Internet Engineering Task Force (IETF)
Request for Comments: 7925
Category: Standards Track
ISSN: 2070-1721

H. Tschofenig, Ed.
ARM Ltd.
T. Fossati
Nokia

July 2016

Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) Profiles for the Internet of Things

Abstract

A common design pattern in Internet of Things (IoT) deployments is the use of a constrained device that collects data via sensors or controls actuators for use in home automation, industrial control systems, smart cities, and other IoT deployments.

This document defines a Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) 1.2 profile that offers communications security for this data exchange thereby preventing eavesdropping, tampering, and message forgery. The lack of communication security is a common vulnerability in IoT products that can easily be solved by using these well-researched and widely deployed Internet security protocols.

Status of This Memo

This is an Internet Standards Track document.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc7925.

Table of Contents

1. Introduction

An engineer developing an Internet of Things (IoT) device needs to investigate the security threats and decide about the security services that can be used to mitigate these threats.

Enabling IoT devices to exchange data often requires authentication of the two endpoints and the ability to provide integrity and confidentiality protection of exchanged data. While these security services can be provided at different layers in the protocol stack, the use of Transport Layer Security (TLS) / Datagram Transport Layer Security (DTLS) has been very popular with many application protocols, and it is likely to be useful for IoT scenarios as well.

Fitting Internet protocols into constrained devices can be difficult, but thanks to the standardization efforts, new profiles and protocols are available, such as the Constrained Application Protocol (CoAP) [RFC7252]. CoAP messages are mainly carried over UDP/DTLS, but other transports can be utilized, such as SMS (as described in Appendix A) or TCP (as currently being proposed with [COAP-TCP-TLS]).

While the main goal for this document is to protect CoAP messages using DTLS 1.2 [RFC6347], the information contained in the following sections is not limited to CoAP nor to DTLS itself.

Instead, this document defines a profile of DTLS 1.2 [RFC6347] and TLS 1.2 [RFC5246] that offers communication security services for IoT applications and is reasonably implementable on many constrained devices. Profile thereby means that available configuration options and protocol extensions are utilized to best support the IoT environment. This document does not alter TLS/DTLS specifications and does not introduce any new TLS/DTLS extension.

The main target audience for this document is the embedded system developer configuring and using a TLS/DTLS stack. This document may, however, also help those developing or selecting a suitable TLS/DTLS stack for an IoT product. If you are familiar with (D)TLS, then skip ahead to Section 4.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

This specification refers to TLS as well as DTLS and particularly to version 1.2, which is the most recent version at the time of writing.

We refer to TLS/DTLS whenever the text is applicable to both versions of the protocol and to TLS or DTLS when there are differences between the two protocols. Note that TLS 1.3 is being developed, but it is not expected that this profile will "just work" due to the significant changes being done to TLS for version 1.3.

Note that "client" and "server" in this document refer to TLS/DTLS roles, where the client initiates the handshake. This does not restrict the interaction pattern of the protocols on top of DTLS since the record layer allows bidirectional communication. This aspect is further described in Section 3.2.

RFC 7228 [RFC7228] introduces the notion of constrained-node networks, which are made of small devices with severe constraints on power, memory, and processing resources. The terms constrained devices and IoT devices are used interchangeably.

The terms "certification authority" (CA) and "distinguished name" (DN) are taken from [RFC5280]. The terms "trust anchor" and "trust anchor store" are defined in [RFC6024] as:

> A trust anchor represents an authoritative entity via a public key and associated data. The public key is used to verify digital signatures, and the associated data is used to constrain the types of information for which the trust anchor is authoritative.
>
> A trust anchor store is a set of one or more trust anchors stored in a device.... A device may have more than one trust anchor store, each of which may be used by one or more applications.

3. Overview

3.1. TLS and DTLS

The TLS protocol [RFC5246] provides authenticated, confidentiality- and integrity-protected communication between two endpoints. The protocol is composed of two layers: the Record Protocol and the handshaking protocols. At the lowest level, layered on top of a reliable transport protocol (e.g., TCP), is the Record Protocol. It provides connection security by using symmetric cryptography for confidentiality, data origin authentication, and integrity protection. The Record Protocol is used for encapsulation of various higher-level protocols. The handshaking protocols consist of three subprotocols -- namely, the handshake protocol, the change cipher spec protocol, and the alert protocol. The handshake protocol allows the server and client to authenticate each other and to negotiate an encryption algorithm and cryptographic keys before the application protocol transmits or receives data.

The design of DTLS [RFC6347] is intentionally very similar to TLS. However, since DTLS operates on top of an unreliable datagram transport, it must explicitly cope with the absence of reliable and ordered delivery assumptions made by TLS. RFC 6347 explains these differences in great detail. As a short summary, for those not familiar with DTLS, the differences are:

- An explicit sequence number and an epoch field is included in the Record Protocol. Section 4.1 of RFC 6347 explains the processing rules for these two new fields. The value used to compute the Message Authentication Code (MAC) is the 64-bit value formed by concatenating the epoch and the sequence number.

- Stream ciphers must not be used with DTLS. The only stream cipher defined for TLS 1.2 is RC4, and due to cryptographic weaknesses, it is not recommended anymore even for use with TLS [RFC7465]. Note that the term "stream cipher" is a technical term in the TLS specification. Section 4.7 of RFC 5246 defines stream ciphers in TLS as follows: "In stream cipher encryption, the plaintext is exclusive-ORed with an identical amount of output generated from a cryptographically secure keyed pseudorandom number generator."

- The TLS handshake protocol has been enhanced to include a stateless cookie exchange for Denial-of-Service (DoS) resistance. For this purpose, a new handshake message, the HelloVerifyRequest, was added to DTLS. This handshake message is sent by the server and includes a stateless cookie, which is returned in a ClientHello message back to the server. Although the exchange is optional for the server to execute, a client implementation has to be prepared to respond to it. Furthermore, the handshake message format has been extended to deal with message loss, reordering, and fragmentation.

3.2. Communication Models

This document describes a profile of DTLS and, to be useful, it has to make assumptions about the envisioned communication architecture.

Two communication architectures (and consequently two profiles) are described in this document.

3.2.1. Constrained TLS/DTLS Clients

The communication architecture shown in Figure 1 assumes a unicast communication interaction with an IoT device utilizing a constrained TLS/DTLS client interacting with one or multiple TLS/DTLS servers.

Before a client can initiate the TLS/DTLS handshake, it needs to know the IP address of that server and what credentials to use. Application-layer protocols, such as CoAP, which is conveyed on top of DTLS, may be configured with URIs of the endpoints to which CoAP needs to register and publish data. This configuration information (including non-confidential credentials, like certificates) may be conveyed to clients as part of a firmware/software package or via a configuration protocol. The following credential types are supported by this profile:

- For authentication based on the Pre-Shared Key (PSK) (see Section 4.2), this includes the paired "PSK identity" and shared secret to be used with each server.

- For authentication based on the raw public key (see Section 4.3), this includes either the server’s public key or the hash of the server’s public key.

- For certificate-based authentication (see Section 4.4), this includes a pre-populated trust anchor store that allows the DTLS stack to perform path validation for the certificate obtained during the handshake with the server.

Figure 1 shows example configuration information stored at the constrained client for use with respective servers.

This document focuses on the description of the DTLS client-side functionality but, quite naturally, the equivalent server-side support has to be available.

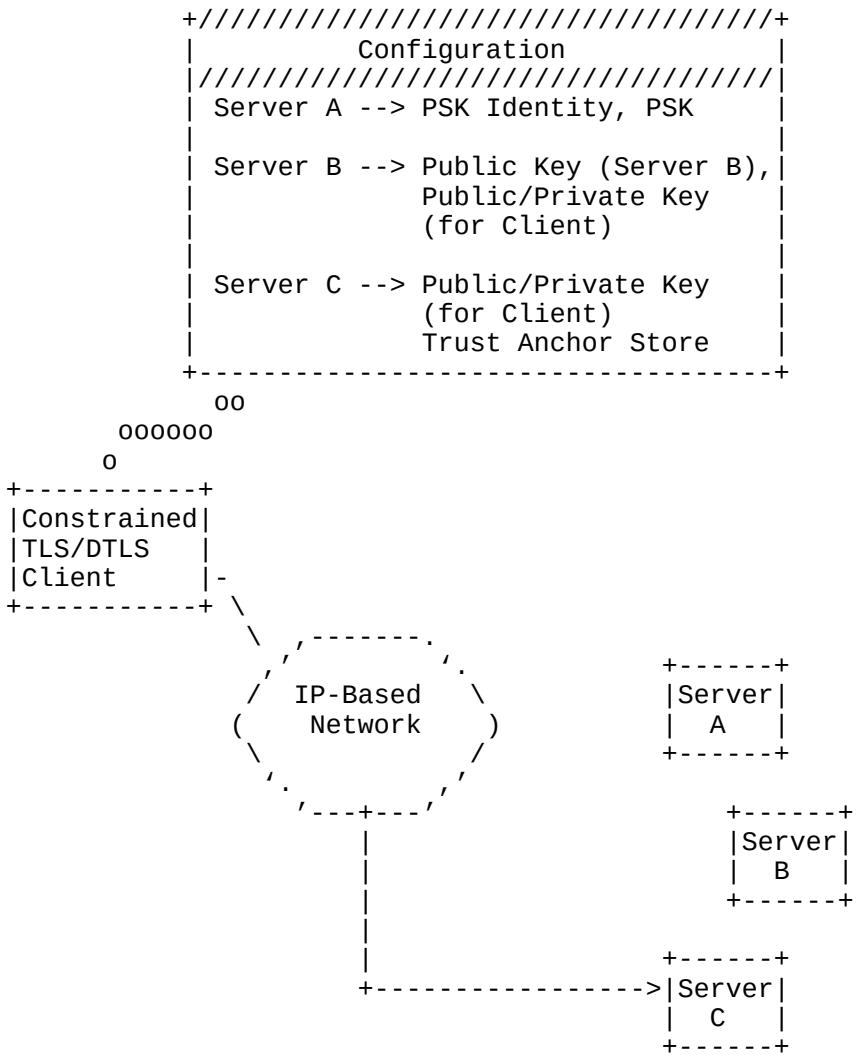


Figure 1: Constrained Client Profile

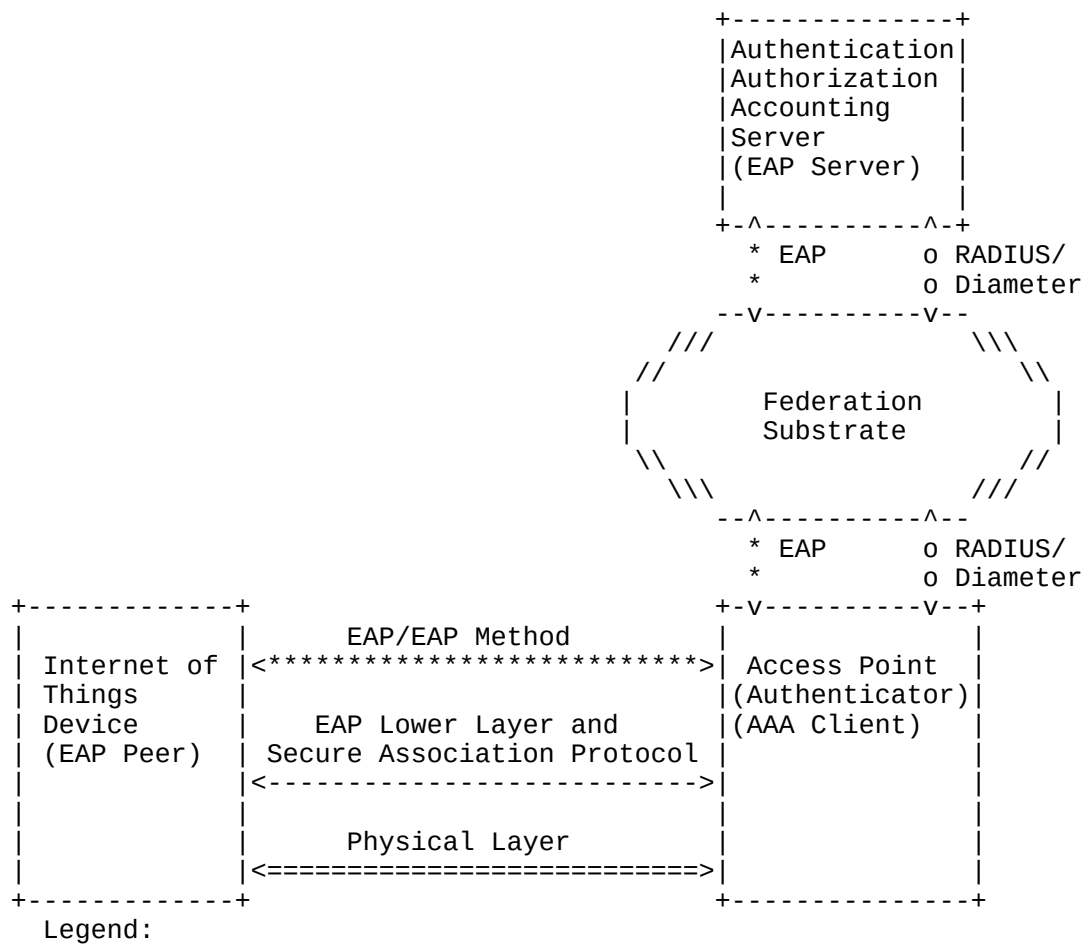
3.2.1.1. Examples of Constrained Client Exchanges

3.2.1.1.1. Network Access Authentication Example

Reuse is a recurring theme when considering constrained environments and is behind a lot of the directions taken in developments for constrained environments. The corollary of reuse is to not add functionality if it can be avoided. An example relevant to the use of TLS is network access authentication, which takes place when a device connects to a network and needs to go through an authentication and access control procedure before it is allowed to communicate with other devices or connect to the Internet.

Figure 2 shows the network access architecture with the IoT device initiating the communication to an access point in the network using the procedures defined for a specific physical layer. Since credentials may be managed and stored centrally, in the Authentication, Authorization, and Accounting (AAA) server, the security protocol exchange may need to be relayed via the Authenticator, i.e., functionality running on the access point to the AAA server. The authentication and key exchange protocol itself is encapsulated within a container, the Extensible Authentication Protocol (EAP) [RFC3748], and messages are conveyed back and forth between the EAP endpoints, namely the EAP peer located on the IoT device and the EAP server located on the AAA server or the access point. To route EAP messages from the access point, acting as a AAA client, to the AAA server requires an adequate protocol mechanism, namely RADIUS [RFC2865] or Diameter [RFC6733].

More details about the concepts and a description about the terminology can be found in RFC 5247 [RFC5247].



Legend:

<****>: Device-to-AAA-Server Exchange
<---->: Device-to-Authenticator Exchange
<oooo>: AAA-Client-to-AAA-Server Exchange
<====>: Physical layer like IEEE 802.11/802.15.4

Figure 2: Network Access Architecture

One standardized EAP method is EAP-TLS, defined in RFC 5216 [RFC5216], which reuses the TLS-based protocol exchange and encapsulates it inside the EAP payload. In terms of reuse, this allows many components of the TLS protocol to be shared between the network access security functionality and the TLS functionality needed for securing application-layer traffic. In the EAP-TLS exchange shown in Figure 3, the IoT device as the EAP peer acts as a TLS client.

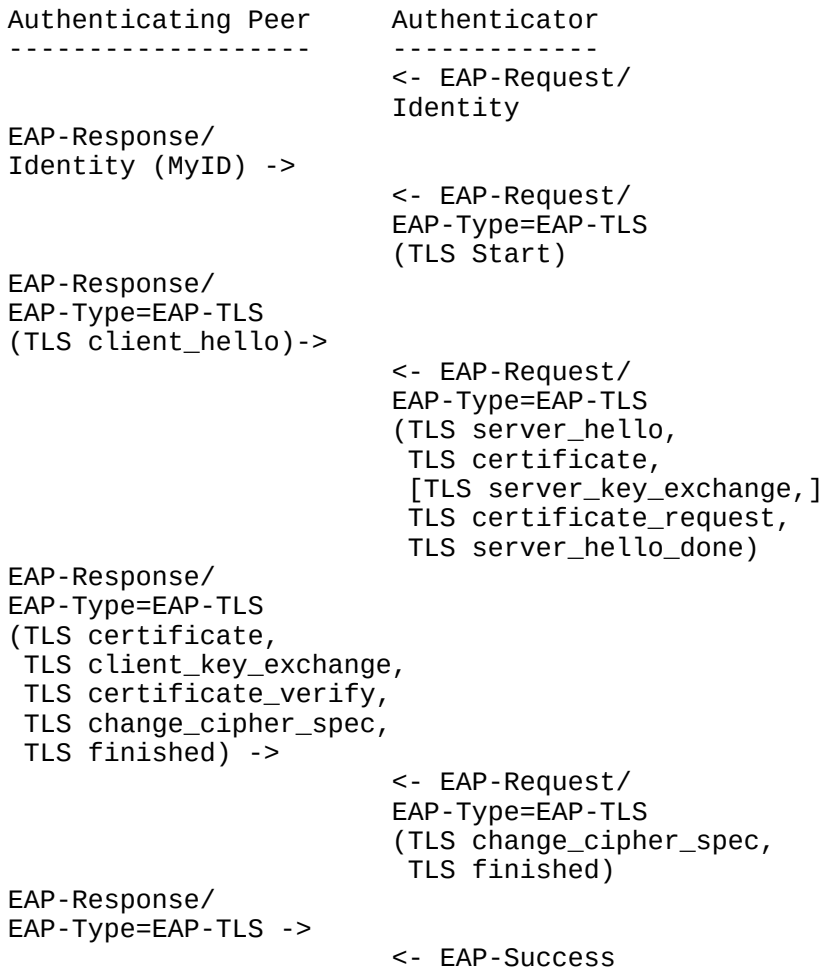


Figure 3: EAP-TLS Exchange

The guidance in this document also applies to the use of EAP-TLS for network access authentication. An IoT device using a network access authentication solution based on TLS can reuse most parts of the code for the use of DTLS/TLS at the application layer, thereby saving a significant amount of flash memory. Note, however, that the credentials used for network access authentication and those used for application-layer security are very likely different.

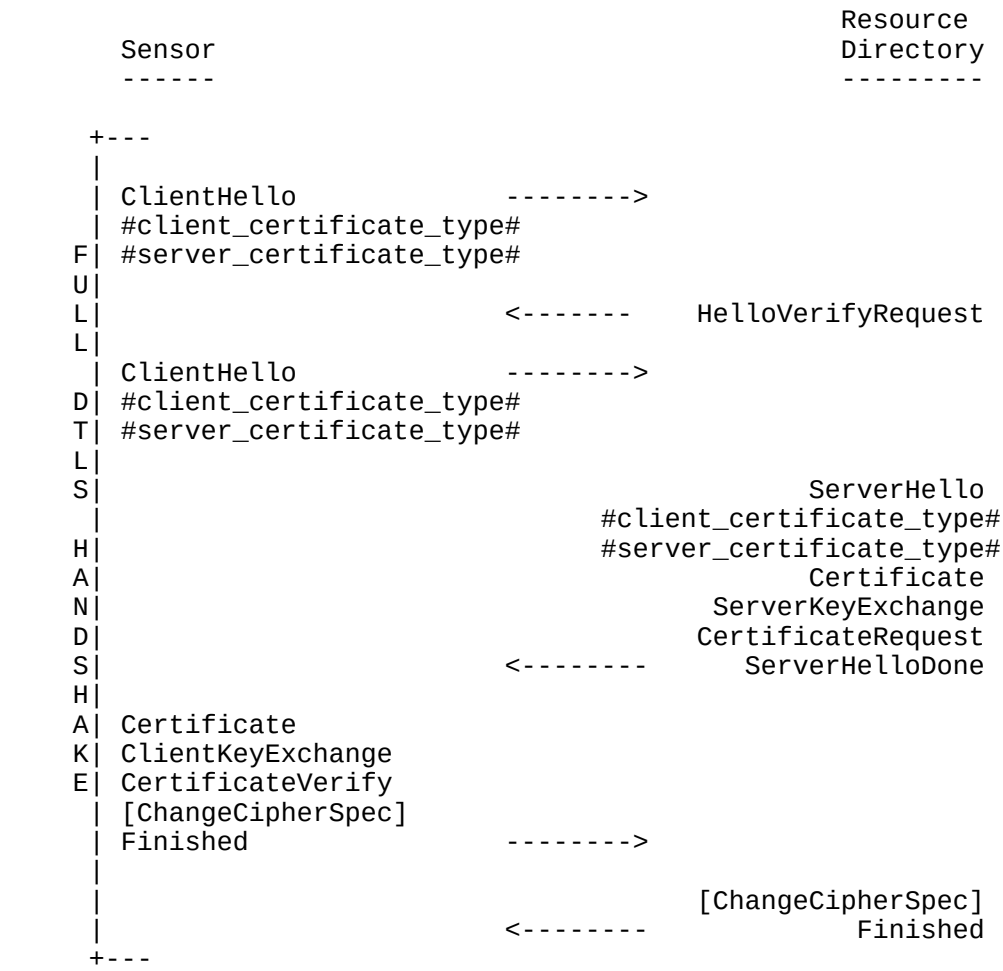
3.2.1.1.2. CoAP-Based Data Exchange Example

When a constrained client uploads sensor data to a server infrastructure, it may use CoAP by pushing the data via a POST message to a preconfigured endpoint on the server. In certain circumstances, this might be too limiting and additional functionality is needed, as shown in Figures 4 and 5, where the IoT device itself runs a CoAP server hosting the resource that is made accessible to other entities. Despite running a CoAP server on the IoT device, it is still the DTLS client on the IoT device that initiates the interaction with the non-constrained resource server in our scenario.

Figure 4 shows a sensor starting a DTLS exchange with a resource directory and uses CoAP to register available resources in Figure 5. [CoRE-RD] defines the resource directory (RD) as a web entity that stores information about web resources and implements Representational State Transfer (REST) interfaces for registration and lookup of those resources. Note that the described exchange is borrowed from the Open Mobile Alliance (OMA) Lightweight Machine-to-Machine (LWM2M) specification [LWM2M] that uses RD but adds proxy functionality.

The initial DTLS interaction between the sensor, acting as a DTLS client, and the resource directory, acting as a DTLS server, will be a full DTLS handshake. Once this handshake is complete, both parties have established the DTLS record layer. Subsequently, the CoAP client can securely register at the resource directory.

After some time (assuming that the client regularly refreshes its registration), the resource directory receives a request from an application to retrieve the temperature information from the sensor. This request is relayed by the resource directory to the sensor using a GET message exchange. The already established DTLS record layer can be used to secure the message exchange.



Note: Extensions marked with "#" were introduced with RFC 7250.

Figure 4: DTLS/CoAP Exchange Using Resource Directory: Part 1 -- DTLS Handshake

Figure 5 shows the DTLS-secured communication between the sensor and the resource directory using CoAP.

Figure 5: DTLS/CoAP Exchange Using Resource Directory: Part 2 -- CoAP/RD Exchange

Note that the CoAP GET message transmitted from the resource server is protected using the previously established DTLS record layer.

3.2.2. Constrained TLS/DTLS Servers

Section 3.2.1 illustrates a deployment model where the TLS/DTLS client is constrained and efforts need to be taken to improve memory utilization, bandwidth consumption, reduce performance impacts, etc. In this section, we assume a scenario where constrained devices run TLS/DTLS servers to secure access to application-layer services running on top of CoAP, HTTP, or other protocols. Figure 6 illustrates a possible deployment whereby a number of constrained servers are waiting for regular clients to access their resources. The entire process is likely, but not necessarily, controlled by a third party, the authentication and authorization server. This authentication and authorization server is responsible for holding authorization policies that govern the access to resources and distribution of keying material.

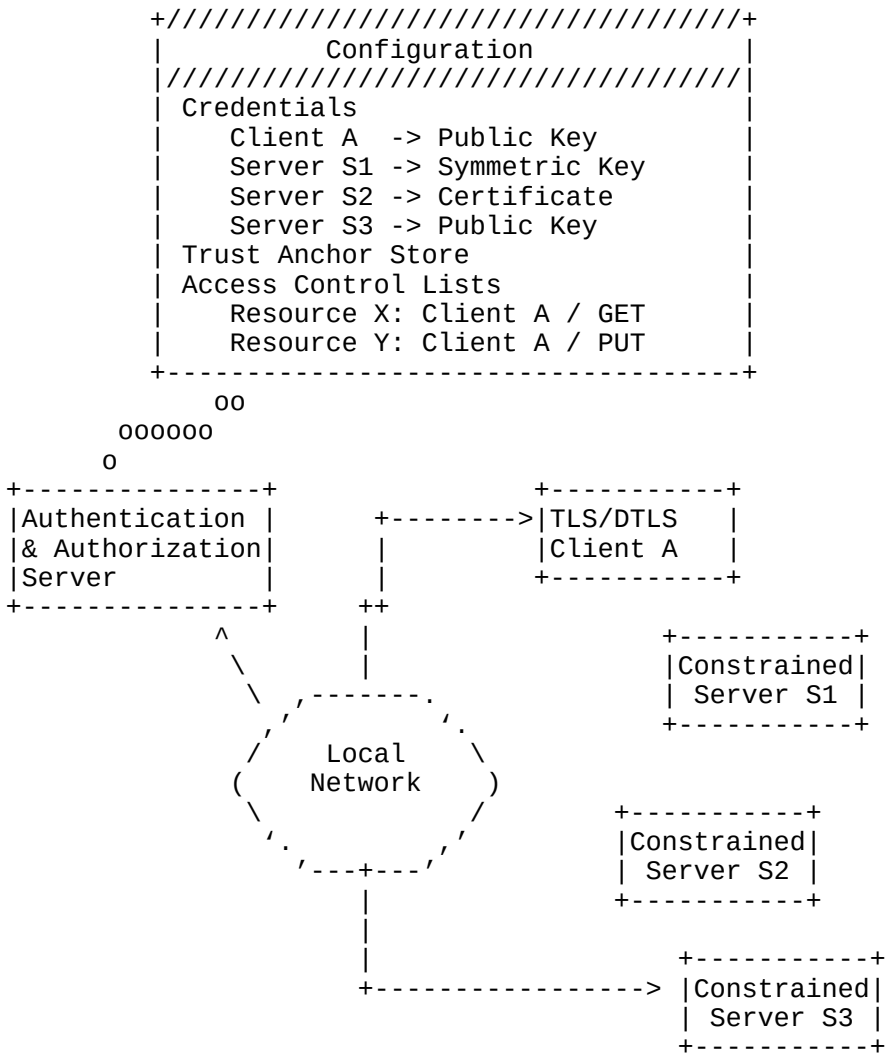


Figure 6: Constrained Server Profile

A deployment with constrained servers has to overcome several challenges. Below we explain how these challenges can be solved with CoAP, as an example. Other protocols may offer similar capabilities. While the requirements for the TLS/DTLS protocol profile change only slightly when run on a constrained server (in comparison to running it on a constrained client), several other ecosystem factors will impact deployment.

There are several challenges that need to be addressed:

Discovery and Reachability:

A client must first and foremost discover the server before initiating a connection to it. Once it has been discovered, reachability to the device needs to be maintained.

In CoAP, the discovery of resources offered by servers is accomplished by sending a unicast or multicast CoAP GET to a well-known URI. The Constrained RESTful Environments (CoRE) Link Format specification [RFC6690] describes the use case (see Section 1.2.1) and reserves the URI (see Section 7.1). Section 7 of the CoAP specification [RFC7252] describes the discovery procedure. [RFC7390] describes the use case for discovering CoAP servers using multicast (see Section 3.3) and specifies the protocol processing rules for CoAP group communications (see Section 2.7).

The use of RD [CoRE-RD] is yet another possibility for discovering registered servers and their resources. Since RD is usually not a proxy, clients can discover links registered with the RD and then access them directly.

Authentication:

The next challenge concerns the provisioning of authentication credentials to the clients as well as servers. In Section 3.2.1, we assume that credentials (and other configuration information) are provisioned to the device, and that those can be used with the authorization servers. Of course, this leads to a very static relationship between the clients and their server-side infrastructure but poses fewer challenges from a deployment point of view, as described in Section 2 of [RFC7452]. In any case, engineers and product designers have to determine how the relevant credentials are distributed to the respective parties. For example, shared secrets may need to be provisioned to clients and the constrained servers for subsequent use of TLS/DTLS PSK. In other deployments, certificates, private keys, and trust anchors for use with certificate-based authentication may need to be utilized.

Practical solutions use either pairing (also called imprinting) or a trusted third party. With pairing, two devices execute a special protocol exchange that is unauthenticated to establish a shared key (for example, using an unauthenticated Diffie-Hellman (DH) exchange). To avoid man-in-the-middle attacks, an out-of-band channel is used to verify that nobody has tampered

with the exchanged protocol messages. This out-of-band channel can come in many forms, including:

* Human involvement by comparing hashed keys, entering passkeys, and scanning QR codes

* The use of alternative wireless communication channels (e.g., infrared communication in addition to Wi-Fi)

* Proximity-based information

More details about these different pairing/imprinting techniques can be found in the Smart Object Security Workshop report [RFC7397] and various position papers submitted on that topic, such as [ImprintingSurvey]. The use of a trusted third party follows a different approach and is subject to ongoing standardization efforts in the Authentication and Authorization for Constrained Environments (ACE) working group [ACE-WG].

Authorization

The last challenge is the ability for the constrained server to make an authorization decision when clients access protected resources. Pre-provisioning access control information to constrained servers may be one option but works only in a small scale, less dynamic environment. For a finer-grained and more dynamic access control solution, the reader is referred to the ongoing work in the IETF ACE working group.

Figure 7 shows an example interaction whereby a device, a thermostat in our case, searches in the local network for discoverable resources and accesses those. The thermostat starts the procedure using a link-local discovery message using the "All CoAP Nodes" multicast address by utilizing the link format per RFC 6690 [RFC6690]. The IPv6 multicast address used for CoAP link-local discovery is FF02::FD. As a result, a temperature sensor and a fan respond. These responses allow the thermostat to subsequently read temperature information from the temperature sensor with a CoAP GET request issued to the previously learned endpoint. In this example we assume that accessing the temperature sensor readings and controlling the fan requires authentication and authorization of the thermostat and TLS is used to authenticate both endpoints and to secure the communication.

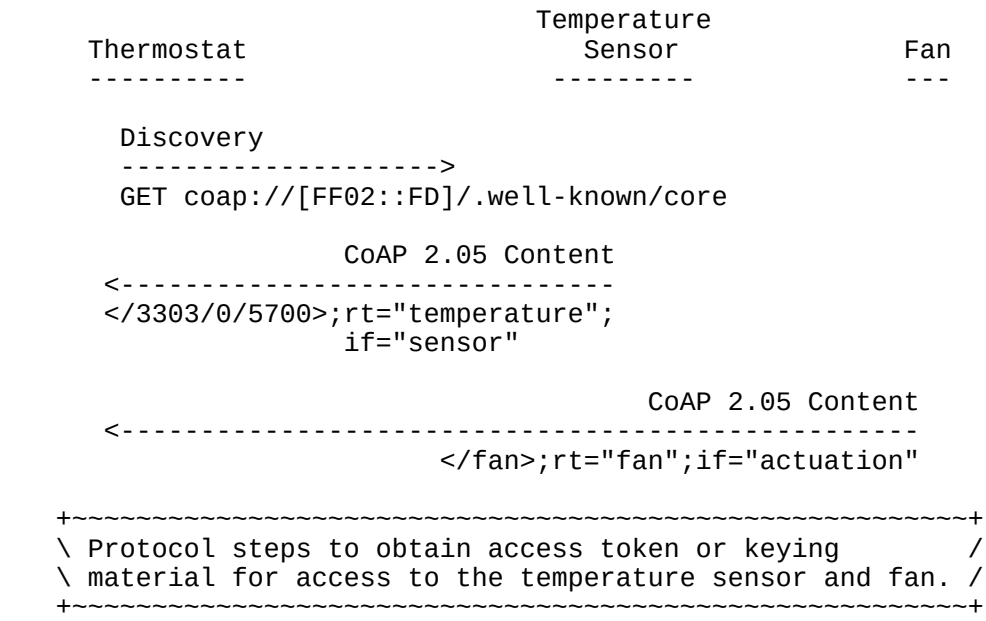


Figure 7: Local Discovery and Resource Access

3.3. The Ciphersuite Concept

TLS (and consequently DTLS) support ciphersuites, and an IANA registry [IANA-TLS] was created to register the suites. A ciphersuite (and the specification that defines it) contains the following information:

- Authentication and key exchange algorithm (e.g., PSK)
- Cipher and key length (e.g., Advanced Encryption Standard (AES) with 128-bit keys [AES])
- Mode of operation (e.g., Counter with CBC-MAC (CCM) mode for AES) [RFC3610]
- Hash algorithm for integrity protection, such as the Secure Hash Algorithm (SHA) in combination with Keyed-Hashing for Message Authentication (HMAC) (see [RFC2104] and [RFC6234])
- Hash algorithm for use with pseudorandom functions (e.g., HMAC with the SHA-256)
- Misc information (e.g., length of authentication tags)
- Information whether the ciphersuite is suitable for DTLS or only for TLS

The TLS ciphersuite TLS_PSK_WITH_AES_128_CCM_8, for example, uses a pre-shared authentication and key exchange algorithm. [RFC6655] defines this ciphersuite. It uses the AES encryption algorithm, which is a block cipher. Since the AES algorithm supports different key lengths (such as 128, 192, and 256 bits), this information has to be specified as well, and the selected ciphersuite supports 128-bit keys. A block cipher encrypts plaintext in fixed-size blocks, and AES operates on a block size of 128 bits. For messages exceeding 128 bits, the message is partitioned into 128-bit blocks, and the AES cipher is applied to these input blocks with appropriate chaining, which is called mode of operation.

TLS 1.2 introduced Authenticated Encryption with Associated Data (AEAD) ciphersuites (see [RFC5116] and [RFC6655]). AEAD is a class of block cipher modes that encrypt (parts of) the message and authenticate the message simultaneously. AES-CCM [RFC6655] is an example of such a mode.

Some AEAD ciphersuites have shorter authentication tags (i.e., message authentication codes) and are therefore more suitable for networks with low bandwidth where small message size matters. The

TLS_PSK_WITH_AES_128_CCM_8 ciphersuite that ends in "_8" has an 8-octet authentication tag, while the regular CCM ciphersuites have, at the time of writing, 16-octet authentication tags. The design of CCM and the security properties are described in [CCM].

TLS 1.2 also replaced the combination of MD5/SHA-1 hash functions in the TLS pseudorandom function (PRF) used in earlier versions of TLS with ciphersuite-specified PRFs. For this reason, authors of more recent TLS 1.2 ciphersuite specifications explicitly indicate the MAC algorithm and the hash functions used with the TLS PRF.

4. Credential Types

The mandatory-to-implement functionality will depend on the credential type used with IoT devices. The subsections below describe the implications of three different credential types, namely pre-shared secrets, raw public keys, and certificates.

4.1. Preconditions

All exchanges described in the subsequent sections assume that some information has been distributed before the TLS/DTLS interaction starts. The credentials are used to authenticate the client to the server, and vice versa. What information items have to be distributed depends on the chosen credential types. In all cases, the IoT device needs to know what algorithms to prefer, particularly if there are multiple algorithm choices available as part of the implemented ciphersuites, as well as information about the other communication endpoint (for example, in the form of a URI) a particular credential has to be used with.

Pre-Shared Secrets: In this case, a shared secret together with an identifier needs to be made available to the device as well as to the other communication party.

Raw Public Keys: A public key together with a private key are stored on the device and typically associated with some identifier. To authenticate the other communication party, the appropriate credential has to be known. If the other end uses raw public keys as well, then their public key needs to be provisioned (out of band) to the device.

Certificates: The use of certificates requires the device to store the public key (as part of the certificate) as well as the private key. The certificate will contain the identifier of the device as well as various other attributes. Both communication parties are assumed to be in possession of a trust anchor store that contains CA certificates and, in case of certificate pinning, end-entity

certificates. Similar to the other credentials, the IoT device needs information about which entity to use which certificate with. Without a trust anchor store on the IoT device, it will not be possible to perform certificate validation.

We call the above-listed information "device credentials" and these device credentials may be provisioned to the device already during the manufacturing time or later in the process, depending on the envisioned business and deployment model. These initial credentials are often called "root of trust". Whatever process is chosen for generating these initial device credentials, it MUST be ensured that a different key pair is provisioned for each device and installed in as secure a manner as possible. For example, it is preferable to generate public/private keys on the IoT device itself rather than generating them outside the device. Since an IoT device is likely to interact with various other parties, the initial device credential may only be used with some dedicated entities, and configuring further configuration and credentials to the device is left to a separate interaction. An example of a dedicated protocol used to distribute credentials, access control lists, and configure information is the LWM2M protocol [LWM2M].

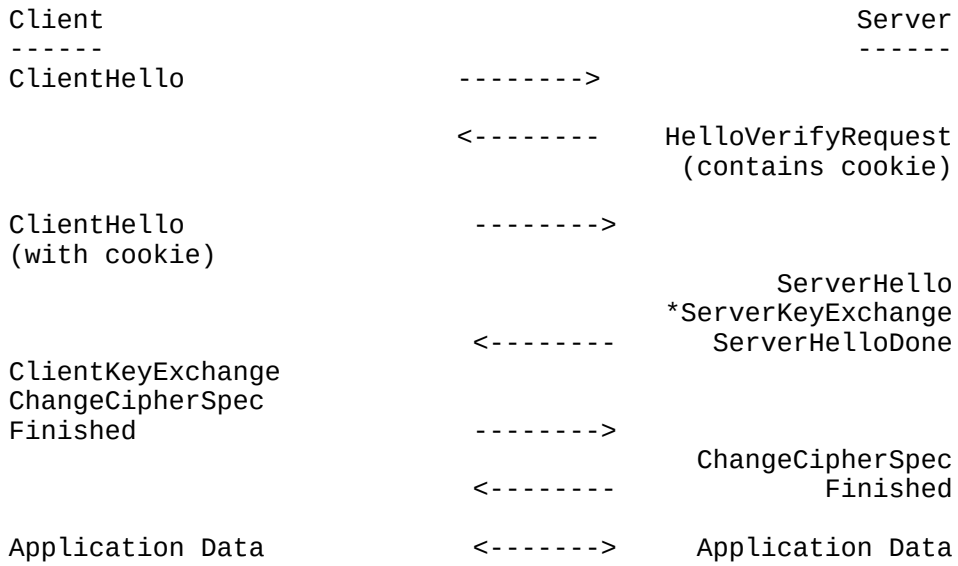
For all the credentials listed above, there is a chance that those may need to be replaced or deleted. While separate protocols have been developed to check the status of these credentials and to manage these credentials, such as the Trust Anchor Management Protocol (TAMP) [RFC5934], their usage is, however, not envisioned in the IoT context so far. IoT devices are assumed to have a software update mechanism built-in, and it will allow updates of low-level device information, including credentials and configuration parameters. This document does, however, not mandate a specific software/firmware update protocol.

With all credentials used as input to TLS/DTLS authentication, it is important that these credentials have been generated with care. When using a pre-shared secret, a critical consideration is using sufficient entropy during the key generation, as discussed in [RFC4086]. Deriving a shared secret from a password, some device identifiers, or other low-entropy sources is not secure. A low-entropy secret, or password, is subject to dictionary attacks. Attention also has to be paid when generating public/private key pairs since the lack of randomness can result in the same key pair being used in many devices. This topic is also discussed in Section 12 since keys are generated during the TLS/DTLS exchange itself as well, and the same considerations apply.

4.2. Pre-Shared Secret

The use of pre-shared secrets is one of the most basic techniques for TLS/DTLS since it is both computationally efficient and bandwidth conserving. Authentication based on pre-shared secrets was introduced to TLS in RFC 4279 [RFC4279].

Figure 8 illustrates the DTLS exchange including the cookie exchange. While the server is not required to initiate a cookie exchange with every handshake, the client is required to implement and to react on it when challenged, as defined in RFC 6347 [RFC6347]. The cookie exchange allows the server to react to flooding attacks.



Legend:

* indicates an optional message payload

Figure 8: DTLS PSK Authentication Including the Cookie Exchange

Note that [RFC4279] used the term "PSK identity" to refer to the identifier used to refer to the appropriate secret. While "identifier" would be more appropriate in this context, we reuse the terminology defined in RFC 4279 to avoid confusion. RFC 4279 does not mandate the use of any particular type of PSK identity, and the client and server have to agree on the identities and keys to be used. The UTF-8 encoding of identities described in Section 5.1 of RFC 4279 aims to improve interoperability for those cases where the identity is configured by a human using some management interface

provided by a web browser. However, many IoT devices do not have a user interface, and most of their credentials are bound to the device rather than to the user. Furthermore, credentials are often provisioned into hardware modules or provisioned alongside with firmware. As such, the encoding considerations are not applicable to this usage environment. For use with this profile, the PSK identities SHOULD NOT assume a structured format (such as domain names, distinguished names, or IP addresses), and a byte-by-byte comparison operation MUST be used by the server for any operation related to the PSK identity. These types of identifiers are called "absolute" per RFC 6943 [RFC6943].

Protocol-wise, the client indicates which key it uses by including a "PSK identity" in the ClientKeyExchange message. As described in Section 3.2, clients may have multiple pre-shared keys with a single server, for example, in a hosting context. The TLS Server Name Indication (SNI) extension allows the client to convey the name of the server it is contacting. A server implementation needs to guide the selection based on a received SNI value from the client.

RFC 4279 requires TLS implementations supporting PSK ciphersuites to support arbitrary PSK identities up to 128 octets in length and arbitrary PSKs up to 64 octets in length. This is a useful assumption for TLS stacks used in the desktop and mobile environments where management interfaces are used to provision identities and keys. Implementations in compliance with this profile MAY use PSK identities up to 128 octets in length and arbitrary PSKs up to 64 octets in length. The use of shorter PSK identities is RECOMMENDED.

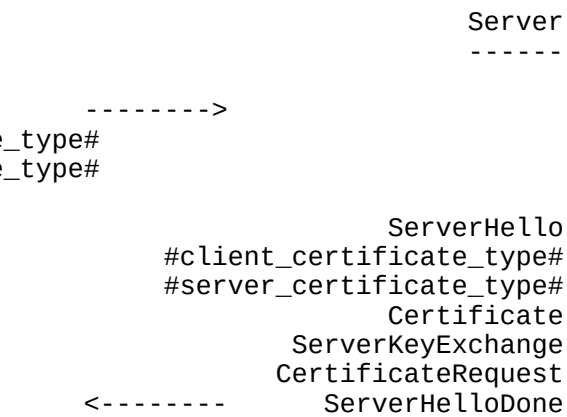
"The Constrained Application Protocol (CoAP)" [RFC7252] currently specifies TLS_PSK_WITH_AES_128_CCM_8 as the mandatory-to-implement ciphersuite for use with shared secrets. This ciphersuite uses the AES algorithm with 128 bit keys and CCM as the mode of operation. The label "_8" indicates that an 8-octet authentication tag is used. Note that the shorted authentication tag increases the chance that an adversary with no knowledge of the secret key can present a message with a MAC that will pass the verification procedure. The likelihood of accepting forged data is explained in Section 5.3.5 of [SP800-107-rev1] and depends on the lengths of the authentication tag and allowed numbers of MAC verifications using a given key.

This ciphersuite makes use of the default TLS 1.2 PRF, which uses an HMAC with the SHA-256 hash function. Note: Starting with TLS 1.2 (and consequently DTLS 1.2), ciphersuites have to specify the PRF. RFC 5246 states that "New cipher suites MUST explicitly specify a PRF and, in general, SHOULD use the TLS PRF with SHA-256 or a stronger standard hash function." The ciphersuites recommended in this document use the SHA-256 construct defined in Section 5 of RFC 5246.

A device compliant with the profile in this section MUST implement TLS_PSK_WITH_AES_128_CCM_8 and follow the guidance from this section.

4.3. Raw Public Key

The use of raw public keys with TLS/DTLS, as defined in [RFC7250], is the first entry point into public key cryptography without having to pay the price of certificates and a public key infrastructure (PKI). The specification reuses the existing Certificate message to convey the raw public key encoded in the SubjectPublicKeyInfo structure. To indicate support, two new extensions had been defined, as shown in Figure 9, namely the server_certificate_type and the client_certificate_type.



Note: Extensions marked with "#" were introduced with RFC 7250.

Figure 9: DTLS Raw Public Key Exchange

The CoAP-recommended ciphersuite for use with this credential type is TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. This AES-CCM TLS ciphersuite based on elliptic curve cryptography (ECC) uses the Ephemeral Elliptic Curve Diffie-Hellman (ECDHE) as the key establishment mechanism and an Elliptic Curve Digital Signature

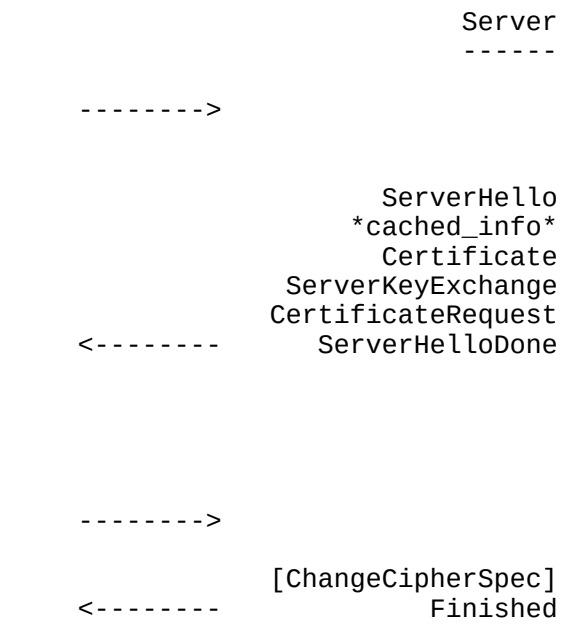
Algorithm (ECDSA) for authentication. The named DH groups [FFDHE-TLS] are not applicable to this profile since it relies on the ECC-based counterparts. This ciphersuite makes use of the AEAD capability in DTLS 1.2 and utilizes an 8-octet authentication tag. The use of a DH key exchange provides perfect forward secrecy (PFS). More details about PFS can be found in Section 9.

[RFC6090] provides valuable information for implementing ECC algorithms, particularly for choosing methods that have been available in the literature for a long time (i.e., 20 years and more).

A device compliant with the profile in this section MUST implement TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 and follow the guidance from this section.

4.4. Certificates

The use of mutual certificate-based authentication is shown in Figure 10, which makes use of the "cached_info" extension [RFC7924]. Support of the "cached_info" extension is REQUIRED. Caching certificate chains allows the client to reduce the communication overhead significantly, otherwise the server would provide the end-entity certificate and the certificate chain with every full DTLS handshake.



Note: Extensions marked with "*" were introduced with RFC 7924.

Figure 10: DTLS Mutual Certificate-Based Authentication

TLS/DTLS offers a lot of choices when selecting ECC-based ciphersuites. This document restricts the use to named curves defined in RFC 4492 [RFC4492]. At the time of writing, the recommended curve is secp256r1, and the use of uncompressed points follows the recommendation in CoAP. Note that standardization for Curve25519 (for ECDHE) is ongoing (see [RFC7748]), and support for this curve will likely be required in the future.

A device compliant with the profile in this section MUST implement TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 and follow the guidance from this section.

4.4.1. Certificates Used by Servers

The algorithm for verifying the service identity, as described in RFC 6125 [RFC6125], is essential for ensuring proper security when certificates are used. As a summary, the algorithm contains the following steps:

1. The client constructs a list of acceptable reference identifiers based on the source domain and, optionally, the type of service to which the client is connecting.

2. The server provides its identifiers in the form of a PKIX certificate.

3. The client checks each of its reference identifiers against the presented identifiers for the purpose of finding a match.

4. When checking a reference identifier against a presented identifier, the client matches the source domain of the identifiers and, optionally, their application service type.

For various terms used in the algorithm shown above, consult RFC 6125. It is important to highlight that comparing the reference identifier against the presented identifier obtained from the certificate is required to ensure the client is communicating with the intended server.

It is worth noting that the algorithm description and the text in RFC 6125 assumes that fully qualified DNS domain names are used. If a server node is provisioned with a fully qualified DNS domain, then the server certificate MUST contain the fully qualified DNS domain name or "FQDN" as dNSName [RFC5280]. For CoAP, the coaps URI scheme is described in Section 6.2 of [RFC7252]. This FQDN is stored in the SubjectAltName or in the leftmost Common Name (CN) component of the subject name, as explained in Section 9.1.3.3 of [RFC7252], and used by the client to match it against the FQDN used during the lookup process, as described in [RFC6125]. For other protocols, the appropriate URI scheme specification has to be consulted.

The following recommendation is provided:

1. Certificates MUST NOT use DNS domain names in the CN of certificates and instead use the subjectAltName attribute, as described in the previous paragraph.

2. Certificates MUST NOT contain domain names with wildcard characters.

3. Certificates MUST NOT contain multiple names (e.g., more than one dNSName field).

Note that there will be servers that are not provisioned for use with DNS domain names, for example, IoT devices that offer resources to nearby devices in a local area network, as shown in Figure 7. When such constrained servers are used, then the use of certificates as described in Section 4.4.2 is applicable. Note that the SNI extension cannot be used in this case since SNI does not offer the ability to convey a 64-bit Extended Unique Identifier (EUI-64) [EUI64]. Note that this document does not recommend use of IP addresses in certificates nor does it discuss the implications of placing IP addresses in certificates.

4.4.2. Certificates Used by Clients

For client certificates, the identifier used in the SubjectAltName or in the leftmost CN component of subject name MUST be an EUI-64.

4.4.3. Certificate Revocation

For certificate revocation, neither the Online Certificate Status Protocol (OCSP) nor Certificate Revocation Lists (CRLs) are used. Instead, this profile relies on a software update mechanism to provision information about revoked certificates. While multiple OCSP stapling [RFC6961] has recently been introduced as a mechanism to piggyback OCSP request/responses inside the DTLS/TLS handshake (to avoid the cost of a separate protocol handshake), further investigations are needed to determine its suitability for the IoT environment.

As stated earlier in this section, modifications to the trust anchor store depends on a software update mechanism as well. There are limitations to the use of a software update mechanism because of the potential inability to change certain types of keys, such as those provisioned during manufacturing. For this reason, manufacturer-provisioned credentials are typically employed only to obtain further certificates (for example, via a key distribution server) for use with servers the IoT device is finally communicating with.

4.4.4. Certificate Content

All certificate elements listed in Table 1 MUST be implemented by clients and servers claiming support for certificate-based authentication. No other certificate elements are used by this specification.

When using certificates, IoT devices MUST provide support for a server certificate chain of at least 3, not including the trust anchor, and MAY reject connections from servers offering chains longer than 3. IoT devices MAY have client certificate chains of any length. Obviously, longer chains require more digital signature verification operations to perform and lead to larger certificate messages in the TLS handshake.

Table 1 provides a summary of the elements in a certificate for use with this profile.

Element	Notes
version	This profile uses X.509 v3 certificates [RFC5280].
serialNumber	Positive integer unique per certificate.
signature	This field contains the signature algorithm, and this profile uses ecdsa-with-SHA256 or stronger [RFC5758].
issuer	Contains the DN of the issuing CA.
validity	Values expressed as UTC time in notBefore and notAfter fields. No validity period mandated.
subject	See rules outlined in this section.
subjectPublicKeyInfo	The SubjectPublicKeyInfo structure indicates the algorithm and any associated parameters for the ECC public key. This profile uses the id-ecPublicKey algorithm identifier for ECDSA signature keys, as defined and specified in [RFC5480].
signatureAlgorithm	The ECDSA signature algorithm with ecdsa-with-SHA256 or stronger.
signatureValue	Bit string containing the digital signature.

Extension: subjectAltName	See rules outlined in this section.
Extension: BasicConstraints	Indicates whether the subject of the certificate is a CA and the maximum depth of valid certification paths that include this certificate. This extension is used for CA certs only, and then the value of the "cA" field is set to TRUE. The default is FALSE.
Extension: Key Usage	The KeyUsage field MAY have the following values in the context of this profile: digitalSignature or keyAgreement, keyCertSign for verifying signatures on public key certificates.
Extension: Extended Key Usage	The ExtKeyUsageSyntax field MAY have the following values in context of this profile: id-kp-serverAuth for server authentication, id-kp-clientAuth for client authentication, id-kp-codeSigning for code signing (for software update mechanism), and id-kp-OCSPSigning for future OCSP usage in TLS.

Table 1: Certificate Content

There are various cryptographic algorithms available to sign digital certificates; those algorithms include RSA, the Digital Signature Algorithm (DSA), and ECDSA. As Table 1 shows, certificates are signed using ECDSA in this profile. This is not only true for the end-entity certificates but also for all other certificates in the chain, including CA certificates. This profiling reduces the amount of flash memory needed on an IoT device to store the code of several algorithm implementations due to the smaller number of options.

Further details about X.509 certificates can be found in Section 9.1.3.3 of [RFC7252].

4.4.5. Client Certificate URLs

RFC 6066 [RFC6066] allows the sending of client-side certificates to be avoided and uses URLs instead. This reduces the over-the-air transmission. Note that the TLS "cached_info" extension does not provide any help with caching client certificates.

TLS/DTLS clients MUST implement support for client certificate URLs for those environments where client-side certificates are used and the server-side is not constrained. For constrained servers this functionality is NOT RECOMMENDED since it forces the server to execute an additional protocol exchange, potentially using a protocol it does not even support. The use of this extension also increases the risk of a DoS attack against the constrained server due to the additional workload.

4.4.6. Trusted CA Indication

RFC 6066 [RFC6066] allows clients to indicate what trust anchor they support. With certificate-based authentication, a DTLS server conveys its end-entity certificate to the client during the DTLS handshake. Since the server does not necessarily know what trust anchors the client has stored, to facilitate certification path construction and validation, it includes intermediate CA certs in the certificate payload.

Today, in most IoT deployments there is a fairly static relationship between the IoT device (and the software running on them) and the server-side infrastructure. For these deployments where IoT devices interact with a fixed, preconfigured set of servers, this extension is NOT RECOMMENDED.

In cases where clients interact with dynamically discovered TLS/DTLS servers, for example, in the use cases described in Section 3.2.2, the use of this extension is RECOMMENDED.

5. Signature Algorithm Extension

The "signature_algorithms" extension, defined in Section 7.4.1.4.1 of RFC 5246 [RFC5246], allows the client to indicate to the server which signature/hash algorithm pairs may be used in digital signatures. The client MUST send this extension to select the use of SHA-256, otherwise if this extension is absent, RFC 5246 defaults to SHA-1 / ECDSA for the ECDH_ECDSA and the ECDHE_ECDSA key exchange algorithms.

The "signature_algorithms" extension is not applicable to the PSK-based ciphersuite described in Section 4.2.

6. Error Handling

TLS/DTLS uses the alert protocol to convey errors and specifies a long list of error types. However, not all error messages defined in the TLS/DTLS specification are applicable to this profile. In general, there are two categories of errors (as defined in Section 7.2 of RFC 5246), namely fatal errors and warnings. Alert

messages with a level of "fatal" result in the immediate termination of the connection. If possible, developers should try to develop strategies to react to those fatal errors, such as restarting the handshake or informing the user using the (often limited) user interface. Warnings may be ignored by the application since many IoT devices will have either limited ways to log errors or no ability at all. In any case, implementers have to carefully evaluate the impact of errors and ways to remedy the situation since a commonly used approach for delegating decision making to users is difficult (or impossible) to accomplish in a timely fashion.

All error messages marked as RESERVED are only supported for backwards compatibility with the Secure Socket Layer (SSL) and MUST NOT be used with this profile. Those include decryption_failed_RESERVED, no_certificate_RESERVED, and export_restriction_RESERVED.

A number of the error messages MUST only be used for certificate-based ciphersuites. Hence, the following error messages MUST NOT be used with PSK and raw public key authentication:

- bad_certificate,
- unsupported_certificate,
- certificate_revoked,
- certificate_expired,
- certificate_unknown,
- unknown_ca, and
- access_denied.

Since this profile does not make use of compression at the TLS layer, the decompression_failure error message MUST NOT be used either.

RFC 4279 introduced the new alert message "unknown_psk_identity" for PSK ciphersuites. As stated in Section 2 of RFC 4279, the decrypt_error error message may also be used instead. For this profile, the TLS server MUST return the decrypt_error error message instead of the unknown_psk_identity since the two mechanisms exist and provide the same functionality.

Furthermore, the following errors should not occur with devices and servers supporting this specification, but implementations MUST be prepared to process these errors to deal with servers that are not compliant to the profiles in this document:

protocol_version: While this document focuses only on one version of the TLS/DTLS protocol, namely version 1.2, ongoing work on TLS/DTLS 1.3 is in progress at the time of writing.

insufficient_security: This error message indicates that the server requires ciphers to be more secure. This document specifies only one ciphersuite per profile, but it is likely that additional ciphersuites will get added over time.

user_canceled: Many IoT devices are unattended and hence this error message is unlikely to occur.

7. Session Resumption

Session resumption is a feature of the core TLS/DTLS specifications that allows a client to continue with an earlier established session state. The resulting exchange is shown in Figure 11. In addition, the server may choose not to do a cookie exchange when a session is resumed. Still, clients have to be prepared to do a cookie exchange with every handshake. The cookie exchange is not shown in the figure.

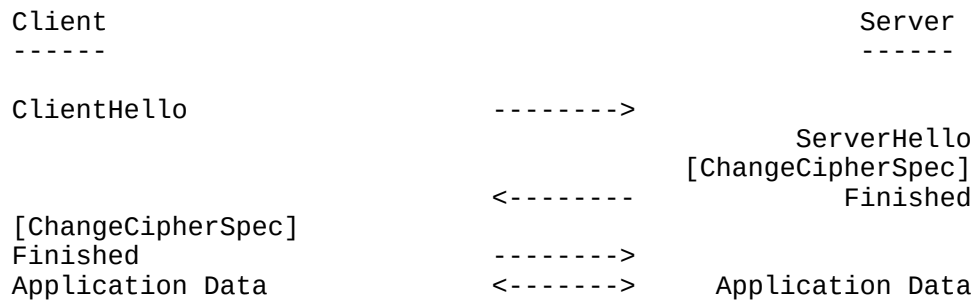


Figure 11: DTLS Session Resumption

Constrained clients MUST implement session resumption to improve the performance of the handshake. This will lead to a reduced number of message exchanges, lower computational overhead (since only symmetric cryptography is used during a session resumption exchange), and session resumption requires less bandwidth.

For cases where the server is constrained (but not the client), the client MUST implement RFC 5077 [RFC5077]. Note that the constrained

server refers to a device that has limitations in terms of RAM and flash memory, which place restrictions on the amount of TLS/DTLS security state information that can be stored on such a device. RFC 5077 specifies a version of TLS/DTLS session resumption that does not require per-session state information to be maintained by the constrained server. This is accomplished by using a ticket-based approach.

If both the client and the server are constrained devices, both devices SHOULD implement RFC 5077 and MUST implement basic session resumption. Clients that do not want to use session resumption are always able to send a ClientHello message with an empty session_id to revert to a full handshake.

8. Compression

Section 3.3 of [RFC7525] recommends disabling TLS-/DTLS-level compression due to attacks, such as CRIME [CRIME]. For IoT applications, compression at the TLS/DTLS layer is not needed since application-layer protocols are highly optimized, and the compression algorithms at the DTLS layer increases code size and complexity.

TLS/DTLS layer compression is NOT RECOMMENDED by this TLS/DTLS profile.

9. Perfect Forward Secrecy

PFS is a property that preserves the confidentiality of past protocol interactions even in situations where the long-term secret is compromised.

The PSK ciphersuite recommended in Section 4.2 does not offer this property since it does not utilize a DH exchange. New ciphersuites that support PFS for PSK-based authentication, such as proposed in [PSK-AES-CCM-TLS], might become available as a standardized ciphersuite in the (near) future. The recommended PSK-based ciphersuite offers excellent performance, a very small memory footprint, and has the lowest on the wire overhead at the expense of not using any public cryptography. For deployments where public key cryptography is acceptable, the use of raw public keys might offer a middle ground between the PSK ciphersuite in terms of out-of-band validation and the functionality offered by asymmetric cryptography.

Physical attacks create additional opportunities to gain access to the crypto material stored on IoT devices. A PFS ciphersuite prevents an attacker from obtaining the communication content exchanged prior to a successful long-term key compromise; however, an implementation that (for performance or energy efficiency reasons)

has been reusing the same ephemeral DH keys over multiple different sessions partially defeats PFS, thus increasing the damage extent. For this reason, implementations SHOULD NOT reuse ephemeral DH keys over multiple protocol exchanges.

The impact of the disclosure of past communication interactions and the desire to increase the cost for pervasive monitoring (as demanded by [RFC7258]) has to be taken into account when selecting a ciphersuite that does not support the PFS property.

Client implementations claiming support of this profile MUST implement the ciphersuites listed in Section 4 according to the selected credential type.

10. Keep-Alive

Application-layer communication may create state at the endpoints, and this state may expire at some time. For this reason, applications define ways to refresh state, if necessary. While the application-layer exchanges are largely outside the scope of the underlying TLS/DTLS exchange, similar state considerations also play a role at the level of TLS/DTLS. While TLS/DTLS also creates state in the form of a security context (see the security parameter described in Appendix A.6 in RFC 5246) at the client and the server, this state information does not expire. However, network intermediaries may also allocate state and require this state to be kept alive. Failure to keep state alive at a stateful packet filtering firewall or at a NAT may result in the inability for one node to reach the other since packets will get blocked by these middleboxes. Periodic keep-alive messages exchanged between the TLS/DTLS client and server keep state at these middleboxes alive. According to measurements described in [HomeGateway], there is some variance in state management practices used in residential gateways, but the timeouts are heavily impacted by the choice of the transport-layer protocol: timeouts for UDP are typically much shorter than those for TCP.

RFC 6520 [RFC6520] defines a heartbeat mechanism to test whether the other peer is still alive. As an additional feature, the same mechanism can also be used to perform Path Maximum Transmission Unit (MTU) Discovery.

A recommendation about the use of RFC 6520 depends on the type of message exchange an IoT device performs and the number of messages the application needs to exchange as part of their application functionality. There are three types of exchanges that need to be analyzed:

Client-Initiated, One-Shot Messages

This is a common communication pattern where IoT devices upload data to a server on the Internet on an irregular basis. The communication may be triggered by specific events, such as opening a door.

The DTLS handshake may need to be restarted (ideally using session resumption, if possible) in case of an IP address change.

In this case, there is no use for a keep-alive extension for this scenario.

Client-Initiated, Regular Data Uploads

This is a variation of the previous case whereby data gets uploaded on a regular basis, for example, based on frequent temperature readings. If neither NAT bindings nor IP address changes occurred, then the record layer will not notice any changes. For the case where the IP address and port number changes, it is necessary to recreate the record layer using session resumption.

In this scenario, there is no use for a keep-alive extension. It is also very likely that the device will enter a sleep cycle in between data transmissions to keep power consumption low.

Server-Initiated Messages

In the two previous scenarios, the client initiates the protocol interaction and maintains it. Since messages to the client may get blocked by middleboxes, the initial connection setup is triggered by the client and then kept alive by the server.

For this message exchange pattern, the use of DTLS heartbeat messages is quite useful but may have to be coordinated with application exchanges (for example, when the CoAP resource directory is used) to avoid redundant keep-alive message exchanges. The MTU discovery mechanism, which is also part of [RFC6520], is less likely to be relevant since for many IoT deployments, the most constrained link is the wireless interface between the IoT device and the network itself (rather than some links along the end-to-end path). Only in more complex network topologies, such as multi-hop mesh networks, path MTU discovery might be appropriate. It also has to be noted that DTLS itself already provides a basic path discovery mechanism (see Section 4.1.1.1 of RFC 6347) by using the fragmentation capability of the handshake protocol.

For server-initiated messages, the heartbeat extension is RECOMMENDED.

11. Timeouts

A variety of wired and wireless technologies are available to connect devices to the Internet. Many of the low-power radio technologies, such as IEEE 802.15.4 or Bluetooth Smart, only support small frame sizes (e.g., 127 bytes in case of IEEE 802.15.4 as explained in [RFC4919]). Other radio technologies, such as the Global System for Mobile Communications (GSM) using the short messaging service (SMS), have similar constraints in terms of payload sizes, such as 140 bytes without the optional segmentation and reassembly scheme known as "Concatenated SMS", but show higher latency.

The DTLS handshake protocol adds a fragmentation and reassembly mechanism to the TLS handshake protocol since each DTLS record must fit within a single transport layer datagram, as described in Section 4.2.3 of [RFC6347]. Since handshake messages are potentially bigger than the maximum record size, the mechanism fragments a handshake message over a number of DTLS records, each of which can be transmitted separately.

To deal with the unreliable message delivery provided by UDP, DTLS adds timeouts and "per-flight" retransmissions, as described in Section 4.2.4 of [RFC6347]. Although the timeout values are implementation specific, recommendations are provided in Section 4.2.4.1 of [RFC6347], with an initial timer value of 1 second and double the value at each retransmission, up to no less than 60 seconds.

TLS protocol steps can take longer due to higher processing time on the constrained side. On the other hand, the way DTLS handles retransmission, which is per-flight instead of per-segment, tends to interact poorly with low-bandwidth networks.

For these reasons, it’s essential that the probability of a spurious retransmit is minimized and, on timeout, the sending endpoint does not react too aggressively. The latter is particularly relevant when the Wireless Sensor Network (WSN) is temporarily congested: if lost packets are reinjected too quickly, congestion worsens.

An initial timer value of 9 seconds with exponential back off up to no less then 60 seconds is therefore RECOMMENDED.

This value is chosen big enough to absorb large latency variance due to either slow computation on constrained endpoints or intrinsic network characteristics (e.g., GSM-SMS), as well as to produce a low

number of retransmission events and relax the pacing between them. Its worst case wait time is the same as using 1s timeout (i.e., 63s), while triggering less than half of the retransmissions (2 instead of 5).

In order to minimize the wake time during DTLS handshake, sleepy nodes might decide to select a lower threshold and, consequently, a smaller initial timeout value. If this is the case, the implementation MUST keep into account the considerations about network stability described in this section.

12. Random Number Generation

The TLS/DTLS protocol requires random numbers to be available during the protocol run. For example, during the ClientHello and the ServerHello exchange, the client and the server exchange random numbers. Also, the use of the DH exchange requires random numbers during the key pair generation.

It is important to note that sources contributing to the randomness pool on laptops or desktop PCs are not available on many IoT devices, such as mouse movement, timing of keystrokes, air turbulence on the movement of hard drive heads, etc. Other sources have to be found or dedicated hardware has to be added.

Lacking sources of randomness in an embedded system may lead to the same keys generated again and again.

The ClientHello and the ServerHello messages contain the "Random" structure, which has two components: gmt_unix_time and a sequence of 28 random bytes. gmt_unix_time holds the current time and date in standard UNIX 32-bit format (seconds since the midnight starting Jan 1, 1970, GMT). Since many IoT devices do not have access to an accurate clock, it is RECOMMENDED that the receiver of a ClientHello or ServerHello does not assume that the value in "Random.gmt_unix_time" is an accurate representation of the current time and instead treats it as an opaque random string.

When TLS is used with certificate-based authentication, the availability of time information is needed to check the validity of a certificate. Higher-layer protocols may provide secure time information. The gmt_unix_time component of the ServerHello is not used for this purpose.

IoT devices using TLS/DTLS must offer ways to generate quality random numbers. There are various implementation choices for integrating a hardware-based random number generator into a product: an implementation inside the microcontroller itself is one option, but

dedicated crypto chips are also reasonable choices. The best choice will depend on various factors outside the scope of this document. Guidelines and requirements for random number generation can be found in RFC 4086 [RFC4086] and in the NIST Special Publication 800-90a [SP800-90A].

Chip manufacturers are highly encouraged to provide sufficient documentation of their design for random number generators so that customers can have confidence about the quality of the generated random numbers. The confidence can be increased by providing information about the procedures that have been used to verify the randomness of numbers generated by the hardware modules. For example, NIST Special Publication 800-22b [SP800-22b] describes statistical tests that can be used to verify random number generators.

13. Truncated MAC and Encrypt-then-MAC Extension

The truncated MAC extension was introduced in RFC 6066 [RFC6066] with the goal to reduce the size of the MAC used at the record layer. This extension was developed for TLS ciphersuites that used older modes of operation where the MAC and the encryption operation were performed independently.

The recommended ciphersuites in this document use the newer AEAD construct, namely the CCM mode with 8-octet authentication tags, and are therefore not applicable to the truncated MAC extension.

RFC 7366 [RFC7366] introduced the encrypt-then-MAC extension (instead of the previously used MAC-then-encrypt) since the MAC-then-encrypt mechanism has been the subject of a number of security vulnerabilities. RFC 7366 is, however, also not applicable to the AEAD ciphers recommended in this document.

Implementations conformant to this specification MUST use AEAD ciphers. RFC 7366 ("encrypt-then-MAC") and RFC 6066 ("truncated MAC extension") are not applicable to this specification and MUST NOT be used.

14. Server Name Indication (SNI)

The SNI extension [RFC6066] defines a mechanism for a client to tell a TLS/DTLS server the name of the server it wants to contact. This is a useful extension for many hosting environments where multiple virtual servers are run on a single IP address.

Implementing the Server Name Indication extension is REQUIRED unless it is known that a TLS/DTLS client does not interact with a server in a hosting environment.

15. Maximum Fragment Length Negotiation

This RFC 6066 extension lowers the maximum fragment length support needed for the record layer from 2^14 bytes to 2^9 bytes.

This is a very useful extension that allows the client to indicate to the server how much maximum memory buffers it uses for incoming messages. Ultimately, the main benefit of this extension is to allow client implementations to lower their RAM requirements since the client does not need to accept packets of large size (such as 16K packets as required by plain TLS/DTLS).

Client implementations MUST support this extension.

16. Session Hash

In order to begin connection protection, the Record Protocol requires specification of a suite of algorithms, a master secret, and the client and server random values. The algorithm for computing the master secret is defined in Section 8.1 of RFC 5246, but it only includes a small number of parameters exchanged during the handshake and does not include parameters like the client and server identities. This can be utilized by an attacker to mount a man-in-the-middle attack since the master secret is not guaranteed to be unique across sessions, as discovered in the "triple handshake" attack [Triple-HS].

[RFC7627] defines a TLS extension that binds the master secret to a log of the full handshake that computes it, thus preventing such attacks.

Client implementations SHOULD implement this extension even though the ciphersuites recommended by this profile are not vulnerable to this attack. For DH-based ciphersuites, the keying material is contributed by both parties and in case of the pre-shared secret key ciphersuite, both parties need to be in possession of the shared secret to ensure that the handshake completes successfully. It is, however, possible that some application-layer protocols will tunnel other authentication protocols on top of DTLS making this attack relevant again.

17. Renegotiation Attacks

TLS/DTLS allows a client and a server that already have a TLS/DTLS connection to negotiate new parameters, generate new keys, etc., by using the renegotiation feature. Renegotiation happens in the existing connection, with the new handshake packets being encrypted along with application data. Upon completion of the renegotiation procedure, the new channel replaces the old channel.

As described in RFC 5746 [RFC5746], there is no cryptographic binding between the two handshakes, although the new handshake is carried out using the cryptographic parameters established by the original handshake.

To prevent the renegotiation attack [RFC5746], this specification REQUIRES the TLS renegotiation feature to be disabled. Clients MUST respond to server-initiated renegotiation attempts with an alert message (no_renegotiation), and clients MUST NOT initiate them.

18. Downgrading Attacks

When a client sends a ClientHello with a version higher than the highest version known to the server, the server is supposed to reply with ServerHello.version equal to the highest version known to the server, and then the handshake can proceed. This behavior is known as version tolerance. Version intolerance is when the server (or a middlebox) breaks the handshake when it sees a ClientHello.version higher than what it knows about. This is the behavior that leads some clients to rerun the handshake with a lower version. As a result, a potential security vulnerability is introduced when a system is running an old TLS/SSL version (e.g., because of the need to integrate with legacy systems). In the worst case, this allows an attacker to downgrade the protocol handshake to SSL 3.0. SSL 3.0 is so broken that there is no secure cipher available for it (see [RFC7568]).

The above-described downgrade vulnerability is solved by the TLS Fallback Signaling Cipher Suite Value (SCSV) [RFC7507] extension. However, the solution is not applicable to implementations conforming to this profile since the version negotiation MUST use TLS/DTLS version 1.2 (or higher). More specifically, this implies:

- Clients MUST NOT send a TLS/DTLS version lower than version 1.2 in the ClientHello.
- Clients MUST NOT retry a failed negotiation offering a TLS/DTLS version lower than 1.2.

- Servers MUST fail the handshake by sending a protocol_version fatal alert if a TLS/DTLS version >= 1.2 cannot be negotiated. Note that the aborted connection is non-resumable.

19. Crypto Agility

This document recommends that software and chip manufacturers implement AES and the CCM mode of operation. This document references the CoAP-recommended ciphersuite choices, which have been selected based on implementation and deployment experience from the IoT community. Over time, the preference for algorithms will, however, change. Not all components of a ciphersuite are likely to change at the same speed. Changes are more likely expected for ciphers, the mode of operation, and the hash algorithms. The recommended key lengths have to be adjusted over time as well. Some deployment environments will also be impacted by local regulation, which might dictate a certain algorithm and key size combination. Ongoing discussions regarding the choice of specific ECC curves will also likely impact implementations. Note that this document does not recommend or mandate a specific ECC curve.

The following recommendations can be made to chip manufacturers:

- Make any AES hardware-based crypto implementation accessible to developers working on security implementations at higher layers in the protocol stack. Sometimes hardware implementations are added to microcontrollers to offer support for functionality needed at the link layer and are only available to the on-chip link-layer protocol implementation. Such a setup does not allow application developers to reuse the hardware-based AES implementation.

- Provide flexibility for the use of the crypto function with future extensibility in mind. For example, making an AES-CCM implementation available to developers is a first step but such an implementation may not be usable due to parameter differences between an AES-CCM implementation. AES-CCM in IEEE 802.15.4 and Bluetooth Smart use a nonce length of 13 octets while DTLS uses a nonce length of 12 octets. Hardware implementations of AES-CCM for IEEE 802.15.4 and Bluetooth Smart are therefore not reusable by a DTLS stack.

- Offer access to building blocks in addition (or as an alternative) to the complete functionality. For example, a chip manufacturer who gives developers access to the AES crypto function can use it to build an efficient AES-GCM implementation. Another example is to make a special instruction available that increases the speed of speed-up carryless multiplications.

As a recommendation for developers and product architects, we suggest that sufficient headroom is provided to allow an upgrade to a newer cryptographic algorithm over the lifetime of the product. As an example, while AES-CCM is recommended throughout this specification, future products might use the ChaCha20 cipher in combination with the Poly1305 authenticator [RFC7539]. The assumption is made that a robust software update mechanism is offered.

20. Key Length Recommendations

RFC 4492 [RFC4492] gives approximate comparable key sizes for symmetric- and asymmetric-key cryptosystems based on the best-known algorithms for attacking them. While other publications suggest slightly different numbers, such as [Keylength], the approximate relationship still holds true. Figure 12 illustrates the comparable key sizes in bits.

Symmetric	ECC	DH/DSA/RSA
80	163	1024
112	233	2048
128	283	3072
192	409	7680
256	571	15360

Figure 12: Comparable Key Sizes (in Bits) Based on RFC 4492

At the time of writing, the key size recommendations for use with TLS-based ciphers found in [RFC7525] recommend DH key lengths of at least 2048 bits, which corresponds to a 112-bit symmetric key and a 233-bit ECC key. These recommendations are roughly in line with those from other organizations, such as the National Institute of Standards and Technology (NIST) or the European Network and Information Security Agency (ENISA). The authors of [ENISA-Report2013] add that a 80-bit symmetric key is sufficient for legacy applications for the coming years, but a 128-bit symmetric key is the minimum requirement for new systems being deployed. The authors further note that one needs to also take into account the length of time data needs to be kept secure for. The use of 80-bit symmetric keys for transactional data may be acceptable for the near future while one has to insist on 128-bit symmetric keys for long-lived data.

Note that the recommendations for 112-bit symmetric keys are chosen conservatively under the assumption that IoT devices have a long expected lifetime (such as 10+ years) and that this key length recommendation refers to the long-term keys used for device authentication. Keys, which are provisioned dynamically, for the

protection of transactional data (such as ephemeral DH keys used in various TLS/DTLS ciphersuites) may be shorter considering the sensitivity of the exchanged data.

21. False Start

A full TLS handshake as specified in [RFC5246] requires two full protocol rounds (four flights) before the handshake is complete and the protocol parties may begin to send application data.

An abbreviated handshake (resuming an earlier TLS session) is complete after three flights, thus adding just one round-trip time if the client sends application data first.

If the conditions outlined in [TLS-FALSESTART] are met, application data can be transmitted when the sender has sent its own "ChangeCipherSpec" and "Finished" messages. This achieves an improvement of one round-trip time for full handshakes if the client sends application data first and for abbreviated handshakes if the server sends application data first.

The conditions for using the TLS False Start mechanism are met by the public-key-based ciphersuites in this document. In summary, the conditions are:

- Modern symmetric ciphers with an effective key length of 128 bits, such as AES-128-CCM
- Client certificate types, such as ecdsa_sign
- Key exchange methods, such as ECDHE_ECDSA

Based on the improvement over a full round-trip for the full TLS/DTLS exchange, this specification RECOMMENDS the use of the False Start mechanism when clients send application data first.

22. Privacy Considerations

The DTLS handshake exchange conveys various identifiers, which can be observed by an on-path eavesdropper. For example, the DTLS PSK exchange reveals the PSK identity, the supported extensions, the session ID, algorithm parameters, etc. When session resumption is used, then individual TLS sessions can be correlated by an on-path adversary. With many IoT deployments, it is likely that keying material and their identifiers are persistent over a longer period of time due to the cost of updating software on these devices.

User participation poses a challenge in many IoT deployments since many of the IoT devices operate unattended, even though they are initially provisioned by a human. The ability to control data sharing and to configure preferences will have to be provided at a system level rather than at the level of the DTLS exchange itself, which is the scope of this document. Quite naturally, the use of DTLS with mutual authentication will allow a TLS server to collect authentication information about the IoT device (likely over a long period of time). While this strong form of authentication will prevent misattribution, it also allows strong identification. Device-related data collection (e.g., sensor recordings) associated with other data types will prove to be truly useful, but this extra data might include personal information about the owner of the device or data about the environment it senses. Consequently, the data stored on the server side will be vulnerable to stored data compromise. For the communication between the client and the server, this specification prevents eavesdroppers from gaining access to the communication content. While the PSK-based ciphersuite does not provide PFS, the asymmetric versions do. This prevents an adversary from obtaining past communication content when access to a long-term secret has been gained. Note that no extra effort to make traffic analysis more difficult is provided by the recommendations made in this document.

Note that the absence or presence of communication itself might reveal information to an adversary. For example, a presence sensor may initiate messaging when a person enters a building. While TLS/DTLS would offer confidentiality protection of the transmitted information, it does not help to conceal all communication patterns. Furthermore, the IP header, which is not protected by TLS/DTLS, additionally reveals information about the other communication endpoint. For applications where such privacy concerns exist, additional safeguards are required, such as injecting dummy traffic and onion routing. A detailed treatment of such solutions is outside the scope of this document and requires a system-level view.

23. Security Considerations

This entire document is about security.

We would also like to point out that designing a software update mechanism into an IoT system is crucial to ensure that both functionality can be enhanced and that potential vulnerabilities can be fixed. This software update mechanism is important for changing configuration information, for example, trust anchors and other keying-related information. Such a suitable software update mechanism is available with the LWM2M protocol published by the OMA [LWM2M].

24. References

24.1. Normative References

[EUI64] IEEE, "Guidelines for 64-bit Global Identifier (EUI-64)", Registration Authority, <https://standards.ieee.org/regauth/oui/tutorials/EUI64.html>.

[GSM-SMS] ETSI, "3rd Generation Partnership Project; Technical Specification Group Core Network and Terminals; Technical realization of the Short Message Service (SMS) (Release 13)", 3GPP TS 23.040 V13.1.0, March 2016.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC4279] Eronen, P., Ed. and H. Tschofenig, Ed., "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, DOI 10.17487/RFC4279, December 2005, <http://www.rfc-editor.org/info/rfc4279>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, DOI 10.17487/RFC5746, February 2010, <http://www.rfc-editor.org/info/rfc5746>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <http://www.rfc-editor.org/info/rfc6066>.

[RFC6125] Saint-Andre, P. and J. Hodges, "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)", RFC 6125, DOI 10.17487/RFC6125, March 2011, <http://www.rfc-editor.org/info/rfc6125>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) Heartbeat Extension", RFC 6520, DOI 10.17487/RFC6520, February 2012, <http://www.rfc-editor.org/info/rfc6520>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <http://www.rfc-editor.org/info/rfc7250>.

[RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-CCM Elliptic Curve Cryptography (ECC) Cipher Suites for TLS", RFC 7251, DOI 10.17487/RFC7251, June 2014, <http://www.rfc-editor.org/info/rfc7251>.

[RFC7627] Bhargavan, K., Ed., Delignat-Lavaud, A., Pironti, A., Langley, A., and M. Ray, "Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension", RFC 7627, DOI 10.17487/RFC7627, September 2015, <http://www.rfc-editor.org/info/rfc7627>.

[RFC7924] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", RFC 7924, DOI 10.17487/RFC7924, July 2016, <http://www.rfc-editor.org/info/rfc7924>.

[WAP-WDP] Open Mobile Alliance, "Wireless Datagram Protocol", Wireless Application Protocol, WAP-259-WDP, June 2001.

24.2. Informative References

[ACE-WG] IETF, "Authentication and Authorization for Constrained Environments (ACE) Working Group", <https://datatracker.ietf.org/wg/ace/charter>.

[AES] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", NIST FIPS PUB 197, November 2001, <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.

[CCM] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation: The CCM Mode for Authentication and Confidentiality", NIST Special Publication 800-38C, May 2004, <http://csrc.nist.gov/publications/nistpubs/800-38C/SP800-38C_updated-July20_2007.pdf>.

[COAP-TCP-TLS] Bormann, C., Lemay, S., Tschofenig, H., Hartke, K., Silverajan, B., and B. Raymor, "CoAP (Constrained Application Protocol) over TCP, TLS, and WebSockets", Work in Progress, draft-ietf-core-coap-tcp-tls-03, July 2016.

[CoRE-RD] Shelby, Z., Koster, M., Bormann, C., and P. Stok, "CoRE Resource Directory", Work in Progress, draft-ietf-core-resource-directory-08, July 2016.

[CRIME] Wikipedia, "CRIME", May 2016, <https://en.wikipedia.org/w/index.php?title=CRIME&oldid=721665716>.

[ENISA-Report2013] ENISA, "Algorithms, Key Sizes and Parameters Report - 2013", October 2013, <https://www.enisa.europa.eu/activities/identity-and-trust/library/deliverables/algorithms-key-sizes-and-parameters-report>.

[FFDHE-TLS] Gillmor, D., "Negotiated Finite Field Diffie-Hellman Ephemeral Parameters for TLS", Work in Progress, draft-ietf-tls-negotiated-ff-dhe-10, June 2015.

[HomeGateway] Eggert, L., Hatoen, S., Kojo, M., Nyrhinen, A., Sarolahti, P., and S. Strowes, "An Experimental Study of Home Gateway Characteristics", In Proceedings of the 10th ACM SIGCOMM conference on Internet measurement, DOI 10.1145/1879141.1879174, 2010, <http://conferences.sigcomm.org/imc/2010/papers/p260.pdf>.

[IANA-TLS] IANA, "Transport Layer Security (TLS) Parameters", <https://www.iana.org/assignments/tls-parameters>.

[ImprintingSurvey]
Chilton, E., "A Brief Survey of Imprinting Options for Constrained Devices", March 2012, <http://www.lix.polytechnique.fr/hipercom/SmartObjectSecurity/papers/EricRescorla.pdf>.

[Keylength]
Giry, D., "Cryptographic Key Length Recommendations", September 2015, <http://www.keylength.com>.

[LWM2M] Open Mobile Alliance, "Lightweight Machine-to-Machine Requirements", Candidate Version 1.0, OMA-RD-LightweightM2M-V1_0-20131210-C, December 2013, <http://openmobilealliance.org/about-oma/work-program/m2m-enablers>.

[PSK-AES-CCM-TLS]
Schmertmann, L. and C. Bormann, "ECDHE-PSK AES-CCM Cipher Suites with Forward Secrecy for Transport Layer Security (TLS)", Work in Progress, draft-schmertmann-dice-ccm-psk-pfs-01, August 2014.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, DOI 10.17487/RFC1981, August 1996, <http://www.rfc-editor.org/info/rfc1981>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <http://www.rfc-editor.org/info/rfc2104>.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, DOI 10.17487/RFC2865, June 2000, <http://www.rfc-editor.org/info/rfc2865>.

[RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3610, DOI 10.17487/RFC3610, September 2003, <http://www.rfc-editor.org/info/rfc3610>.

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <http://www.rfc-editor.org/info/rfc3748>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4492] Blake-Wilson, S., Bolyard, N., Gupta, V., Hawk, C., and B. Moeller, "Elliptic Curve Cryptography (ECC) Cipher Suites for Transport Layer Security (TLS)", RFC 4492, DOI 10.17487/RFC4492, May 2006, <http://www.rfc-editor.org/info/rfc4492>.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, DOI 10.17487/RFC4821, March 2007, <http://www.rfc-editor.org/info/rfc4821>.

[RFC4919] Kushalnagar, N., Montenegro, G., and C. Schumacher, "IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs): Overview, Assumptions, Problem Statement, and Goals", RFC 4919, DOI 10.17487/RFC4919, August 2007, <http://www.rfc-editor.org/info/rfc4919>.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <http://www.rfc-editor.org/info/rfc5077>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/RFC5116, January 2008, <http://www.rfc-editor.org/info/rfc5116>.

[RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <http://www.rfc-editor.org/info/rfc5216>.

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, DOI 10.17487/RFC5247, August 2008, <http://www.rfc-editor.org/info/rfc5247>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <http://www.rfc-editor.org/info/rfc5280>.

[RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois Counter Mode (GCM) Cipher Suites for TLS", RFC 5288, DOI 10.17487/RFC5288, August 2008, <http://www.rfc-editor.org/info/rfc5288>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <http://www.rfc-editor.org/info/rfc5480>.

[RFC5758] Dang, Q., Santesson, S., Moriarty, K., Brown, D., and T. Polk, "Internet X.509 Public Key Infrastructure: Additional Algorithms and Identifiers for DSA and ECDSA", RFC 5758, DOI 10.17487/RFC5758, January 2010, <http://www.rfc-editor.org/info/rfc5758>.

[RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor Management Protocol (TAMP)", RFC 5934, DOI 10.17487/RFC5934, August 2010, <http://www.rfc-editor.org/info/rfc5934>.

[RFC6024] Reddy, R. and C. Wallace, "Trust Anchor Management Requirements", RFC 6024, DOI 10.17487/RFC6024, October 2010, <http://www.rfc-editor.org/info/rfc6024>.

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, DOI 10.17487/RFC6090, February 2011, <http://www.rfc-editor.org/info/rfc6090>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <http://www.rfc-editor.org/info/rfc6234>.

[RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for Transport Layer Security (TLS)", RFC 6655, DOI 10.17487/RFC6655, July 2012, <http://www.rfc-editor.org/info/rfc6655>.

[RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link Format", RFC 6690, DOI 10.17487/RFC6690, August 2012, <http://www.rfc-editor.org/info/rfc6690>.

[RFC6733] Fajardo, V., Ed., Arkko, J., Loughney, J., and G. Zorn, Ed., "Diameter Base Protocol", RFC 6733, DOI 10.17487/RFC6733, October 2012, <http://www.rfc-editor.org/info/rfc6733>.

[RFC6943] Thaler, D., Ed., "Issues in Identifier Comparison for Security Purposes", RFC 6943, DOI 10.17487/RFC6943, May 2013, <http://www.rfc-editor.org/info/rfc6943>.

[RFC6961] Pettersen, Y., "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", RFC 6961, DOI 10.17487/RFC6961, June 2013, <http://www.rfc-editor.org/info/rfc6961>.

[RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for Constrained-Node Networks", RFC 7228, DOI 10.17487/RFC7228, May 2014, <http://www.rfc-editor.org/info/rfc7228>.

[RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an Attack", BCP 188, RFC 7258, DOI 10.17487/RFC7258, May 2014, <http://www.rfc-editor.org/info/rfc7258>.

[RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7366, DOI 10.17487/RFC7366, September 2014, <http://www.rfc-editor.org/info/rfc7366>.

[RFC7390] Rahman, A., Ed. and E. Dijk, Ed., "Group Communication for the Constrained Application Protocol (CoAP)", RFC 7390, DOI 10.17487/RFC7390, October 2014, <http://www.rfc-editor.org/info/rfc7390>.

[RFC7397] Gilger, J. and H. Tschofenig, "Report from the Smart Object Security Workshop", RFC 7397, DOI 10.17487/RFC7397, December 2014, <http://www.rfc-editor.org/info/rfc7397>.

[RFC7400] Bormann, C., "6LoWPAN-GHC: Generic Header Compression for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs)", RFC 7400, DOI 10.17487/RFC7400, November 2014, <http://www.rfc-editor.org/info/rfc7400>.

[RFC7452] Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson, "Architectural Considerations in Smart Object Networking", RFC 7452, DOI 10.17487/RFC7452, March 2015, <http://www.rfc-editor.org/info/rfc7452>.

[RFC7465] Popov, A., "Prohibiting RC4 Cipher Suites", RFC 7465, DOI 10.17487/RFC7465, February 2015, <http://www.rfc-editor.org/info/rfc7465>.

[RFC7507] Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher Suite Value (SCSV) for Preventing Protocol Downgrade Attacks", RFC 7507, DOI 10.17487/RFC7507, April 2015, <http://www.rfc-editor.org/info/rfc7507>.

[RFC7525] Sheffer, Y., Holz, R., and P. Saint-Andre, "Recommendations for Secure Use of Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", BCP 195, RFC 7525, DOI 10.17487/RFC7525, May 2015, <http://www.rfc-editor.org/info/rfc7525>.

[RFC7539] Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF Protocols", RFC 7539, DOI 10.17487/RFC7539, May 2015, <http://www.rfc-editor.org/info/rfc7539>.

[RFC7568] Barnes, R., Thomson, M., Pironti, A., and A. Langley, "Deprecating Secure Sockets Layer Version 3.0", RFC 7568, DOI 10.17487/RFC7568, June 2015, <http://www.rfc-editor.org/info/rfc7568>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <http://www.rfc-editor.org/info/rfc7748>.

[SP800-107-rev1] National Institute of Standards and Technology, "Recommendation for Applications Using Approved Hash Algorithms", NIST Special Publication 800-107, Revision 1, DOI 10.6028/NIST.SP.800-107r1, August 2012, <http://csrc.nist.gov/publications/nistpubs/800-107-rev1/sp800-107-rev1.pdf>.

[SP800-22b] National Institute of Standards and Technology, "A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications", NIST Special Publication 800-22, Revision 1a, April 2010, <http://csrc.nist.gov/publications/nistpubs/800-22-rev1a/SP800-22rev1a.pdf>.

[SP800-90A]
National Institute of Standards and Technology, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators", NIST Special Publication 800-90A Revision 1, DOI 10.6028/NIST.SP.800-90Ar1, June 2015, <http://csrc.nist.gov/publications/drafts/800-90/sp800-90a_r1_draft_november2014_ver.pdf>.

[TLS-FALSESTART]
Langley, A., Modadugu, N., and B. Moeller, "Transport Layer Security (TLS) False Start", Work in Progress, draft-ietf-tls-falsestart-02, May 2016.

[Triple-HS]
Bhargavan, K., Delignat-Lavaud, C., Pironti, A., and P. Yves Strub, "Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS", In Proceedings of the IEEE Symposium on Security and Privacy, Pages 98-113, DOI 10.1109/SP.2014.14, 2014.

Appendix A. Conveying DTLS over SMS

This section is normative for the use of DTLS over SMS. Timer recommendations are already outlined in Section 11 and also applicable to the transport of DTLS over SMS.

This section requires readers to be familiar with the terminology and concepts described in [GSM-SMS] and [WAP-WDP].

The remainder of this section assumes Mobile Stations are capable of producing and consuming Transport Protocol Data Units (TPDUs) encoded as 8-bit binary data.

A.1. Overview

DTLS adds an additional round-trip to the TLS [RFC5246] handshake to serve as a return-routability test for protection against certain types of DoS attacks. Thus, a full-blown DTLS handshake comprises up to 6 "flights" (i.e., logical message exchanges), each of which is then mapped on to one or more DTLS records using the segmentation and reassembly (SaR) scheme described in Section 4.2.3 of [RFC6347]. The overhead for said scheme is 6 bytes per handshake message which, given a realistic 10+ messages handshake, would amount to around 60 bytes across the whole handshake sequence.

Note that the DTLS SaR scheme is defined for handshake messages only. In fact, DTLS records are never fragmented and MUST fit within a single transport layer datagram.

SMS provides an optional segmentation and reassembly scheme as well, known as Concatenated short messages (see Section 9.2.3.24.1 of [GSM-SMS]). However, since the SaR scheme in DTLS cannot be circumvented, the Concatenated short messages mechanism SHOULD NOT be used during handshake to avoid redundant overhead. Before starting the handshake phase (either actively or passively), the DTLS implementation MUST be explicitly configured with the Path MTU (PMTU) of the SMS transport in order to correctly instrument its SaR function. The PMTU SHALL be 133 bytes if multiplexing based on the Wireless Datagram Protocol (WDP) is used (see Appendix A.3); 140 bytes otherwise.

It is RECOMMENDED that the established security context over the longest possible period be used (possibly until a Closure Alert message is received or after a very long inactivity timeout) to avoid the expensive re-establishment of the security association.

A.2. Message Segmentation and Reassembly

The content of an SMS message is carried in the TP-UserData field, and its size may be up to 140 bytes. As already mentioned in Appendix A.1, longer (i.e., up to 34170 bytes) messages can be sent using Concatenated SMS.

This scheme consumes 6-7 bytes (depending on whether the short or long segmentation format is used) of the TP-UserData field, thus reducing the space available for the actual content of the SMS message to 133-134 bytes per TPDU.

Though in principle a PMTU value higher than 140 bytes could be used, which may look like an appealing option given its more efficient use of the transport, there are disadvantages to consider. First, there is an additional overhead of 7 bytes per TPDU to be paid to the SaR function (which is in addition to the overhead introduced by the DTLS SaR mechanism. Second, some networks only partially support the Concatenated SMS function, and others do not support it at all.

For these reasons, the Concatenated short messages mechanism SHOULD NOT be used, and it is RECOMMENDED to leave the same PMTU settings used during the handshake phase, i.e., 133 bytes if WDP-based multiplexing is enabled; 140 bytes otherwise.

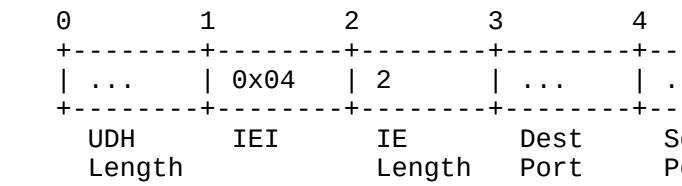
Note that, after the DTLS handshake has completed, any fragmentation and reassembly logic that pertains the application layer (e.g., segmenting CoAP messages into DTLS records and reassembling them after the crypto operations have been successfully performed) needs to be handled by the application that uses the established DTLS tunnel.

A.3. Multiplexing Security Associations

Unlike IPsec Encapsulating Security Payload (ESP) / Authentication Header (AH), DTLS records do not contain any association identifiers. Applications must arrange to multiplex between associations on the same endpoint which, when using UDP/IP, is usually done with the host/port number.

If the DTLS server allows more than one client to be active at any given time, then the Wireless Application Protocol (WAP) User Datagram Protocol [WAP-WDP] can be used to achieve multiplexing of the different security associations. (The use of WDP provides the additional benefit that upper-layer protocols can operate independently of the underlying wireless network, hence achieving application-agnostic transport handover.)

The total overhead cost for encoding the WDP source and destination ports is either 5 or 7 bytes out of the total available for the SMS content depending on if 1-byte or 2-byte port identifiers are used, as shown in Figures 13 and 14.



Legend:
UDH = user data header
IEI = information element identifier

Figure 13: Application Port Addressing Scheme (8-Bit Address)

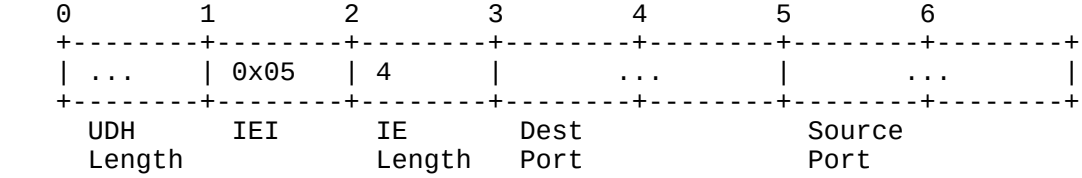


Figure 14: Application Port Addressing Scheme (16-Bit Address)

The receiving side of the communication gets the source address from the originator address (TP-OA) field of the SMS-DELIVER TPDU. This way, a unique 4-tuple identifying the security association can be reconstructed at both ends. (When replying to its DTLS peer, the sender will swap the TP-OA and destination address (TP-DA) parameters and the source and destination ports in the WDP.)

A.4. Timeout

If SMS-STATUS-REPORT messages are enabled, their receipt is not to be interpreted as the signal that the specific handshake message has been acted upon by the receiving party. Therefore, it MUST NOT be taken into account by the DTLS timeout and retransmission function.

Handshake messages MUST carry a validity period (TP-VP parameter in a SMS-SUBMIT TPDU) that is not less than the current value of the retransmission timeout. In order to avoid persisting messages in the network that will be discarded by the receiving party, handshake messages SHOULD carry a validity period that is the same as, or just slightly higher than, the current value of the retransmission timeout.

Appendix B. DTLS Record Layer Per-Packet Overhead

Figure 15 shows the overhead for the DTLS record layer for protecting data traffic when AES-128-CCM with an 8-octet Integrity Check Value (ICV) is used.

```
DTLS Record Layer Header.............13 bytes
Nonce (Explicit).....................8 bytes
ICV..................................8 bytes
-----------------------------------------------
Overhead.............................29 bytes
-----------------------------------------------
```

Figure 15: AES-128-CCM-8 DTLS Record Layer Per-Packet Overhead

The DTLS record layer header has 13 octets and consists of:

- 1-octet content type field,
- 2-octet version field,
- 2-octet epoch field,
- 6-octet sequence number, and
- 2-octet length field.

The "nonce" input to the AEAD algorithm is exactly that of [RFC5288], i.e., 12 bytes long. It consists of two values, namely a 4-octet salt and an 8-octet nonce_explicit:

The salt is the "implicit" part and is not sent in the packet. Instead, the salt is generated as part of the handshake process.

The nonce_explicit value is 8 octets long and it is chosen by the sender and carried in each TLS record. RFC 6655 [RFC6655] allows the nonce_explicit to be a sequence number or something else. This document makes this use more restrictive for use with DTLS: the 64-bit none_explicit value MUST be the 16-bit epoch concatenated with the 48-bit seq_num. The sequence number component of the nonce_explicit field at the AES-CCM layer is an exact copy of the sequence number in the record layer header field. This leads to a duplication of 8-bytes per record.

To avoid this 8-byte duplication, RFC 7400 [RFC7400] provides help with the use of the generic header compression technique for IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). Note that this header compression technique is not available when DTLS

is exchanged over transports that do not use IPv6 or 6LoWPAN, such as the SMS transport described in Appendix A of this document.

Appendix C. DTLS Fragmentation

Section 4.2.3 of [RFC6347] advises DTLS implementations to not produce overlapping fragments. However, it requires receivers to be able to cope with them. The need for the latter requisite is explained in Section 4.1.1.1 of [RFC6347]: accurate PMTU estimation may be traded for shorter handshake completion time.

In many cases, the cost of handling fragment overlaps has proved to be unaffordable for constrained implementations, particularly because of the increased complexity in buffer management.

In order to reduce the likelihood of producing different fragment sizes and consequent overlaps within the same handshake, this document RECOMMENDs:

- clients (handshake initiators) to use reliable PMTU information for the intended destination; and
- servers to mirror the fragment size selected by their clients.

The PMTU information comes from either a "fresh enough" discovery performed by the client [RFC1981] [RFC4821] or some other reliable out-of-band channel.

Acknowledgments

Thanks to Derek Atkins, Paul Bakker, Olaf Bergmann, Carsten Bormann, Ben Campbell, Brian Carpenter, Robert Cragie, Spencer Dawkins, Russ Housley, Rene Hummen, Jayaraghavendran K, Sye Loong Keoh, Matthias Kovatsch, Sandeep Kumar, Barry Leiba, Simon Lemay, Alexey Melnikov, Gabriel Montenegro, Manuel Pegourie-Gonnard, Akbar Rahman, Eric Rescorla, Michael Richardson, Ludwig Seitz, Zach Shelby, Michael StJohns, Rene Struik, Tina Tsou, and Sean Turner for their helpful comments and discussions that have shaped the document.

A big thanks also to Klaus Hartke, who wrote the initial draft version of this document.

Finally, we would like to thank our area director (Stephen Farrell) and our working group chairs (Zach Shelby and Dorothy Gellert) for their support.

Authors' Addresses

Hannes Tschofenig (editor)
ARM Ltd.
110 Fulbourn Rd
Cambridge CB1 9NJ
United Kingdom

Email: Hannes.tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Thomas Fossati
Nokia
3 Ely Road
Milton, Cambridge CB24 6DD
United Kingdom

Email: thomas.fossati@nokia.com