A TLS/DTLS 1.2 Profile for the Internet of Things
draft-ietf-dice-profile-08.txt

Abstract

   A common design pattern in Internet of Things (IoT) deployments is
   the use of a constrained device (typically providing sensor data)
   that makes data available for home automation, industrial control
   systems, smart cities and other IoT deployments.

   This document defines a Transport Layer Security (TLS) and Datagram
   TLS 1.2 profile that offers communications security for this data
   exchange thereby preventing eavesdropping, tampering, and message
   forgery.

Status of This Memo

   This Internet-Draft is submitted in full conformance with the
   provisions of BCP 78 and BCP 79.

   Internet-Drafts are working documents of the Internet Engineering
   Task Force (IETF). Note that other groups may also distribute
   working documents as Internet-Drafts. The list of current Internet-
   Drafts is at http://datatracker.ietf.org/drafts/current/.

   Internet-Drafts are draft documents valid for a maximum of six months
   and may be updated, replaced, or obsoleted by other documents at any
   time. It is inappropriate to use Internet-Drafts as reference
   material or to cite them other than as "work in progress."

   This Internet-Draft will expire on June 24, 2015.

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 make data available 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 TLS
   (DTLS) has been very popular with many application protocols and it
   is likely to be useful for IoT scenarios as well.

   To make Internet protocols fit 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]. UDP is mainly used to carry CoAP messages but other
   transports can be utilized, such as SMS or even TCP.

   While this document is inspired by the desire to protect CoAP
   messages using DTLS 1.2 [RFC6347] the guidance in this document 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 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 extensions.

   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 Internet of Things product development.

2. Terminology

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

Note that "Client" and "Server" in this document refer to TLS/DTLS
roles, where the Client initiates the TLS/DTLS handshake. This does
not restrict the interaction pattern of the protocols on top of TLS/
DTLS since the record layer allows bi-directional communication.
This aspect is further described in Section 4.

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

3. TLS/DTLS Protocol Overview

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
Handshake Protocol. 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. One such encapsulated 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.
Since DTLS operates on top of an unreliable datagram transport a few
enhancements to the TLS structure are, however necessary. RFC 6347
explains these differences in great detail. As a short summary, for
those not familiar with DTLS the differences are:

o 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 MAC
   is the 64-bit value formed by concatenating the epoch and the
   sequence number.

o 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
   [I-D.ietf-tls-prohibiting-rc4]. 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
   dentical amount of output generated from a cryptographically
   secure keyed pseudorandom number generator.

o 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. Retransmission timers have been included to
   deal with message loss.

4. Communication Models

This document describes a profile of TLS/DTLS 1.2 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.

4.1. 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, conveyed on top of DTLS
may need additional information, such information about URLs of the
endpoints the CoAP needs to register and publish information to.
This configuration information (including credentials) 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:

o For PSK-based authentication (see Section 6.1), this includes the
   paired "PSK identity" and shared secret to be used with each
   server.

o For raw public key-based authentication (see Section 6.2), this
   includes either the server's public key or the hash of the
   server's public key.

o For certificate-based authentication (see Section 6.3), 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.

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

```
            +////////////////////////////////+
            |           Configuration         |
            |////////////////////////////////|
            | Server A --> PSK Identity, PSK  |
            | Server B --> Public Key (Server B),|
            |              Public Key (Client)  |
            | Server C --> Public Key (Client),  |
            |              Trust Anchor Store   |
            +----------------------------------+
              oo
          oooooo
        o
  +-----------+
  |Constrained|
  |TLS/DTLS   |
  |Client     |-
  +-----------+ \
              \   ,-------.
               ,'          `.            +------+
              /   IP-based   \           |Server|
             (     Network     )         |  A   |
              \               /          +------+
               `.          ,'
                 '---+---'              +------+
                     |                  |Server|
                     |                  |  B   |
                     |                  +------+
                     |
                     |                  +------+
             +----------------->|Server|
                                |  C   |
                                +------+
```
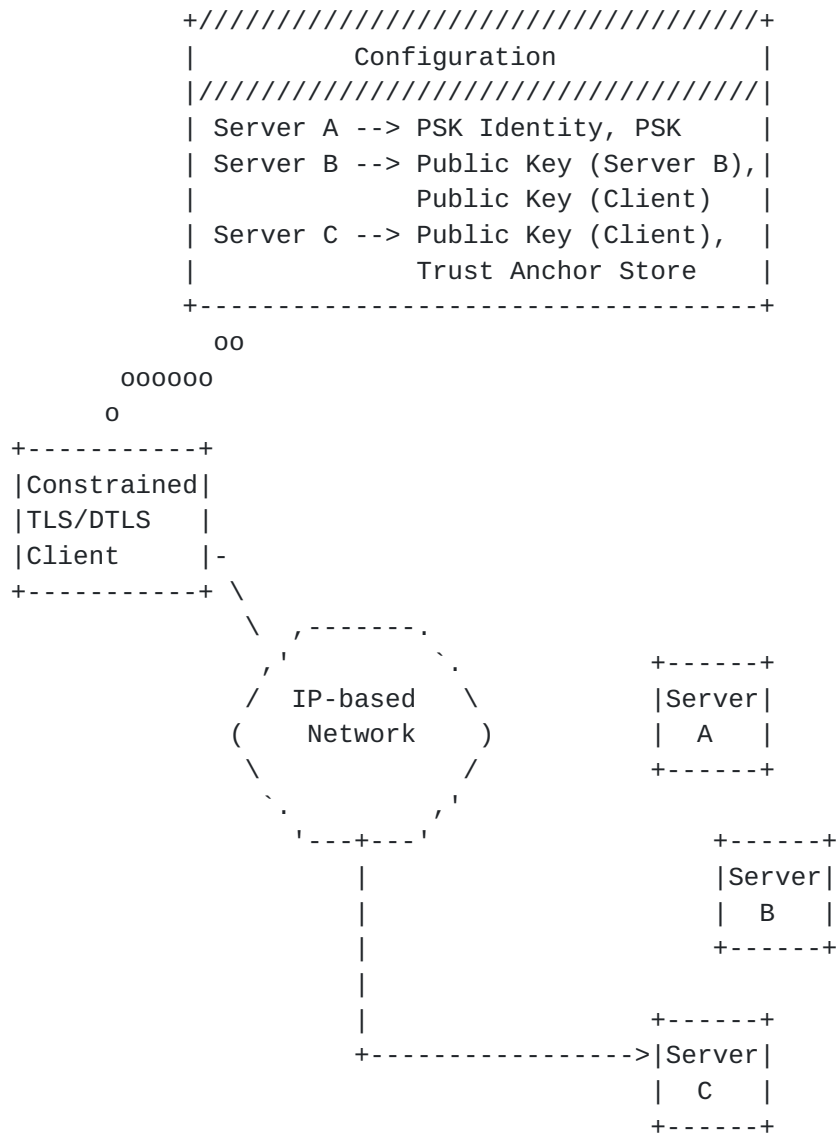
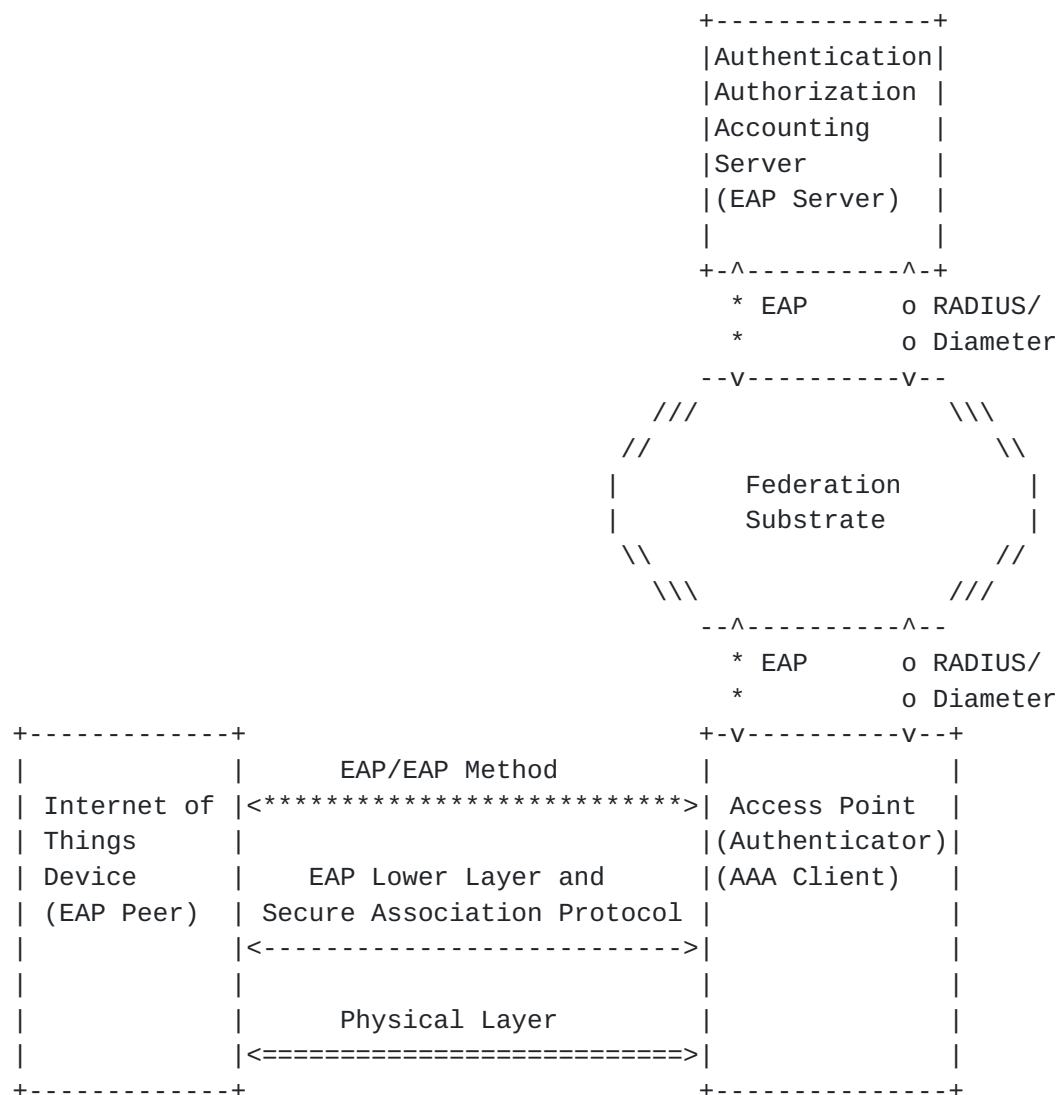Figure 1: Constrained Client Profile.

4.1.1. Examples of Constrained Client Exchanges

4.1.1.1. Network Access Authentication Example

   Re-use 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 re-use 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), 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, name RADIUS or
   Diameter.

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

```
                                        +--------------+
                                        |Authentication|
                                        |Authorization |
                                        |Accounting    |
                                        |Server        |
                                        |(EAP Server)  |
                                        |              |
                                        +-^----------^-+
                                          * EAP      o RADIUS/
                                          *          o Diameter
                                        --v----------v--
                                      ///              \\\
                                     //                  \\
                                     |      Federation     |
                                     |      Substrate      |
                                      \\                  //
                                       \\\              ///
                                        --^----------^--
                                          * EAP      o RADIUS/
                                          *          o Diameter
   +-------------+                       +-v----------v--+
   |             |     EAP/EAP Method    |              |
   | Internet of |<*********************>| Access Point |
   | Things      |                       |(Authenticator)|
   | Device      |    EAP Lower Layer and|(AAA Client)  |
   | (EAP Peer)  | Secure Association Protocol|          |
   |             |<--------------------------->|         |
   |             |                       |              |
   |             |      Physical Layer   |              |
   |             |<=========================>|          |
   +-------------+                       +--------------+

   Legend:

     <****>: Device-to-AAA Server Exchange
     <---->: Device-to-Authenticator Exchange
     <oooo>: AAA Client-to-AAA Server Exchange
     <====>: Phyiscal 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 re-uses the TLS-based protocol exchange and
   encapsulates it inside the EAP payload. In terms of re-use 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. The EAP-TLS exchange
   is shown in Figure 3 where it is worthwhile to point out that in EAP

the client / server roles are reversed but with the use of EAP-TLS
the IoT device acts as a TLS client.

```
     Authenticating Peer       Authenticator
     -------------------       -------------
                               <- EAP-Request/
                               Identity
     EAP-Response/
     Identity (MyID) ->
                               <- EAP-Request/
                               EAP-Type=EAP-TLS
                               (TLS Start)
     EAP-Response/
     EAP-Type=EAP-TLS
     (TLS client_hello)->
                               <- EAP-Request/
                               EAP-Type=EAP-TLS
                               (TLS server_hello,
                                 TLS certificate,
                        [TLS server_key_exchange,]
                         TLS certificate_request,
                            TLS server_hello_done)
     EAP-Response/
     EAP-Type=EAP-TLS
     (TLS certificate,
      TLS client_key_exchange,
      TLS certificate_verify,
      TLS change_cipher_spec,
      TLS finished) ->
                               <- EAP-Request/
                               EAP-Type=EAP-TLS
                               (TLS change_cipher_spec,
                                TLS finished)
     EAP-Response/
     EAP-Type=EAP-TLS ->
                               <- EAP-Success
```

                    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 re-use 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.
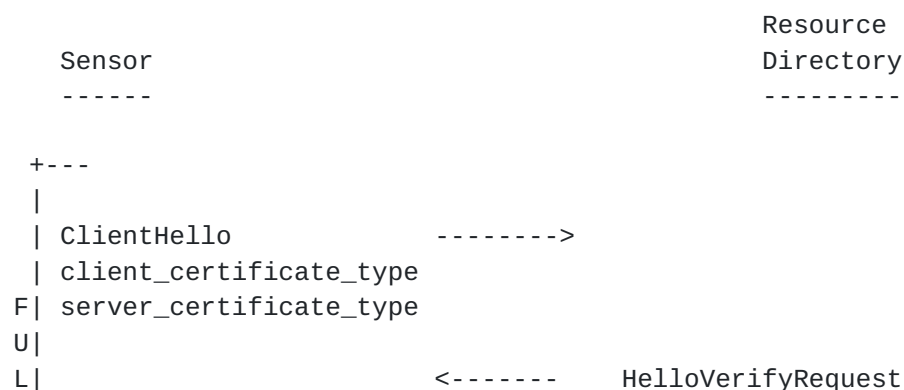
4.1.1.2. CoAP-based Data Exchange Example

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 pre-configured endpoint on the server. In certain circumstances
this might be too limiting and additional functionality is needed, as
shown in Figure 4, 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 with a DTLS exchange with a resource
directory to register available resources.
[I-D.ietf-core-resource-directory] defines the resource directory as
a web entity that stores information about web resources and
implements the REST interfaces defined in
[I-D.ietf-core-resource-directory] for registration and lookup of
those resources.

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. Details
about the capabilities of the resource directory can be found in
[I-D.ietf-core-resource-directory].

After some time (assuming that the client regularly refreshes its
registration) the resource directory receives a request (not shown in
the figure) 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.

```
                                            Resource
        Sensor                              Directory
        ------                              ---------


     +---
      |
      | ClientHello              -------->
      | client_certificate_type
    F| server_certificate_type
    U|
    L|                           <-------      HelloVerifyRequest
```

```
    L|
     | ClientHello           -------->
    D| client_certificate_type
    T| server_certificate_type
    L|
    S|                                          ServerHello
     |                            client_certificate_type
    H|                            server_certificate_type
    A|                                          Certificate
    N|                                    ServerKeyExchange
    D|                                   CertificateRequest
    S|                          <--------    ServerHelloDone
    H|
    A| Certificate
    K| ClientKeyExchange
    E| CertificateVerify
     | [ChangeCipherSpec]
     | Finished              -------->
     |
     |                                     [ChangeCipherSpec]
     |                          <--------          Finished
     +---

     +---                                           ///+
    C|                                              \ D
    O| Req: POST coap://rd.example.com/rd?ep=node1  \ T
    A| Payload:                                      \ L
    P| </temp>;ct=41;                                \ S
     |    rt="temperature-c";if="sensor",            \
    R| </light>;ct=41;                               \ R
    D|    rt="light-lux";if="sensor"                 \ E
     |                          -------->            \ C
    R|                                               \ O
    E|                                               \ R
    G|                              Res: 2.01 Created \ D
    .|                          <-------- Location: /rd/4521 \
     |                                               \ L
     +---                                            \ A
                                                     \ Y
                          *                          \ E
                          * (time passes)            \ R
                          *                          \
     +---                                            \ P
    C|                                               \ R
    O|            Req: GET coaps://sensor.example.com/temp \ O
    A|                          <--------            \ T
    P|                                               \ E
     | Res:  2.05 Content                            \ C
```

```
 G| Payload:                                              \ T
 E| 25.5                        -------->                 \ E
 T|                                                       \ D
  +---                                                    ///+
```

Figure 4: DTLS/CoAP exchange using Resource Directory.

4.2. Constrained TLS/DTLS Servers

Section 4.1 illustrates deployment models 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 look at cases where constrained devices run TLS/
DTLS servers to secure access to an application layer protocol
server, like a CoAP or HTTP server. Running server functionality on
a constrained node is typically more demanding since servers have to
listen and wait for incoming requests. Therefore, they will have
fewer possibilities to enter sleeping cycles. Nevertheless, there
are legitimate reasons to deploy servers as constrained devices.

A deployment with constrained servers has to overcome several
challenges. Later we will explain how these challenges has been
solved using 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 eco-
system factor will impact deployment.

The challenges are:

Discovery and Reachability:

   Before initiating a connection to a constrained server a client
   first needs to discover that server and, once discovered, it needs
   to maintain reachability with that device.

   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 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. RFC 7390 [RFC7390] describes
   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).

Authentication:

The next challenge concerns the provisioning of authentication
credentials to the clients as well as servers. In Section 4.1 we
assumed 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
[I-D.iab-smart-object-architecture] these different communication
patterns. 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 either use pairing (also called imprinting) or
a trusted third party. With pairing two devices execute a special
protocol exchange that is unauthenticated to establish an shared
key (for example using an unauthenticated Diffie-Hellman exchange)
key. 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,
   scanning QR codes

* The use of alternative wireless communication channels (e.g.,
   infra-red communication in addition to WiFi)

* 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 to 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.

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 more fine-grained and
dynamic access control the reader is referred to the ongoing work
in the ACE working group.

5. **The TLS/DTLS Ciphersuite Concept**

TLS (and consequently DTLS) has the concept of 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:

o Authentication and key exchange algorithm (e.g., PSK)

o Cipher and key length (e.g., Advanced Encryption Standard (AES)
   with 128 bit keys [AES])

o Mode of operation (e.g., Counter with Cipher Block Chaining -
   Message Authentication Code (CBC-MAC) Mode (CCM) for AES)
   [RFC3610]

o Hash algorithm for integrity protection, such as the Secure Hash
   Algorithm (SHA) in combination with Keyed-Hashing for Message
   Authentication (HMAC) (see [RFC2104] and [RFC4634])

o Hash algorithm for use with the pseudorandom function (e.g., HMAC
   with the SHA-256)

o Misc information (e.g., length of authentication tags)

o 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. RFC 6655
[RFC6655] defines this ciphersuite. It uses the Advanced Encryption
Standard (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 fixed 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 which encrypt (parts of) the message and

authenticate the message simultaneously. Examples of such modes
include the Counter with Cipher Block Chaining - Message
Authentication Code (CBC-MAC) Mode (CCM) mode, and the Galois/Counter
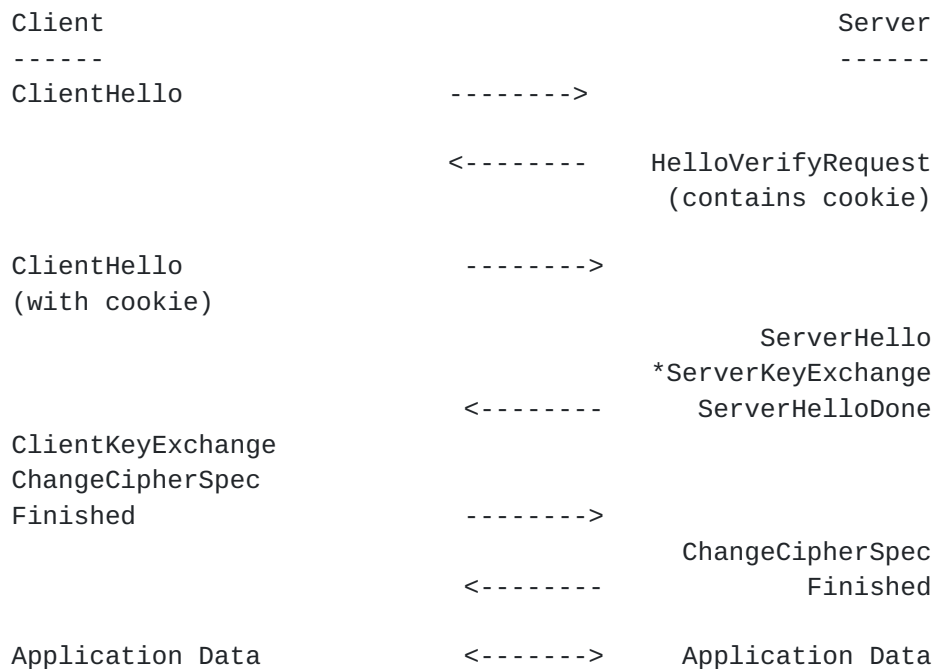Mode (GCM) (see [RFC5288] and [RFC7251]).

Some AEAD ciphersuites have shorter authentication tags 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.

TLS 1.2 also replaced the combination of MD5/SHA-1 hash functions in
the TLS pseudo random function (PRF) used in earlier versions of TLS
with cipher-suite-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.

6. Credential Types

6.1. Pre-Shared Secret

The use of pre-shared secret credentials is one of the most basic
techniques for TLS/DTLS since it is both computational efficient and
bandwidth conserving. Pre-shared secret based authentication was
introduced to TLS with RFC 4279 [RFC4279]. The exchange shown in
Figure 5 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. The cookie exchange allows the server to react
to flooding attacks.

```
        Client                                          Server
        ------                                          ------
        ClientHello               -------->

                                  <--------      HelloVerifyRequest
                                                    (contains cookie)

        ClientHello               -------->
        (with cookie)
                                                          ServerHello
                                                   *ServerKeyExchange
                                  <--------        ServerHelloDone
        ClientKeyExchange
        ChangeCipherSpec
        Finished                  -------->
                                                      ChangeCipherSpec
                                  <--------                  Finished

        Application Data          <------->      Application Data
```

   Legend:

   * indicates an optional message payload

     Figure 5: DTLS PSK Authentication including the Cookie Exchange.

   [RFC4279] does not mandate the use of any particular type of client
   identity and the client and server have to agree on the identities
   and keys to be used. The mandated encoding of identities in
   Section 5.1 of RFC 4279 aims to improve interoperability for those
   cases where the identity is configured by a person using some
   management interface. Many IoT devices do, however, not have a user
   interface and most of their credentials are bound to the device
   rather than the user. Furthermore, credentials are often provisioned
   into trusted hardware modules or in the firmware by developers. 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 (as domain names, Distinguished Names, or
   IP addresses have) and a bit-by-bit comparison operation can then be
   used by the server-side infrastructure.

   The client indicates which key it uses by including a "PSK identity"
   in the ClientKeyExchange message. As described in Section 4 clients
   may have multiple pre-shared keys with a single server and to help
   the client in selecting which PSK identity / PSK pair to use, the
   server can provide a "PSK identity hint" in the ServerKeyExchange
   message. If the hint for PSK key selection is based on the domain

name of the server then servers SHOULD NOT send the "PSK identity hint" in the ServerKeyExchange message. In general, servers SHOULD NOT send the "PSK identity hint" in the ServerKeyExchange message and client MUST ignore the message. This approach is inline with RFC 4279 [RFC4279]. Note: The TLS Server Name Indication (SNI) extension allows the client to tell a server the name of the server it is contacting, which is relevant for hosting environments. A server using the identity hint 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. For the IoT environment, keys are distributed as part of hardware modules or are embedded into the firmware. 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 and shorter PSKs is RECOMMENDED.

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. This ciphersuite makes use of the default TLS 1.2 Pseudorandom Function (PRF), which uses an HMAC with the SHA-256 hash function. (Note that all IoT implementations will need a SHA-256 implementation due to the construction of the pseudo-random number function in DTLS/TLS 1.2.)

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.

6.2. 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 re-uses 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 6, namely the server_certificate_type*' and the client_certificate_type. To operate this mechanism securely it is necessary to authenticate and authorize the public keys out-of-band. This document therefore assumes that a client implementation comes with one or multiple raw public keys of servers, it has to communicate with, pre-provisioned. Additionally, a device will have

its own raw public key. To replace, delete, or add raw public key to
this list requires a software update, for example using a firmware
update mechanism.

```
 Client                                          Server
 ------                                          ------

 ClientHello               -------->
 *client_certificate_type*
 *server_certificate_type*

                                            ServerHello
                             *client_certificate_type*
                             *server_certificate_type*
                                            Certificate
                                      ServerKeyExchange
                                      CertificateRequest
                          <--------      ServerHelloDone

 Certificate
 ClientKeyExchange
 CertificateVerify
 [ChangeCipherSpec]
 Finished                  -------->

                                      [ChangeCipherSpec]
                          <--------            Finished
```

Note: Extensions marked with '*' were introduced with
      RFC 7250.

Figure 6: DTLS Raw Public Key Exchange including the Cookie Exchange.

The CoAP recommended ciphersuite for use with this credential type is
TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8 [RFC7251]. This elliptic curve
cryptography (ECC) based AES-CCM TLS ciphersuite uses the Ephemeral
Elliptic Curve Diffie-Hellman (ECDHE) as the key establishment
mechanism and an Elliptic Curve Digital Signature Algorithm (ECDSA)
for authentication. Due to the use of Ephemeral Elliptic Curve
Diffie-Hellman (ECDHE) the recently introduced named Diffie-Hellman
groups [I-D.ietf-tls-negotiated-dl-dhe] are not applicable to this
profile. This ciphersuite make use of the AEAD capability in DTLS
1.2 and utilizes an eight-octet authentication tag. The use of a
Diffie-Hellman key exchange provides perfect forward secrecy (PFS).
More details about PFS can be found in Section 11.

RFC 6090 [RFC6090] provides valuable information for implementing
Elliptic Curve Cryptography 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.

6.3. Certificates

The use of mutual certificate-based authentication is shown in
Figure 7, which makes use of the cached info extension
[I-D.ietf-tls-cached-info]. Support of the cached info extension is
REQUIRED. Caching certificate chains allows the client to reduce the
communication overhead significantly since otherwise the server would
provide the end entity certificate, and the certificate chain.
Because certificate validation requires that root keys be distributed
independently, the self-signed certificate that specifies the root
certificate authority is omitted from the chain. Client
implementations MUST be provisioned with a trust anchor store that
contains the root certificates. The use of the Trust Anchor
Management Protocol (TAMP) [RFC5934] is, however, not envisioned.
Instead IoT devices using this profile MUST use a software update
mechanism to populate the trust anchor store.

```
   Client                                            Server
   ------                                            ------

   ClientHello              -------->
   *cached_information*

                                                   ServerHello
                                          *cached_information*
                                                   Certificate
                                             ServerKeyExchange
                                            CertificateRequest
                              <--------       ServerHelloDone

   Certificate
   ClientKeyExchange
   CertificateVerify
   [ChangeCipherSpec]
   Finished                 -------->

                                             [ChangeCipherSpec]
                              <--------               Finished
```

   Note: Extensions marked with '*' were introduced with
         [I-D.ietf-tls-cached-info].

           Figure 7: DTLS Mutual Certificate-based Authentication.

   When DTLS is used to secure CoAP messages then the server provided
   certificates MUST contain the fully qualified DNS domain name or
   "FQDN" as dNSName. The coaps URI scheme is described in Section 6.2
   of [RFC7252]. This FQDN is stored in the SubjectAltName or in the
   leftmost CN component of 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 look-up process, as described in RFC
   6125 [RFC6125]. For the profile in this specification does not
   assume dynamic discovery of local servers.

   For client certificates the identifier used in the SubjectAltName or
   in the CN MUST be an EUI-64 [EUI64], as mandated in Section 9.1.3.3
   of [RFC7252].

   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. 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.

Regarding the ciphersuite choice the discussion in Section 6.2
applies. Further details about X.509 certificates can be found in
Section 9.1.3.3 of [RFC7252]. The TLS_ECDHE_ECDSA_WITH_AES_128_CCM_8
ciphersuite description in Section 6.2 is also applicable to this
section.

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 resources to process, transmit
or receive.

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.

6.3.1. Client Certificate URLs

RFC 6066 [RFC6066] allows to avoid sending client-side certificates
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.

6.3.2. 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 exchange
provides. Since the server does not necessarily know what trust
anchors the client has stored it includes intermediate CA certs in
the certificate payload as well to facilitate with certification path
construction and path validation.

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 and no such dynamic indication of trust
anchors is needed.

For deployments where IoT devices interact with a fixed, pre-
configured set of servers and where a software update mechanism is
available this extension is NOT RECOMMENDED. Environments where the

client needs to interact with dynamically discovered TLS/DTLS servers
the use of this extension is RECOMMENDED.

7. 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
since otherwise absent this extension 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 6.1.

8. Error Handling

TLS/DTLS uses the Alert protocol to convey error messages and
specifies a longer list of errors. 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 re-starting the
handshake or informing the user using the (often limited) user
interface. Warnings may be ignored by the application since many IoT
devices will either have 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 SSL and are therefore not applicable to
this profile. Those include decryption_failed_RESERVED,
no_certificate_RESERVE, and export_restriction_RESERVED.

A number of the error messages are applicable only for certificate-
based authentication ciphersuites. Hence, for PSK and raw public key
use the following error messages are not applicable:

o bad_certificate,

o unsupported_certificate,

o certificate_revoked,

o certificate_expired,

o certificate_unknown,

o unknown_ca, and

o access_denied.

Since this profile does not make use of compression at the TLS layer
the decompression_failure error message is not applicable either.

RFC 4279 introduced a new alert message unknown_psk_identity for PSK
ciphersuites. As stated in Section 2 of RFC 4279 the
decryption_error error message may also be used instead. For this
profile the TLS server MUST return the decryption_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
   ciphtersuites get added over time.

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

9. 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 8. 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.

```
     Client                                          Server
     ------                                          ------

     ClientHello                 -------->
                                                  ServerHello
                                         [ChangeCipherSpec]
                                 <--------           Finished
     [ChangeCipherSpec]
     Finished                    -------->
     Application Data            <------->     Application Data
```

Figure 8: 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]. 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.

10. Compression

Section 3.3 of [I-D.ietf-uta-tls-bcp] recommends to disable TLS/DTLS-
level compression due to attacks, such as 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.

This TLS/DTLS profile MUST NOT implement TLS/DTLS layer compression.

11. Perfect Forward Secrecy

Perfect forward secrecy (PFS) is a property that preserves the
confidentiality of past conversations even in situations where the
long-term secret is compromised.

The PSK ciphersuite recommended in Section 6.1 does not offer this
property since it does not utilize a Diffie-Hellman exchange. New

ciphersuites that support PFS for PSK-based authentication, such as proposed in [I-D.schmertmann-dice-ccm-psk-pfs], might become available as 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 raw public might offer an acceptable middleground between the PSK ciphersuite in terms of out-of-band validation and the functionality offered by asymmetric cryptography.

The use of PFS is a trade-off decision since on one hand the compromise of long-term secrets of embedded devices is more likely than with many other Internet hosts but on the other hand a Diffie-Hellman exchange requires ephemeral key pairs to be generated, which is demanding from a performance point of view. For performance reasons some implementations re-use key pairs over multiple exchanges (rather than generating new keys for each exchange) for the obvious performance improvement. Note, however, that such key re-use over long periods voids the benefits of forward secrecy when an attack gains access to this DH key pair.

The impact of the disclosure of past conversations and the desire to increase the cost for pervasive monitoring (as demanded by [RFC7258]) has to be taken into account when making a deployment decision.

Client implementations claiming support of this profile MUST implement the ciphersuites listed in Section 6 according to the selected credential type.

12. Keep-Alive

RFC 6520 [RFC6520] defines a heartbeat mechanism to test whether the other peer is still alive. 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. There are three types of exchanges that need to be analysed:

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.

Since the upload happens on an irregular and unpredictable basis
and due to renumbering and Network Address Translation (NAT) the
DTLS handshake may need to be re-started (ideally using session
resumption, if possible).

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 re-create 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 initiated the protocol
interaction but in this case we consider server-initiated
messages. Since messages to the client may get blocked by
intermediaries, such as NATs (including IPv4/IPv6 protocol
translators) and stateful packet filtering firewalls, the initial
connection setup is triggered by the client and then kept alive.
Since state at middleboxes expires fairly quickly (according to
measurements described in [HomeGateway]), regular heartbeats are
necessary whereby these keep-alive messages may be exchanged at
the application layer or within DTLS itself.

For this message exchange pattern the use of DTLS heartbeat
messages is quite useful but may interfere with registrations kept
at the application layer (for example when the CoAP resource
directory is used). 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.

13. Timeouts

To connect to the Internet a variety of wired and wireless
technologies are available. 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 RFC 4919 [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 re-transmissions, 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 twice the value at each
retransmission up to no less than 60 seconds. Due to the nature of
some radio technologies, these values are too aggressive and lead to
spurious failures when messages in flight need longer.

Note: If a round-trip time estimator (such as proposed in
[I-D.bormann-core-cocoa]) is available in the protocol stack of the
device, it could be used to dynamically update the setting of the
retransmit timeout.

Choosing appropriate timeout values is difficult with infrequent data
transmissions, changing network conditions, and large variance in
latency. This specification therefore RECOMMENDS an initial timer
value of 10 seconds with exponential back off up to no less then 60
seconds. Appendix A provides additional normative text for carrying
DTLS over SMS.

14. 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 Diffie-Hellman exchange requires
random numbers during the key pair generation. Special care has to
be paid when generating random numbers in embedded systems as many
entropy sources available on desktop operating systems or mobile
devices might be missing, as described in [Heninger]. Consequently,
if not enough time is given during system start time to fill the
entropy pool then the output might be predictable and repeatable, for
example leading to the same keys generated again and again.

It is important to note that sources contributing to the randomness
pool on laptops, or desktop PCs are not available on many IoT device,
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.

The ClientHello and the ServerHello messages contains the 'Random'
structure, which has two components: gmt_unix_time and a random
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). [I-D.mathewson-no-gmtunixtime] argues
that the entire ClientHello.Random value (including gmt_unix_time)
should be set to a cryptographically random sequence because of
privacy concerns regarding device fingerprinting. Since many IoT
devices do not have access to a real-time clock this recommendation
it is RECOMMENDED to follow the guidance outlined in
[I-D.mathewson-no-gmtunixtime] regarding the content of the
ClientHello.Random field. However, for the ServerHello.Random
structure it is RECOMMENDED to maintain the existing structure with
gmt_unix_time followed by a random sequence of 28 random bytes since
the client can use the received time information to securely obtain
time information. For constrained servers it cannot be assumed that
they maintain accurate time information; these devices MUST include
time information in the Server.Random structure when they actually
obtain accurate time information that can be utilized by clients.
Clients MUST only use time information obtained from servers they
trust.

IoT devices using TLS/DTLS MUST offer ways to generate quality random
numbers. Guidelines and requirements for random number generation
can be found in RFC 4086 [RFC4086].

15. Truncated MAC and Encrypt-then-MAC Extension

The truncated MAC extension was introduced with 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 was
performed independently.

The recommended ciphersuites in this document use the newer
Authenticated Encryption with Associated Data (AEAD) construct,
namely the CBC-MAC mode (CCM) with eight-octet authentication tags,
and are therefore not appliable 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 and RFC 7366 and RFC 6066 MUST NOT be implemented.

16. Server Name Indication (SNI)

The Server Name Indication extension defined in [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 single
IP address.

This specification RECOMMENDs the implementation of RFC 6066 unless
it is known that a TLS/DTLS client does not interact with a server in
a hosting environment.

17. 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 it to
allows 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.

18. 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 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
[Tripple-HS].

[I-D.ietf-tls-session-hash] 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 Diffie-Hellman-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.

19. Re-Negotiation Attacks

TLS/DTLS allows a client and a server who already have a TLS/DTLS
connection to negotiate new parameters, generate new keys, etc by
using the re-negotiation feature. Renegotiation happens in the
existing connection, with the new handshake packets being encrypted
along with application data. Upon completion of the re-negotiation
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 re-negotiation attack [RFC5746] this specification
RECOMMENDS to disable the TLS renegotigation feature. Clients MUST
respond to server-initiated re-negotiation attempts with an alert
message (no_renegotiation) and clients MUST NOT initiate them.

20. 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 the handshake can proceed. This behaviour 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 behaviour that leads
some clients to re-run the handshake with 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
[I-D.ietf-tls-sslv3-diediedie]).

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

o Clients MUST NOT send a TLS/DTLS version lower than version 1.2 in
   the ClientHello.

o Clients MUST NOT retry a failed negotiation offering a TLS/DTLS
   version lower than 1.2.

o 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.

If at some time in the future the TLS/DTLS 1.2 profile reaches the
quality of SSL 3.0 a software update mechanism is needed since
constrained devices are unlikely to run multiple TLS/DTLS versions
due to memory size restrictions.

21. Crypto Agility

This document recommends software and chip manufacturers to 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. Some deployment environments will also be
impacted by local regulation, which might dictate a certain cipher
and key size. Ongoing discussions regarding the choice of specific
ECC curves will also likely to impact implementations.

The following recommendations can be made to chip manufacturers:

o Make any AES hardware-based crypto implementation accessible to
   developers working on security implementations at higher layers.

Sometimes hardware implementatios 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.

o 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 implementations. AES-CCM in IEEE 802.15.4 and Bluetooth Smart uses 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 re-usable by a DTLS stack.

o 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 an the AES crypto function can use it in functions to build an efficient AES-GCM implementations. 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 recommend that sufficient headroom is provided to allow an upgrade to a newer cryptographic algorithms over the lifetime of the product. As an example, while AES-CCM is recommended thoughout this specification future products might use the ChaCha20 cipher in combination with the Poly1305 authenticator [I-D.irtf-cfrg-chacha20-poly1305]. The assumption is made that a robust software update mechanism is offered.

22. 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 9 illustrates the comparable key sizes in bits.

At the time of writing the key size recommendations for use with TLS-based ciphers found in [I-D.ietf-uta-tls-bcp] recommend DH key lengths of at least 2048 bit, which corresponds to a 112-bit symmetric key and a 233 bit ECC keys. These recommendations are inline with those from other organizations, such as National Institute of Standards and Technology (NIST) or European Network and Information Security Agency (ENISA). The authors of [ENISA-Report2013] add that a symmetric 80-bit security level is sufficient for legacy applications for the coming years, but a

128-bit security level 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 80-bit encryption for transactional data may be acceptable
for the near future while one has to insist on 128-bit encryption for
long lived data.

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

Figure 9: Comparable Key Sizes (in bits).

23. 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 [I-D.bmoeller-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

o Modern symmetric ciphers with an effective key length of 128 bits,
   such as AES-128-CCM

o Client certificate types, such as ecdsa_sign

o Key exchange methods, such as ECDHE_ECDSA

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

[24]. 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 with many IoT deployments poses a challenge since
many of the IoT devices operate unattended, even though they will
initially be provisioned by a human. The ability to control data
sharing and to configure preference 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 mis-attribution it also allows strong identification.
Device-related data collection (e.g., sensor recordings) will be
associated with other data to be truly useful and this extra data
might include personal data 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 to gain 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.

[25]. 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 also useful for changing

configuration information, for example, trust anchors and other
keying related information.

[26](). IANA Considerations

This document includes no request to IANA.

[27](). Acknowledgements

Thanks to Paul Bakker, Robert Cragie, Russ Housley, Rene Hummen,
Matthias Kovatsch, Sandeep Kumar, Sye Loong Keoh, Alexey Melnikov,
Manuel Pegourie-Gonnard, Akbar Rahman, Eric Rescorla, Michael
Richardson, Zach Shelby, Michael StJohns, Rene Struik, and Sean
Turner for their helpful comments and discussions that have shaped
the document.

Big thanks also to Klaus Hartke, who wrote the initial 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.

[28](). References

[28.1](). Normative References

[EUI64]   "GUIDELINES FOR 64-BIT GLOBAL IDENTIFIER (EUI-64)
            REGISTRATION AUTHORITY", April 2010,
            <http://standards.ieee.org/regauth/oui/tutorials/
            EUI64.html>.

[GSM-SMS]   ETSI, "3GPP TS 23.040 V7.0.1 (2007-03): 3rd Generation
            Partnership Project; Technical Specification Group Core
            Network and Terminals; Technical realization of the Short
            Message Service (SMS) (Release 7)", March 2007.

[I-D.ietf-tls-cached-info]
            Santesson, S. and H. Tschofenig, "Transport Layer Security
            (TLS) Cached Information Extension", draft-ietf-tls-
            cached-info-17 (work in progress), November 2014.

[I-D.ietf-tls-session-hash]
            Bhargavan, K., Delignat-Lavaud, A., Pironti, A., Langley,
            A., and M. Ray, "Transport Layer Security (TLS) Session
            Hash and Extended Master Secret Extension", draft-ietf-
            tls-session-hash-03 (work in progress), November 2014.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119, March 1997.

   [RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites
              for Transport Layer Security (TLS)", RFC 4279, December
              2005.

   [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security
              (TLS) Protocol Version 1.2", RFC 5246, August 2008.

   [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov,
              "Transport Layer Security (TLS) Renegotiation Indication
              Extension", RFC 5746, February 2010.

   [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions:
              Extension Definitions", RFC 6066, January 2011.

   [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, March 2011.

   [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer
              Security Version 1.2", RFC 6347, January 2012.

   [RFC6520] Seggelmann, R., Tuexen, M., and M. Williams, "Transport
              Layer Security (TLS) and Datagram Transport Layer Security
              (DTLS) Heartbeat Extension", RFC 6520, February 2012.

   [RFC7250] Wouters, P., Tschofenig, H., Gilmore, J., Weiler, S., and
              T. Kivinen, "Using Raw Public Keys in Transport Layer
              Security (TLS) and Datagram Transport Layer Security
              (DTLS)", RFC 7250, June 2014.

   [RFC7251] McGrew, D., Bailey, D., Campagna, M., and R. Dugal, "AES-
              CCM Elliptic Curve Cryptography (ECC) Cipher Suites for
              TLS", RFC 7251, June 2014.

   [WAP-WDP] Wireless Application Protocol Forum, "Wireless Datagram
              Protocol", June 2001.

28.2. Informative References

   [AES]   NIST, "FIPS PUB 197, Advanced Encryption Standard (AES)",
              http://www.iana.org/assignments/tls-parameters/
              tls-parameters.xhtml#tls-parameters-4, November 2001.

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

[Heninger]
          Heninger, N., Durumeric, Z., Wustrow, E., and A.
          Halderman, "Mining Your Ps and Qs: Detection of Widespread
          Weak Keys in Network Devices", 21st USENIX Security
          Symposium,
          https://www.usenix.org/conference/usenixsecurity12/
          technical-sessions/presentation/heninger, 2012.

[HomeGateway]
          Eggert, L., "An experimental study of home gateway
          characteristics, In Proceedings of the '10th annual
          conference on Internet measurement'", 2010.

[I-D.bmoeller-tls-falsestart]
          Langley, A., Modadugu, N., and B. Moeller, "Transport
          Layer Security (TLS) False Start", draft-bmoeller-tls-
          falsestart-01 (work in progress), November 2014.

[I-D.bormann-core-cocoa]
          Bormann, C., Betzler, A., Gomez, C., and I. Demirkol,
          "CoAP Simple Congestion Control/Advanced", draft-bormann-
          core-cocoa-02 (work in progress), July 2014.

[I-D.iab-smart-object-architecture]
          Tschofenig, H., Arkko, J., Thaler, D., and D. McPherson,
          "Architectural Considerations in Smart Object Networking",
          draft-iab-smart-object-architecture-06 (work in progress),
          October 2014.

[I-D.ietf-core-resource-directory]
          Shelby, Z. and C. Bormann, "CoRE Resource Directory",
          draft-ietf-core-resource-directory-02 (work in progress),
          November 2014.

[I-D.ietf-lwig-tls-minimal]
          Kumar, S., Keoh, S., and H. Tschofenig, "A Hitchhiker's
          Guide to the (Datagram) Transport Layer Security Protocol
          for Smart Objects and Constrained Node Networks", draft-
          ietf-lwig-tls-minimal-01 (work in progress), March 2014.

[I-D.ietf-tls-downgrade-scsv]
          Moeller, B. and A. Langley, "TLS Fallback Signaling Cipher
          Suite Value (SCSV) for Preventing Protocol Downgrade
          Attacks", draft-ietf-tls-downgrade-scsv-03 (work in
          progress), December 2014.

[I-D.ietf-tls-negotiated-dl-dhe]
          Gillmor, D., "Negotiated Discrete Log Diffie-Hellman
          Ephemeral Parameters for TLS", draft-ietf-tls-negotiated-
          dl-dhe-00 (work in progress), July 2014.

[I-D.ietf-tls-prohibiting-rc4]
          Popov, A., "Prohibiting RC4 Cipher Suites", draft-ietf-
          tls-prohibiting-rc4-01 (work in progress), October 2014.

[I-D.ietf-tls-sslv3-diediedie]
          Barnes, R., Thomson, M., Pironti, A., and A. Langley,
          "Deprecating Secure Sockets Layer Version 3.0", draft-
          ietf-tls-sslv3-diediedie-00 (work in progress), December
          2014.

[I-D.ietf-uta-tls-bcp]
          Sheffer, Y., Holz, R., and P. Saint-Andre,
          "Recommendations for Secure Use of TLS and DTLS", draft-
          ietf-uta-tls-bcp-08 (work in progress), December 2014.

[I-D.irtf-cfrg-chacha20-poly1305]
          Nir, Y. and A. Langley, "ChaCha20 and Poly1305 for IETF
          protocols", draft-irtf-cfrg-chacha20-poly1305-03 (work in
          progress), November 2014.

[I-D.mathewson-no-gmtunixtime]
          Mathewson, N. and B. Laurie, "Deprecating gmt_unix_time in
          TLS", draft-mathewson-no-gmtunixtime-00 (work in
          progress), December 2013.

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

[IANA-TLS]
          IANA, "TLS Cipher Suite Registry",
          http://www.iana.org/assignments/tls-parameters/
          tls-parameters.xhtml#tls-parameters-4, 2014.

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

[Keylength]
          Giry, D., "Cryptographic Key Length Recommendations",
          http://www.keylength.com, November 2014.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191,
          November 1990.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery
          for IP version 6", RFC 1981, August 1996.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-
          Hashing for Message Authentication", RFC 2104, February
          1997.

[RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with
          CBC-MAC (CCM)", RFC 3610, September 2003.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness
          Requirements for Security", BCP 106, RFC 4086, June 2005.

[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, May 2006.

[RFC4634] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms
          (SHA and HMAC-SHA)", RFC 4634, July 2006.

[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, August 2007.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig,
          "Transport Layer Security (TLS) Session Resumption without
          Server-Side State", RFC 5077, January 2008.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated
          Encryption", RFC 5116, January 2008.

[RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS
          Authentication Protocol", RFC 5216, March 2008.

   [RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible
              Authentication Protocol (EAP) Key Management Framework",
              RFC 5247, August 2008.

   [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, May 2008.

   [RFC5288] Salowey, J., Choudhury, A., and D. McGrew, "AES Galois
              Counter Mode (GCM) Cipher Suites for TLS", RFC 5288,
              August 2008.

   [RFC5934] Housley, R., Ashmore, S., and C. Wallace, "Trust Anchor
              Management Protocol (TAMP)", RFC 5934, August 2010.

   [RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic
              Curve Cryptography Algorithms", RFC 6090, February 2011.

   [RFC6655] McGrew, D. and D. Bailey, "AES-CCM Cipher Suites for
              Transport Layer Security (TLS)", RFC 6655, July 2012.

   [RFC6690] Shelby, Z., "Constrained RESTful Environments (CoRE) Link
              Format", RFC 6690, August 2012.

   [RFC6961] Pettersen, Y., "The Transport Layer Security (TLS)
              Multiple Certificate Status Request Extension", RFC 6961,
              June 2013.

   [RFC7228] Bormann, C., Ersue, M., and A. Keranen, "Terminology for
              Constrained-Node Networks", RFC 7228, May 2014.

   [RFC7252] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained
              Application Protocol (CoAP)", RFC 7252, June 2014.

   [RFC7258] Farrell, S. and H. Tschofenig, "Pervasive Monitoring Is an
              Attack", BCP 188, RFC 7258, May 2014.

   [RFC7366] Gutmann, P., "Encrypt-then-MAC for Transport Layer
              Security (TLS) and Datagram Transport Layer Security
              (DTLS)", RFC 7366, September 2014.

   [RFC7390] Rahman, A. and E. Dijk, "Group Communication for the
              Constrained Application Protocol (CoAP)", RFC 7390,
              October 2014.

   [RFC7397] Gilger, J. and H. Tschofenig, "Report from the Smart
              Object Security Workshop", RFC 7397, December 2014.

   [RFC7400]   Bormann, C., "6LoWPAN-GHC: Generic Header Compression for
               IPv6 over Low-Power Wireless Personal Area Networks
               (6LoWPANs)", RFC 7400, November 2014.

   [Tripple-HS]
               Bhargavan, K., Delignat-Lavaud, C., Pironti, A., and P.
               Strub, "Triple Handshakes and Cookie Cutters: Breaking and
               Fixing Authentication over TLS", IEEE Symposium on
               Security and Privacy, pages 98-113, 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 13 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 8-bit binary data encoded Transport Protocol
   Data Units (TPDU).

A.1. Overview

   DTLS adds an additional roundtrip 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 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 PMTU of the SMS
   transport in order to correctly instrument its SaR function. The

PMTU SHALL be 133 bytes if WDP-based multiplexing is used (see
Appendix A.3), 140 bytes otherwise.

It is RECOMMENDED to use the established security context over the
longest possible period (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 Re-Assembly

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 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 ESP/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 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 7 bytes out of the total available for the SMS content.

The receiving side of the communication gets the source address from
the originator address (TP-OA) field of the SMS-DELIVER TPDU. This
way an unique 4-tuple identifying the security association can be
reconstructed at both ends. (When replying to its DTLS peer, the
sender will swaps the TP-OA and 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 10 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 10: AES-128-CCM-8 DTLS Record Layer Per-Packet Overhead.

The DTLS record layer header has 13 octets and consists of

o 1 octet content type field,

o 2 octet version field,

o 2 octet epoch field,

o 6 octet sequence number,

o 2 octet length field.

The "nonce" input to the AEAD algorithm is exactly that of [RFC5288],
i.e., 12 bytes long. It consists of a 4 octet salt and an 8 octet
nonce. The salt is the "implicit" part of the nonce and is not sent
in the packet. Since the nonce_explicit may be the 8 octet sequence
number and, in DTLS, it is the 8 octet epoch concatenated with the 6
octet sequence number.

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 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.

Appendix C. **DTLS Fragmentation**

[Editor's Note: Proposed text that requires discussion.]

Section 4.2.3 of [RFC6347] advises DTLS implementations to not
produce overlapping fragments, but 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 path MTU (PMTU) estimation may
be traded for shorter handshake completion time. This approach may
be beneficial in unconstrained networks where a PMTU of 1280 bytes
can be pretty much universally assumed. However, when the handshake
is carried over a narrow-band radio technology, such as IEEE 802.15.4
or GSM-SMS, and the client is lacking reliable PMTU data to inform
fragmentation (e.g., using [RFC1981] or [RFC1191]) can put a cost on

the constrained implementation in terms of memory (due to re-
buffering) and latency (due to re-transmission) much higher than the
benefit that it would get from a shorter handshake.

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

o for clients (handshake initiators), to perform PMTU discovery
   towards the server before handshake starts, and not rely on any
   guesses (unless the network path characteristics are reliably
   known from another source);

o for servers, to mirror the fragment size selected by their
   clients.

Authors' Addresses

   Hannes Tschofenig (editor)
   ARM Ltd.
   110 Fulbourn Rd
   Cambridge CB1 9NJ
   Great Britain

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

   Thomas Fossati
   Alcatel-Lucent
   3 Ely Road
   Milton, Cambridge CB24 6DD
   UK

   Email: thomas.fossati@alcatel-lucent.com