```
0220 - f3 0e 9d f2 d8 39 69 a7-a9 23 16 03 03 00 93 0c   .....9i..#......
0230 - 00 00 8f 03 00 17 41 04-7c bb 4e eb 3e 14 b8 97   ......A.|.N.>...
0240 - e9 f6 6c 3c 0f a4 84 be-de 6c c6 47 f4 ca eb d3   ..l<.....l.G....
0250 - 6b 92 34 c4 fc 95 c5 46-4a 67 a1 08 9f 99 0d 87   k.4....FJg......
0260 - b1 7c 4a a5 08 84 72 c9-dd 48 1d 63 7d 81 50 d3   .|J...r..H.c}.P.
0270 - 7e ea df fa e8 79 85 d1-04 03 00 46 30 44 02 20   ~....y.....F0D.
0280 - 69 48 4a c1 14 14 ff c3-43 2b 48 15 93 78 c6 5c   iHJ.....C+H..x.\
0290 - 45 91 2d 06 0c 04 b4 1c-ea 7b de 62 f0 1d 92 d0   E.-......{.b....
02a0 - 02 20 5e ab cb ef 3a c0-0f f2 90 ce 2d 0f 98 11   . ^...:.....-...
02b0 - 46 bd d3 99 8c 54 0b 7e-6b 95 89 26 3f ca c0 91   F....T.~k..&?...
02c0 - 64 0f 16 03 03 00 04 0e-00 00 00                  d..........
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 46                                    ....F
read from 0x564024a33b40 [0x564024a3f758] (70 bytes => 70 (0x46))
0000 - 10 00 00 42 41 04 e4 3e-28 aa d8 88 56 b7 02 3b   ...BA..>(...V..;
0010 - 22 c6 76 bb b5 18 cd 4d-26 58 76 ce b1 14 92 28   ".v....M&Xv....(
0020 - 78 9a d4 15 b6 1c da 3e-f8 28 3b fb 39 82 e6 6d   x......>.(;.9..m
0030 - 50 88 a3 bf 58 6d 89 13-67 6f 44 97 6d 4d 02 44   P...Xm..goD.mM.D
0040 - a8 aa 96 cc 04 dc                                 ......
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 14 03 03 00 01                                    .....
read from 0x564024a33b40 [0x564024a3f758] (1 bytes => 1 (0x1))
0000 - 01                                                .
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 16 03 03 00 40                                    ....@
read from 0x564024a33b40 [0x564024a3f758] (64 bytes => 64 (0x40))
0000 - 64 2c 08 4b 6d c7 a5 36-18 20 98 ac ef 05 96 5d   d,.Km..6. .....]
0010 - a1 b9 e7 be a2 40 c7 f0-52 e3 b9 9a dd 67 cf 15   .....@..R....g..
0020 - 7e 5e 85 14 75 e1 ff 25-b1 11 fc 19 59 20 c5 05   ~^..u..%....Y ..
0030 - d0 9e f9 f3 d2 88 bd f5-76 40 8b 9b 2c 46 48 26   ........v@..,FH&
write to 0x564024a33b40 [0x564024a48a20] (75 bytes => 75 (0x4B))
0000 - 14 03 03 00 01 01 16 03-03 00 40 54 78 c6 4d 3d   ..........@Tx.M=
0010 - 05 69 fd 87 94 f5 0d cb-4b 02 cc 53 cc 0d cf 6d   .i......K..S...m
0020 - 6c f2 fe e4 14 87 50 4b-e2 9c d6 ff 60 77 be 33   l.....PK....`w.3
0030 - 83 ed f5 4d 0d 7d 7c 04-ee 64 c6 92 43 bc 0c 8b   ...M.}|..d..C...
0040 - eb 77 58 3f e8 2b 03 a3-72 04 5c                  .wX?.+..r.\
-----BEGIN SSL SESSION PARAMETERS-----
MHUCAQECAgMDBALACQQgpgWUa6UA8vdeBd+n2zgyH08GmG6mCLIW0LqRMSGtXRME
MCe5f2t8lF1q+m6pt9g4ezEJFv9zIydYlWXrfBK5uesOTtNqWwsMz/88qI4YOX8w
RaEGAgRezmDDogQCAhwgpAYEBAEAAAA=
-----END SSL SESSION PARAMETERS-----
Shared ciphers:TLS_AES_128_GCM_SHA256:TLS_AES_256_GCM_SHA384:
TLS_CHACHA20_POLY1305_SHA256:ECDHE-ECDSA-AES128-SHA:
ECDHE-ECDSA-AES128-SHA256:ECDHE-ECDSA-AES256-SHA:
ECDHE-ECDSA-AES256-SHA384:ECDHE-ECDSA-AES128-GCM-SHA256:
ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES128-SHA:
ECDHE-RSA-AES128-SHA256:ECDHE-RSA-AES256-SHA:ECDHE-RSA-AES256-SHA384:
ECDHE-RSA-AES128-GCM-SHA256:ECDHE-RSA-AES256-GCM-SHA384:DHE-RSA-AES128-SHA:
DHE-RSA-AES128-SHA256:DHE-RSA-AES256-SHA:DHE-RSA-AES256-SHA256:
DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:AES128-SHA:
AES128-SHA256:AES256-SHA:AES256-SHA256:AES128-GCM-SHA256:AES256-GCM-SHA384
Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:ECDSA+SHA1:
RSA+SHA256:RSA+SHA384:RSA+SHA1
Shared Signature Algorithms: ECDSA+SHA256:ECDSA+SHA384:RSA+SHA256:RSA+SHA384
Supported Elliptic Curve Point Formats: uncompressed
Supported Elliptic Groups: P-256:P-384:P-521:P-224:P-192:X25519
Shared Elliptic groups: P-256:P-384:P-521:X25519
CIPHER is ECDHE-ECDSA-AES128-SHA
Secure Renegotiation IS NOT supported
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                    ....0
read from 0x564024a33b40 [0x564024a3f758] (48 bytes => 48 (0x30))
0000 - 7b aa ca 02 ad e6 44 9d-11 5c e1 42 38 f0 19 dd   {.....D..\.B8...
0010 - 30 61 34 d7 78 ff 8e ac-1e 44 6c d2 41 f2 56 17   0a4.x....Dl.A.V.
0020 - 8c 6b a0 a9 cd 01 cb 24-20 6b 39 54 40 9e 97 c5   .k.....$ k9T@...

read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 5 (0x5))
0000 - 17 03 03 00 30                                    ....0
read from 0x564024a33b40 [0x564024a3f758] (48 bytes => 48 (0x30))
0000 - ec 03 47 e7 63 cd 0e 2d-4e 14 6a da 46 be 6c e6   ..G.c..-N.j.F.l.
0010 - a3 d3 d0 8d 39 a8 d7 14-e4 d2 3c b0 54 fd f2 aa   ....9.....<.T...
0020 - 87 84 da 7d e5 f8 92 4a-48 85 11 1c fa c3 5a 7d   ...}...JH.....Z}
ping
pong
write to 0x564024a33b40 [0x564024a438a3] (53 bytes => 53 (0x35))
0000 - 17 03 03 00 30 73 93 88-2f 7f b2 db 66 e5 45 d1   ....0s../...f.E.
0010 - 66 d3 57 94 a9 d9 c9 8b-2c b1 f9 ed 8a 12 07 1d   f.W.....,.......
0020 - 81 f8 71 08 cc bd 5d f6-dd e1 74 66 4b 76 8c bd   ..q...]...tfKv..
0030 - f8 c8 c8 91 7e                                    ....~
read from 0x564024a33b40 [0x564024a3f753] (5 bytes => 0 (0x0))
ERROR
shutting down SSL
CONNECTION CLOSED
```

A successful TLS 1.2 handshake, client side

```
scripts/tls_test --connect www.test.local --port 8443 --cert
↪ /home/pascal/Documents/Bildung/fh/hsr/sem6/SA/sa-strongswan-doku/scripts/
↪ server-client/tls_ecdsa/server.crt

negotiated TLS 1.2 using suite TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
received TLS server certificate 'C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local'
no issuer certificate found for "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
  issuer is "C=AU, ST=Some-State, O=Internet Widgits Pty Ltd"
  using trusted certificate "C=CN, ST=GD, L=SZ, O=Magrathea, Inc., CN=www.test.local"
ping
pong
^C
```