

Transport Layer Security (TLS) Next Protocol Negotiation Extension
draft-agl-tls-nextprotoneg-04

Abstract

This document describes a Transport Layer Security (TLS) extension for application layer protocol negotiation. This allows the application layer to negotiate which protocol should be performed over the secure connection.

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 October 31, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	2
2. Requirements Notation	2
3. Next Protocol Negotiation Extension	2
4. Protocol selection	4
5. Design discussion	5

6. Security considerations	5
7. IANA Considerations	5
8. Acknowledgments	5
9. References	6
9.1. Normative References	6
9.2. Informative References	6
Author's Address	6

1. Introduction

The Next Protocol Negotiation extension (NPN) is currently used to negotiate the use of SPDY [spdy] as an application level protocol on port 443, and to perform SPDY version negotiation. However, it is not SPDY specific in any way.

Designers of new application level protocols are faced with a problem: there are no good options for establishing a clean transport for a new protocol and negotiating its use. Negotiations on port 80 will run afoul of intercepting proxies. Ports other than 80 and 443 are likely to be firewalled without any fast method of detection, and are also unlikely to traverse HTTP proxies with CONNECT. Negotiating on port 443 is possible, but may run afoul of MITM proxies and also uses a round trip for negotiation on top of the round trips for establishing the TLS connection. Negotiation at that level is also dependent on the application level protocol, i.e. the real world tolerance of servers to HTTP Upgrade requests.

Next Protocol Negotiation allows application level protocols to be negotiated without additional round trips and with clean fallback in the case of an unsupportive MITM proxy.

2. Requirements Notation

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

3. Next Protocol Negotiation Extension

A new extension type ("next_protocol_negotiation(TBD)") is defined and MAY be included by the client in its "ClientHello" message. If, and only if, the server sees this extension in the "ClientHello", it MAY choose to echo the extension in its "ServerHello".

```
enum {  
    next_protocol_negotiation(TBD), (65535)  
} ExtensionType;
```

The "extension_data" field of a "next_protocol_negotiation" extension in a "ClientHello" MUST be empty.

The "extension_data" field of a "next_protocol_negotiation" extension in a "ServerHello" contains an optional list of protocols advertised by the server. Protocols are named by opaque, non-empty byte strings and the list of protocols is serialized as a concatenation of 8-bit, length prefixed byte strings. Implementations MUST ensure that the empty string is not included and that no byte strings are truncated.

A new handshake message type ("encrypted_extensions(TBD)") is defined. If the server included a "next_protocol_negotiation" extension in its "ServerHello" message, the client MUST send a "EncryptedExtensions" message after its "ChangeCipherSpec" and before its "Finished" message.

```
enum {
    encrypted_extensions(TBD), (65535)
} HandshakeType;
```

Therefore a full handshake with "EncryptedExtensions" has the following flow (contrast with section 7.3 of RFC 5246 [RFC5246]):

Client	Server
ClientHello (NPN extension) ----->	
	ServerHello (NPN extension & list of protocols)
	Certificate*
	ServerKeyExchange*
	CertificateRequest*
	ServerHelloDone
	<-----
Certificate*	
ClientKeyExchange	
CertificateVerify*	
[ChangeCipherSpec]	
EncryptedExtensions	
Finished ----->	
	[ChangeCipherSpec]
	<----- Finished
Application Data <----->	Application Data

An abbreviated handshake with "EncryptedExtensions" has the following flow:

```

Client                                                    Server

ClientHello (NPN extension)  ----->

                                ServerHello
                                (NPN extension &
                                list of protocols)
                                [ChangeCipherSpec]
                                Finished
                                <-----
[ChangeCipherSpec]
EncryptedExtensions
Finished                ----->
Application Data        <----->    Application Data

```

The "EncryptedExtensions" message contains a series of "Extension" structures (see section 7.4.1.4 of RFC 5246 [RFC5246])

If the server included a "next_protocol_negotiation" extension in its "ServerHello" message, the client MUST include an "Extension" with "extension_type" equal to "next_protocol_negotiation(TBD)". The "extension_data" of which has the following format:

```

struct {
    opaque selected_protocol<0..255>;
    opaque padding<0..255>;
} NextProtocolNegotiationEncryptedExtension;

```

The contents of "selected_protocol" are an opaque protocol string, but need not have been advertised by the server. The length of "padding" SHOULD be $32 - ((\text{len}(\text{selected_protocol}) + 2) \% 32)$. Note that $\text{len}(\text{selected_protocol})$ does not include its length prefix.

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

For the same reasons, after a handshake has been performed for a given connection, renegotiations on the same connection MUST NOT include the "next_protocol_negotiation" extension.

4. Protocol selection

It's expected that a client will have a list of protocols that it supports, in preference order, and will only select a protocol if the server supports it. In that case, the client SHOULD select the first protocol advertised by the server that it also supports. In the event that the client doesn't support any of server's protocols, or the server doesn't advertise any, it SHOULD select the first protocol that it supports.

There may be cases where the client knows, via other means, that a server supports an unadvertised protocol. In these cases the client can simply select that protocol.

5. Design discussion

NPN is an outlier from TLS in several respects: firstly that it introduces a handshake message between the "ChangeCipherSpec" and "Finished" message, that the handshake message is padded, and that the negotiation isn't done purely with the hello messages. All these aspects of the protocol are intended to prevent middle-ware discrimination based on the negotiated protocol and follow the general principle that anything that can be encrypted, should be encrypted. The server's list of advertised protocols is in the clear as a compromise between performance and robustness.

6. Security considerations

The server's list of supported protocols is still advertised in the clear with this extension. This may be undesirable for certain protocols (such as Tor [tor]) where one could imagine that hostile networks would terminate any TLS connection with a server that advertised such a capability. In this case, clients may wish to opportunistically select a protocol that wasn't advertised by the server. However, the workings of such a scheme are outside the scope of this document.

7. IANA Considerations

This document requires IANA to update its registry of TLS extensions to assign an entry referred to here as "next_protocol_negotiation".

This document also requires IANA to update its registry of TLS handshake types to assign an entry referred to here as "encrypted_extensions".

This document also requires IANA to create a registry of TLS Next Protocol Negotiation protocol strings on a first come, first served basis, initially containing the following entries:

- o "http/1.1": HTTP/1.1[RFC2616]
- o "spdy/1": (obsolete) SPDY version 1
- o "spdy/2": SPDY version 2
- o "spdy/3": SPDY version 3

8. Acknowledgments

This document benefited specifically from discussions with Wan-Teh Chang and Nagendra Modadugu.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

9.2. Informative References

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P. and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.
- [tor] Dingledine, R., Matthewson, N. and P. Syverson, "Tor: The Second-Generation Onion Router", August 2004.
- [spdy] Belshe, M. and R. Peon, "SPDY Protocol (Internet Draft)", Feb 2012.

Author's Address

Adam Langley
Google Inc

Email: agl@google.com

Transport Layer Security
Internet-Draft
Intended status: Informational
Expires: August 6, 2015

D. Thakore
CableLabs
February 2, 2015

Transport Layer Security (TLS) Authorization Using DTCP Certificate
draft-dthakore-tls-authz-09

Abstract

This document specifies the use of Digital Transmission Content Protection (DTCP) certificates as an authorization data type in the authorization extension for the Transport Layer Security (TLS) Protocol. This is in accordance with the guidelines for authorization extensions as specified in [RFC5878]. As with other TLS extensions, this authorization data can be included in the client and server Hello messages to confirm that both parties support the desired authorization data types. If supported by both the client and the server, DTCP certificates are exchanged in the supplemental data TLS handshake message as specified in RFC4680. This authorization data type extension is in support of devices containing DTCP certificates, issued by the Digital Transmission Licensing Administrator [DTLA].

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 August 6, 2015.

Copyright Notice

Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Introduction	2
1.1. Applicability Statement	3
1.2. Conventions	3
2. Overview	4
2.1. Overview of DTCP Certificates	4
2.2. Overview of Supplemental Data handshake	4
2.3. Overview of authorization extensions	5
2.4. Overview of supplemental data usage for authorization . .	6
3. DTCP Authorization Data Format	6
3.1. DTCP Authorization Type	6
3.2. DTCP Authorization Data	6
3.3. Usage rules for clients to exchange DTCP Authorization data	8
3.4. Usage rules for servers to exchange DTCP Authorization data	8
3.5. TLS message exchange with dtcp_authz_data	8
3.6. Alert Messages	9
4. IANA Considerations	10
5. Security Considerations	10
6. Acknowledgements	11
7. References	11
7.1. Normative References	11
7.2. Informative References	12
Appendix A. Additional Stuff	13
Author's Address	15

1. Introduction

The Transport Layer Security (TLS) protocol (TLS1.0 [RFC2246], TLS1.1 [RFC4346], TLS1.2 [RFC5246]) is being used in an ever increasing variety of operational environments, the most common among which is its use in securing HTTP traffic ([RFC2818]). [RFC5878] introduces extensions that enable TLS to operate in environments where authorization information needs to be exchanged between the client and the server before any protected data is exchanged. The use of these TLS authorization extensions is especially attractive since it allows the client and server to determine the type of protected data to exchange based on the authorization information received in the extensions.

A substantial number of deployed consumer electronics devices such as televisions, tablets, game consoles, set-top boxes and other multimedia devices contain [DTLA] issued Digital Transmission Content Protection [DTCP] certificates. These DTCP certificates enable secure transmission of premium audio-visual content between devices over various types of links (e.g., DTCP over IP [DTCP-IP]). These DTCP certificates can also be used to verify device functionality (e.g., supported device features)

This document describes the format and necessary identifiers to exchange DTCP certificates within the supplemental data message (see [RFC4680]) while negotiating a TLS session. The DTCP certificates are then used independent of their use for content protection (e.g., to verify supported features) and the corresponding DTCP Authentication and Key Exchange (AKE) protocol. This communication allows either the client or the server or both to perform certain actions or provide specific services. The actual semantics of the authorization decision by the client/server are beyond the scope of this document. The DTCP certificate, which is not an X.509 certificate, can be cryptographically tied to the X.509 certificate being used during the TLS tunnel establishment by an EC-DSA [DTCP] signature.

1.1. Applicability Statement

DTCP-enabled consumer electronics devices (eg. televisions, game consoles) use DTCP certificates for secure transmission of audio-visual content. The Authentication and Key Exchange (AKE) protocol defined in [DTCP] is used to exchange DTCP Certificates and allows a device to be identified and authenticated based on the information in the DTCP Certificate. However these DTCP-enabled devices offer additional functionality (e.g., via HTML5 User Agents or web enabled applications) that is distinct from its capability to transmit and play audio-visual content. The mechanism outlined in this document allows a DTCP-enabled consumer electronics device to authenticate and authorize using its DTCP Certificate when accessing services over the internet; for example ,web applications on televisions that can enable value-added services. This is anticipated to be very valuable since there are a considerable number of such devices. The re-use of well-known web security will also keep such communication consistent with existing standards and best practices.

1.2. Conventions

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

2. Overview

2.1. Overview of DTCP Certificates

DTCP certificates issued by [DTLA] to DTLA-compliant devices come in three general variations (see [DTCP], Section 4.2.3.1)

- * Restricted Authentication device certificate format (Format 0):
Typically issued to devices with limited computation resources.

- * Baseline Full Authentication device certificate format (Format 1):

This is the most commonly issued certificate format. Format 1 certificates include a unique Device ID and device EC-DSA public/private key pair generated by the DTLA. (See Section 4.3 of [DTCP])

- * Extended Full Authentication device certificate format (Format 2):

This is issued to devices that possess additional functions (e.g., additional channel ciphers, specific device properties). The presence of these additional functions is indicated by the device capability mask as specified in Section 4.2.3.2 of the DTCP specification [DTCP]. Format 2 certificates also include a unique Device ID and device EC-DSA public/private key pair generated by the DTLA. (See Section 4.3 of [DTCP])

The mechanism specified in this document allows only Format 1 and Format 2 DTCP certificates to be exchanged in the supplemental data message since it requires the use of the EC-DSA private key associated with the certificate.

2.2. Overview of Supplemental Data handshake

Figure 1 illustrates the exchange of SupplementalData message during the TLS handshake as specified in [RFC4680] and is repeated here for convenience:

```

Client                                     Server

ClientHello (with extensions) ----->

                                     ServerHello(with extensions)
                                     SupplementalData*
                                     Certificate*
                                     ServerKeyExchange*
                                     CertificateRequest*
<----- ServerHelloDone

SupplementalData*
Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished ----->
                                     [ChangeCipherSpec]
<----- Finished
Application Data <-----> Application Data

* Indicates optional or situation-dependent messages that are
  not always sent.

[] Indicates that ChangeCipherSpec is an independent TLS
  protocol content type; it is not a TLS handshake message.

```

TLS handshake message exchange with Supplemental Data

Figure 1

2.3. Overview of authorization extensions

[RFC5878] defines two authorization extension types that are used in the ClientHello and ServerHello messages and are repeated below for convenience:

```

enum {
    client_authz(7), server_authz(8), (65535)
} ExtensionType;

```

A client uses the client_authz and server_authz extensions in the ClientHello message to indicate that it will send client

authorization data and receive server authorization data respectively in the SupplementalData messages. A server uses the extensions in a similar manner in its ServerHello message. [RFC5878] also establishes a registry that is maintained by IANA for registering authorization data formats. This document defines a new authorization data type for both the client_authz and server_authz extensions and allows the client and server to exchange DTCP certificates in the SupplementalData message.

2.4. Overview of supplemental data usage for authorization

Section 3 of [RFC5878] specifies the syntax of the supplemental data message when carrying the authz_data message that is negotiated in the client_authz and/or server_authz types. This document defines a new authorization data format that is used in the authz_data message when sending DTCP Authorization data.

3. DTCP Authorization Data Format

3.1. DTCP Authorization Type

The DTCP Authorization type definition in the TLS Authorization Data Formats registry is:

```
dtcp_authorization(TBA);
```

Note to RFC Editor: Please populate the number assigned by IANA

3.2. DTCP Authorization Data

The DTCP Authorization data is used when the AuthzDataFormat type is dtcp_authorization. The syntax of the authorization data is:

```
struct {  
    opaque random_bytes[32];  
} RandomNonce;  
  
struct {  
    opaque RandomNonce nonce;  
    opaque DTCPCert<0..2^24-1>;  
    opaque ASN.1Cert<0..2^24-1>;  
    opaque signature<0..2^16-1>;  
} dtcp_authz_data;
```

RandomNonce is generated by the server and consists of 32 bytes generated by a high quality secure random number generator. The client always sends back the server generated RandomNonce in its dtcp_authz_data structure. The RandomNonce helps the server in detecting replay attacks. A client can detect replay attacks by associating the ASN.1 Certificate in the dtcp_authz_data structure with the certificate received in the Certificate message of the TLS handshake so a separate nonce for the client is not required.

DTCPCert is the sender's DTCP certificate, see Section 4.2.3.1 of the DTCP Specification [DTCP]

ASN.1Cert is the sender's certificate used to establish the TLS session, i.e. sent in the Certificate or ClientCertificate message using the Certificate structure defined in Section 7.4.2 of [RFC5246].

The DTCPCert and ASN.1Cert are variable length vectors as specified in Section 4.3 of [RFC5246]. Hence the actual length precedes the vector's contents in the byte stream. If the ASN.1Cert is not being sent, the ASN.1Cert_length MUST be zero.

dtcp_authz_data - contains the RandomNonce, DTCP Certificate and the optional ASN.1 Certificate. This is then followed by the digital signature covering the RandomNonce, DTCP Certificate and the ASN.1 certificate (if present). The signature is generated using the private key associated with the DTCP certificate using the Signature Algorithm and Hash Algorithm as specified in Section 4.4 of [DTCP]. This signature provides proof of the possession of the private key by the sender. A sender sending its own DTCP Certificate MUST populate this field. The length of the signature field is determined by the Signature Algorithm and Hash Algorithm as specified in Section 4.4 of [DTCP] and so it is not explicitly encoded in the dtcp_authz_data structure (e.g. The length will be 40 bytes for SHA1+ECDSA algorithm combination).

3.3. Usage rules for clients to exchange DTCP Authorization data

A client includes both the `client_authz` and `server_authz` extensions in the extended client hello message when indicating its desire to exchange DTCP authorization data with the server. Additionally, the client includes the `AuthzDataFormat` type specified in Section 3.1 in the `extension_data` field to specify the format of the authorization data.

A client will receive the server's `dtcp_authz_data` before it sends its own `dtcp_authz_data`. When sending its own `dtcp_authz_data` message, the client includes the same `RandomNonce` that it receives in the server's `dtcp_authz_data` message. Clients **MUST** include its DTCP Certificate in the `dtcp_authz_data` message. Clients **MAY** include its ASN.1 Certificate (certificate in the `ClientCertificate` message) in the `ASN.1Cert` field of the `dtcp_authz_data` to cryptographically tie the `dtcp_authz_data` with its ASN.1Cert being used to establish the TLS session (i.e. sent in the `ClientCertificate` message).

3.4. Usage rules for servers to exchange DTCP Authorization data

A server responds with both the `client_authz` and `server_authz` extensions in the extended server hello message when indicating its desire to exchange `dtcp_authorization` data with the client. Additionally, the server includes the `AuthzDataFormat` type specified in Section 3.1 in the `extension_data` field to specify the format of the `dtcp_authorization` data. A client may or may not include an ASN.1 Certificate during the TLS handshake. However, the server will not know that at the time of sending the `SupplementalData` message. Hence a server **MUST** generate and populate the `RandomNonce` in the `dtcp_authz_data` message. If the client's hello message does not contain both the `client_authz` and `server_authz` extensions with `dtcp_authorization` type, the server **MUST NOT** include support for `dtcp_authorization` data in its hello message. A server **MAY** include its DTCP Certificate in the `dtcp_authz_data` message. If the server does not send a DTCP Certificate, it will send only the `RandomNonce` in its `dtcp_authz_data` message. If the server includes its DTCP Certificate, it **MUST** also include its server certificate (sent in the `TLS Certificate` message) in the `certs` field to cryptographically tie its `dtcp_authz_data` with the ASN.1 Certificate used in the TLS session being established. This also helps the client in detecting replay attacks.

3.5. TLS message exchange with `dtcp_authz_data`

Based on the usage rules in the sections above, Figure 2 (Figure 2) below provides one possible TLS message exchange where the client

sends its DTCP Certificate to the server within the dtcp_authz_data message.

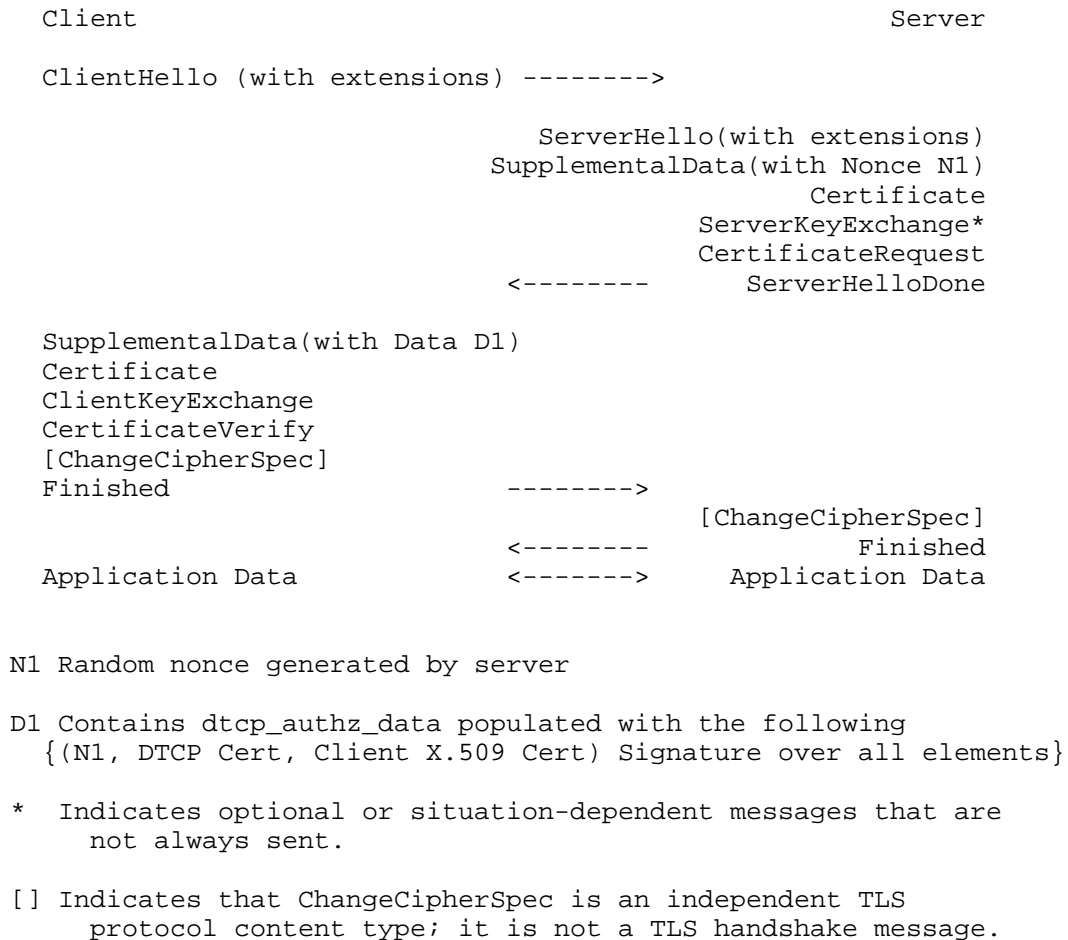


Figure 2

3.6. Alert Messages

This document reuses TLS Alert messages for any errors that arise during authorization processing, and reuses the AlertLevels as specified in [RFC5878]. Additionally the following AlertDescription values are used to report errors in dtcp_authorization processing:

unsupported_extension:

During processing of dtcp_authorization, a client uses this when it receives a server hello message that includes support for dtcp_authorization in only one of client_authz or server_authz but not in both the extensions.

This message is always fatal.

Note: Completely omitting the dtcp_authorization extension and/or omitting the client_authz and server_authz completely is allowed and should not constitute the reason that this alert is sent.

certificate_unknown:

During processing of dtcp_authorization, a client or server uses this when it has received an X.509 certificate in the dtcp_authorization data and that X.509 certificate does not match the certificate sent in the corresponding ClientCertificate or Certificate message.

4. IANA Considerations

This document proposes a new entry to be registered in the IANA-maintained TLS Authorization Data Formats registry for dtcp_authorization(TBA). This registry is defined in [RFC5878] and defines two ranges: one is IETF review and the other is specification required. The value for dtcp_authorization should be assigned via [RFC5226] Specification Required. The extension defined in this document is compatible with DTLS [RFC6347] and the registry assignment should be marked "Y" for DTLS-OK.

Note to RFC Editor: Please populate the number assigned by IANA in TBA above.

5. Security Considerations

The dtcp_authorization data as specified in this document carries the DTCP Certificate that identifies the associated device. Inclusion of the X.509 Certificate being used to establish a TLS Session in the dtcp_authorization data allows an application to cryptographically tie them. However a TLS Client is not required to (and may not possess) an X.509 Certificate. In this case, the dtcp_authorization data exchange is prone to a man-in-the-middle attack. In such situations, a TLS server MUST deny access to the application features dependent on the DTCP Certificate or use a double handshake. The double handshake mechanism is also vulnerable to the TLS MITM Renegotiation exploit as explained in [RFC5746]. In order to address this vulnerability, clients and servers MUST use the

secure_renegotiation extension as specified in [RFC5746] when exchanging dtcp_authorization data. Additionally, the renegotiation is also vulnerable to the Triple Handshake exploit. To mitigate this, servers MUST use the same ASN.1 certificate during renegotiation as the one used in the initial handshake.

It should be noted that for double handshake to succeed, any extension (e.g., TLS Session Ticket [RFC5077]) that results in the TLS Handshake sequence being modified may result in failure to exchange SupplementalData.

Additionally the security considerations specified in [RFC5878] and [RFC5246] apply to the extension specified in this document. In addition, the dtcp_authorization data may be carried along with other supplemental data or some other authorization data and that information may require additional protection. Finally, implementers should also reference [DTCP] and [DTCP-IP] for more information regarding DTCP certificates, their usage and associated security considerations.

6. Acknowledgements

The author wishes to thank Mark Brown, Sean Turner, Sumanth Channabasappa; and the Chairs (EKR, Joe Saloway) and members of the TLS Working Group who provided feedback and comments on one or more revisions of this document.

This document derives its structure and much of its content from [RFC4680], [RFC5878] and [RFC6042].

7. References

7.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999, <<http://tools.ietf.org/html/rfc2246>>.
- [RFC4346] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.1", RFC 4346, April 2006, <<http://tools.ietf.org/html/rfc4346>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The TLS Protocol Version 1.2", RFC 5246, August 2008, <<http://tools.ietf.org/html/rfc5246>>.

- [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, February 2010, <<http://tools.ietf.org/html/rfc5746>>.
- [RFC4680] Santesson, S., "TLS Handshake Message for Supplemental Data", September 2006, <<http://tools.ietf.org/html/rfc4680>>.
- [RFC5878] Brown, M. and R. Housley, "Transport Layer Security (TLS) Authorization Extensions", RFC 5878, May 2010, <<http://tools.ietf.org/html/rfc5878>>.
- [RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, Jan 2012, <<http://tools.ietf.org/html/rfc6347>>.
- [DTCP] Digital Transmission Licensing Administrator, "Digital Transmission Content Protection", <<http://www.dtcp.com/documents/dtcp/info-20130605-dtcp-v1-rev-1-7-ed2.pdf>>.
- [DTCP-IP] Digital Transmission Licensing Administrator, "DTCP Volume 1 Supplement E", <<http://www.dtcp.com/documents/dtcp/info-20130605-dtcp-vlse-ip-rev-1-4-ed3.pdf>>.

7.2. Informative References

- [RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [DTLA] Digital Transmission Licensing Administrator, "DTLA", <<http://www.dtcp.com>>.
- [RFC2818] Rescorla, E., "HTTP over TLS", RFC 2818, May 2000, <<http://tools.ietf.org/html/rfc2818>>.
- [RFC5077] Salowey, J. and P. Eronen, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008, <<http://tools.ietf.org/html/rfc5077>>.

[RFC6042] Keromytis, A., "Transport Layer Security (TLS) Authorization Using KeyNote", RFC 6042, October 2010, <<http://tools.ietf.org/html/rfc6042>>.

Appendix A. Additional Stuff

This document specifies a TLS authorization data extension that allows TLS clients and servers to exchange DTCP certificates during a TLS handshake exchange. In cases where the supplemental data contains sensitive information, the double handshake technique described in [RFC4680] can be used to provide protection for the supplemental data information. The double handshake specified in [RFC4680] assumes that the client knows the context of the TLS session that is being set up and uses the authorization extensions as needed. Figure 3 illustrates a variation of the double handshake that addresses the case where the client may not have a priori knowledge that it will be communicating with a server capable of exchanging dtcp_authz_data (typical for https connections; see [RFC2818]). In Figure 3 (Figure 3), the client's Hello messages includes the client_authz and server_authz extensions. The server simply establishes an encrypted TLS session with the client in the first handshake by not indicating support for any authz extensions. The server initiates a second handshake by sending a HelloRequest. The second handshake will include server's support for authz extensions which will result in SupplementalData being exchanged.

Alternately, it is also possible to do a double handshake where the server sends the authorization extensions during both the first and the second handshake. Depending on the information received in the first handshake, the server can decide if a second handshake is needed or not.

Double Handshake to protect Supplemental Data

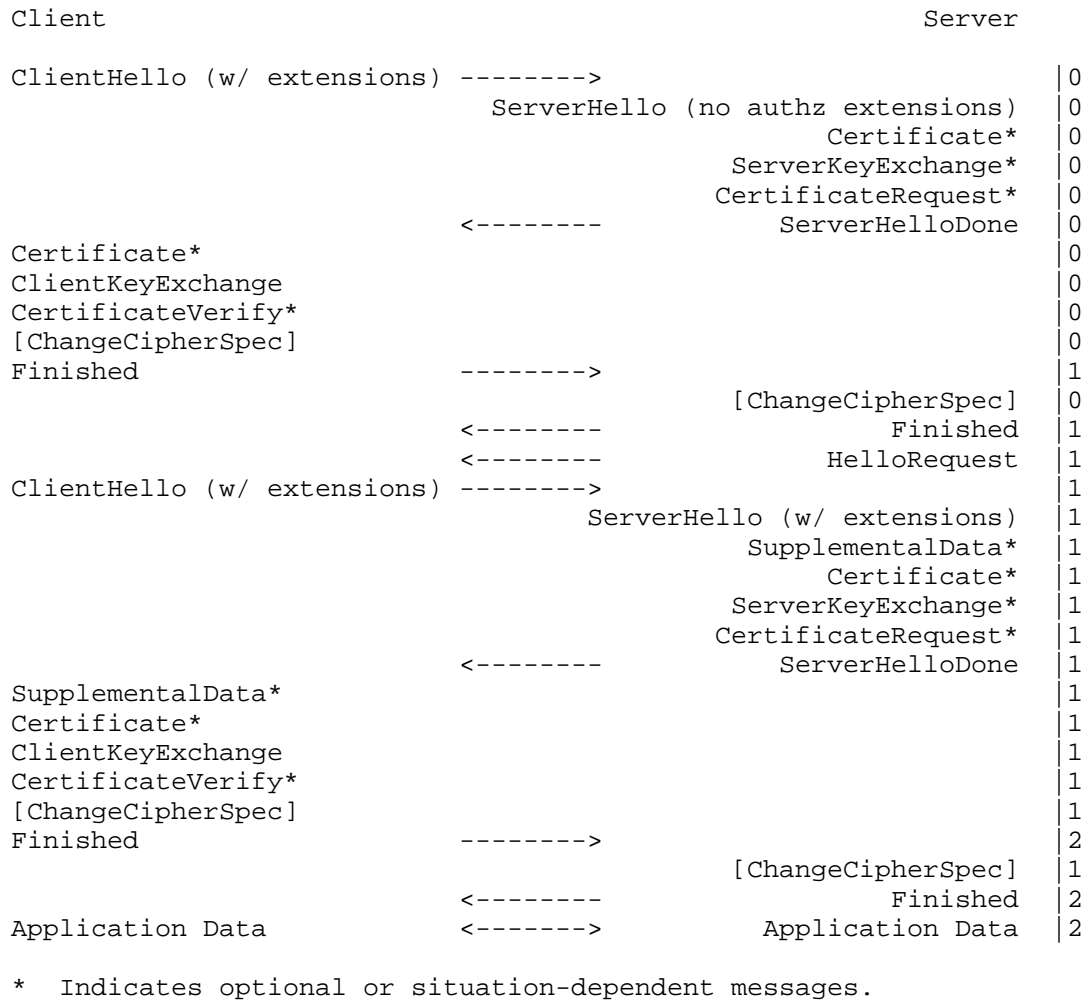


Figure 3

Author's Address

D. Thakore
Cable Television Laboratories, Inc.
858 Coal Creek Circle
Louisville, CO 80023
USA

Email: d.thakore@cablelabs.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 25, 2013

S. Friedl
Cisco Systems, Inc.
A. Popov
Microsoft Corp.
February 21, 2013

Transport Layer Security (TLS) Application Layer Protocol Negotiation
Extension
draft-friedl-tls-applayerprotoneg-02

Abstract

This document describes a Transport Layer Security (TLS) extension for application layer protocol negotiation within the TLS handshake. For instances in which the TLS connection is established over a well known TCP/IP port not associated with the desired application layer protocol, this extension allows the application layer to negotiate which protocol will be used within the TLS session.

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 August 25, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
2. Requirements Language
3. Application Layer Protocol Negotiation
 - 3.1. The Application Layer Protocol Negotiation Extension

3.2.	Protocol Selection
4.	Design Considerations
5.	Security Considerations
6.	IANA Considerations
7.	Acknowledgements
8.	References
8.1.	Normative References
8.2.	Informative References
	Authors' Addresses

1. Introduction

Currently, the Next Protocol Negotiation extension (NPN) is used to establish a SPDY [spdy] protocol session within a TLS RFC 5246 [RFC5246] session on port 443. NPN is not specific to SPDY and can be used to negotiate sessions for a wide variety of protocols within the TLS handshake.

NPN seeks to provide a reliable mechanism for application developers to establish secure sessions for arbitrary protocols without interference from firewalls, HTTP proxies and MITM proxies. It addresses this goal by introducing a protocol negotiation process into the TLS handshake under the constraints that no additional roundtrips be added to the handshake and that the final protocol selection be opaque to the network carrying the TLS session. Within the NPN extension, it is the server that first generates and transmits an offer of supported protocols to the client. The offer is sent as part of the TLS ServerHello message before the [ChangeCipherSpec] subprotocol has been started, therefore the list of protocols supported by the server is transmitted in plaintext. The client chooses a protocol which may or may not appear in the offer from the server and then responds with the definitive protocol selection answer. The client response is sent after the [ChangeCipherSpec] subprotocol has been initiated, so the protocol selected is encrypted in the client response.

In many other application layer protocol negotiation processes, it is the client that first sends an offer of protocols it supports to the server. The server then selects the protocol to be used in the session and includes this answer in the response. RFC 3264 [RFC3264] describes a SDP based offer/answer model which is not proscriptive in terms of which party generates the offer, however in practice it is typically the client generating the offer and the server replying with the answer. This permits the server to act as the definitive entity for selection of the application layer protocol.

This draft proposes an alternative formulation of the NPN protocol which 1) brings the offer/answer negotiation into alignment with the majority of other application layer protocol negotiation standards, 2) allows certificate selection based on the application protocol and 3) makes the definitive protocol selection answer from the server visible to the network, when the parties so desire.

2. Requirements Language

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

3. Application Layer Protocol Negotiation

3.1. The Application Layer Protocol Negotiation Extension

A new extension type ("application_layer_protocol_negotiation(TBD)") is defined and MAY be included by the client in its "ClientHello" message.

```
enum {  
    application_layer_protocol_negotiation(TBD), (65535)  
} ExtensionType;
```

The "extension_data" field of the ("application_layer_protocol_negotiation(TBD)") extension SHALL contain a "ProtocolNameList" value.

opaque ProtocolName<1..2⁸-1>;

```
struct {  
    ProtocolName protocol_name_list<2..216-1>  
} ProtocolNameList;
```

"ProtocolNameList" contains the list of protocols advertised by the client, in descending order of preference. Protocols are named by IANA registered, opaque, non-empty byte strings. Implementations MUST ensure that an empty string is not included and that no byte strings are truncated.

Experimental protocol names, which are not registered by IANA, will start with the following sequence of bytes: 0x65, 0x78, 0x70 ("exp").

Servers that receive a client hello containing the "application_layer_protocol_negotiation" extension, MAY return a suitable protocol selection response to the client. The server will ignore any protocol name that it does not recognize. A new ServerHello extension type ("application_layer_protocol_negotiation(TBD)") MAY be returned to the client within the extended ServerHello message. The "extension_data" field of the ("application_layer_protocol_negotiation(TBD)") extension SHALL be structured the same as described above for the client "extension_data", except that the "ProtocolNameList" MUST contain exactly one "ProtocolName".

The additional content associated with this extension MUST be included in the hash calculations associated with the "Finished" messages.

Therefore, a full handshake with the "application_layer_protocol_negotiation" extension in the ClientHello and ServerHello messages has the following flow (contrast with section 7.3 of RFC 5246 [RFC5246]):

Client		Server
ClientHello	----->	ServerHello
(ALPN extension & list of protocols)		(ALPN extension & selected protocol)
		Certificate*
		ServerKeyExchange*
		CertificateRequest*
	<-----	ServerHelloDone


```

Certificate*
ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished          ----->
                  <----- [ChangeCipherSpec]
                  Finished
Application Data  <-----> Application Data

```

Figure 1

An abbreviated handshake with the "application_layer_protocol_negotiation" extension has the following flow:

Client		Server
ClientHello (ALPN extension & list of protocols)	----->	ServerHello (ALPN extension & selected protocol)
	<-----	[ChangeCipherSpec] Finished
[ChangeCipherSpec] Finished	----->	
Application Data	<----->	Application Data

Figure 2

Unlike many other TLS extensions, this extension does not establish properties of the session, only of the connection. When session resumption or session tickets RFC 5077 [RFC5077] are used, the previous contents of this extension are irrelevant and only the values in the new handshake messages are considered.

3.2. Protocol Selection

It is expected that a server will have a list of protocols that it supports, in preference order, and will only select a protocol if the client supports it. In that case, the server SHOULD select the most highly preferred protocol it supports which is also advertised by the client. In the event that the server supports no protocols that the client advertises, then the server SHALL respond with a fatal "no_application_protocol" alert.

```

enum {
    no_application_protocol(120),
    (255)
} AlertDescription;

```

The "no_application_protocol" fatal alert is only defined for the "application_layer_protocol_negotiation" extension and MUST NOT be sent unless the server has received a ClientHello message containing this extension.

The protocol identified in the "application_layer_protocol_negotiation" extension type in the ServerHello SHALL be definitive for the connection. The server SHALL NOT respond with a selected protocol and subsequently use a different protocol for application data exchange.

4. Design Considerations

The ALPN extension is intended to follow the typical design of TLS protocol extensions. Specifically, the negotiation is performed entirely within the client/server hello exchange in accordance with established TLS architecture. The "application_layer_protocol_negotiation" ServerHello extension is intended to be definitive for the connection and is sent in plaintext to permit network elements to provide differentiated service for the connection when the TCP/IP port number is not definitive for the application layer protocol to be used in the connection. By placing ownership of protocol selection on the server, ALPN facilitates scenarios in which certificate selection or connection rerouting may be based on the negotiated protocol.

Finally, by managing protocol selection in the clear as part of the handshake, ALPN avoids introducing false confidence with respect to the ability to hide the negotiated protocol in advance of establishing the connection. If hiding the protocol is required, then renegotiation after connection establishment, which would provide true TLS security guarantees, would be a preferred methodology.

5. Security Considerations

The ALPN extension does not impact the security of TLS session establishment or application data exchange. ALPN serves to provide an externally visible marker for the application layer protocol associated with the TLS connection. Historically, the application layer protocol associated with a connection could be ascertained from the TCP/IP port number in use.

Encrypting the selected application protocol information and sending it before the Finished messages are exchanged, as done in NPN, does not provide confidentiality guarantees due to the possibility of man-in-the-middle attacks.

6. IANA Considerations

This document requires the IANA to update its registry of TLS extensions to assign an entry referred to here as "application_layer_protocol_negotiation" for extended ClientHello and ServerHello messages.

This document also requires the IANA to create a registry of Application Layer Protocol Negotiation protocol byte strings, initially containing the following entries:

- "http/1.1": HTTP/1.1 [RFC2616];
- "http/2.0": HTTP/2.0;
- "spdy/1": (obsolete) SPDY version 1;
- "spdy/2": SPDY version 2;
- "spdy/3": SPDY version 3.

A namespace will be assigned for experimental protocols, comprising byte strings which start with the following sequence of bytes: 0x65, 0x78, 0x70 ("exp"). Assignments in this namespace do not need IANA

registration.

7. Acknowledgements

This document benefitted specifically from the NPN extension draft authored by Adam Langley of Google and from discussions with Tom Wesselman and Cullen Jennings both of Cisco.

8. References

8.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3264] Rosenberg, J. and H. Schulzrinne, "An Offer/Answer Model with Session Description Protocol (SDP)", RFC 3264, June 2002.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

8.2. Informative References

- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.
- [spdy] Belshe, M. and R. Peon, "SPDY Protocol (Internet Draft)", 2012.

Authors' Addresses

Stephan Friedl
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134
USA

Phone: (720)562-6785
Email: sfriedl@cisco.com

Andrei Popov
Microsoft Corp.
One Microsoft Way
Redmond, WA 98052
USA

Email: andreipo@microsoft.com

TLS
Internet-Draft
Intended status: Standards Track
Expires: March 16, 2013

S. Santesson
3xA Security AB
H. Tschofenig
Nokia Siemens Networks
September 12, 2012

Transport Layer Security (TLS) Cached Information Extension
draft-ietf-tls-cached-info-13.txt

Abstract

Transport Layer Security (TLS) handshakes often include fairly static information, such as the server certificate and a list of trusted Certification Authorities (CAs). This information can be of considerable size, particularly if the server certificate is bundled with a complete certificate path (including all intermediary certificates up to the trust anchor public key).

This document defines an extension that omits the exchange of already available information. The TLS client informs a server of cached information, for example from a previous TLS handshake, allowing the server to omit the already available information.

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 March 16, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents

(<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
2. Terminology	4
3. Cached Information Extension	5
4. Exchange Specification	7
4.1. Fingerprint of the Certificate Chain	7
4.2. Fingerprint for Trusted CAs	8
5. Example	10
6. Security Considerations	12
7. IANA Considerations	13
7.1. New Entry to the TLS ExtensionType Registry	13
7.2. New Registry for CachedInformationType	13
8. Acknowledgments	14
9. References	15
9.1. Normative References	15
9.2. Informative References	15
Authors' Addresses	16

1. Introduction

Transport Layer Security (TLS) handshakes often include fairly static information, such as the server certificate and a list of trusted Certification Authorities (CAs). This information can be of considerable size, particularly if the server certificate is bundled with a complete certificate path (including all intermediary certificates up to the trust anchor public key).

Optimizing the exchange of information to a minimum helps to improve performance in environments where devices are connected to a network with characteristics like low bandwidth, high latency and high loss rate. These types of networks exist, for example, when smart objects are connected using a low power IEEE 802.15.4 radio. For more information about the challenges with smart object deployments please see [RFC6574].

This specification defines a TLS extension that allows a client and a server to exclude transmission of cached information from the TLS handshake.

A typical example exchange may therefore look as follows. First, the TLS exchange executes the usual TLS handshake. It may decide to store the certificate provided by the server for a future exchange. When the TLS client then connects to the TLS server some time in the future, without using session resumption, it then attaches the `cached_information` extension defined in this document to the client hello message to indicate that it had cached the certificate, and it provides the fingerprint of it. If the server's certificate had not changed then the TLS server does not need to send the full certificate to the client again. In case the information had changed, the certificate payload is transmitted to the client to allow the client to update it's state information.

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].

3. Cached Information Extension

This document defines a new extension type (`cached_information(TBD)`), which is used in client hello and server hello messages. The extension type is specified as follows.

```
enum {  
    cached_information(TBD), (65535)  
} ExtensionType;
```

The `extension_data` field of this extension, when included in the client hello, MUST contain the `CachedInformation` structure.

```
enum {  
    certificate_chain(1), trusted_cas(2) (255)  
} CachedInformationType;  
  
struct {  
    CachedInformationType type;  
    HashAlgorithm hash;  
    opaque hash_value<1..255>;  
} CachedObject;  
  
struct {  
    CachedObject cached_info<1..2^16-1>;  
} CachedInformation;
```

When the `CachedInformationType` identifies a `certificate_chain`, then the `hash_value` field MUST include the hash calculated over the `certificate_list` element of the `Certificate` payload provided by the TLS server in an earlier exchange, excluding the three length bytes of the `certificate_list` vector.

When the `CachedInformationType` identifies a `trusted_cas`, then the `hash_value` MUST include a hash calculated over the `certificate_authorities` element of the `CertificateRequest` payload provided by the TLS server in an earlier exchange, excluding the two length bytes of the `certificate_authorities` vector.

The hash algorithm used to calculate hash values is conveyed in the 'hash' field of the `CachedObject` element. The list of registered hash algorithms can be found in the TLS HashAlgorithm Registry, which was created by RFC 5246 [RFC5246]. The value zero (0) for 'none' is not an allowed choice for a hash algorithm and MUST NOT be used.

This document establishes a registry for `CachedInformationType` types and additional values can be added following the policy described in Section 7.

4. Exchange Specification

Clients supporting this extension MAY include the "cached_information" extension in the (extended) client hello, which MAY contain zero or more CachedObject attributes.

Server supporting this extension MAY include the "cached_information" extension in the (extended) server hello, which MAY contain one or more CachedObject attributes. By returning the "cached_information" extension the server indicates that it supports caching of each present CachedObject that matches the specified hash value. The server MAY support other cached objects that are not present in the extension.

Note: Clients may need the ability to cache different values depending on other information in the Client Hello that modify what values the server uses, in particular the Server Name Indication [RFC6066] value.

Following a successful exchange of "cached_information" extensions, the server MAY send fingerprints of the cached information in the handshake exchange as a replacement for the exchange of the full data. Section 4.1 and Section 4.2 defines the syntax of the fingerprinted information.

The handshake protocol MUST proceed using the information as if it was provided in the handshake protocol. The Finished message MUST be calculated over the actual data exchanged in the handshake protocol. That is, the Finished message will be calculated over the hash values of cached information objects and not over the cached information that were omitted from transmission.

The server MUST NOT include more than one fingerprint for a single information element, i.e., at maximum only one CachedObject structure per replaced information is provided.

4.1. Fingerprint of the Certificate Chain

When an object of type 'certificate_chain' is provided in the client hello, the server MAY send a fingerprint instead of the complete certificate chain as shown below.

The original handshake message syntax is defined in RFC 5246 [RFC5246] and has the following structure:

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    ASN.1Cert certificate_list<0..2^24-1>;
} Certificate;
```

By using the extension defined in this document the following information is sent:

```
struct {
    CachedObject cached_objects<1..2^24-1>;
} Certificate;
```

The `certificate_list` vector of opaque `ASN.1Cert` elements in the original syntax is replaced with a vector holding `CachedObject` structures as defined in this document.

Note: [I-D.ietf-tls-oob-pubkey] allows a PKIX certificate containing only the `SubjectPublicKeyInfo` instead of the full information typically found in a certificate. Hence, when this specification is used in combination with [I-D.ietf-tls-oob-pubkey] and the negotiated certificate type is a raw public key then the TLS server sends the hashed Certificate payload that contains a `ASN.1Cert` structure of the `SubjectPublicKeyInfo`.

4.2. Fingerprint for Trusted CAs

When a hash for an object of type `'trusted_cas'` is provided in the client hello, the server MAY send a fingerprint instead of the complete certificate authorities information as shown below.

The original handshake message syntax is defined in RFC 5246 [RFC5246] and has the following structure:

```
opaque DistinguishedName<1..2^16-1>;

struct {
    ClientCertificateType certificate_types<1..2^8-1>;
    SignatureAndHashAlgorithm
        supported_signature_algorithms<2^16-1>;
    DistinguishedName certificate_authorities<0..2^16-1>;
} CertificateRequest;
```

By using the extension defined in this document the following

information is sent:

```
struct {  
    ClientCertificateType certificate_types<1..2^8-1>;  
    SignatureAndHashAlgorithm  
        supported_signature_algorithms<2^16-1>;  
    CachedObject cached_objects<1..2^16-1>;  
} CertificateRequest;
```

The `certificate_authorities` vector of opaque `DistinguishedName` elements in the original syntax is replaced with a vector holding `CachedObject` structures as defined in this document.

5. Example

Figure 1 illustrates an example exchange using the TLS cached info extension. In the normal TLS handshake exchange shown in flow (A) the TLS server provides its certificate in the Certificate payload to the client, see step [1]. This allows the client to store the certificate for future use. After some time the TLS client again interacts with the same TLS server and makes use of the TLS cached info extension, as shown in flow (B). The TLS client indicates support for this specification via the `cached_information` extension, see [2], and indicates that it has stored the `certificate_chain` from the earlier exchange. With [3] the TLS server indicates that it also supports this specification and informs the client that it also supports caching of other objects beyond the `'certificate_chain'`, namely `'trusted_cas'` (also defined in this document), and the `'foo-bar'` extension (i.e., an imaginary extension that yet needs to be defined). With [4] the TLS server provides the fingerprint of the certificate chain as described in Section 4.1.

(A) Initial (full) Exchange

```
client_hello ->
               <-  server_hello,
                   certificate, // [1]
                   server_key_exchange,
                   server_hello_done

client_key_exchange,
change_cipher_spec,
finished
               ->
               <-  change_cipher_spec,
                   finished

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

(B) TLS Cached Extension Usage

```
client_hello,
cached_information=(certificate_chain) -> // [2]
               <-  server_hello,
                   cached_information= // [3]
                       (certificate_chain, trusted_cas, foo-bar)
                   certificate, // [4]
                   server_key_exchange,
                   server_hello_done

client_key_exchange,
change_cipher_spec,
finished
               ->
               <-  change_cipher_spec,
                   finished

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

Figure 1: Example Message Exchange

6. Security Considerations

This specification defines a mechanism to reference stored state using a fingerprint. The hash algorithm used in this specification is required to have reasonable random properties in order to provide reasonably unique identifiers. There is no requirement that this hash algorithm must have strong collision resistance.

Caching information in an encrypted handshake (such as a renegotiated handshake) and sending a hash of that cached information in an unencrypted handshake might introduce integrity or data disclosure issues as it enables an attacker to identify if a known object (such as a known server certificate) has been used in previous encrypted handshakes. Information object types defined in this specification, such as server certificates, are public objects and usually not sensitive in this regard, but implementers should be aware if any cached information are subject to such security concerns and in such case SHOULD NOT send a hash over encrypted data in unencrypted handshake.

7. IANA Considerations

7.1. New Entry to the TLS ExtensionType Registry

IANA is requested to add an entry to the existing TLS ExtensionType registry, defined in RFC 5246 [RFC5246], for cached_information(TBD) defined in this document.

7.2. New Registry for CachedInformationType

IANA is requested to establish a registry for TLS CachedInformationType values. The first entries in the registry are

- o certificate_chain(1)
- o trusted_cas(2)

The policy for adding new values to this registry, following the terminology defined in RFC 5226 [RFC5226], is as follows:

- o 0-63 (decimal): Standards Action
- o 64-223 (decimal): Specification Required
- o 224-255 (decimal): reserved for Private Use

8. Acknowledgments

We would like to thank the following persons for your detailed document reviews:

- o Paul Wouters and Nikos Mavrogiannopoulos (December 2011)
- o Rob Stradling (February 2012)
- o Ondrej Mikle in March 2012)

Additionally, we would like to thank the TLS working group chairs, Eric Rescorla and Joe Salowey, as well as the security area directors, Sean Turner and Stephen Farrell, for their feedback and support.

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3874] Housley, R., "A 224-bit One-way Hash Function: SHA-224", RFC 3874, September 2004.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

9.2. Informative References

- [I-D.ietf-tls-oob-pubkey]
Wouters, P., Gilmore, J., Weiler, S., Kivinen, T., and H. Tschofenig, "Out-of-Band Public Key Validation for Transport Layer Security", draft-ietf-tls-oob-pubkey-04 (work in progress), July 2012.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6574] Tschofenig, H. and J. Arkko, "Report from the Smart Object Workshop", RFC 6574, April 2012.

Authors' Addresses

Stefan Santesson
3xA Security AB
Scheelev. 17
Lund 223 70
Sweden

Email: sts@aaa-sec.com

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 10, 2013

Y. Pettersen
February 6, 2013

The TLS Multiple Certificate Status Request Extension
draft-ietf-tls-multiple-cert-status-extension-04

Abstract

This document defines the Transport Layer Security (TLS) Certificate Status Version 2 Extension to allow clients to specify and support multiple certificate status methods. Also defined is a new method based on the Online Certificate Status Protocol (OCSP) that servers can use to provide status information not just about the server's own certificate, but also the status of intermediate certificates in the chain.

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 August 10, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

1. Introduction

The Transport Layer Security (TLS) Extension [RFC6066] framework defines, among other extensions, the Certificate Status Extension that clients can use to request the server's copy of the current status of its certificate. The benefits of this extension include a reduced number of roundtrips and network delays for the client to verify the status of the server's certificate and a reduced load on the certificate issuer's status response servers, thus solving a problem that can become significant when the issued certificate is presented by a frequently visited server.

There are two problems with the existing Certificate Status extension. First, it does not provide functionality to request the status information about intermediate Certification Authority (CA) certificates, which means the client has to request status information through other methods, such as Certificate Revocation Lists (CRLs), thus adding additional delay. Second, the current format of the extension and requirements in the TLS protocol prevents a client from offering the server multiple status methods.

Many CAs are now issuing intermediate CA certificates that not only specify the publication point for their CRLs in a CRL Distribution Point [RFC5280], but also specify a URL for their OCSP [RFC2560] server in Authority Information Access [RFC5280]. Given that client-cached CRLs are frequently out of date, clients would benefit from using OCSP to access up-to-date status information about intermediate CA certificates. The benefit to the issuing CA is less clear, as providing the bandwidth for the OCSP responder can be costly, especially for CAs with many high-traffic subscriber sites, and this cost is a concern for many CAs. There are cases where OCSP requests for a single high-traffic site caused significant network problems

for the issuing CA.

Clients will benefit from the TLS server providing certificate status information regardless of type, not just for the server certificate, but also for the intermediate CA certificates. Combining the status checks into one extension will reduce the roundtrips needed to complete the handshake by the client to just those needed for negotiating the TLS connection. Also, for the Certification Authorities, the load on their servers will depend on the number of certificates they have issued, not on the number of visitors to those sites.

For such a new system to be introduced seamlessly, clients need to be able to indicate support for the existing OCSP Certificate Status method, and a new multiple-OCSP mode.

Unfortunately, the definition of the Certificate Status extension only allows a single Certificate Status extension to be defined in a single extension record in the handshake, and the TLS Protocol [RFC5246] only allows a single record in the extension list for any given extension. This means that it is not possible for clients to indicate support for new methods while still supporting older methods, which would cause problems for interoperability between newer clients and older servers. This will not just be an issue for the multiple status request mode proposed above, but also for any other future status methods that might be introduced. This will be true not just for the current PKIX infrastructure [RFC5280], but also for alternative PKI structures.

The solution to this problem is to define a new extension, `status_request_v2`, with an extended format that allows the client to indicate support for multiple status request methods. This is implemented using a list of `CertificateStatusRequestItem` records in the extension record. As the server will select the single status method based on the selected cipher suite and the certificate presented, no significant changes are needed in the server's extension format.

1.1. Requirements Language

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

2. Multiple Certificate Status Extension

2.1. New extension

The extension defined by this document is indicated by the "status_request_v2" in the ExtensionType enum, which uses the following value:

```
enum {
    status_request_v2(XX), (65535)
} ExtensionType;
```

[[EDITOR: The value used for status_request_v2 has been left as "XX". This value will be assigned when this draft progresses to RFC.]]

2.2. Multiple Certificate Status Request record

Clients that support a certificate status protocol like OCSP may send the status_request_v2 extension to the server in order to use the TLS handshake to transfer such data instead of downloading it through separate connections. When using this extension, the "extension_data" field of the extension SHALL contain a CertificateStatusRequestList where:

```
struct {
    CertificateStatusType status_type;
    uint16 request_length; /* Length of request field in bytes */
    select (status_type) {
        case ocsp: OCSPStatusRequest;
        case ocsp_multi: OCSPStatusRequest;
    } request;
} CertificateStatusRequestItem;

enum { ocsp(1), ocsp_multi(2), (255) } CertificateStatusType;

struct {
    ResponderID responder_id_list<0..2^16-1>;
    Extensions request_extensions;
} OCSPStatusRequest;

opaque ResponderID<1..2^16-1>;
opaque Extensions<0..2^16-1>;

struct {
    CertificateStatusRequestItem certificate_status_req_list<1..2^16-1>;
} CertificateStatusRequestList;
```

In the OCSPStatusRequest, the "ResponderIDs" provides a list of OCSP

responders that the client trusts. A zero-length "responder_id_list" sequence has the special meaning that the responders are implicitly known to the server, e.g., by prior arrangement, or are identified by the certificates used by the server. "Extensions" is a DER encoding [CCITT.X690.2002] of the OCSRP request extensions.

Both "ResponderID" and "Extensions" are DER-encoded ASN.1 types as defined in [RFC2560]. "Extensions" is imported from [RFC5280]. A zero-length "request_extensions" value means that there are no extensions (as opposed to a zero-length ASN.1 SEQUENCE, which is not valid for the "Extensions" type).

In the case of the "id-pkix-ocsp-nonce" OCSRP extension, [RFC2560] is unclear about its encoding; for clarification, the nonce MUST be a DER-encoded OCTET STRING, which is encapsulated as another OCTET STRING (note that implementations based on an existing OCSRP client will need to be checked for conformance to this requirement).

The list of CertificateStatusRequestItem entries MUST be in order of preference.

A server that receive a client hello containing the "status_request_v2" extension MAY return a suitable certificate status response message to the client along with the server's certificate message. If OCSRP is requested, it SHOULD use the information contained in the extension when selecting an OCSRP responder and SHOULD include request_extensions in the OCSRP request.

The server returns a certificate status response along with its certificate by sending a "CertificateStatus" message immediately after the "Certificate" message (and before any "ServerKeyExchange" or "CertificateRequest" messages). If a server returns a "CertificateStatus" message in response to a status_request_v2 request, then the server MUST have included an extension of type "status_request_v2" with empty "extension_data" in the extended server hello. The "CertificateStatus" message is conveyed using the handshake message type "certificate_status" as follows (see also [RFC6066]):

```
struct {
    CertificateStatusType status_type;
    select (status_type) {
        case ocsp: OCSPResponse;
        case ocsp_multi: OCSPResponseList;
    } response;
} CertificateStatus;

opaque OCSPResponse<0..2^24-1>;

struct {
    OCSPResponse ocsp_response_list<1..2^24-1>;
} OCSPResponseList;
```

An "OCSPResponse" element contains a complete, DER-encoded OCSP response (using the ASN.1 [CCITT.X680.2002] type OCSPResponse defined in [RFC2560]). Only one OCSP response, with a length of at least one byte, may be sent for status_type "ocsp".

An "ocsp_response_list" contains a list of "OCSPResponse" elements, as specified above, each containing the OCSP response for the matching corresponding certificate in the server's Certificate TLS handshake message. That is, the first entry is the OCSP response for the first certificate in the Certificate list, the second entry is the response for the second certificate, and so on. The list MAY contain fewer OCSP responses than there were certificates in the Certificate handshake message, but there MUST NOT be more responses than there were certificates in the list. Individual elements of the list MAY have a length of 0 (zero) bytes, if the server does not have the OCSP response for that particular certificate stored, in which case, the client MUST act as if a response was not received for that particular certificate. If the client receives a "ocsp_response_list" that does not contain a response for one or more of the certificates in the completed certificate chain, the client SHOULD attempt to validate the certificate using an alternative retrieval method, such as downloading the relevant CRL; OCSP SHOULD in this situation only be used for the end entity certificate, not intermediate CA certificates, for reasons stated above.

Note that a server MAY also choose not to send a "CertificateStatus" message, even if it has received a "status_request_v2" extension in the client hello message and has sent a "status_request_v2" extension in the server hello message. Additionally, note that that a server MUST NOT send the "CertificateStatus" message unless it received either a "status_request" or "status_request_v2" extension in the client hello message and sent a corresponding "status_request" or "status_request_v2" extension in the server hello message.

Clients requesting an OCSP response and receiving one or more OCSP responses in a "CertificateStatus" message MUST check the OCSP response(s) and abort the handshake if the response is a revoked status or other unacceptable responses (as determined by client policy), with a `bad_certificate_status_response(113)` alert. This alert is always fatal.

If the response is inconclusive, then the client MAY decide to allow the connection if it believes it will have the opportunity to check the validity of the certificate through another means, e.g., by directly querying the issuer's CRL or OCSP responders. The client MUST abort the connection if it needs to engage in activities that require trust in the server, and the server certificate has not been sufficiently validated. An example of where the client might wish to continue is with EAP-TLS, where the client can use another mechanism to check the status of a certificate once it obtains network access. In this case, the client could continue with the handshake, but it would be inappropriate for the client to disclose a username and password until it has fully validated the server certificate.

3. IANA Considerations

Section 2.1 defines the new TLS Extension `status_request_v2` enum, which should be added to the ExtensionType Values list in the IANA Transport Layer Security (TLS) Extensions registry.

Section 2.2 describes a TLS CertificateStatusType Registry to be maintained by the IANA. The new registry is called TLS Certificate Status Types and should be defined under the Transport Layer Security (TLS) Extensions registry. CertificateStatusType values are to be assigned via IETF Review as defined in [RFC5226]. The initial registry corresponds to the definition of "ExtensionType" in Section 2.2.

Value	Description	Reference
1	ocsp	[This RFC]
2	ocsp_multi	[This RFC]

4. Security Considerations

General Security Considerations for TLS Extensions are covered in [RFC5246]. Security Considerations for the particular extension specified in this document are given below. In general, implementers should continue to monitor the state of the art and address any weaknesses identified.

4.1. Security Considerations for status_request_v2

If a client requests an OCSP response, it must take into account that an attacker's server using a compromised key could (and probably would) pretend not to support the extension. In this case, a client that requires OCSP validation of certificates SHOULD either contact the OCSP server directly or abort the handshake.

Use of the OCSP nonce request extension (id-pkix-ocsp-nonce) may improve security against attacks that attempt to replay OCSP responses; see Section 4.4.1 of [RFC2560] for further details.

The security considerations of [RFC2560] apply to OCSP requests and responses.

5. Acknowledgements

This document is based on [RFC6066] authored by Donald Eastlake 3rd.

6. Normative References

- [CCITT.X680.2002] International International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", CCITT Recommendation X.680, July 2002.
- [CCITT.X690.2002] International International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security

(TLS) Protocol Version 1.2", RFC 5246, 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.

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

Author's Address

Yngve N. Pettersen

Email: yngve@spec-work.net

TLS
Internet-Draft
Intended status: Standards Track
Expires: August 19, 2013

P. Wouters, Ed.
Red Hat
H. Tschofenig, Ed.
Nokia Siemens Networks
J. Gilmore

S. Weiler
SPARTA, Inc.
T. Kivinen
AuthenTec
February 15, 2013

Out-of-Band Public Key Validation for Transport Layer Security (TLS)
draft-ietf-tls-oob-pubkey-07.txt

Abstract

This document specifies a new certificate type for exchanging raw public keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) for use with out-of-band public key validation. Currently, TLS authentication can only occur via X.509-based Public Key Infrastructure (PKI) or OpenPGP certificates. By specifying a minimum resource for raw public key exchange, implementations can use alternative public key validation methods.

One such alternative public key validation method is offered by the DNS-Based Authentication of Named Entities (DANE) together with DNS Security. Another alternative is to utilize pre-configured keys, as is the case with sensors and other embedded devices. The usage of raw public keys, instead of X.509-based certificates, leads to a smaller code footprint.

This document introduces the support for raw public keys in TLS.

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 August 19, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	4
2. Terminology	4
3. New TLS Extension	5
4. TLS Handshake Extension	8
4.1. Client Hello	8
4.2. Server Hello	9
4.3. Certificate Request	9
4.4. Other Handshake Messages	9
4.5. Client authentication	9
5. Examples	10
6. Security Considerations	12
7. IANA Considerations	13
8. Acknowledgements	13
9. References	14
9.1. Normative References	14
9.2. Informative References	14
Appendix A. Example Encoding	15
Authors' Addresses	16

1. Introduction

Traditionally, TLS server public keys are obtained in PKIX containers in-band using the TLS handshake and validated using trust anchors based on a [PKIX] certification authority (CA). This method can add a complicated trust relationship that is difficult to validate. Examples of such complexity can be seen in [Defeating-SSL].

Alternative methods are available that allow a TLS client to obtain the TLS server public key:

- o The TLS server public key is obtained from a DNSSEC secured resource records using DANE [RFC6698].
- o The TLS server public key is obtained from a [PKIX] certificate chain from an Lightweight Directory Access Protocol (LDAP) [LDAP] server.
- o The TLS client and server public key is provisioned into the operating system firmware image, and updated via software updates.

Some smart objects use the UDP-based Constrained Application Protocol (CoAP) [I-D.ietf-core-coap] to interact with a Web server to upload sensor data at a regular intervals, such as temperature readings. CoAP [I-D.ietf-core-coap] can utilize DTLS for securing the client-to-server communication. As part of the manufacturing process, the embedded device may be configured with the address and the public key of a dedicated CoAP server, as well as a public key for the client itself. The usage of X.509-based PKIX certificates [PKIX] may not suit all smart object deployments and would therefore be an unnecessary burden.

The Transport Layer Security (TLS) Protocol Version 1.2 [RFC5246] provides a framework for extensions to TLS as well as guidelines for designing such extensions. This document registers a new value to the IANA certificate types registry for the support of raw public keys.

2. Terminology

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

3. New TLS Extension

This section describes the changes to the TLS handshake message contents when raw public key certificates are to be used. Figure 4 illustrates the exchange of messages as described in the sub-sections below. The client and the server exchange make use of two new TLS extensions, namely 'client_certificate_type' and 'server_certificate_type', and an already available IANA TLS Certificate Type registry [TLS-Certificate-Types-Registry] to indicate their ability and desire to exchange raw public keys. These raw public keys, in the form of a SubjectPublicKeyInfo structure, are then carried inside the Certificate payload. The Certificate and the SubjectPublicKeyInfo structure is shown in Figure 1.

```
opaque ASN.1Cert<1..2^24-1>;

struct {
    select(certificate_type){
        // certificate type defined in this document.
        case RawPublicKey:
            opaque ASN.1_subjectPublicKeyInfo<1..2^24-1>;

        // X.509 certificate defined in RFC 5246
        case X.509:
            ASN.1Cert certificate_list<0..2^24-1>;

        // Additional certificate type based on TLS
        // Certificate Type Registry
    };
} Certificate;
```

Figure 1: TLS Certificate Structure.

The SubjectPublicKeyInfo structure is defined in Section 4.1 of RFC 5280 [PKIX] and does not only contain the raw keys, such as the public exponent and the modulus of an RSA public key, but also an algorithm identifier. The structure, as shown in Figure 2, is encoded in an ASN.1 format and therefore contains length information as well. An example is provided in Appendix A.

```

SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm            AlgorithmIdentifier,
    subjectPublicKey     BIT STRING }

```

Figure 2: SubjectPublicKeyInfo ASN.1 Structure.

The algorithm identifiers are Object Identifiers (OIDs). RFC 3279 [RFC3279], for example, defines the following OIDs shown in Figure 3.

Key Type	Document	OID
RSA	Section 2.3.1 of RFC 3279	1.2.840.113549.1.1
.....
Digital Signature Algorithm (DSS)	Section 2.3.2 of RFC 3279	1.2.840.10040.4.1
.....
Elliptic Curve Digital Signature Algorithm (ECDSA)	Section 2.3.5 of RFC 3279	1.2.840.10045.2.1

Figure 3: Example Algorithm Identifiers.

The message exchange in Figure 4 shows the 'client_certificate_type' and 'server_certificate_type' extensions added to the client and server hello messages.

```

client_hello,
client_certificate_type
server_certificate_type  ->

                                <-  server_hello,
                                client_certificate_type,
                                server_certificate_type,
                                certificate,
                                server_key_exchange,
                                certificate_request,
                                server_hello_done

certificate,
client_key_exchange,
certificate_verify,
change_cipher_spec,
finished                                ->

                                <-  change_cipher_spec,
                                finished

Application Data          <----->          Application Data

```

Figure 4: Basic Raw Public Key TLS Exchange.

The semantic of the two extensions is defined as follows:

The 'client_certificate_type' and 'server_certificate_type' sent in the client hello, may carry a list of supported certificate types, sorted by client preference. It is a list in the case where the client supports multiple certificate types. These extension MUST be omitted if the client only supports X.509 certificates. The 'client_certificate_type' sent in the client hello indicates the certificate types the client is able to provide to the server, when requested using a certificate_request message. The 'server_certificate_type' in the client hello indicates the type of certificates the client is able to process when provided by the server in a subsequent certificate payload.

The 'client_certificate_type' returned in the server hello indicates the certificate type found in the attached certificate payload. Only a single value is permitted. The 'server_certificate_type' in the server hello indicates the type of certificates the client is requested to provide in a subsequent certificate payload. The value conveyed in the 'server_certificate_type' MUST be selected from one of the values provided in the 'server_certificate_type' sent in the client hello. If the server does not send a certificate_request payload

or none of the certificates supported by the client (as indicated in the 'server_certificate_type' in the client hello) match the server-supported certificate types the 'server_certificate_type' payload sent in the server hello is omitted.

The "extension_data" field of this extension contains the ClientCertTypeExtension or the ServerCertTypeExtension structure, as shown in Figure 5. The CertificateType structure is an enum with values from TLS Certificate Type Registry.

```
struct {
    select(ClientOrServerExtension)
        case client:
            CertificateType client_certificate_types<1..2^8-1>;
        case server:
            CertificateType client_certificate_type;
    }
} ClientCertTypeExtension;

struct {
    select(ClientOrServerExtension)
        case client:
            CertificateType server_certificate_types<1..2^8-1>;
        case server:
            CertificateType server_certificate_type;
    }
} ServerCertTypeExtension;
```

Figure 5: CertTypeExtension Structure.

No new cipher suites are required to use raw public keys. All existing cipher suites that support a key exchange method compatible with the defined extension can be used.

4. TLS Handshake Extension

4.1. Client Hello

In order to indicate the support of out-of-band raw public keys, clients MUST include the 'client_certificate_type' and 'server_certificate_type' extensions extended client hello message. The hello extension mechanism is described in TLS 1.2 [RFC5246].

4.2. Server Hello

If the server receives a client hello that contains the 'client_certificate_type' and 'server_certificate_type' extensions and chooses a cipher suite then three outcomes are possible:

1. The server does not support the extension defined in this document. In this case the server returns the server hello without the extensions defined in this document.
2. The server supports the extension defined in this document and has at least one certificate type in common with the client. In this case it returns the 'server_certificate_type' and indicates the selected certificate type value.
3. The server supports the extension defined in this document but does not have a certificate type in common with the client. In this case the server terminate the session with a fatal alert of type "unsupported_certificate".

If the TLS server also requests a certificate from the client (via the certificate_request) it MUST include the 'client_certificate_type' extension with a value chosen from the list of client-supported certificates types (as provided in the 'client_certificate_type' of the client hello).

If the client indicated the support of raw public keys in the 'client_certificate_type' extension in the client hello and the server is able to provide such raw public key then the TLS server MUST place the SubjectPublicKeyInfo structure into the Certificate payload. The public key algorithm MUST match the selected key exchange algorithm.

4.3. Certificate Request

The semantics of this message remain the same as in the TLS specification.

4.4. Other Handshake Messages

All the other handshake messages are identical to the TLS specification.

4.5. Client authentication

Client authentication by the TLS server is supported only through authentication of the received client SubjectPublicKeyInfo via an out-of-band method.

5. Examples

Figure 6, Figure 7, and Figure 8 illustrate example exchanges.

The first example shows an exchange where the TLS client indicates its ability to receive and validate raw public keys from the server. In our example the client is quite restricted since it is unable to process other certificate types sent by the server. It also does not have credentials (at the TLS layer) it could send. The 'client_certificate_type' extension indicates this in [1]. When the TLS server receives the client hello it processes the 'client_certificate_type' extension. Since it also has a raw public key it indicates in [2] that it had chosen to place the SubjectPublicKeyInfo structure into the Certificate payload [3]. The client uses this raw public key in the TLS handshake and an out-of-band technique, such as DANE, to verify its validity.

```

client_hello,
server_certificate_type=(RawPublicKey) -> // [1]

      <- server_hello,
      server_certificate_type=(RawPublicKey), // [2]
      certificate, // [3]
      server_key_exchange,
      server_hello_done

client_key_exchange,
change_cipher_spec,
finished                                ->

      <- change_cipher_spec,
      finished

Application Data      <----->      Application Data

```

Figure 6: Example with Raw Public Key provided by the TLS Server

In our second example the TLS client as well as the TLS server use raw public keys. This is a use case envisioned for smart object networking. The TLS client in this case is an embedded device that is configured with a raw public key for use with TLS and is also able to process raw public keys sent by the server. Therefore, it indicates these capabilities in [1]. As in the previously shown example the server fulfills the client's request, indicates this via the "RawPublicKey" value in the server_certificate_type payload, and

provides a raw public key into the Certificate payload back to the client (see [3]). The TLS server, however, demands client authentication and therefore a `certificate_request` is added [4]. The `certificate_type` payload in [2] indicates that the TLS server accepts raw public keys. The TLS client, who has a raw public key pre-provisioned, returns it in the Certificate payload [5] to the server.

```

client_hello,
client_certificate_type=(RawPublicKey) // [1]
server_certificate_type=(RawPublicKey) // [1]
->
    <-  server_hello,
        server_certificate_type=(RawPublicKey)//[2]
        certificate, // [3]
        client_certificate_type=(RawPublicKey)//[4]
        certificate_request, // [4]
        server_key_exchange,
        server_hello_done

certificate, // [5]
client_key_exchange,
change_cipher_spec,
finished
->
    <-  change_cipher_spec,
        finished

Application Data      <----->      Application Data

```

Figure 7: Example with Raw Public Key provided by the TLS Server and the Client

In our last example we illustrate a combination of raw public key and X.509 usage. The client uses a raw public key for client authentication but the server provides an X.509 certificate. This exchange starts with the client indicating its ability to process X.509 certificates provided by the server, and the ability to send raw public keys (see [1]). The server provides the X.509 certificate in [3] with the indication present in [2]. For client authentication the server indicates in [4] that it selected the raw public key format and requests a certificate from the client in [5]. The TLS client provides a raw public key in [6] after receiving and processing the TLS server hello message.

```

client_hello,
server_certificate_type=(X.509)
client_certificate_type=(RawPublicKey) // [1]
->
<- server_hello,
    server_certificate_type=(X.509)//[2]
    certificate, // [3]
    client_certificate_type=(RawPublicKey)//[4]
    certificate_request, // [5]
    server_key_exchange,
    server_hello_done
certificate, // [6]
client_key_exchange,
change_cipher_spec,
finished
->
<- change_cipher_spec,
    finished

Application Data      <----->      Application Data

```

Figure 8: Hybrid Certificate Example

6. Security Considerations

The transmission of raw public keys, as described in this document, provides benefits by lowering the over-the-air transmission overhead since raw public keys are quite naturally smaller than an entire certificate. There are also advantages from a codesize point of view for parsing and processing these keys. The cryptographic procedures for associating the public key with the possession of a private key also follows standard procedures.

The main security challenge is, however, how to associate the public key with a specific entity. This information will be needed to make authorization decisions. Without a secure binding, man-in-the-middle attacks may be the consequence. This document assumes that such binding can be made out-of-band and we list a few examples in Section 1. DANE [RFC6698] offers one such approach. If public keys are obtained using DANE, these public keys are authenticated via DNSSEC. Pre-configured keys is another out of band method for authenticating raw public keys. While pre-configured keys are not suitable for a generic Web-based e-commerce environment such keys are a reasonable approach for many smart object deployments where there is a close relationship between the software running on the device and the server-side communication endpoint. Regardless of the chosen

mechanism for out-of-band public key validation an assessment of the most suitable approach has to be made prior to the start of a deployment to ensure the security of the system.

7. IANA Considerations

IANA is asked to register a new value in the "TLS Certificate Types" registry of Transport Layer Security (TLS) Extensions [TLS-Certificate-Types-Registry], as follows:

Value: 2
Description: Raw Public Key
Reference: [[THIS RFC]]

This document asks IANA to allocate two new TLS extensions, "client_certificate_type" and "server_certificate_type", from the TLS ExtensionType registry defined in [RFC5246]. These extensions are used in both the client hello message and the server hello message. The new extension type is used for certificate type negotiation. The values carried in these extensions are taken from the TLS Certificate Types registry [TLS-Certificate-Types-Registry].

8. Acknowledgements

The feedback from the TLS working group meeting at IETF#81 has substantially shaped the document and we would like to thank the meeting participants for their input. The support for hashes of public keys has been moved to [I-D.ietf-tls-cached-info] after the discussions at the IETF#82 meeting.

We would like to thank the following persons for their review comments: Martin Rex, Bill Frantz, Zach Shelby, Carsten Bormann, Cullen Jennings, Rene Struik, Alper Yegin, Jim Schaad, Barry Leiba, Paul Hoffman, Robert Cragie, Nikos Mavrogiannopoulos, Phil Hunt, John Bradley, Klaus Hartke, Stefan Jucker, Kovatsch Matthias, Daniel Kahn Gillmor, and James Manger. Nikos Mavrogiannopoulos contributed the design for re-using the certificate type registry. Barry Leiba contributed guidance for the IANA consideration text. Stefan Jucker, Kovatsch Matthias, and Klaus Hartke provided implementation feedback regarding the SubjectPublicKeyInfo structure.

Finally, we would like to thank our TLS working group chairs, Eric Rescorla and Joe Salowey, for their guidance and support.

9. References

9.1. Normative References

- [PKIX] 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.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [TLS-Certificate-Types-Registry] "TLS Certificate Types Registry", February 2013, <<http://www.iana.org/assignments/tls-extensiontype-values#tls-extensiontype-values-2>>.

9.2. Informative References

- [ASN.1-Dump] Gutmann, P., "ASN.1 Object Dump Program", February 2013, <<http://www.cs.auckland.ac.nz/~pgut001/>>.
- [Defeating-SSL] Marlinspike, M., "New Tricks for Defeating SSL in Practice", February 2009, <<http://www.blackhat.com/presentations/bh-dc-09/Marlinspike/BlackHat-DC-09-Marlinspike-Defeating-SSL.pdf>>.
- [I-D.ietf-core-coap] Shelby, Z., Hartke, K., Bormann, C., and B. Frank, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-13 (work in progress), December 2012.
- [I-D.ietf-tls-cached-info] Santesson, S. and H. Tschofenig, "Transport Layer Security (TLS) Cached Information Extension", draft-ietf-tls-cached-info-13 (work in progress), September 2012.
- [LDAP] Sermersheim, J., "Lightweight Directory Access Protocol (LDAP): The Protocol", RFC 4511, June 2006.
- [RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key

Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

Appendix A. Example Encoding

For example, the following hex sequence describes a SubjectPublicKeyInfo structure inside the certificate payload:

	0	1	2	3	4	5	6	7	8	9
1	0x30,	0x81,	0x9f,	0x30,	0x0d,	0x06,	0x09,	0x2a,	0x86,	0x48,
2	0x86,	0xf7,	0x0d,	0x01,	0x01,	0x01,	0x05,	0x00,	0x03,	0x81,
3	0x8d,	0x00,	0x30,	0x81,	0x89,	0x02,	0x81,	0x81,	0x00,	0xcd,
4	0xfd,	0x89,	0x48,	0xbe,	0x36,	0xb9,	0x95,	0x76,	0xd4,	0x13,
5	0x30,	0x0e,	0xbf,	0xb2,	0xed,	0x67,	0x0a,	0xc0,	0x16,	0x3f,
6	0x51,	0x09,	0x9d,	0x29,	0x2f,	0xb2,	0x6d,	0x3f,	0x3e,	0x6c,
7	0x2f,	0x90,	0x80,	0xa1,	0x71,	0xdf,	0xbe,	0x38,	0xc5,	0xcb,
8	0xa9,	0x9a,	0x40,	0x14,	0x90,	0x0a,	0xf9,	0xb7,	0x07,	0x0b,
9	0xe1,	0xda,	0xe7,	0x09,	0xbf,	0x0d,	0x57,	0x41,	0x86,	0x60,
10	0xa1,	0xc1,	0x27,	0x91,	0x5b,	0x0a,	0x98,	0x46,	0x1b,	0xf6,
11	0xa2,	0x84,	0xf8,	0x65,	0xc7,	0xce,	0x2d,	0x96,	0x17,	0xaa,
12	0x91,	0xf8,	0x61,	0x04,	0x50,	0x70,	0xeb,	0xb4,	0x43,	0xb7,
13	0xdc,	0x9a,	0xcc,	0x31,	0x01,	0x14,	0xd4,	0xcd,	0xcc,	0xc2,
14	0x37,	0x6d,	0x69,	0x82,	0xd6,	0xc6,	0xc4,	0xbe,	0xf2,	0x34,
15	0xa5,	0xc9,	0xa6,	0x19,	0x53,	0x32,	0x7a,	0x86,	0x0e,	0x91,
16	0x82,	0x0f,	0xa1,	0x42,	0x54,	0xaa,	0x01,	0x02,	0x03,	0x01,
17	0x00,	0x01								

Figure 9: Example SubjectPublicKeyInfo Structure Byte Sequence.

The decoded byte-sequence shown in Figure 9 (for example using Peter's ASN.1 decoder [ASN.1-Dump]) illustrates the structure, as shown in Figure 10.

Offset	Length	Description
0	3+159:	SEQUENCE {
3	2+13:	SEQUENCE {
5	2+9:	OBJECT IDENTIFIER Value (1 2 840 113549 1 1 1)
	:	PKCS #1, rsaEncryption
16	2+0:	NULL
	:	}
18	3+141:	BIT STRING, encapsulates {
22	3+137:	SEQUENCE {
25	3+129:	INTEGER Value (1024 bit)
157	2+3:	INTEGER Value (65537)
	:	}
	:	}
	:	}

Figure 10: Decoding of Example SubjectPublicKeyInfo Structure.

Authors' Addresses

Paul Wouters (editor)
Red Hat

Email: paul@nohats.ca

Hannes Tschofenig (editor)
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
Email: Hannes.Tschofenig@gmx.net
URI: <http://www.tschofenig.priv.at>

John Gilmore
PO Box 170608
San Francisco, California 94117
USA

Phone: +1 415 221 6524
Email: gnu@toad.com
URI: <https://www.toad.com/>

Samuel Weiler
SPARTA, Inc.
7110 Samuel Morse Drive
Columbia, Maryland 21046
US

Email: weiler@tislabs.com

Tero Kivinen
AuthenTec
Eerikinkatu 28
HELSINKI FI-00180
FI

Email: kivinen@iki.fi

TLS Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom ParisTech

February 7, 2013

Expires: August 2013

LLCPS
draft-urien-tls-llcp-01.txt

Abstract

This document describes the implementation, named LLCPS, of the TLS protocol over the NFC (Near Field Communication) LLCP (Logical Link Control Protocol) layer. The NFC peer to peer (P2P) protocol may be used by any application that needs communication between two devices at very small distances (a few centimeters). LLCPS enforces a strong security in NFC P2P exchanges, and may be deployed for many services, in the Internet Of Things ecosystem, such as access control or ticketing operations.

Requirements Language

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

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 August 2013.

.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

Abstract.....	1
Requirements Language.....	1
Status of this Memo.....	1
Copyright Notice.....	2
1 Overview.....	5
1.1 About the NFC protocol.....	5
1.2 The LLCP layer.....	7
1.3 LLCPS basic guidelines.....	9
2 TLS support over LLCP, Connection-oriented Transport.....	10
2.1 Peer To Peer Link Establishment.....	10
2.3 Connection Process, the Initiator is Server, the Target is Client.....	13
2.3.1 Initiator side	13
2.3.2 Target side	14
2.3.3 Connection choreography	14
2.4 Connection Process, the Initiator is Client, the Target is Server.....	14
2.4.1 Initiator side	14
2.4.2 Target side	15
2.4.3 Connection choreography	15
2.5 Disconnection Process.....	15
2.5.1 Disconnection initiated by the Initiator	15
2.5.2 Disconnection initiated by the Target	15
2.5.3 Disconnection choreography	16
2.6 Sending Process.....	16
2.7 Receiving Process.....	18
3 TLS support over LLCP, Connectionless Transport.....	21
3.1 Peer To Peer Link Establishment.....	23
3.2 Inactivity Process.....	24
3.3 Connection Process, the Initiator is Server, the Target is Client.....	24
3.3.1 Initiator side	24
3.3.2 Target side	25
3.3.3 Connection choreography	25
3.4 Connection Process, the Initiator is Client, the Target is Server.....	25
3.4.1 Initiator side	25
3.4.2 Target side	25
3.4.3 Connection choreography	26
3.5 Disconnection Process.....	26
3.5.1 Disconnection initiated by the Initiator	26
3.5.2 Disconnection initiated by the Target	26
3.5.3 Disconnection choreography	27
3.6 Sending Process.....	27
3.7 Receiving Process.....	29
4 Example of LLCPS session, connected mode.....	32
4.1 Protocol Activation and Parameters Selection.....	32
4.1.1 Initiator ATR-REQ	32
4.1.2 Target ATR-RESP	32

4.2	LLCP connection.....	32
4.3	Target: sending Client Hello.....	33
4.4	Inactivity Process.....	33
4.5	Server: sending Server Hello.....	33
4.6	LLCP Inactivity Process.....	34
4.7	Client: sending Client Finished.....	34
4.8	Exchanging Data.....	35
4.8.1	Sending data from client to server	35
4.8.2	Sending data from server to client	35
4.9	Closing TLS session, initiated by the Initiator.....	36
5	Example of LLCPS session, Connectionless mode.....	36
5.1	Protocol Activation and Parameters Selection.....	36
5.1.1	Initiator ATR-REQ	36
5.1.2	Target ATR-RESP	36
5.2	LLCP connection.....	37
5.3	Client Hello.....	37
5.4	Server Hello.....	37
5.5	Client Finished.....	38
5.6	Exchanging Data.....	38
5.6.1	Sending data from client to server	38
5.6.2	Sending data from server to client	39
5.7	End of Session.....	39
6	Security Considerations.....	40
7	IANA Considerations.....	40
8	References.....	40
8.1	Normative References.....	40
8.2	Informative References.....	40
9	Authors' Addresses.....	41

1 Overview

1.1 About the NFC protocol

The Near Field Communication protocol (NFC) is based on standards such as [ECMA340] or [ISO/IEC 18092]. It uses the 13,56 Mhz frequency, with data rates ranging from 106 To 848 kbps. The working distance between two nodes is about a few centimeters, with electromagnetic fields ranging between 1 and 10 A/M.

There are two classes of working operations:

- Reader/Writer and Card Emulation. A device named "Reader" feeds another device called "Card", thanks to a 13,56 MHz electromagnetic field coupling. This mode is typically used with [ISO7816] contactless smartcards or with NFC RFIDs.
- Peer To Peer (P2P). Two devices, the "Initiator" and the "Target" establish a NFC communication link. In the "Active" mode these two nodes are managing their own energy resources. In the "Passive" mode the Initiator powers the Target via a 13,56 MHz electromagnetic field coupling.

This draft focuses on P2P security, which is required by many applications, targeting access control, transport, or other Internet Of Things (IoT) items. Although the NFC protocol enables data exchange at small physical distances, it doesn't support standardized security features providing privacy or integrity. Thus, protocols such as [SNEP] or [NPP], whose goal is to push NDEF [NDEF] contents, are not today secured. In this draft we define a profile for TLS support in P2P operations.

A P2P session (see figure 1) occurs in four logical phases:

- 1) Initialization and Anti-collision. The Initiator periodically sends a request packet (and therefore generates a RF field), which is acknowledged by a Target response packet. Because several Targets may be located near the Initiator, an anti-collision mechanism is managed by the Initiator in order to establish a session with a single Target.
- 2) Protocol Activation and Parameters Selection. The Initiator starts a logical session with a detected Target by sending a ATR-REQ (Attribute-Request) message, which is confirmed by a Target ATR-RESP (Attribute-Response) message. These messages fix the device IDs (DIDi, Device ID Initiator and DIDt, Device ID Target) used in further packet exchanges. Optional information fields (Gi for the Initiator, and Gt for the Target) identify the protocol to be used over the MAC level; in this document it is assumed that the LLCPP [LLCP] (Logical Link Control Protocol) protocol is selected by the Gi and Gt bytes. Optionally some parameters are negotiated by additional packets.

3) Data Exchange. Frames are exchanged via the DEP (Data Exchange Protocol) protocol. DEP works with DEP-REQ (DEP-Request) transmitted by the Initiator and DEP-RESP (DEP-Response) delivered by the Target. DEP provides error detection and recovery. It uses small data unit size (from 64 to 256 bytes); however it supports a chaining mode for larger sizes. DEP frames typically transport LLCP packets, and provide an error free service

4) De-Activation. The Initiator may deactivate the Target by sending a RLS-REQ (Release Request) message acknowledged by a RLS-RESP (Release Response).

Usually, and for practical reasons, P2P sessions are established between a unique Target and an Initiator, for example a mobile phone and another NFC device. They are automatically started when the distance between the two NFC modes is sufficiently small. The MAC link may be broken at any time, as soon as the distance disables radio operations.

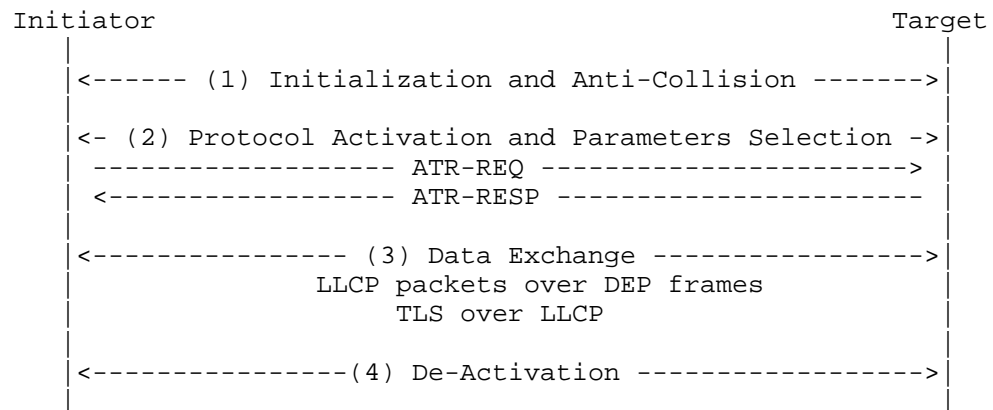


Figure 1. A NFC P2P Session

Due to the dissymmetry of the DEP protocol (see figure 2), in which the Initiator sends requests and Target returns responses, the NFC-P2P MAC services are dissymmetric on the Initiator and Target sides.

- The Initiator delivers Data.Request-i and gets Data.Indication-i.
- The Target gets Data.Indication-t and delivers Data.Request-t

MAC services implemented by NFC controllers usually support such dissymmetric primitives for Initiator and Target procedures (MAC Data.request-i/t and Data.Indication-i/t).

The timeout value (between DEP-REQ and DEP-RESP messages) is deduced from the RWT attribute (Response Waiting Time) returned by the Target in the ATR-RESP message. RWT ranges between 0,6 ms and 9,9 ms. It may be extended to the RWT-INT by a factor RTOX (RWT-INT = RTOX x RWT) between 1 and 60, so the maximum value is about 6s.

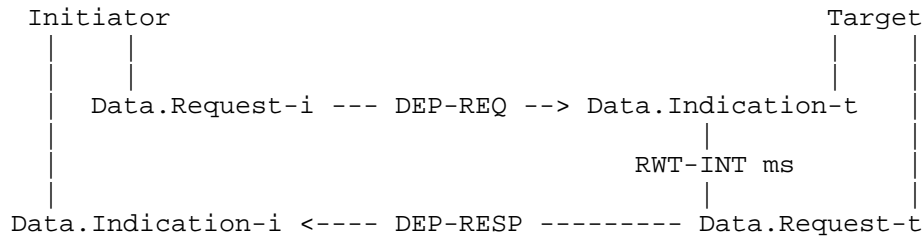


Figure 2. NFC-P2P MAC layer service, based on DEP frames

1.2 The LLCP layer

The LLCP [LLCP] protocol works like a light LLC [IEEE 802.2] layer. It provides two classes of services, connectionless transport and connection-oriented transport.

This draft focuses both on connection-oriented transport, in which TLS services are identified by a Service Name (SN), and on non-connected mode, in which a fix (well-known) Service Access Point (SAP) is used.

A LLCP packet (see figure 3) comprises three mandatory fields, DSAP (Destination Service Access Point, 6 bits), SSAP (Source Service Access Point, 6 bits), and PTYPE (Protocol data unit type field, 4 bits).

An optional sequence field (8 bits) contains two 4 bits number N(S) and N(R) respectively giving the number of the information SDU to be sent and the number of the next information PDU to be received.

An optional Information field transports the LLCP payload.

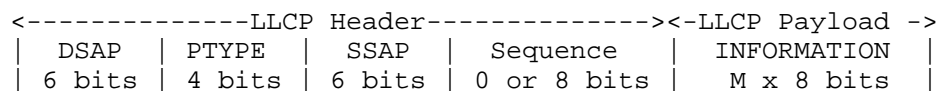


Figure 3. Structure of an LLCP packet

There are sixteen types of LLCP packets, identified by PTYPE values ranging between 0 and 15. In this draft we use only eight of these PDUs.

1) Symmetry (SYMM, PTYPE=0, DSAP=SSAP=0, No Sequence, No Information). This PDU is produced as soon as there is no information to provide. This mechanism avoids timeout at the MAC (DEP) level. SYMM SHOULD be generated after an inactivity period of about LTO/2, where LTO is the link timeout.

2) Connect (CONNECT, PTYPE=4, No sequence, Information). This PDU MUST include a SN (service name parameter) that identified the TLS service ("com.ietf.tls"). It uses a DSAP value set to 1 (the SAP of the Service Discovery Protocol, SDP) and a SSAP value ranging between 16 and 31. It indicates the connection the well-known service (WKS) SDP (SAP=1), which SHOULD deliver an ephemeral SAP (SAP-client) ranging between 16 and 31.

3) Connection Complete (CC, PTYPE=6, No sequence, Optional Information). This PDU notifies the successful connection to the "com.ietf.tls" service. It allocates the SAP (DSAP=SAP-client) to be used for this session identified by the tuple (SAP-server, SAP-client)

4) Disconnection (DISC, PTYPE=5, No sequence, No Information). This PDU indicates the disconnection of the (SAP-server, SAP-client) session. Null SAP values MAY be used to notify the disconnection of the LLCPS entity.

5) Disconnected Mode (DM, TYPE=7, No sequence, one byte of Information). This PDU confirms the disconnection of the (SAP-server, SAP-client) session; one information byte gives the "Disconnected Mode Reasons". Null SAP values notify the disconnection of the LLCPS entity.

6) Information (INFORMATION, PTYPE=10, Sequence, information). This PDU transport a SDU; N(S) indicates the SDU number, N(R) indicates the next SDU number to be received. In this draft the Receive Windows Size (RW) MUST be set to one, which is the default LLCPS value.

7) Receive Ready (RR, PTYPE=11, sequence N(R) only, no Information). This PDU is used for the acknowledgment of previously received information PDU. It indicates the next sequence number (N(R)) to be received.

8) Unnumbered Information (UI, PTYPE=3, no Sequence, Optional Information). This PDU is used to transfer service data units to the peer LLC without prior establishment of a data link connection.

According to [LLCP] some LLCPS functional parameters are updated by LLCPS-Parameter attributes exchanged in LLCPS packets or in ATR-REQ and ATR-RESP messages. Parameters are encoding according to TLV format, in which Type size is one byte, Length size is one byte and Value is a set of L bytes. In this document we use 6 parameters.

1) Version Number (VERSION, T=01h, L=01h, V=10h). In this document this option MUST be included in the general bytes of ATR-REQ and ATR-RESP.

2) Maximum Information Unit Extension (MIUX, T=02h, L=02h). This parameter extends the maximum size of the LLC PDU (MIU), whose default value is 128 bytes, according to the relation: $MIU = MIUX + 128$. The MIUX parameter MAY be inserted in general bytes of ATR-REQ and ATR-RESP, and in LLC PDUs CONNECT and CC.

3) Well-Known Service List (WKS, T=03h, L=02h). This parameter associates a bit to the instance of a well-known LLC parameter. A typical value is 00001h, indicating the availability of the DSP service. WKS MAY be inserted in general bytes of ATR-REQ and ATR-RESP.

4) Link Timeout (LTO, T=04h, L=01h). This parameter indicates the timeout value for the LLC layer, in multiples of 10ms. LTO MAY be inserted in general bytes of ATR-REQ and ATR-RESP.

5) Receive Windows Frame (RW, T=05h, L=01h). This parameter indicates the size of the receive windows, its value ranges between 0 and 15. The default value is one, and MUST be set to one according to this document. It MAY be inserted in LLC PDUs CONNECT or CC.

6) Service Name (SN, T=06h). This parameter indicates the name of a service. It MUST be inserted in the CONNECT PDU. In this document its value is set to "com.ietf.tls".

1.3 LLCPS basic guidelines

The TLS protocol is a series of record messages, which MAY be encrypted or integrity-protected. Each record message includes a five bytes prefix that comprises three attributes:

- The type (one byte) of the message,
- The version (two bytes),
- The message length (two bytes).

The client and the server exchange RECORD messages whose meaning is deduced from the TLS protocol rules, according to a half-duplex paradigm. Therefore as soon as the beginning of the TLS session is detected, the two TLS entities alternatively send and receive a set of record messages, whose synchronization is handled by the knowledge of TLS protocol.

LLCPS specifies the TLS session establishment and release, and the transport of TLS packets in a NFC P2P context.

2 TLS support over LLCP, Connection-oriented Transport

In NFC P2P mode the Initiator detects a Target and afterwards starts and manages a data exchange session; it may optionally feed the Target device. The Initiator has consequently a longer useful life than the Target; it is a legitimate place to host TLS server in a permanent way.

However the TLS server MAY be hosted on the Initiator or on the Target side.

Each entity manages five exclusive processes

- The Connection Process (CP)
- The Disconnection Process (DP)
- The Sending Process (SP)
- The Receiving Process (RP)
- The Inactivity Process (IP)

The Inactivity Process MAY be started (see figure 4) each time a receiving or sending buffer is empty; in this case it is assumed that the computing time or the delay required before the next input/output operation is greater than the LLCP timeout (LTO).

2.1 Peer To Peer Link Establishment

As described in section 1, the Initiator periodically probes the presence of a Target. At the end of the "Protocol Activation and Parameters Selection" phase, ATR-REQ and ATR-RESP messages have been exchanged, and LLCP services are available on both Initiator and Target nodes, including in particular the Data-Request-i/t and Data-Indication-i/t primitives.

Due to the ephemeral intrinsic nature of an NFC connection, the P2P session may be broken at any time, which implies transmission or reception errors notified by the MAC primitives.

As a consequence an LLCP session is assumed to be released at the first MAC error.

Once a NFC P2P link is established, TLS server and client software entities are activated. Procedures such as:

- SOCKET acceptllcp (char *ServiceName), and
- SOCKET connectllcp(char *ServiceName)

MAY be used respectively on Initiator and Target sides, in order to get a SOCKET.

A SOCKET object supports additional facilities, typically the following procedures:

- int sendllcp(SOCKET s, char *buffer, int length)
- int recvllcp(SOCKET s, char *buffer, int length)
- int closellcp(SOCKET s)

which are used for the LLCPS session management.

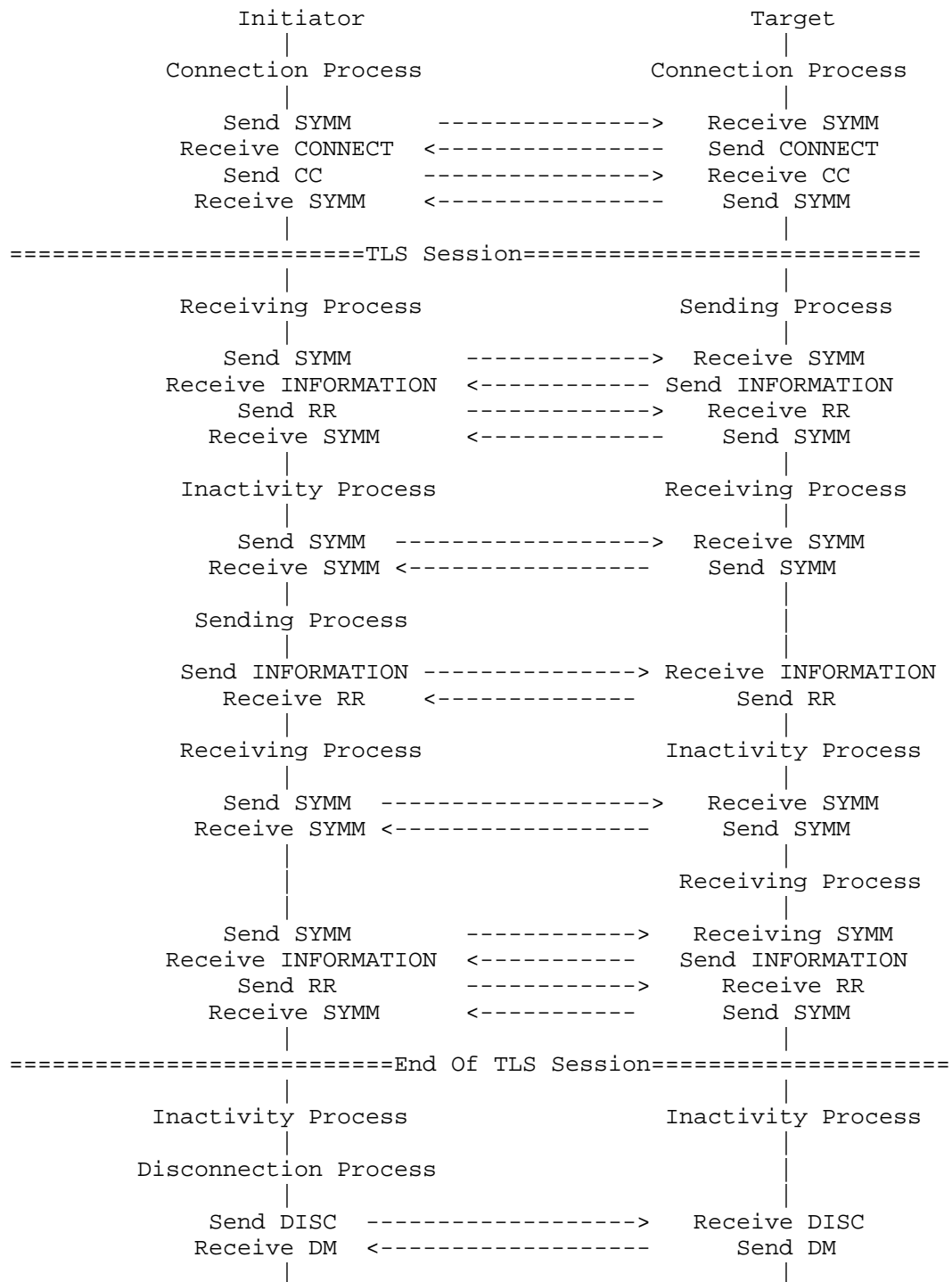


Figure 4. Overview of Operations, Connected Mode

2.2 Inactivity Process

When the LLCPS layer detects an inactivity period greater than a given timeout value (see figure 5), it generates a SYMM PDU. Therefore each time a LLCPS layer is waiting for a non SYMM PDU, and receives a SYMM PDU, it MUST acknowledge it by sending a SYMM PDU. A maximum number (SYMM-Ct-i/t) of echoed SYMM PDU SHOULD be defined.

The Inactivity Process (IP) MAY start between the Receiving Process (RP) and the Sending Process (SP).

Upon the reception of an INFORMATION PDU, the packet is stored in the reception buffer. Thereafter the IP sends a SYMM, in order to block further remote SP packets.

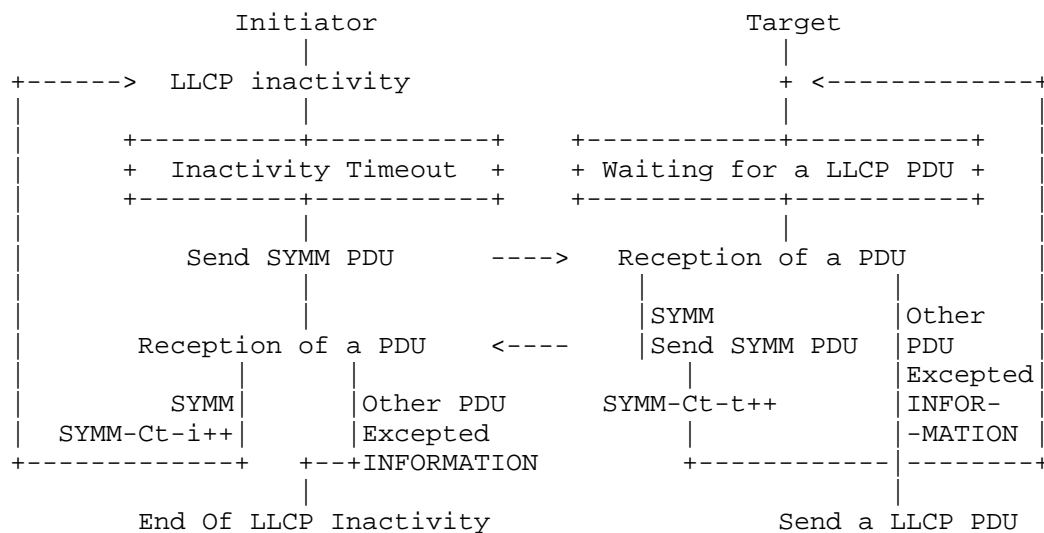


Figure 5. Inactivity Process

2.3 Connection Process, the Initiator is Server, the Target is Client

2.3.1 Initiator side

The Initiator MUST transmit a SYMM LLCPS PDU.

The Initiator MUST receive a CONNECT PDU, with DSAP=1, including the SN option, whose value MUST be set to "com.ietf.tls". If the SN value is incorrect the Initiator transmits a DM PDU with a reason code.

The Initiator MUST send a CC PDU, with an SSAP ranging between 16 and 31.

The Initiator SHOULD receive a SYMM PDU. It MAY receive an INFORMATION PDU but this behavior is not recommended, since it complicates the implementation of the acceptllcp (and connectllcp) procedure.

2.3.2 Target side

The Target MUST wait for the reception of a SYMM PDU

The Target MUST send a CONNECT PDU, with DSAP=1 and SSAP ranging between 16 and 31, including the option SN, whose value MUST be set to "com.ietf.tls".

The Target MUST receive a CC PDU.

The Target SHOULD send a SYMM PDU. It MAY send an INFORMATION PDU but this behavior is not recommended, since it complicates the implementation of the connectllcp (and acceptllcp) procedure.

2.3.3 Connection choreography

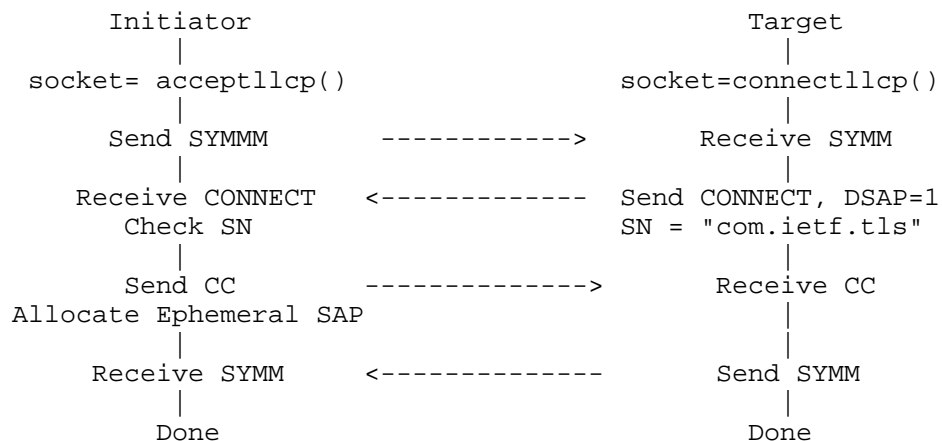


Figure 6. Connection Choreography

2.4 Connection Process, the Initiator is Client, the Target is Server

2.4.1 Initiator side

The Initiator MUST send a CONNECT PDU, with DSAP=1 and SSAP ranging between 16 and 31, including the SN option, whose value MUST be set to "com.ietf.tls".

The Initiator MUST receive a CC PDU.

2.4.2 Target side

The Target MUST receive a CONNECT PDU, with DSAP=1, including the SN option, whose value MUST be set to "com.ietf.tls". If the SN value is incorrect the Initiator transmits a DM PDU with a reason code.

The Target MUST send a CC PDU, with an SSAP ranging between 16 and 31.

2.4.3 Connection choreography

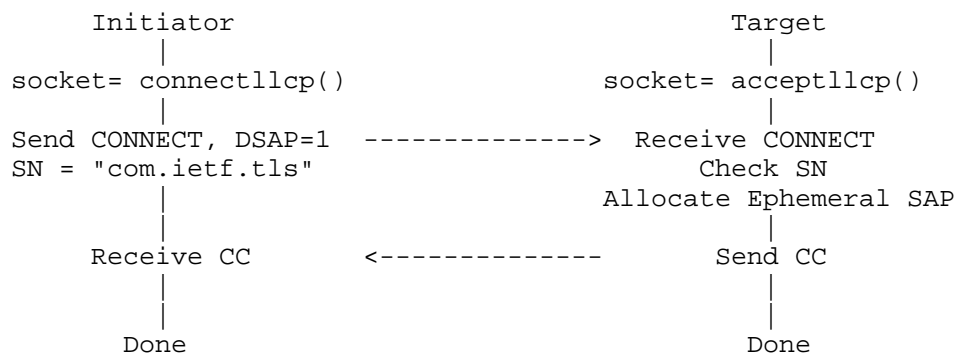


Figure 7. Connection Choreography

2.5 Disconnection Process

Due to the ephemeral nature of P2P NFC session, the disconnection process MAY be unavailable. Nevertheless it SHOULD be used for a graceful closing of a TLS session.

The Disconnection Process is started by the Initiator or the Target.

2.5.1 Disconnection initiated by the Initiator

The Initiator MUST send a DISC PDU.

The Target receives the DISC PDU.

The Target MUST send the DM PDU.

The Initiator MUST receive the DM PDU.

2.5.2 Disconnection initiated by the Target

The Target receives a LLCPS PDU. If it receives DISC then it sends DM; else it sends the DISC PDU.

The target waits for an LLC PDU. Upon reception of a LLC PDU it MUST send the SYMM or the DM PDU.

2.5.3 Disconnection choreography

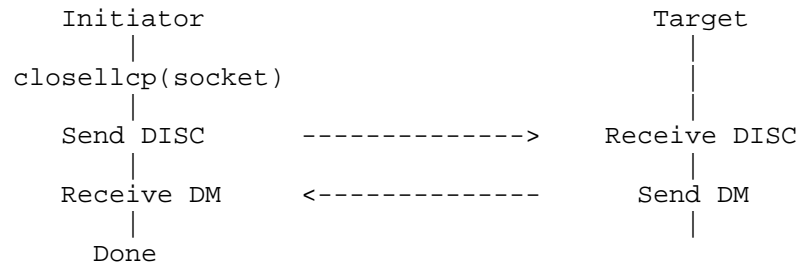


Figure 8. Disconnection started by the Initiator

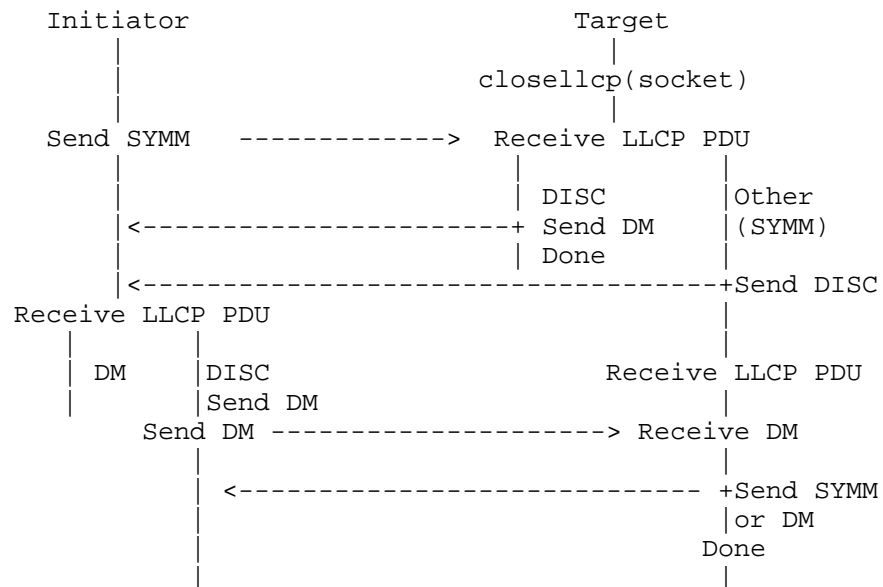


Figure 9. Disconnection started by the Target

2.6 Sending Process

The data transmission is managed by the `sendllcp(SOCKET s, char *buffer, int length)` procedure.

2.6.1 Initiator side

The buffer to be transmitted is segmented in LLC INFORMATION packets.

Each packet MUST be acknowledged by the Target with a RR PDU

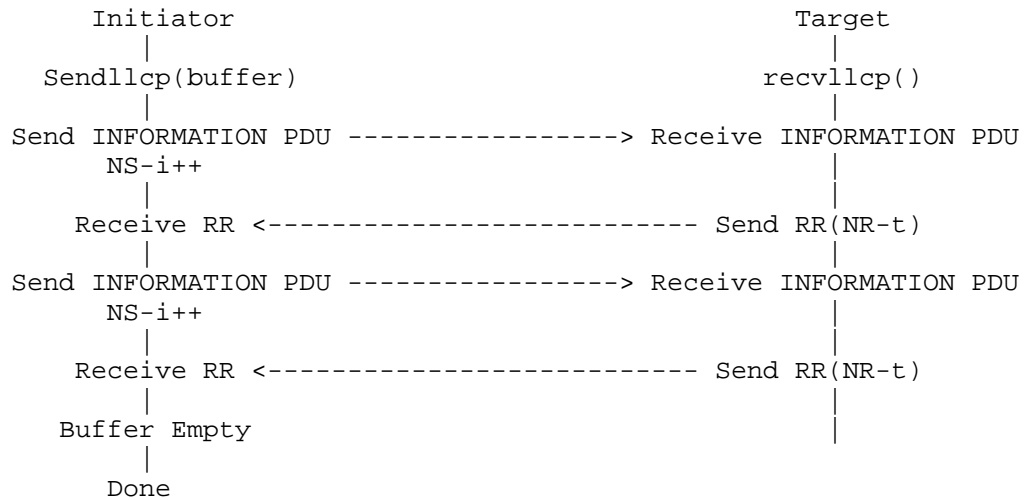


Figure 10. Sending Process, Initiator side.

2.6.2 Target side

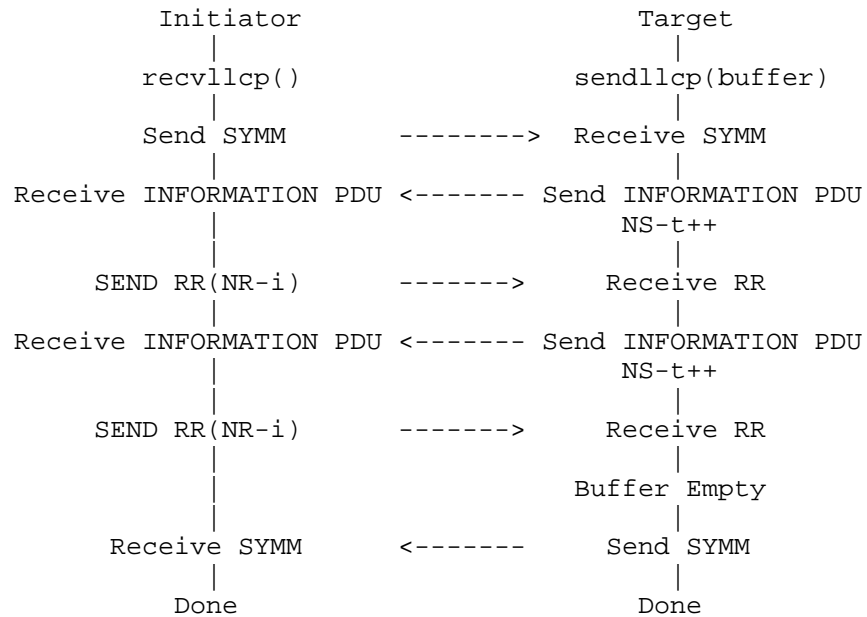
The Target switches to the sending process, managed by the `sendllcp()` procedure.

The Target MUST receive a SYMM PDU.

The buffer to be sent is segmented in INFORMATION PDUs.

Each INFORMATION PDU is sent by the Target to the Initiator and MUST be acknowledged by a RR PDU.

Upon the reception of the last RR PDU a SYMM PDU MUST be sent by the Target to the Initiator.



operation, a RR PDU is sent to the Target. The PDU received from the Target is either an INFORMATION PDU or a SYMM PDU.

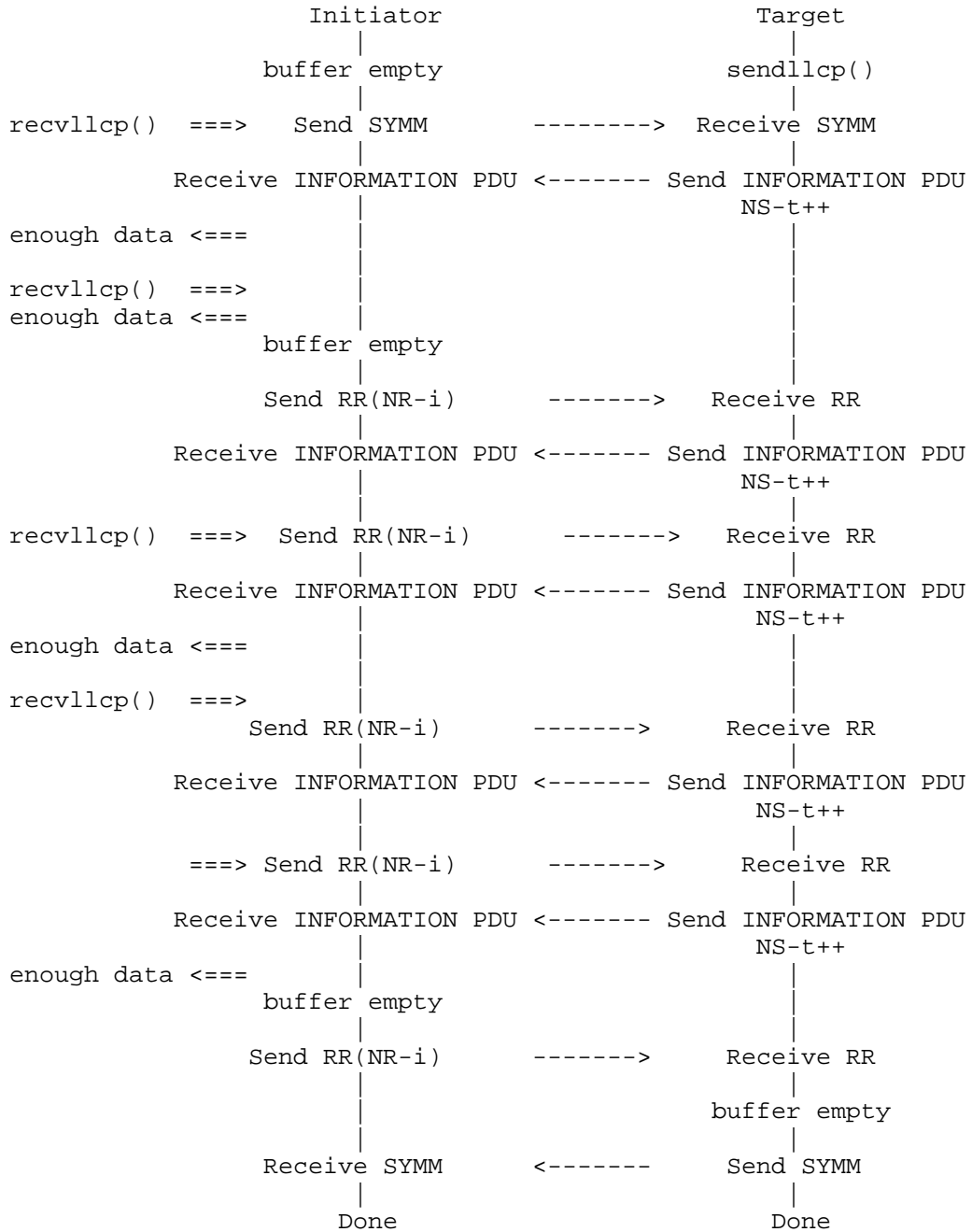
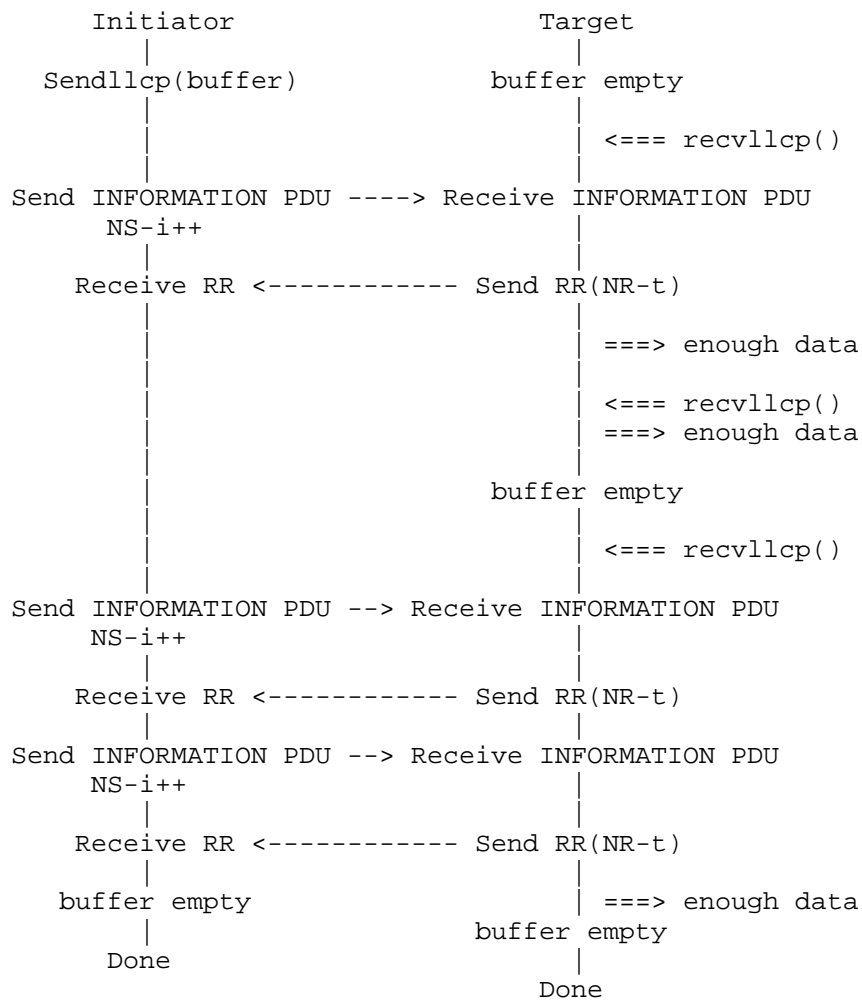


Figure 12. Receiving Process, Initiator side.

A1) If the reception buffer stores enough data, then the size requested by the `recvllcp()` procedure is returned.

- Receive INFORMATION PDU
- Send RR PDU



3 TLS support over LLCPS, Connectionless Transport

In NFC P2P mode the Initiator detects a Target and afterwards starts and manages a data exchange session; it may optionally feed the Target device.

The Initiator has consequently a longer useful life than the Target; it is a legitimate place to host TLS server in a permanent way.

However the TLS server MAY be hosted on the Initiator or on the Target side.

Each entity manages five exclusive processes

- The Connection Process (CP)
- The Disconnection Process (DP)
- The Sending Process (SP)
- The Receiving Process (RP)
- The Inactivity Process (IP)

The Inactivity Process MAY be started (see figure 14) each time a receiving or sending buffer is empty; in this case it is assumed that the computing time or the delay required before the next input/output operation is greater than the LLCPS timeout (LTO).

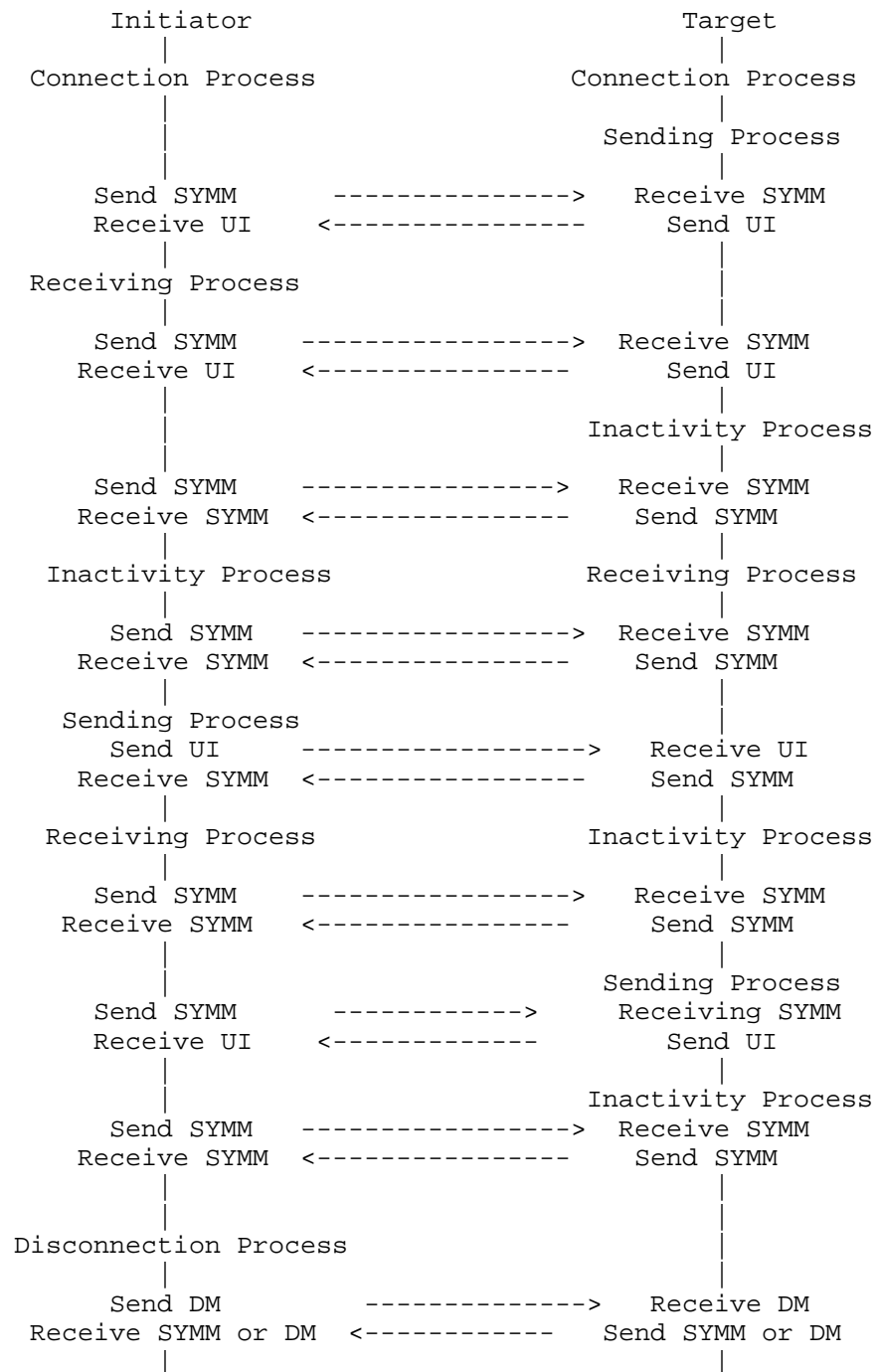


Figure 14. Overview of Process Operations, connectionless mode

3.1 Peer To Peer Link Establishment

As described in section 1, the Initiator periodically probes the presence of a Target. At the end of the "Protocol Activation and Parameters Selection" phase, ATR-REQ and ATR-RESP messages have been exchanged, and LLCPS services are available on both Initiator and Target nodes, including in particular the Data-Request-i/t and Data-Indication-i/t primitives.

Due to the ephemeral intrinsic nature of an NFC connection, the P2P session may be broken at any time, which implies transmission or reception errors notified by the MAC primitives.

As a consequence an LLCPS session is assumed to be released at the first MAC error.

Once a NFC P2P link is established, TLS server and client software entities are activated. Procedures such as:

- SOCKET acceptllcp(char TLS-SAP), and
- SOCKET connectllcp(char TLS-SAP)

MAY be used respectively on Initiator and Target sides, in order to get a SOCKET. This object supports additional facilities, typically the following procedures:

- int sendllcp(SOCKET s, char *buffer, int length)
- int recvllcp(SOCKET s, char *buffer, int length)
- int closellcp(SOCKET s)

which are used for the LLCPS session management.

3.2 Inactivity Process

When the LLCPS layer detects an inactivity period greater than a given timeout value (see figure 15), it generates a SYMM PDU. Therefore each time a LLCPS layer is waiting for a non SYMM PDU, and receives a SYMM PDU, it MUST acknowledge it by sending a SYMM PDU. A maximum number (SYMM-Ct-i/t) of echoed SYMM PDU SHOULD be defined.

Upon the reception of an UI PDU, the packet is stored in the reception buffer.

The Inactivity Process (IP) MAY start between the Receiving Process (RP) and the Sending Process (SP).

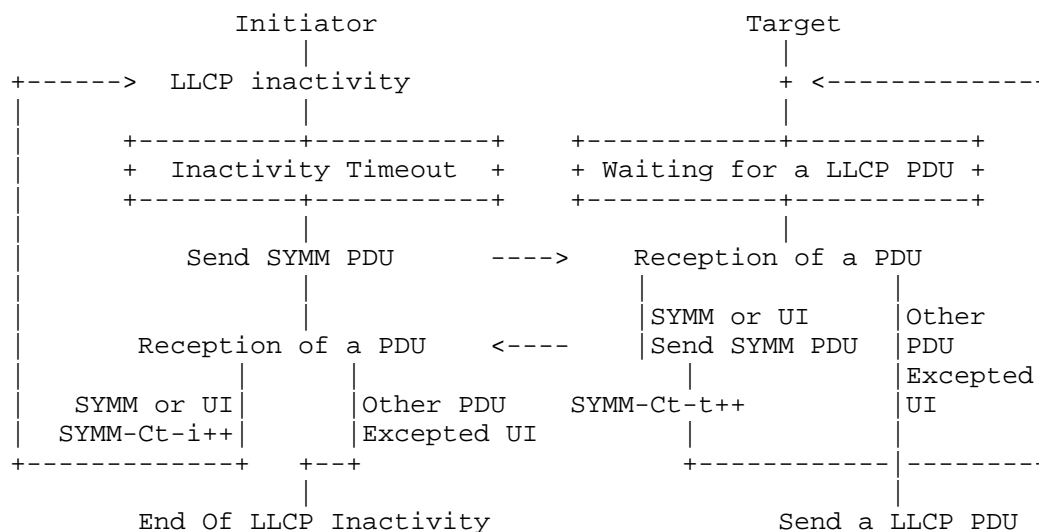


Figure 15. Inactivity Process

3.3 Connection Process, the Initiator is Server, the Target is Client

3.3.1 Initiator side

The Initiator MUST transmit a SYMM PDU.

If the Initiator receives a SYMM then it sends a SYMM.

If the Initiator receives an UI PDU, with the DSAP set to a well-known value that identifies the TLS service, then the service data unit transported by the UI is stored in the reception buffer.

If the DSAP value is incorrect the Initiator transmits a DM PDU with a reason code.

3.3.2 Target side

The Target allocates an ephemeral SSAP ranging between 16 and 31, and sends a SYMM.

The DSAP of UI PDU will use the allocated SSAP, and DSAP set to a well-known value that identifies the TLS service.

3.3.3 Connection choreography

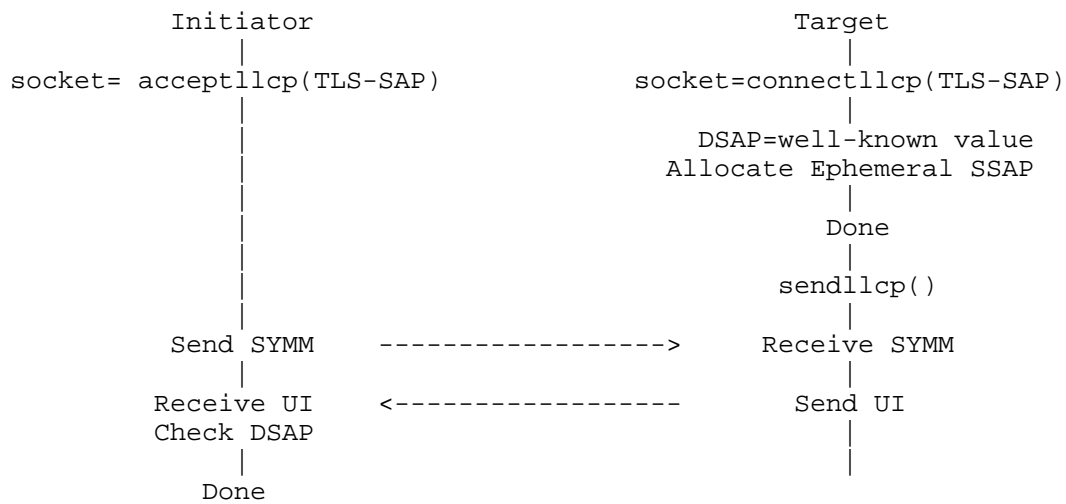


Figure 15. Connection Choreography

3.4 Connection Process, the Initiator is Client, the Target is Server

3.4.1 Initiator side

The initiator allocates an ephemeral SSAP ranging between 16 and 31, and sends a SYMM.

The DSAP of UI PDU will use the allocated SSAP, and DSAP set to a well-known value that identifies the TLS service.

3.4.2 Target side

If target receives a SYMM, then it sends A SYMM.

If the Target receives an UI PDU, with the DSAP set to a well-known value that identifies the TLS service, then the service data unit transported by the UI is stored in the reception buffer.

Upon success the Target sends a SYMM.

If the DSAP value is incorrect the Initiator transmits a DM PDU with a reason code.

3.4.3 Connection choreography

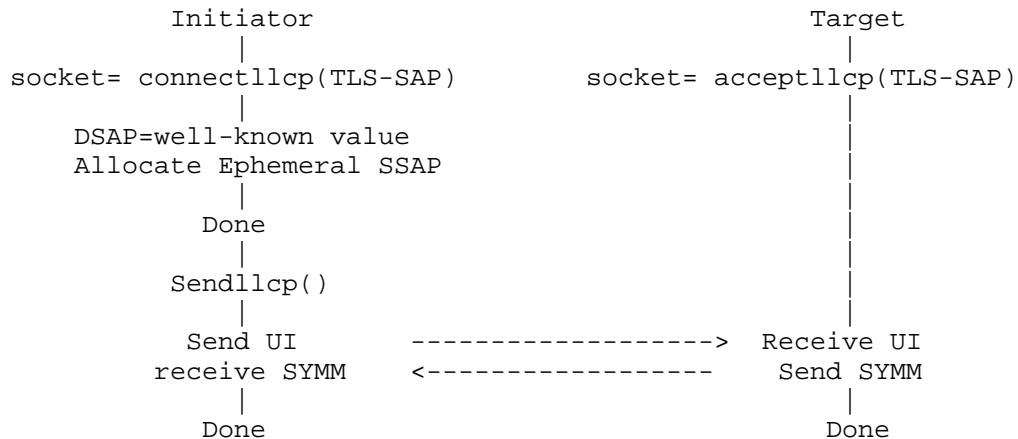


Figure 16. Connection Choreography

3.5 Disconnection Process

Due to the ephemeral nature of P2P NFC session, the disconnection process MAY be unavailable. Nerveless it SHOULD be used for a graceful closing of a TLS session. The Disconnection Process is initiated by the Initiator or the Target.

3.5.1 Disconnection initiated by the Initiator

The Initiator MUST send a DM PDU

The Target receives the DM PDU.

The Target sends a SYMM or a DM PDU.

3.5.2 Disconnection initiated by the Target

If the Target receives a DM PDU, then it sends the DM or the SYMM PDU.

Else the Target sends the DM PDU.

3.5.3 Disconnection choreography

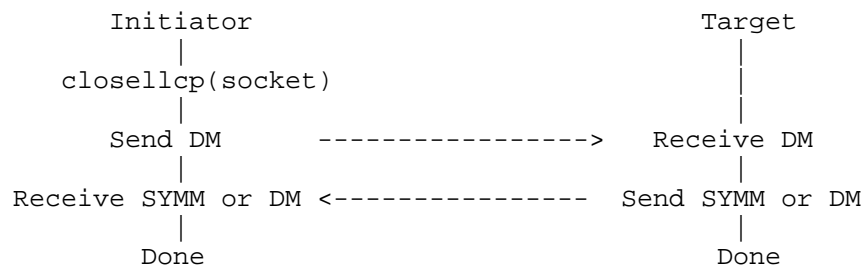


Figure 17. Disconnection initiated by the Initiator

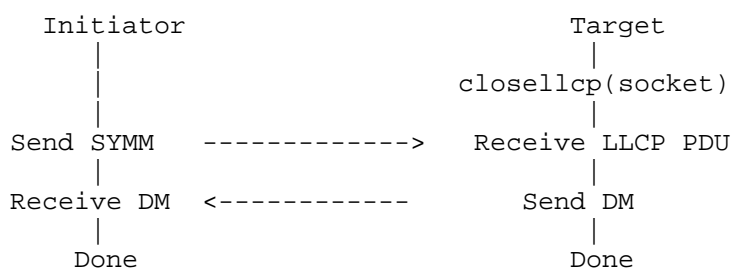


Figure 18. Disconnection initiated by the Target

3.6 Sending Process

The data transmission is managed by the
`sendllcp(SOCKET s, char *buffer, int length)`
 procedure.

3.6.1 Initiator side

The buffer to be transmitted is segmented in LLC P UI packets.



Figure 19. Sending Process, Initiator side.

The following loop is performed

- The Initiator sends an UI PDU
- The initiator receive a SYMM PDU

3.6.2 Target side

The Target switches to the sending process, managed by the `sendllcp()` procedure.

The Target MUST receive a SYMM PDU.

The buffer to be sent is segmented in UI PDUs.

The following loop is performed

- The Target sends an UI PDU
- The Target receives a SYMM PDU

When the buffer is empty a last SYMM is sent.

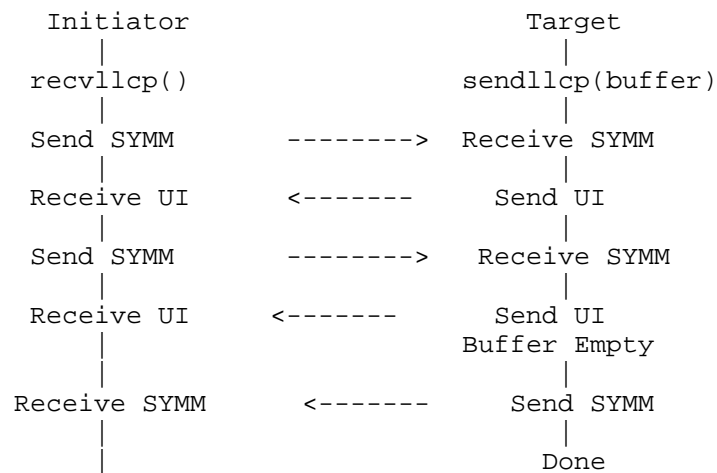


Figure 20. Sending Process, Target side.

3.7 Receiving Process

The Receiving process is handled by the
 `recvllcp(SOCKET s, char *buffer, int length)`
procedure, which manages a reception buffer.

3.7.1 Initiator side

A1) If the reception buffer is empty, the Initiator sends a SYMM PDU. This PDU starts the Target receiving process. The expected PDU received from the Target is either an UI PDU or a SYMM PDU (notifying an ephemeral inactivity state).

B1) If the reception buffer stores enough data, then the size requested by the `recvllcp()` procedure is returned. If the buffer gets empty after this operation, the SYMM PDU SHOULD be sent to the Target. The PDU received from the Target is either an UI PDU or a SYMM PDU.

B2) Else, while there is not enough data in the buffer, the following loop is performed

- Send SYMM
- Receive UI PDU

B2.1) at this end of this loop the size requested by the `recvllcp()` procedure is returned. If the buffer gets empty after this operation, the SYMM PDU SHOULD be sent to the Target. The PDU received from the Target is either an UI PDU or a SYMM PDU.

In B1 and B2.1 a SYMM PDU SHOULD be sent when the reception buffer gets empty. This rule avoids un-needed transition to the IP process. It is a "double checking" of the empty buffer event.

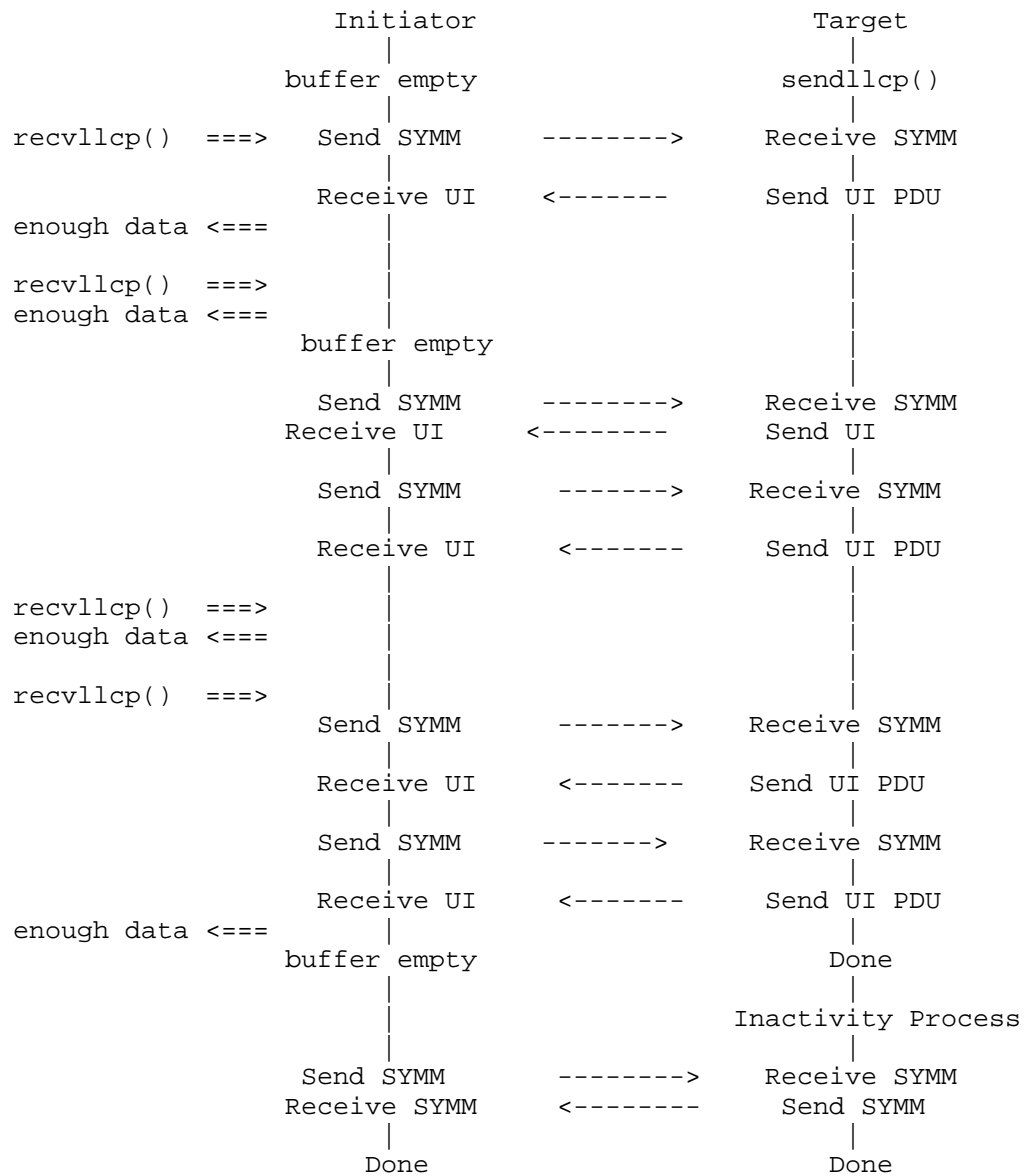


Figure 21. Receiving Process, Initiator side.

A1) If the reception buffer stores enough data, then the size requested by the `recvllcp()` procedure is returned.

- Receive UI PDU
- Send SYMM PDU

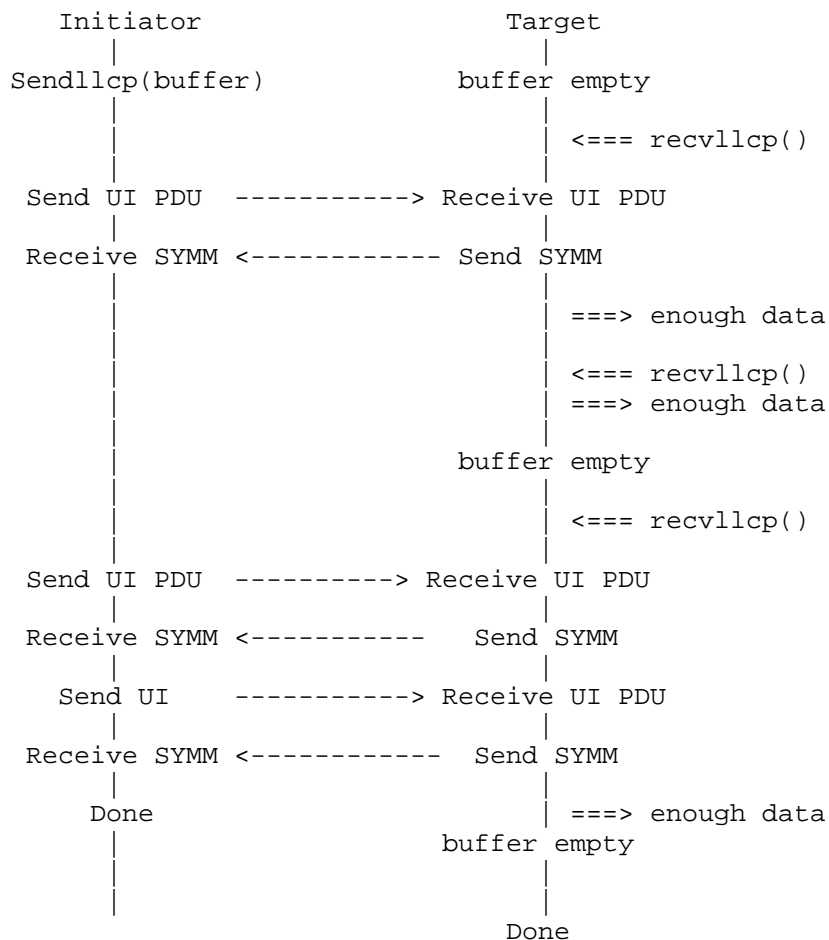


Figure 22. Receiving Process, Target side.

4 Example of LLCPS session, connected mode

4.1 Protocol Activation and Parameters Selection

4.1.1 Initiator ATR-REQ

Raw-data:

```
5C A9 BE E1 C0 35 A0 BF 16 0F 00 00 00 02 46 66
6D 01 01 10 03 02 00 01 04 01 01 10 64
```

NFCID3i= 5C A9 BE E1 C0 35 A0 BF 16 0F

DIDi (Initiator ID) = 00

BSi= 00

BRi= 00

PPi= 02, 64 bytes of Transport Data, Gt bytes available

Magic Bytes: 46666d

Option: Version, Major=1, Minor=0

Option: WKS: Well-Known Service List 0x0001

Option: LTO: Link TimeOut 0x64 (1000 ms)

4.1.2 Target ATR-RESP

Raw-Data:

```
AA 99 88 77 66 55 44 33 22 11 00 00 00 09 03 46
66 6D 01 01 10 03 02 00 01 04 01 64
```

NFCID3t= AA 99 88 77 66 55 44 33 22 11

DIDt (Target ID)= 00

BSt= 00

BRt= 00

TO= 09, WT= 6363 ms

PPt= 03, 64 bytes of Transport Data, NAD available, Gt bytes available

Magic Bytes: 46666d

Option: Version, Major=1, Minor=0

Option: WKS: Well-Known Service List 0x0001

Option: LTO: Link TimeOut 0x64 (1000 ms)

4.2 LLCP connection

Initiator: Sending SYMM, ssap=0 dsap=0

Tx-i: 00 00

Target: Sending CONNECT, ssap=27 dsap=1, option=SN("com.ietf.tls")

Rx_i: 05 1B 06 0C 63 6F 6D 2E 69 65 74 66 2E 74 6C 73

Initiator: Sending ConnectionComplete, ssap=16 dsap=27

Tx-i: 6D 90

Target: Sending SYMM, ssap=0 dsap=0

Rx-i: 00 00

4.3 Target: sending Client Hello

RecvLLCP Initiator: request size=5, buffer empty, sending SYMM
 Initiator: Sending SYMM, ssap=0 dsap=0
 Tx-i: 00 00

SendLLCP Target: request size=82 bytes, Waiting for SYMM
 Target: Receiving SYMM, ssap=0 dsap=0
 Target: Sending INFORMATION, ssap=27 dsap=16 Nr=0, Ns=0
 Rx-i: 43 1B 00 16 03 01 00 4D 01 00 00 49 03 01 50 1A
 A9 6B 82 55 1C B5 AD FF BC 87 21 66 5F B5 98 41
 9E 17 33 39 45 F9 78 86 46 D6 F6 75 51 10 20 E7
 0A 41 FE 8C F9 A0 38 D3 28 72 E8 04 7E C2 37 22
 05 13 24 AA DE 2F 6B 67 4C 19 CE A5 7D A0 86 00
 02 00 04 01 00

RecvLLCP_Initiator: request size=5 bytes, buffer=82 bytes
 RecvLLCP_Initiator: request size=77 bytes, buffer=77 bytes
 RecvLLCP_Initiator: buffer empty, sending RR(1), ssap=16 dsap=27
 Tx-i: 6F 50 01

SendLLCP_Target: Receiving RR(1), ssap=16 dsap=27
 SendLLCP_Target: empty buffer, Done, Sending SYMM
 Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM ssap=0 dsap=0
 Rx-i: 00 00

4.4 Inactivity Process

Initiator: Sending SYMM, ssap=0 dsap=0
 Tx-i: 00 00

RecvLLCP Target: request size=5 bytes, buffer empty
 Target: Receiving SYMM, ssap=0 dsap=0
 Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM, ssap=0 dsap=0
 Rx-i: 00 00

4.5 Server: sending Server Hello

SendLLCP_Initiator: request size=122 bytes
 Initiator: Sending INFORMATION, ssap=16 dsap=27 Nr=1 Ns=0
 Tx-i: 6F 10 01 16 03 01 00 4A 02 00 00 46 03 01 50 1A
 A9 6B 6C 0E 31 E1 F3 0E CD 18 E7 6F 81 BF 5F 3C
 FD DE 00 4C A4 12 AE DC DF E4 FF 82 09 5E 20 E7
 0A 41 FE 8C F9 A0 38 D3 28 72 E8 04 7E C2 37 22
 05 13 24 AA DE 2F 6B 67 4C 19 CE A5 7D A0 86 00
 04 00 14 03 01 00 01 01 16 03 01 00 20 83 18 D1

E3 BC 3A 94 26 91 3D FC F3 8E 01 46 5E 52 8E 67
 A2 66 FC 5F D5 89 78 59 66 14 BA D3 B0

RecvLLCP_Target: Receiving INFORMATION, ssap=16 dsap=27 Nr=1 Ns=0
 RecvLLCP_Target: sending RR(1), ssap=27 dsap=16
 RecvLLCP_Target: request size=74 bytes
 RecvLLCP_Target: request size=5 bytes
 RecvLLCP_Target: request size=1 byte

SendLLCP Initiator: Receiving RR(1), ssap=27 dsap=16
 Rx-i: 43 5B 01
 SendLLCP_Initiator: buffer empty, Done

RecvLLCP_Target: request size=5 bytes
 RecvLLCP_Target: request size=32 bytes, Done, empty buffer

4.6 LLCP Inactivity Process

RecvLLCP_Initiator: request size=5, empty buffer, sending SYMM
 Initiator: Sending SYMM, ssap=0 dsap=0
 Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0
 Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM ssap=0 dsap=0
 Rx-i: 00 00

4.7 Client: sending Client Finished

Initiator: Receiving SYMM ssap=0 dsap=0
 Tx-i: 00 00

SendLLCP_Target: request size=43 bytes, Waiting for SYMM
 Target: Receiving SYMM, ssap=0 dsap=0
 Target: Sending INFORMATION, ssap=27 dsap=16 Nr=1, Ns=1
 Rx-i: 43 1B 11 14 03 01 00 01 01 16 03 01 00 20 57 DD
 DE 29 9E E4 EF DD C5 18 87 50 C6 C7 B9 56 AD FA
 EF 65 B2 24 48 04 2E FE 7D BD 97 E1 F3 3A

Initiator: Receiving INFORMATION, ssap=27 dsap=16 Nr=1, Ns=1
 RecvLLCP_Initiator: request size= 5 bytes, buffer=43 bytes
 RecvLLCP_Initiator: request size= 1 bytes, buffer=38 bytes
 RecvLLCP_Initiator: request size= 5 bytes, buffer=37 bytes
 RecvLLCP_Initiator: request size=32 bytes, buffer=32 bytes
 RecvLLCP_Initiator: empty buffer, sending RR(2)
 Initiator: Sending RR(2), ssap=16 dsap=27
 Tx-i: 6F 50 02

Target: Receiving RR(2), ssap=16 dsap=27 Nr=2

SendLLC_Target: empty buffer, Done, sending SYMM
 Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM ssap=0 dsap=0
 Rx-i: 00 00

4.8 Exchanging Data

4.8.1 Sending data from client to server

RecvLLCP_Initiator: request size=5 bytes, empty buffer, sending SYMM
 Initiator: Sending SYMM, ssap=0 dsap=0
 Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0
 SendLLCP_Target: sending 27 bytes
 Target: Sending INFORMATION, ssap=27 dsap=16 Nr=1, Ns=2

Initiator: Receiving INFORMATION, ssap=27 dsap=16 Nr=1, Ns=2
 Rx-i: 43 1B 21 17 03 01 00 16 C2 D5 18 CB 0D AB 44 E5
 0F 25 DB 83 6D 26 B7 74 E7 90 EF 33 8C FE
 RecvLLCP_Initiator: request size= 5 bytes, buffer=27 bytes
 RecvLLCP_Initiator: request size=22 bytes, buffer=22 bytes
 Initiator: Sending RR(3), ssap=16 dsap=27
 Tx-i: 6F 50 03

Target: Receiving RR(3), ssap=16 dsap=27
 SendLLC_Target: empty buffer, Done, sending SYMM
 Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM ssap=0 dsap=0
 Rx-i: 00 00

4.8.2 Sending data from server to client

SendLLCP Initiator: request size=27 bytes
 Initiator: Sending INFORMATION, ssap=16 dsap=27 Nr=3 Ns=1
 Tx-i: 6F 10 13 17 03 01 00 16 DC 82 FE B9 EA 1C 63 5C
 AC 8C FE C9 A2 4F 8A FD 54 EE 18 F5 DB 30

RecvLLCP_Target: request size= 5 bytes
 Target: Receiving INFORMATION, ssap=16 dsap=27 Nr=3 Ns=1
 RecvLLCP_Target: sending RR(2)
 Target: Sending RR(2), ssap=27 dsap=16
 RecvLLCP_Target: request size=22 bytes, buffer=22 bytes, Done

Initiator: Receiving RR(2), ssap=27 dsap=16
 Rx-i: 43 5B 02
 SendLLCP Initiator: empty buffer, Done

4.9 Closing TLS session, initiated by the Initiator

Initiator: Sending DISC, ssap=16 dsap=27
Tx-i: 6D 50

Target: Receiving DISC, ssap=16 dsap=27
Target: Sending DM, ssap=27 dsap=16

Initiator: Receiving DM, ssap=27 dsap=16
Rx-i: 41 DB 00

5 Example of LLCPS session, Connectionless mode

5.1 Protocol Activation and Parameters Selection

5.1.1 Initiator ATR-REQ

Raw-data:
5C A9 BE E1 C0 35 A0 BF 16 0F 00 00 00 02 46 66
6D 01 01 10 03 02 00 01 04 01 01 10 64

NFCID3i= 5C A9 BE E1 C0 35 A0 BF 16 0F
DIDi (Initiator ID) = 00
BSi= 00
BRi= 00
PPi= 02, 64 bytes of Transport Data, Gt bytes available
Magic Bytes: 46666d
Option: Version, Major=1, Minor=0
Option: WKS: Well-Known Service List 0x0001
Option: LTO: Link TimeOut 0x64 (1000 ms)

5.1.2 Target ATR-RESP

Raw-Data:
AA 99 88 77 66 55 44 33 22 11 00 00 00 09 03 46
66 6D 01 01 10 03 02 00 01 04 01 64

NFCID3t= AA 99 88 77 66 55 44 33 22 11
DIDt (Target ID)= 00
BSt= 00
BRt= 00
TO= 09, WT= 6363 ms
PPt= 03, 64 bytes of Transport Data, NAD available, Gt bytes available
Magic Bytes: 46666d
Option: Version, Major=1, Minor=0
Option: WKS: Well-Known Service List 0x0001
Option: LTO: Link TimeOut 0x64 (1000 ms)

5.2 LLCP connection

Initiator: Sending SYMM, ssap=0 dsap=0

Tx-i: 00 00

Target: Setting DSAP to 13 (well known-value), setting ephemeral
SSAP to 27

5.3 Client Hello

Target: Receiving SYMM, ssap=0 dsap=0

Target: Sending UI, dsap=13 ssap=27, 82 bytes

Initiator: Receiving UI, ssap=27 dsap=13

Rx-i: 34 DB 16 03 01 00 4D 01 00 00 49 03 01 51 09 2E
3A CC 72 28 FE F5 D3 6F A8 D9 E7 55 67 6C 3B C3
7C 6C AF 18 1A 7F C6 81 1A 9D 0F 3D F8 20 04 E2
26 36 24 92 33 68 48 C7 34 A4 44 E3 70 8C 6C 11
44 53 54 20 B1 A9 3D 47 A8 3F E5 C5 D5 D2 00 02
00 04 01 00 90 00

RecvLLC Initiator: request size=5 buffer size=82

RecvLLC Initiator: request size=77 buffer size=77

RecvLLC Initiator: buffer empty

Initiator: Sending SYMM, ssap=0 dsap=0

Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0

Target: Sending SYMM, ssap=0, dsap=0

Rx-i: 00 00

5.4 Server Hello

SendLLC Initiator: request size=122

Initiator: Sending UI, ssap=13 dsap=27

Tx-i: 6C CD 16 03 01 00 4A 02 00 00 46 03 01 51 09 2E
3A 23 03 7D 28 AF D1 71 B4 0F 60 ED 3D A0 86 4B
67 36 A8 80 AB 34 78 21 63 1B D8 F5 81 20 04 E2
26 36 24 92 33 68 48 C7 34 A4 44 E3 70 8C 6C 11
44 53 54 20 B1 A9 3D 47 A8 3F E5 C5 D5 D2 00 04
00 14 03 01 00 01 01 16 03 01 00 20 B9 0C 3F E8
C8 48 F3 8B 1A 1C 59 01 6C C9 A0 7F 33 FB E9 A3
1E 9E 25 B8 FA AE FE 77 06 51 3D E4

Target: Receiving UI, ssap=13 dsap=27, 122 bytes

RecvLLC Target: request size= 5, buffer size= 122

RecvLLC Target: request size=74, buffer size= 117

RecvLLC Target: request size= 5, buffer size= 43
RecvLLC Target: request size= 1, buffer size= 42
RecvLLC Target: request size= 5, buffer size= 37
RecvLLC Target: request size=32, buffer size= 32
RecvLLC Target: empty buffer

Target: Sending SYMM, ssap=0 dsap=0

Initiator: Receiving SYMM, ssap=0 dsap=0
Rx-i: 00 00

5.5 Client Finished

Initiator: Sending SYMM, ssap=0 dsap=0
Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0

SendLLC Target: sending 43 bytes
Target: Sending UI, ssap=27 dsap=13, 43 bytes

Initiator: Receiving UI, ssap=27 dsap=13, 43 bytes

Rx-i: 34 DB 14 03 01 00 01 01 16 03 01 00 20 7E 92 D1
D1 78 C4 39 2D 8D 11 9A DF 0F 0B E5 7C 33 BA DC
3D B0 33 CD 5E 27 BE A4 6C 62 78 F3 D8

RecvLLC Initiator: request size=5 buffer size=43
RecvLLC Initiator: request_size=1 buffer size=38
RecvLLC Initiator: request_size=5 buffer_size=37
RecvLLC Initiator: request_size=32 buffer size=32
RecvLLC_Initiator: buffer empty

Initiator: Sending SYMM, ssap=0 dsap=0
Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0
Target: Sending SYMM, ssap=0,dsap=0
Rx-i: 00 00

5.6 Exchanging Data

5.6.1 Sending data from client to server

Initiator: Sending SYMM, ssap=0 dsap=0
Tx-i: 00 00

Target: Receiving SYMM, ssap=0 dsap=0
Target: Sending UI, ssap=27 dsap=13, 27 bytes

Rx-i: 34 DB 17 03 01 00 16 EA 91 72 8A DA 5A DD F0 C7
6A E0 82 15 B4 8F 5E 72 F6 BE 64 9D 0E

Initiator: Receiving UI, ssap=27 dsap=13, 27 bytes

SendLLC Initiator: request size= 5, buffer size=32
SendLLC Initiator: request size=27, buffer size=27
SendLLC Initiator: buffer empty

Initiator: Sending SYMM, ssap=0 dsap=0
Tx-i: 00 00

Target: Sending SYMM, ssap=0,dsap=0
Initiator: Receiving SYMM, ssap=0 dsap=0

Rx-i: 00 00

5.6.2 Sending data from server to client

Initiator: Sending UI, ssap=13 dsap=27, 27 bytes

Tx-i: 6C CD 17 03 01 00 16 93 48 F4 7F 67 F8 6E A1 94
15 BB AF D1 BD CA 2D AE 48 0B A6 9B 9D

Target: Receiving UI, ssap=13 dsap=27, 27 bytes
Target: Sending SYMM, ssap=0,dsap=0

RecvLLC Target: request size= 5, buffer size=32
RecvLLC Target: request size=27, buffer size=27
RecvLLC Target: buffer empty

Initiator: Receiving SYMM, ssap=0 dsap=0
Rx-i: 00 00

5.7 End of Session

Initiator: Sending DM, ssap=0 dsap=0
Target: Receiving DM, ssap=0 dsap=0
Target: Sending SYMM, ssap=0 dsap=0
Initiator: Receiving SYMM, ssap=0 dsap=0

6 Security Considerations

To be done.

7 IANA Considerations

8 References

8.1 Normative References

[TLS 1.0] Dierks, T., C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999

[TLS 1.1] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006

[TLS 1.2] Dierks, T., Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.1", draft-ietf-tls-rfc4346-bis-10.txt, March 2008

[ECMA340] "Near Field Communication Interface and Protocol (NFCIP-1)", Standard ECMA-340, December 2004

[ISO/IEC 18092] "Information technology - Telecommunications and information exchange between systems - Near Field Communication - Interface and Protocol (NFCIP-1)", April 2004

[LLCP] "Logical Link Control Protocol", Technical Specification, NFC ForumTM, LLCP 1.1, June 2011

[SNEP] "Simple NDEF Exchange Protocol", Technical Specification, NFC ForumTM, SNEP 1.0, August 2011

[NDEF] "NFC Data Exchange Format (NDEF)", Technical Specification NFC ForumTM, NDEF 1.0, July 2006.

[ISO7816] ISO 7816, "Cards Identification - Integrated Circuit Cards with Contacts", The International Organization for Standardization (ISO)

[IEEE 802.2] IEEE Std 802.2, "IEEE Standard for Information technology Telecommunications and information exchange between systems Local and metropolitan area networks, Specific requirements, Part 2: Logical Link Control", 1998

8.2 Informative References

[NPP] "Android NDEF Push Protocol Specification Version 1", February 2011

9 Authors' Addresses

Pascal Urien
Telecom ParisTech
23 avenue d' Italie
75013 Paris
France

Phone: NA
Email: Pascal.Urien@telecom-paristech.fr