# A Light-Weight TLS and X.509 Profile


## *Abstract*

*This document describes a restricted profile for TLS 1.2 suitable for application in constrained devices such as IoT devices. It also describes a profile for X.509 public key certificates. The main advantage of the TLS profile is that it severely restricts the number of optional features being used and limits the choice of cipher suites to a single one. This can result in smaller and less complex implementations, with a smaller attack surface. The profile for X.509 has a similar aim: improving security by reducing complexity and optional features. It is assumed that the reader is familiar with the details of TLS 1.2 as specified in RFC 5246 and with the X.509v3 standard described in RFC 5280 and how it is used by web servers as specified by the CA/Browser Forum.*


## 1. Introduction

Recently, the standardization of TLS version 1.3 has been finalized (see [RFC8446]). However, it is expected that TLS version 1.2 [RFC5246] will still be around for quite some time to come.
There are good reasons that TLS 1.2 is being succeeded by TLS 1.3. Since 2008, when TLS 1.2 was first published, the network security landscape has evolved significantly. Multiple security vulnerabilities have been discovered in these past years: in the specification of TLS 1.2 itself, the cryptography it uses and the various commonly-used implementations of it.
However, all kinds of relatively low-cost consumer and embedded devices, particularly IoT devices, as well as the networking infrastructure on which it depends, will likely keep using TLS 1.2 for the foreseeable future. These devices are typically constrained devices, having little processing power, working memory and storage. They often run a software stack that has not been developed and is not actively maintained by the device manufacturer, but is, for example, an open source stack or provided by the board and/or IC manufacturer(s).

### 1.1. Light-Weight TLS profile

Even though TLS 1.2 is no longer state of the art and over the years multiple security problems have been discovered around its use, this does not mean that TLS 1.2 is completely insecure in and of itself. A significant source of security problems with implementations of TLS 1.2 has been the large number of optional features, the various choices that can or have to be made during connection establishment and backwards compatibility options to support earlier and weaker versions of the protocol.

Examples of these include:

- CVE-2012-4929, a.k.a. “CRIME”: a session hijacking attack using information leaked when using optional compression feature
- CVE-2013-0169, a.k.a. “Lucky Thirteen”: a timing side-channel attack during handling of invalid messages
- CVE-2014-0160, a.k.a. “Heartbleed”: attacking the implementation of an optional heart-beat extension
- CVE-2014-8730, a.k.a. “POODLE” (newest variant): attacking TLS/SSL versions before TLS 1.2 on the improper checking of padding bytes after decryption
- CVE-2015-7575, a.k.a. “SLOTH”: attacking old hash functions
- CVE-2015-0204, a.k.a. “FREAK”: a downgrade attack to export-grade RSA
- CVE-2015-0205, a.k.a. “SKIP-TLS”: an attack on the TLS state machine implementations to skip encryption altogether
- CVE-2015-4000, a.k.a. “Logjam”: a downgrade attack to export-grade Diffie-Hellman
- CVE-2016-0800, a.k.a. “DROWN”: a protocol attack on servers that still support SSLv2
- CVE-2017-13099 (and many others), a.k.a. “ROBOT””: attacking a vulnerability from 1998 with RSA, that was still not properly fixed

An overview of various other attacks and vulnerabilities is also provided by [RFC7457].

When making sensible and secure choices by using the right configuration parameters, choosing the correct cryptographic algorithms, restricting most of the optional features in TLS 1.2, and removing the backwards compatibility features, we believe it can still be used in a way that is sufficiently secure for a large number of use cases.

This can be accomplished in such a way that the resulting profile is significantly reduced in complexity while remaining backwards compatible to existing TLS 1.2 implementations commonly deployed in cloud servers. The resulting security level can and will never be exactly comparable to what TLS 1.3 can offer. However, for many IoT devices and use cases, this may simply not be necessary in the near future.

This document specifies exactly such a configuration as a TLS 1.2 profile. It is called Light-Weight TLS (LWTLS).

## 1.2. Light-Weight X.509 certificate profile

In all versions of TLS and DTLS, authentication based on a Public Key Infrastructure (PKI) using X.509 certificates[1] is the default and the most widely used method for the

[1] X.509 is defined by the standardization sector (ITU-T) of the International Telecommunications Union (ITU) as Recommendation X.509 (for the latest version from 2016 see [X.509]) and is also

communication peers to authenticate each other. Like TLS (including its predecessor SSL), X.509 has a very long history and has been continuously extended by adding new options and features as well as deprecating older features. Due to the complexity of the certificate format and the certificate validation rules that is further increased by the abundance of options and features that need to be taken into account, implementing algorithms to parse and validate certificates is a very difficult task and countless security vulnerabilities have been discovered in implementations over the years. Apart from its inherent implementation security issues, the complexity of X.509 certificate validation also significantly increases processing overhead and code size. In addition, certificates and certificate chains can grow very large depending on which options and features are selected during creation, how much additional metadata is added and how it is encoded.

Therefore, constrained embedded devices are especially impacted by authentication based on X.509 certificates due to their very limited amount of memory, processing capacity and bandwidth. Also, the software (firmware) on constrained devices is typically difficult to keep updated over the device lifetime in case security vulnerabilities are discovered or in case configuration changes in the PKI need to be supported.

In contrast to TLS, no successor to X.509 is in sight. Several extensions of TLS have been standardized to add support for alternatives that do not rely on a PKI (like pre-shared keys (PSK) [RFC4279], raw public keys [RFC7250] and OpenPGP keys [RFC6091]) and the latest version TLS 1.3 added optional support for pre-shared keys and raw public keys in the base standard without extensions. These alternatives (especially PSKs) have been included in many connectivity standards for constrained devices (*e.g.* CoAP, Thread) as they significantly decrease the burden on the constrained device. For larger scale deployments or in open networks however, only a PKI based authentication provides enough flexibility and ease of deployment to be feasible. Attempts have also been made to define extensions specifying PKI-based alternatives for X.509 certificates for TLS (*e.g.* ECQV implicit certificates [SEC4] [ECQV], ETSI/IEEE certificates [certIEEE1] [certIEEE2]), but none have ever left draft status.

Similar to the situation with TLS 1.2, X.509 certificates will still have to be supported for the foreseeable future (in case of X.509 probably even longer as no alternative is in sight), even by low-cost constrained devices. As attempted by the Light-Weight profile for TLS 1.2 in section 2, the goal of section 3 is to define a Light-Weight profile for X.509 that improves implementation security and reduces memory, processing

---

standardized by the International Organization for Standardization (ISO) as ISO/IEC 9594-8 (for the latest version ISO/IEC 9594-8:2017 see [ISO9594]). In the context of TLS, the term "X.509 certificate" refers to the profile for version 3 of the X.509 certificate format (X.509v3) defined by the IETF in [RFC5280] and [RFC3280].

and bandwidth requirements. This is achieved by pre-defining sensible and secure choices, removing optional or outdated features, simplifying validation rules and providing guidance on creating compact certificates and certificate chains. To ensure interoperability with existing cloud services, the choices made are based on how X.509 certificates are typically used in practice.
The Light-Weight X.509 profile is meant to complement the Lightweight TLS profile (*e.g.* by matching the crypto choices made there for TLS 1.2), but it can also be used independently.

## 2. Light-Weight TLS

As mentioned in the introduction, the main target for LWTLS is the class of constrained devices, especially IoT devices. However, in the definition of LWTLS there is nothing that restricts it to this class of devices.[2] The LWTLS profile can be used by anyone wishing to configure TLS 1.2 as secure as possible without losing compatibility to plain standard TLS 1.2.

The first and foremost requirement of LWTLS is that it is compatible with servers implementing regular TLS 1.2. This means that it does not introduce any new extensions, cryptographic algorithms or protocol options. (This also somewhat restricts what can be done to achieve as much security as possible.)

The second requirement is that it removes or disables any obsolete, unused or insecure TLS 1.2 protocol options. All backward compatibility to earlier TLS versions is dropped, as this automatically prevents any version downgrade attacks. No outdated and insecure cryptographic algorithms are supported. Only up-to-date secure cryptographic algorithms are supported.

The third requirement is that the LWTLS profile is fixed. Extensibility is considered undesirable as it is a potential future security risk. As mentioned before, LWTLS does not introduce any new extensions itself. Also, runtime reconfigurability can pose a security risk, so an established LWTLS connection is not reconfigurable or renegotiable.

As an added benefit, both for security and device resource requirements, it is expected that an optimized LWTLS implementation is less complex, smaller and requires less device resources such as RAM and processing power.

The following description follows closely the description and ordering of the sections of the original TLS 1.2 protocol in [RFC5246]. It is assumed that the reader is well-acquainted with the contents of that document, especially with the concepts and terms introduced in it. Technical details and behavior of LWTLS that are not specifically defined below are understood to be as specified for TLS 1.2 in [RFC5246].

In addition to [RFC5246], LWTLS makes use of the following additional RFCs:

- RFC 8422 “Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier” for the ECDHE_ECDSA key exchange algorithm and client authentication mechanism ‘ECDSA_sign’
- RFC 6655 “AES-CCM Cipher Suites for Transport Layer Security (TLS)” for the AES-CCM data encryption algorithm
- RFC 7251 “AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS” for the definition of the TLS_ECDHE_ECDSA_WITH_AES_128_CCM cipher suite

[2] LWTLS can also be used in the boot-ROM of an SoC or in trusted firmware running in internal RAM, where there are code size restrictions, for example, for implementing a firmware update procedure.

- RFC 6066 “Transport Layer Security (TLS) Extensions: Extension Definitions”, section 3 “Server Name Indication”
- RFC 5077 “Transport Layer Security (TLS) Session Resumption without Server-Side State”
- RFC 4347 “Datagram Transport Layer Security Version 1.2”

## 2.1. HMAC and the Pseudorandom Function

Like TLS 1.2, LWTLS 1.2 uses the HMAC construction to compute a pseudo-random function (PRF) needed to do expansion of secrets into blocks of data for the purposes of key generation or validation. LWTLS uses the exact same HMAC based on SHA-256 as TLS 1.2.

## 2.2. LWTLS Record Protocol

The TLS 1.2 Record Protocol definitions are used without any modifications to preserve backwards compatibility. However, a number of restrictions and simplifications are made to both simplify implementations and ensure improved security.
The four protocols described for TLS 1.2 that use the record protocol are also used in LWTLS: the handshake protocol, the alert protocol, the change cipher spec protocol, and the application data protocol. Apart from the four content types specified in [RFC5246], no additional record content types are supported by LWTLS.
So, every LWTLS Record Protocol message is a valid one in standard TLS 1.2, however, the opposite may not always be the case.

### 2.2.1. Connection states

The connection state for LWTLS is identical to that of TLS 1.2. As in TLS 1.2, the initial current state always specifies that no encryption, compression, or MAC will be used. While in this state, application data must not be sent.
When the sequence number reaches $2^{64}$-1, no renegotiation is done (as that is not supported by LWTLS). Instead, a fatal alert is sent and the connection is terminated.

### 2.2.2. Record layer and protection

In contrast to TLS 1.2, LWTLS has no support for compression and decompression, nor for fragmentation and defragmentation. Applications using LWTLS must therefore take care that their messages do not exceed $2^{14}$ bytes.
In the initial current state, the cipher suite is TLS_NULL_WITH_NULL_NULL. When this cipher suite is selected, no application data must be sent.
LWTLS only supports AEAD mode for record payload protection; CBC mode is not supported. All key calculation is done as for TLS 1.2.

## 2.3. LWTLS Handshaking Protocols

LWTLS supports the same three subprotocols for handshaking as [RFC5246]:

- Change Cipher Spec protocol
- Alert protocol
- Handshake protocol

### 2.3.1. Change Cipher Spec protocol

The Change Cipher Spec protocol used by LWTLS is unchanged from that specified by TLS 1.2. However, as renegotiation is not supported by LWTLS, the complications for the Change Cipher Spec protocol associated with that do not exist for LWTLS.

### 2.3.2. Alert protocol

LWTLS does not support the AlertLevel "warning". Sent alert messages in LWTLS must always be "fatal", therefore any received alert shall always be considered fatal, resulting in the immediate termination of the connection. In case any other connections exist corresponding to the same session, these must be terminated too. Such connections must not be resumed.

### 2.3.3. Handshake protocol overview

In contrast to TLS 1.2, the Server must always send a Server Certificate message. Server authentication is required. In essentially all real-world use cases, especially for IoT, it makes no sense to connect to an unknown server.

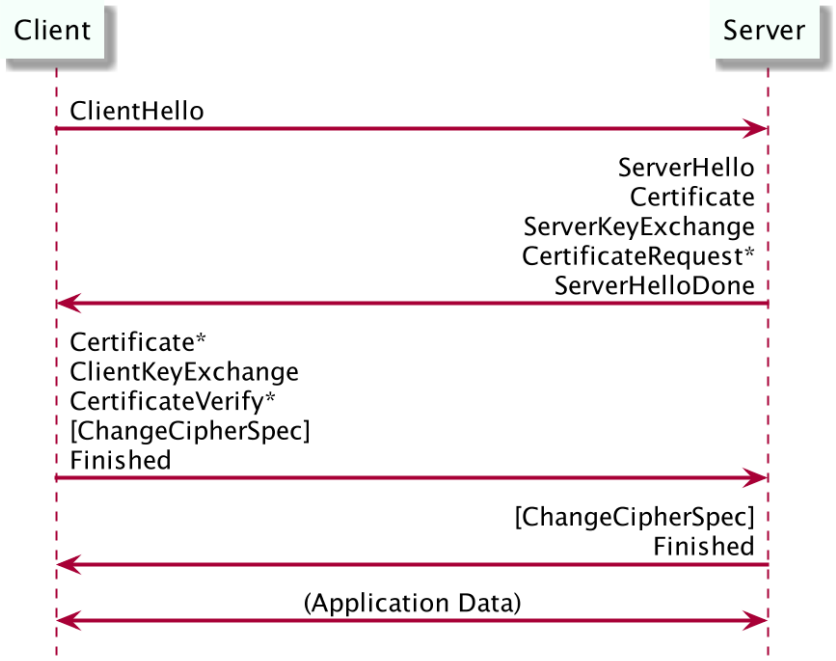


*Figure 1 Message flow for a full LWTLS handshake (* indicates optional messages)*

### 2.3.4. Hello Request

As renegotiation is not supported by LWTLS, a server supporting LWTLS should not send any Hello Request messages. An LWTLS client must ignore any such message.

### 2.3.5. Client Hello

The ClientHello message is as defined in TLS 1.2 with the following restrictions:

- client_version: This field must be set to 3.3 (the highest version supported by TLS 1.2).
- cipher_suites: This field must only contain TLS_ECDHE_ECDSA_WITH_AES_128_CCM.
- compression_methods: This field must only contain CompressionMethod.null ('0', no compression).

For further restrictions on extensions, see 2.3.7 below.
As in [RFC5246], after sending the ClientHello message, the client waits for a ServerHello message. Any handshake message returned by the server, except for a HelloRequest, is treated as a fatal error.

### 2.3.6. Server Hello

The ServerHello message is exactly the same as for TLS 1.2. If the server only supports LWTLS, it should reject any ClientHello message that does not conform to the restrictions specified in section 2.3.5 above. It should also reject any ClientHello message that does not contain the mandatory signature algorithms extension, or if it contains any other extensions than the allowed ones, as specified in section 2.3.7 below.

### 2.3.7. Hello extensions

LWTLS supports the following extensions in the ClientHello message:

- Signature algorithms extension, as defined in TLS 1.2
- Server Name Indication (SNI) extension, as defined in section 3 of [RFC6066]
- SessionTicket extension, as defined in [RFC5077]

For the signature algorithms extension in LWTLS, the only supported hash algorithm is SHA-256 and the only supported signature algorithm is ECDSA. This extension is mandatory for LWTLS. The client must send it and if the server does not accept it, the connection must be terminated.
The Server Name Indication (SNI) extension is required by some cloud service operators to be able to present multiple certificates from the same IP address. It is not mandatory, but if it is used, the client must check that the server name in the extension matches that on the certificate (as specified in [RFC6066]).
The ticket-based session resumption is used exactly as specified in [RFC5077]. The use of this extension is optional.

No other extensions are supported by LWTLS. The procedure for the detection of the presence of extensions is as specified in [RFC5246]. When a client requests session resumption, it must send the same extensions as it would send if it were not attempting resumption.

#### 2.3.8. Server Certificate

As server authentication is mandatory in LWTLS, a server must always send a Certificate message during the handshake. As LWTLS clients only support ECC algorithms, all public keys included in the certificate chain, including the server public key itself, must be ECDSA keys.

#### 2.3.9. Certificate Request

The server can optionally request client authentication. If it does, it must request only the ECDSA_sign mechanism in its CertificateRequest message.

#### 2.3.10. Server Key Exchange Message

The server must send a ServerKeyExchange message. It must contain an ephemeral ECDH public key (freshly generated) and the specification of the elliptic curve secp256r1 (NamedCurve value 23, as defined in [RFC8422]). This message must be signed using the ECDSA private key corresponding to the public key in the server's certificate.

#### 2.3.11. Client Certificate

In case of client authentication, the client public key in the Certificate message must be an ECDSA key. Any intermediate public keys in the certificate chain, if present, are preferably also ECDSA keys.

#### 2.3.12. Client Key Exchange message

As LWTLS only supports ECDHE, the ClientKeyExchange message will always be of the ClientDiffieHellmanPublic variant.

#### 2.3.13. Certificate Verify

As specified in [RFC5246], this message is only sent following a ClientCertificate message. LWTLS only supports client authentication through ECDSA_sign. The structure of the message is as defined in [RFC8422], with sha_hash being a SHA-256 hash value.

### 2.4. Cryptographic computations

The calculation of the master_secret value is done exactly as specified for TLS 1.2.

## 2.5. LWTLS mandatory cipher suite

As listed in the restrictions on the ClientHello above, LWTLS only supports the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM (value {0xC0, 0xAC}, as defined in [RFC7251] and [RFC8422]) and only ECDSA_sign as client authentication mechanism (as defined in [RFC8422]). The support for ECC is even further restricted so that LWTLS clients (and, optionally, servers) only need to support a limited set of ECC options.
The only ECC curve supported by LWTLS is secp256r1 (a.k.a. P-256). Furthermore, LWTLS implementations only support uncompressed points.

## 2.6. Application data protocol

Application data messages are carried by the record layer. They are encrypted based on the current connection state. Application protocols must not continue to exchange data (in plain) after the TLS connection is closed.

## 2.7. Additional behavioral requirements

In addition to the requirements above, closure of the TLS connection must also result in closure of the network connection.

## 2.8. Security considerations

The choice for the cipher suite TLS_ECDHE_ECDSA_WITH_AES_128_CCM results in the use of the most up-to-date and state-of-the-art cryptographic algorithms defined in accepted standards and supported by most TLS implementations.
The choice for ECDHE results in some amount of forward secrecy: the security of connections based on different ECDH keys will not depend on each other's secret keys. LWTLS implementations must therefore always generate a new private key for each new connection. However, the RNG of a constrained device (such as typically the case with LWTLS) may still have a bias, making the security perhaps less than optimal.
In case of a resumed connection, however, the connection security will depend on the confidentiality of the keys of the original connection. In case ticket-based session resumption is used, the security may also depend on the key storage and management practices on the server side. This may seem like session resumption is not the most secure feature to support. However, session resumption allows a constrained device to set up a TLS connection very quickly without doing any public key calculations, which can be very costly and slow on a constrained device.
The choice for AES-CCM instead of, for example, AES-GCM, which seems to be commonly preferred by server implementations, is driven by the fact that the implementation of AES-CCM on a constrained device without (complete) hardware acceleration is typically more efficient than that of AES-GCM.

## 2.9. Light-Weight DTLS

The LWTLS profile can also be applied to DTLS 1.2 [RFC6347], resulting in a Light-Weight DTLS (LWDTLS) profile. All the same restrictions that apply for LWTLS also apply for LWDTLS.
The result is large compatible with the DTLS profile for CoAP [RFC7252]. The main difference is that CoAP has opted for the TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 cipher suite, using 8 bytes for authentication, instead of TLS_ECDHE_ECDSA_WITH_AES_128_CCM, which uses the full 16 bytes. Another difference is that LWDTLS lacks support for pre-shared AES keys and raw public ECDSA keys.

# 3. Light-Weight X.509 profile

The Light-Weight X.509 certificate profile is mostly defined as a simplified subset of the "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" standardized by the IETF as RFC5280 (for further reference see [RFC5280]). The guidelines for compact certificate creation are additionally based on the "Baseline Requirements for Contents of Publicly Trusted SSL/TLS Certificates" specified by the CA/Browser Forum (For further reference see [CABreq]). As in the Lightweight TLS profile, a deliberate choice is made to support only one cryptographic algorithm for each kind of cryptographic operation and the choices made match those from the TLS profile.

## 3.1. Certificate and certificate path validation rules

Because of the fixed choices for cryptographic algorithms and supported options, certificate parsing and validation can be simplified in many cases (*e.g.* by just comparing individual fields or even a complete substructure of the certificate against the only supported values). The certificate validation rules are detailed in the following subsections based on the certificate type they are applied against. Only the contents (payload) of the fields listed here are validated, for the rest of the certificate structure only consistency with the X.509 certificate format is checked (*e.g.* field type, consistent length). This also applies to X.509 extensions, so Certificate Policies, Policy Mappings, Policy Constraints, Name Constraints, Certificate Revocation Lists and all related X.509 extensions as well as any other extensions not explicitly mentioned in this profile are not supported.

### 3.1.1. All certificates

All types of certificates will contain the following fixed value fields with only the predefined payload value allowed as specified by the following table (Table 1).
The X.509v3 extension fields of the certificate are validated based on the type of certificate, but for all types of certificates it must be checked that there are no critical extensions present that are not supported.

### 3.1.2. Intermediate CA certificates

In addition to the checks that apply to all types of certificates, it must be verified for all intermediate CA certificates that the Basic Constraints x509 extension is present and its cA field is set to true. If the optional maxPathLen field is present it must be verified as well.

*Table 1: Fixed value fields in all types of certificates*

| Certificate field | Type | Fixed value |
|---|---|---|
| tbsCertificate.version | INTEGER | 2 (X.509v3 only) |
| tbsCertificate.signature.algorithm | OID | ecdsaWithSHA256 (RFC5758: {iso(1), member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)}) |
| tbsCertificate.signature.parameters | ANY | Omitted completely (not encoded as NULL parameter, see RFC5758 section 3.2.) |
| tbsCertificate.subjectPublicKeyInfo .algorithm.algorithm | OID | ecPublicKey (RFC3279: {iso(1) member-body(2) us(840) ansi-X9-62(10045) id-public-key-type(2) id-ecPublicKey(1)}) |
| tbsCertificate.subjectPublicKeyInfo .algorithm.parameters .namedCurve | ANY<br><br>OID | EcpkParameters, namedCurve (RFC3279) prime256v1 (RFC5758: {iso(1) member-body(2) us(840) ansi-X9-62(10045) curves(3) prime(1) prime256v1(7)}) |
| signatureAlgorithm.algorithm | OID | ecdsaWithSHA256 (RFC5758: {iso(1), member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)}) |
| signatureAlgorithm.parameters | ANY | Omitted completely (not encoded as NULL parameter, see RFC5758 section 3.2.) |

### 3.1.3. End entity certificates

For server certificates, performing server hostname validation is mandatory, using the dNSName field of the Subject Alternative Name x509 extension only. As defined by RFC6125 (For further reference see [RFC6125] section 6.4.3. Checking of Wildcard Certificates), wildcards are only supported for leftmost part of the hostname. Using the legacy Common Name in the Subject field for hostname validation is not supported. If the Extended Key Usage x509 extension is present, it must be verified that it allows the use of the certificate for server authentication (*i.e.* it contains the object identifier id-kp-serverAuth).
For client certificates, verification of the identity of the end entity is completely application specific and is therefore out of scope of this profile. If the Extended Key Usage x509 extension is present, it must be verified that it allows the use of the certificate for client authentication (*i.e.* it contains the object identifier id-kp-clientAuth).

### 3.1.4. Certification path validation

As already indicated, the simplified certification path validation for the Lightweight X.509 certificate profile is based on the Basic Path Validation from RFC5280 (For further reference see [RFC5280] section 6.1: Basic Path Validation), but it is

simplified based on pragmatic assumptions on practical security requirements and current deployment of X.509 certificates in major cloud services.
The certification path validation starts at a trusted CA certificate that is used as Root of Trust. As specified by RFC5280, multiple Roots of Trust can be supported at the same time, but only one will lead to a successful validation of a given path. It is assumed that the certificate chain received is complete and already in the correct order (starting with the end entity certificate). For each certificate in the chain it must be verified that its Issuer field matches the Subject field of the certificate that issued it until the Root of Trust is reached. It is assumed that the Subject and Issuer fields are identically encoded along the chain, so a simple comparison (*e.g.* using memcmp) of the entire structure of both fields can be used for this verification and no decoding of their internal structure is needed. In almost all cases this simplifying assumption is true in practice, as there is no valid reason to encode these structures differently for the same entity. In addition, the signature of each certificate must be verified against the public key of the certificate that issued it and the validity period of each certificate needs to be checked.

## 3.2. Compact certificate creation guidelines

The goal of the compact certificate creation guidelines defined in the following subsections is to provide guidance on how to create compact certificates and certificate chains that are fully compliant with the Lightweight X.509 certificate profile. In contrast to the simplified certificate and certificate chain validation rules, the creation guidelines can be more restrictive as they do not have to consider interoperability with certificates that have already been deployed.

### 3.2.1. All certificates

The following fixed values and other restrictions apply to fields in all types of certificates as specified by the following table (Table 2).

*Table 2: Restrictions for all types of certificates*

| Certificate field | Type | Restriction description |
|---|---|---|
| tbsCertificate.version | INTEGER | Fixed value: 2 (X.509v3 only) |
| tbsCertificate.signature.algorithm | OID | Fixed value: ecdsaWithSHA256 (RFC5758: {iso(1), member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)}) |
| tbsCertificate.signature.parameters | ANY | Omitted completely (not encoded as NULL parameter, see RFC5758 section 3.2.) |
| tbsCertificate.subjectPublicKeyInfo .algorithm.algorithm | OID | Fixed value: ecPublicKey (RFC3279: {iso(1) member-body(2) us(840) ansi-X9-62(10045) id-public-key-type(2) id-ecPublicKey(1)}) |
| tbsCertificate.subjectPublicKeyInfo .algorithm.parameters .namedCurve | ANY<br><br>OID | Fixed value: EcpkParameters, namedCurve (RFC3279) prime256v1 (RFC5758: {iso(1) member-body(2) us(840) ansi-X9-62(10045) curves(3) prime(1) prime256v1(7)}) |
| signatureAlgorithm.algorithm | OID | Fixed value: ecdsaWithSHA256 (RFC5758: {iso(1), member-body(2) us(840) ansi-X9-62(10045) signatures(4) ecdsa-with-SHA2(3) ecdsa-with-SHA256(2)}) |
| signatureAlgorithm.parameters | ANY | Omitted completely (not encoded as NULL parameter, see RFC5758 section 3.2.) |
| issuerUniqueID | | Not allowed (see RFC5280: CAs conforming to this profile MUST NOT generate certificates with unique identifiers.) |
| subjectUniqueID | | Not allowed (see RFC5280: CAs conforming to this profile MUST NOT generate certificates with unique identifiers.) |
| serialNumber | | Max 20 bytes (see RFC5280) |
| Issuer | | Only PrintableString or UTF8String supported for DirectoryString (see RFC5280 section 4.1.2.4.)<br>X.500 Names identically encoded along the chain (to avoid having to decode and match the individual parts on the client) |
| Subject | | Only PrintableString or UTF8String supported for DirectoryString (see RFC5280 section 4.1.2.4.)<br>X.500 Names identically encoded along the chain (to avoid having to decode and match the individual parts on the client) |

### 3.2.2. Root and intermediate CA certificates

In all created CA certificates, the Basic Constraints X.509 extension must be present and must be marked critical (see also CA/Browser Forum baseline requirements

[CABreq]). The cA field of the Basic Constraints extension must be set to True and the pathLenConstraint field should be omitted if not needed to restrict the length of the chain.
In addition, the Key Usage X.509 extension must be present and must be marked critical. Its bit positions for keyCertSign and cRLSign must be set (see also CA/Browser Forum baseline requirements [CABreq]).
As the Subject Alternative Name X.509 extension is not needed for CA certificates it should be omitted completely.

### 3.2.3. End entity certificates

All end entity certificates should have an empty Subject field as the Subject Alternative Name X.509 extension is used instead. Therefore, the Subject Alternative Name extension must be present and must be marked as critical (see also CA/Browser Forum baseline requirements [CABreq]). Both the Basic Constraints and the Key Usage X.509 extensions should be omitted completely (they are optional in the CA/Browser Forum baseline requirements [CABreq]).
Server certificates must contain a dNSName field in their Subject Alternative Name for hostname validation. For client certificates, verification of the identity of the end entity is completely application specific and is therefore out of scope of this profile.
The Extended Key Usage X.509 extension must be present and must contain the OID id-kp-serverAuth for server certificates and the OID id-kp-clientAuth for client certificates (see also CA/Browser Forum baseline requirements [CABreq]).

## 4. Conclusion

It is expected that TLS version 1.2 will still be around for quite some time to come, especially in the area of IoT. Therefore, a Light-Weight TLS (LWTLS) profile was presented here, geared towards IoT devices. The LWTLS profile significantly reduces the number of optional features in TLS 1.2, removes any insecure backwards compatibility features, and strongly restricts the options available during connection establishment while remaining compatible with existing implementations commonly deployed in cloud servers. The resulting security level of LWTLS can be compared to what TLS 1.3 offers (except for new features such as handshake encryption, key update messages and forward-secrecy in session resumption). However, for many IoT devices and use cases, this may simply not be necessary in the near future, while the additional complexity makes TLS 1.3 unfeasible for these devices. We believe that LWTLS can be used in a way that is sufficiently secure for a large number of use cases, particularly in IoT.

In contrast to TLS 1.2, no successor to X.509v3 public-key certificates is in sight, even though the verification of these certificates has been a major source of vulnerabilities in the past. This is due to the complexity of its format, its validation rules and the large amount of options and features that need to be taken into account. Therefore, the Light-Weight X.509 profile was presented here that improves implementation security and reduces memory, processing and bandwidth requirements. This is achieved by pre-defining sensible and secure choices, removing optional or outdated features, simplifying validation rules and providing guidance on creating compact certificates and certificate chains. To ensure interoperability with existing cloud services, the choices made are based on how X.509 certificates are typically used in practice.

The Light-Weight X.509 profile is meant to complement the Light-Weight TLS profile (*e.g.* by matching the crypto choices made there for TLS 1.2), but it can also be used independently.

## 5. References


[CABreq] CA/Browser forum baseline requirements: https://cabforum.org/baseline-requirements-certificate-contents/ https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.5.7-29-Apr-2018.pdf

[certIEEE1] draft-lonc-tls-certieee1609-01: "Transport Layer Security (TLS) Authentication using ITS ETSI and IEEE certificates" (https://tools.ietf.org/html/draft-lonc-tls-certieee1609-01)

[certIEEE2] draft-tls-certieee1609-01: "TLS 1.3 Authentication using ETSI TS 103 097 and IEEE 1609.2 certificates" (https://tools.ietf.org/html/draft-tls-certieee1609-01)

[ECQV] draft-campagna-tls-ecmqv-ecqv-01: "ECMQV_ECQV Cipher Suites for Transport Layer Security (TLS)" (https://tools.ietf.org/html/draft-campagna-tls-ecmqv-ecqv-01)

[ISO9594] ISO/IEC 9594-8:2017 "Information technology -- Open Systems Interconnection -- The Directory -- Part 8: Public-key and attribute certificate frameworks" (https://www.iso.org/standard/72557.html)

[RFC3279] RFC 3279 "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" (https://tools.ietf.org/html/rfc3279)

[RFC3280] RFC 3280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" (https://tools.ietf.org/html/rfc3280), superseded by RFC 5280

[RFC4279] RFC 4279 "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)" (https://tools.ietf.org/html/rfc4279)

[RFC5077] RFC 5077 "Transport Layer Security (TLS) Session Resumption without Server-Side State"

[RFC5246] RFC 5246 "The Transport Layer Security (TLS) Protocol Version 1.2"

[RFC5280] RFC 5280 "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" (https://tools.ietf.org/html/rfc5280)

[RFC5758] RFC 5758 "Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA" (https://tools.ietf.org/html/rfc5758)

[RFC6066] RFC 6066 "Transport Layer Security (TLS) Extensions: Extension Definitions"

[RFC6091] RFC 6091 "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication" (https://tools.ietf.org/html/rfc6091)

[RFC6125] RFC 6125 "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)" (https://tools.ietf.org/html/rfc6125)

[RFC6347] RFC 6347 "Datagram Transport Layer Security Version 1.2"

[RFC6655] RFC 6655 "AES-CCM Cipher Suites for Transport Layer Security (TLS)"

[RFC7250] RFC 7250 "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)"

[RFC7251] RFC 7251 "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS"

[RFC7252] RFC 7252 "The Constrained Application Protocol (CoAP)"

[RFC7457] RFC 7457 "Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS)"

[RFC8422] RFC 8422 "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS) Versions 1.2 and Earlier" (replaces older RFC 4492)

[RFC8446] RFC 8446 "The Transport Layer Security (TLS) Protocol Version 1.3" (https://tools.ietf.org/html/rfc8446)

[SEC4] SEC 4: Elliptic Curve Qu-Vanstone Implicit Certificate Scheme (ECQV) (http://www.secg.org/sec4-1.0.pdf)

[X.509] ITU-T Recommendation X.509 (10/16): "X.509 : Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks" (http://www.itu.int/rec/T-REC-X.509-201610-I/en)