Table of Contents

TLS For IoT

II

Transport Layer Security Protocol For Internet Of Things

Illya Gerasymchuk

`illya.gerasymchuk@tecnico.uliboa.pt`,

WWW home page: `https://iluxonchik.github.io/`

Instituto Superior Tcnico

Supervisors: Ricardo Chaves, Aleksandar Ilic

Abstract. The abstract should summarize the contents of the paper using at least 70 and at most 150 words. It will be set in 9-point font size and be inset 1.0 cm from the right and left margins. There will be two blank lines before and after the Abstract. . . .

Keywords: TLS, IoT, cryptography, protocol, lightweight cryptography

1 Intoduction

TODO: Intruduce the topic: explain what is IoT; what is TLS; what are the issues with using RAW TLS with IoT(power, computation, limited resources).

1.1 Goals

2 Related Work

TODO: Tell that first I describe the parts of TLS that are common to both and then specialize for TLS 1.2 and TLS 1.3

3 The TLS Protocol

TLS stands for Transport Layer Security, it's a **client-server** protocol that runs on top a **connection-oriented and reliable transport protocol**, such as **TCP**. Its main goal is to provide **privacy** and **integrity** between the two communicating peers. Privacy implies that a third party will not be able to read the data, while integrity means that a third party will not be able to alter the data.

In the TCP/IP Protocol Stack, Transport Layer Security (TLS) is placed between the **Transport** and **Application** layers. It's designed to make the application developer's life easier: all the developer has to do is create a "secure" connection, instead of a "normal" one.

TODO: Re-write what's below. It's good to include something like this, but I need to work on the wording. From the top-level view, in a typical connection, there are three basic steps that TLS is responsible for:

1. **Negotiate security parameters** - the communicating peers agree on a set of security parameters to be used in a TLS connection, such as the algorithm used for bulk data encrytion, as well as the secret keys.
2. **Authenticate one to another** - usually only the server authenticates to the client.
3. **Communicate securely** - use the negotiated security parameters to encrypt and authenticate the data, communicating securely one with another.

SSL vs TLS: What's The Difference? You will find the names Secure Sockets Layer (SSL) and TLS used interchangeably in the literature, so I think it's important to distinguish both. TLS is an evolution of the SSL protocol. The protocol changed its name from SSL to TLS when it was standardized by the Internet Engineering Task Force (IETF).SSL was a proprietary protocol owned by Netscape Communications, and The IETF decided that it was a good idea to standarize it, which resulted in `RFC 2246` [5], specifying TLS 1.0, which was nothing more than a new version SSL 3.0, very few changes were made. In this document, I'll be concentrating on TLS 1.2 and TLS 1.3 protocols. The first one is the most recently standardized version of TLS and the latter is currently and in-draft version with many improvements and optimizations relevant for the topic of this dissertation. Despite the protocol name not suggesting it TLS 1.3 is very different from TLS 1.2, in fact, it should've probably been called TLS 2.0 instead. For this reason, I will first describe what is common to both protocols and then go into the relevant details about each one.

TODO: Explain what RFCs are?

3.1 Security Services

TLS provides the following 3 security services:

– **authentication** - both, **peer entity** and **data origin** (or **integrity**) authentication.
  – **peer entity authentication** - we can be sure that were talking to certain entity, for example, `www.google.com`. This is achieved thought the use of **asymmetrical** or Public Key Cryptography (PKC) (for example, `RSA` and `DSA`) or **symmetric key cryptography**, using a Pre-Shared Key (PSK).
– **confidentiality** - the data transmitted between the communicating entities (the client and the server) is encrypted. Symmetric cryptography is used of data encryption (for exmaple, `AES`).
– **integrity** (also called **data origin authentication**) - we can be sure that the data was not modified or forged, *i.e.*, be sure that the data that were receiving is coming from the expected entity (for example, we can be sure that the `index.html` file sent to us when we connected to `www.google.com` in

fact came from `www.google.com` and that it was not modified (i.e tampered with) en route by an attacker (**data integrity**). This is achieved through the use of a keyed Message Authentication Code (MAC) or an Authenticated Encryption With Associated Data (AEAD) cipher.

Despite using PKC, TLS does **not** provide **non-repudiation services**: neither **non-repudiation with proof of origin**, which addresses the user denying having sent a message, not **non-repudiation with proof of delivery**, which addresses the user denying the receipt of a message. This is due to the fact, that instead of using **digital signatures**, either a keyed MAC or an AEAD cipher is used, both of which require a **shared secret** to be used.

You are not required to use all of the 3 security services in every situation. You can think of TLS as a framework that allows you to select which security services you want to use for a communication session. As an example, you might ignore certificate validation, which means you're ignoring the **authentication** guarantee. There are some differences regarding this claim between TLS 1.2 and TLS 1.3, for example, while in the first you have a `null` cipher (no authentication, no confidentiality, no integrity), in the latter this is not true, since it deprecated all non-AEAD ciphers in favor of AEAD ones.

Cipher Spec vs Cipher Suite The meaning of these terms differs in TLS 1.2 and TLS 1.3. For TLS 1.2, **cipher spec** is the message encryption algorithm and the message authentication algorithm, while the **cipher suite** is the **cipher spec**, as well as the **key exchange** algorithm. In TLS 1.3, the **cipher spec** has been removed altogether, since the **ChangeCipherSpec** protocol has been removed. The concept of **cipher suite** has been updated to define the pair of AEAD algorithm and hash function to be used with HMAC-based Extract-and-Expand Key Derivation Function (HKDF): in TLS 1.3 the **key exchange** algorithm is negotiated via extensions. You'll find more details on this below.

3.2 TLS (Sub)Protocols

In reality TLS is composed of several protocols(illustrated in 2), a brief description of each one of which follows:

– **TLS Record Protocol** - the lowest layer in TLS. It's the layer that runs directly on top of **TCP/IP** and it serves as an **encapsulation for the remaining sub-protocols** (4 in case of TLS 1.2 and 3 in case of TLS 1.3). To the **Record Protocol**, the remaining sub-protocols are what `TCP/IP` is to `HTTP`. The TLS Record header is comprised of 4 fields: a 1-byte `record type`, specifying the type of record that's encapsulated (ex: value `0x16` for the handshake protocol), a 2-byte `TLS version` field, a 2-byte `length` field, specifying the length of the data in the record, excluding the header itself and a `fragment` field whose size in bytes is specified by the `length` field, which contains data that's transparent to the Record layer and should be dealt by a higher-level protocol, specified by the `type` field. This is illustrated in figure **??**.

- **TLS Handshake Protocol** - the core protocol of TLS. Allows the communicating peers to **authenticate** one to another and negotiate a **cipher suite** (**cipher suite** and key exchange algorithm in case of TLS 1.3) which will be used to provide the security services. For TLS 1.2, **compression** method is also negotiated here.
- **TLS Alert Protocol** - allows the communicating peers to signal potential problems.
- **TLS Application Data Protocol** - used to transmit data securely.
- **TLS Change Cipher Spec Protocol** (removed in TLS 1.3) - used to activate the initial **cipher spec** or change it during the connection.
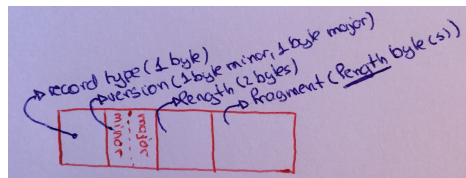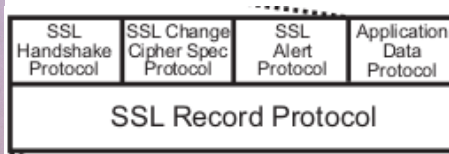

Fig. 1. TLS Record header

Fig. 2. TLS (Sub)protocols and Layers

TLS Connections and Sessions TODO: define what it means to be cryptographically protected?

It's important to distinguish between a **TLS session** and a **TLS connection**.

- **TLS sesion** - assosciation between two communicationg peers that's created by the **TLS Handshake Protocol**, wich defines a set of negotiated paramters (cyrptographic and others, depending on the TLS version, such as the compression algorithm) that are used by the **TLS connections associated with that session**. A single **TLS session** can be shared among multiple **TLS connections** and its main purpuse is to avoid the expensive negotiation of new parameters for each **TLS connection**. For example, let's say you download an Hypertext Markup Language (HTML) page over Hypertext Transfer Protocol Secure (HTTPS) and that page referrences some images from that same server, also using HTTPS, instead of your web browser negotiating a new TLS session again, it can re-use the the one you established to download the HTML page in the first place, saving time and computational resources. Session resumption can be done using various approaches, such as **session identifiers**, described throughout Section 7.4 of RFC 5246 [4], **session tickets**, defined in RFC 5077 [**?**]. TODO: Re-write example better.
- **TLS connection** - used to actually transmit the cryptographically protected data. For the data to be cryptographically protected, some parameters, such as the secret keys used to encrypt and authenticate the trans-

mitted data need to be established; this is done when a **TLS session** is created, during the **TLS Handshake Protocol**.

TLS Record Processing A TLS record must go through some processing before it can tbe sent over the netwrok. This processing involves the following steps (4 for TLS 1.2 and 3 for TLS 1.3):

1. **Fragmentation** - the TLS `Record Layer` takes arbitrary-length data and **fragments** it into manageable pieces: each one of the resulting fragments is called a `TLS Plaintext`. Client message boundaries are not preserved, which means that multiple messages of the same type may be placed into the same fragment or a single message may be fragmented across several records.
2. **Compression** (removed in TLS 1.3) - the `TLS Record Layer` compresses the `TLSPlaintext` structure according to the negotiated compression method, outputting `TLSCompressed`. Compression is optional. If the negotiated compression method is `null`, `TLSCompressed` is the same as `TLSPlaintext`.
3. **Cryptographic Protection** - in case of TLS 1.2, either an AEAD cipher or a separate encryption and MAC functions transform a `TLSCompressed` fragment into a `TLSCipherText` fragment. In case of TLS 1.3, the `TLSPlaintext` fragment is transformed into a `TLSCipherText` by applying an AEAD cipher.
4. Append the **TLS Record Header** - encapsulate `TLSCipherText` in a `TLS Record`.

The process described above, as well as the structure names are depicted in figure 3. Step 2 is not present in TLS 1.3. The structure names are exactly as the appear in the TLS specifications.

3.3 TLS Keying Material

Secret keys are at the base of most cryptographic operations. In order for both communicating peers to be able to encrypt and decrypt data using symmetric cyrpto aglorithms, they need to **share** the same key somehow. In TLS, both, the client and a server derive the **same set of keys** independetely, through the exchanged messages in the TLS Handshake Protocol.

When communicating with one anohter, the client uses one key to encrypt the data to be sent to the server and another different key to decrypt the data that it receives from the server. This means that in order to deal with data encryption and decryption, both of
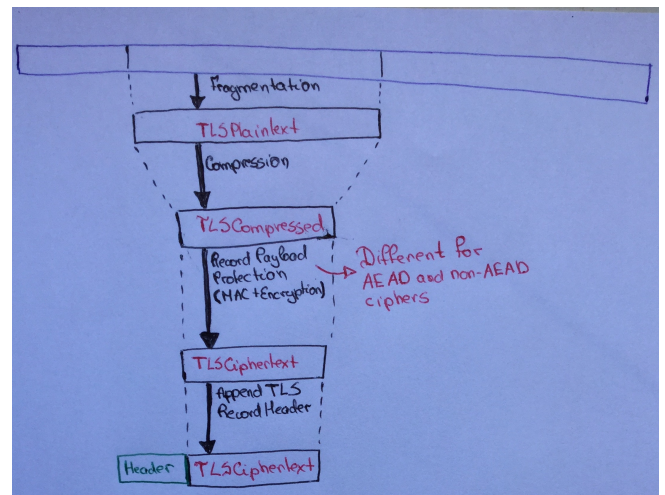

Fig. 3. TLS Record Processing

the communicating entities have two keys: one
to encrypt the outgoing data and one to decrypt the incoming data. Those keys
have different names in TLS 1.2 and TLS 1.3, but they serve the same basic pur-
pose. In this general description, I'll refer to them as `client_write key` (used
by the client to encrypt the data to be sent), `client_read_key`(used by the
client to decrypt the incoming data from the server), `server_write_key`(used
by the server to encrypt the data to be sent) and `server_read_key`(used by
the server to decrypt the incoming data from the client). Note, that the fol-
lowing relationships must hold: `client_write_key == server_write_key` and
`client_read_key == server_write_key`.

Besides the secret keys mentioned previously, in TLS 1.2 you might also have
other ones, depending on the cipher suite in use. TODO: Describe this in a little
more detail, giving examples, when describing TLS 1.2 Key Managment.

TLS 1.3's keying material generation is a little more complex, since different
keys are used to encrypt data throughout the Handshake Protocol, as well a
new key is generated for the Application Data protocol. This can be explained
by the fact that while in TLS 1.2 the data only begins to be encrypted after
the handshake is complete, in the Application Data protocol, the encryption
begins earlier, TLS 1.3, with some of the Hanshake messages encrypted, as well
as features such as **early client/server data** and **0-RTT Data**.

With this the common description of the TLS of protocols ends and we'll
jump into the specifics of the two verions. I'll be mostly concentrating on the
Handshake Protocol, since this is where my work will be concentrated and
it's the main part, where the most interesting and important things happen.

3.4   TLS 1.2

The latest standardized version of TLS is 1.2 and it's defined in `RFC 5246` [4].
TODO: DESCRIBE TLS 1.2 in genreal, put images of handshakes here, later
refer to them in the specific parts, just like the tls RFCs do.

4   TLS 1.2 Keying Material Generation

The generation of secret keys, used for various cryptographic operations involves
the following steps (in order):

- Generate the **premaster secret**
- From the **premaster secret** generate the **master secret**
- From the **master secret** generate the various secret keys, which will be used
  in the cryptographic operations.

TODO: talk about all of the keys present in TLS 1.2 HERE

5 TLS 1.2 Key Exchange Methods

The way the **permaster secret** is generated depends on the key exchange method used. In fact, this is the only phase of the keying material generation phase that is variable for a fixed cipher suite (because a cipher suite defines the Pseudo-Random Function (PRF) function to be used), the rest remains exactly the same. The derivation of the **master secret** from the **premaster secret**, as well as the derivation of the bulk encryption keys, MAC keys and Initialization Vector (IV)s from the **master secret** that follows **is not impacted by the key exchange method** in use.

You have quite a few choices when it comes to key exchange methods. Some of them are defined in the base spec (`RFC5246` [4]), while others in separate `RFCs` (such as the Elliptic Curve Cryptography (ECC) based key exchange, specified in `RFC4492` [2]).

The base spec specifies 4 key exchange methods, one using Rivest-Shamir-Adleman (RSA) and 3 using Diffie-Hellman (DH):

- static RSA (`RSA`) [removed in TLS 1.3] - the client generates the premaster secret (PMS), encrypts it with the server's Public Key (PubK) (which it obtained from the server's `X.509`certificate), sending it to the server, which decrypts it using the corresponding Private Key (PrivK). This key exchange method offers authenticity, but does not offer Perfect Forward Secrecy (PFS).
- anonymous DH (`DH_annon`) [removed in TLS 1.3] - a DH key exchange is performed and an **ephemeral** key is generated, but the exchanged DH parameters are **not authenticated**, making the resulting key exchange vulnerable to Man In The Middle (MITM) attacks. TLS 1.2 spec states that cipher suites using `DH_annon` **must not** be used, unless the application layer explicitly requests so. This key exchange offers PFS, but no authenticity.
- fixed/static DH (`DH`) [removed in TLS 1.3] - the server's/client's public DH parameter is embedded in its certificate. This key exchange method offers authenticity, but does not offer PFS.
- epehemeral DH (`DHE`) - each run of the protocol, uses different pubic DH parameters, which are generated dynamically. This results in a different, epehemeral key being generated every time. The public parameters are then digitally signed in some way, usually using the sender's private RSA (`DHE_RSA)`) `or \gls{dsa} DHE_DSS`) key. This key exchange offers both authenticity and PFS.

When either of the DH variants is used, the value resulting from the exchange is used as the PMS (without the leading `0`'s). Usually, only the server's authenticity is desired, but client's can also be achieved if it provides the server its certificate. Below is a table that summarizes the security properties offered by each key exchange method.

Note that in TLS 1.3, all of static RSA and DH cipher suites have been removed: all of the PubK exchange methods now provide PFS. Even though,

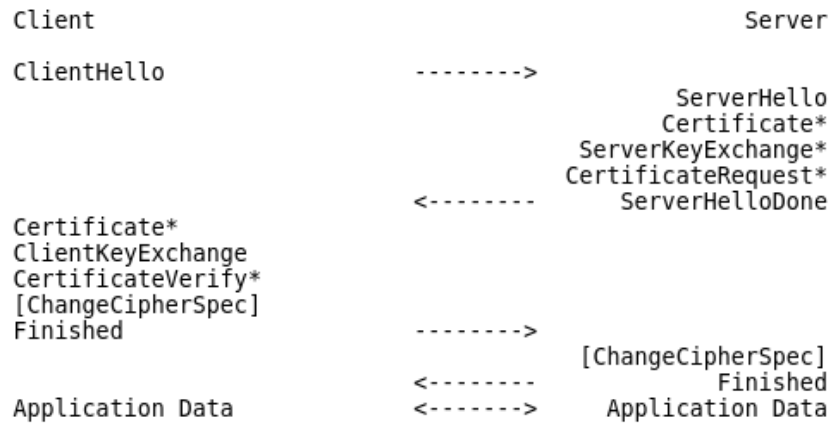**Table 1.** Key exchange methods and security properties

Key Exch Meth	Authentication	PFS
RSA	X	
DH_anon		X
DH	X	
DHE	X	X

anonymous DH has also been removed from TLS 1.3, you can still have unauthenticated connections by either using **raw public keys** [9] or by not verifying the certificate chain and any of it's contents.

TODO: NOTE: I did't cover specifics of how the client generates the premaster secret, etc

The ECC-based key exchange (Elliptic Curve Diffie-Hellman (ECDH) and Elliptic Curve Diffie-Hellman Ephemeral (ECDHE)) and authentication (Elliptic Curve Digital Signature Algorithm (ECDSA)) algorithms are defined in RFC4292 [?], which is also referenced in RFC5246 [?]. The document introduces five new ECC-based key exchange algorithms, all of which use ECC to compute the **premaster secret**, differing only in whether the negotiated keys are epehemeral (ECDH) or long-term (ECDHE), as well as the mechanism (if any) used to authenticate them. Three new ECDSA **client authentication** mechanisms are also defined, differing in the algorithms that the certificate must be signed with, as well as the key exchange algorithms that they can be used with. Those features are negotiated through the TLS Extension Mechanism.

5.1   TLS 1.2 Handshake Protocol

```
Client                                          Server

ClientHello                  -------->
                                               ServerHello
                                               Certificate*
                                         ServerKeyExchange*
                                         CertificateRequest*
                             <--------      ServerHelloDone
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished                     -------->
                                          [ChangeCipherSpec]
                             <--------             Finished
Application Data             <------->      Application Data
```

Fig. 4. TLS 1.2 message flow for a full handshake

In this phase the client and the server agree on which version of the TLS protocol to use, authenticate one to another and negotiate items like the cipher suites and the compression method to use. Figure 4 shows the message flow for the full TLS 1.2 handshake. Note that * indicates situation-dependent messages that are not always sent, while `ChangeCiperSpec` is a separate protocol, rather than a message type.

Every message sent in TLS comes with a TLS Record header. Each record header has 4 fields:

TODO: Mention HelloRequest

Notes and Comments. This is an example of a paragraph. Note the styling.

5.2   TLS 1.3

Despite the protocol name not suggesting it TLS 1.3 is very different from TLS 1.2, in fact, it should've probably been called TLS 2.0 instead.

How Do Peers Distinguish Different TLS Versions? TODO: Talk about version numbers

5.3   TLS Extension Mechanism

TODO: Describe the Extended ClientHello/ServerHello. Use one description for both, TLS 1.2 and TLS 1.3

5.4   The Problem With Compression In TLS

TODO: explain why compression was removed (BEAST and CRIME attacks) and how it can be fixed.

5.5   Theory

TODO: Explain: public key crypto, certificates, AEAD ciphers

References

1. Badra, M., Hajjeh, I.: ECDHE$_P SK Cipher Suites for Transport Layer Security(TLS).RFC5489, RFC Editor(March$
2. Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., Moeller, B.: Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS). RFC 4492, RFC Editor (May 2006), http://www.rfc-editor.org/rfc/rfc4492.txt, http://www.rfc-editor.org/rfc/rfc4492.txt
3. Blumenthal, U., Goel, P.: Pre-Shared Key (PSK) Ciphersuites with NULL Encryption for Transport Layer Security (TLS). RFC 4785, RFC Editor (January 2007)
4. Dierks, T., Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, RFC Editor (August 2008), http://www.rfc-editor.org/rfc/rfc5246.txt, http://www.rfc-editor.org/rfc/rfc5246.txt

5. Dierks, T., Allen, C.: The TLS Protocol Version 1.0. RFC 2246, RFC Editor (January 1999), http://www.rfc-editor.org/rfc/rfc2246.txt, http://www.rfc-editor.org/rfc/rfc2246.txt
6. Eronen, P., Tschofenig, H.: Pre-Shared Key Ciphersuites for Transport Layer Security (TLS). RFC 4279, RFC Editor (December 2005), http://www.rfc-editor.org/rfc/rfc4279.txt, http://www.rfc-editor.org/rfc/rfc4279.txt
7. Rescorla, E.: The Transport Layer Security (TLS) Protocol Version 1.3. Internet-Draft draft-ietf-tls-tls13-21, IETF Secretariat (July 2017), http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-21.txt, http://www.ietf.org/internet-drafts/draft-ietf-tls-tls13-21.txt
8. Tschofenig, H., Fossati, T.: Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things. RFC 7925, RFC Editor (July 2016)
9. Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., Kivinen, T.: Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7250, RFC Editor (June 2014)

Glossary

AEAD Authenticated Encryption With Associated Data. 3, 5

DH Diffie-Hellman. 7, 8

ECC Elliptic Curve Cryptography. 7, 8
ECDH Elliptic Curve Diffie-Hellman. 8
ECDHE Elliptic Curve Diffie-Hellman Ephemeral. 8
ECDSA Elliptic Curve Digital Signature Algorithm. 8

HKDF HMAC-based Extract-and-Expand Key Derivation Function. 3
HTML Hypertext Markup Language. 4
HTTPS Hypertext Transfer Protocol Secure. 4

IETF Internet Engineering Task Force. 2
IV Initialization Vector. 7

MAC Message Authentication Code. 3, 5, 7
MITM Man In The Middle. 7

PFS Perfect Forward Secrecy. 7
PKC Public Key Cryptography. 2, 3
PMS premaster secret. 7
PRF Pseudo-Random Function. 7
PrivK Private Key. 7
PSK Pre-Shared Key. 2
PubK Public Key. 7

RSA Rivest-Shamir-Adleman. 7

SSL Secure Sockets Layer. 2

TLS Transport Layer Security. 1–9