TLS Working Group
Internet Draft
Intended status: Experimental

P. Urien
Telecom ParisTech

August 14, 2012

Expires: February 2013

LLCPS
draft-urien-tls-llcp-00.txt

Abstract

This document describes the implementation, named LLCPS, of the TLS protocol over the NFC (Near Field Communication) LLCP 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

.

Table of Contents

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 LLCP [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.

```
       Initiator                                                Target
          |                                                      |
          |<------ (1) Initialization and Anti-Collision ------->|
          |                                                      |
          |<- (2) Protocol Activation and Parameters Selection ->|
          | ------------------- ATR-REQ ----------------------> |
          | <------------------ ATR-RESP ---------------------- |
          |                                                      |
          |<---------------- (3) Data Exchange ----------------->|
          |               LLCP packets over DEP frames           |
          |                    TLS over LLCP                     |
          |                                                      |
          |<----------------(4) De-Activation ------------------>|
          |                                                      |
```

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.

```
     Initiator                                        Target
      |    |                                          |    |
      |    |                                          |    |
      |  Data.Request-i --- DEP-REQ --> Data.Indication-t  |
      |                                        |           |
      |                                    RWT-INT ms      |
      |                                        |           |
    Data.Indication-i <---- DEP-RESP --------- Data.Request-t
```

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 on connection-oriented transport because we believe that TLS services MUST be identified by a Service Name (SN).

An 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.

```
     <--------------LLCP Header-------------><-LLCP Payload ->
     |  DSAP  | PTYPE  |  SSAP  |  Sequence   |  INFORMATION  |
     | 6 bits | 4 bits | 6 bits | 0 or 8 bits |   M x 8 bits  |
```

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 six 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 LLCP 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 LLCP 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 LLCP 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.

According to [LLCP] some LLCP functional parameters are updated by LLCP-Parameter attributes exchanged in LLCP 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 LLCP 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 LLCP 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 LLCP 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 LLCP 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 LLCP 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".

2 TLS support over LLCP

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. Therefore in this section we assume that the Initiator acts as the TLS server and the Target as the TLS client.

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).

```
                Initiator                           Target
                   |                                  |
           Connection Process                Connection Process
                   |                                  |
               Send SYMM   ------------------->  Receive SYMM
             Receive DISC <-------------------    Send DISC
               Send DM    -------------------->   Receive DM
             Receive SYMM <--------------------    Send SYMM
                   |                                  |
=========================TLS Session============================
                   |                                  |
            Receiving Process                  Sending Process
                   |                                  |
               Send SYMM        -------------> Receive SYMM
           Receive INFORMATION  <------------ Send INFORMATION
                Send RR         ------------->  Receive RR
              Receive SYMM      <-------------    Send SYMM
                   |                                  |
            Inactivity Process               Receiving Process
                   |                                  |
                Send SYMM  ------------------> Receive SYMM
              Receive SYMM <-----------------    Send SYMM
                   |                                  |
             Sending Process                          |
                   |                                  |
            Send INFORMATION ---------------> Receive INFORMATION
               Receive RR    <--------------       Send RR
                   |                                  |
            Receiving Process                Inactivity Process
                   |                                  |
               Send SYMM  -------------------->  Receive SYMM
             Receive SYMM <------------------     Send SYMM
                   |                                  |
               Send SYMM        ------------->  Receiving Process
           Receive INFORMATION  <-----------   Send INFORMATION
                Send RR         ------------->    Receive RR
              Receive SYMM      <-----------      Send SYMM
                   |                                  |
===========================End Of TLS Session=====================
                   |                                  |
          Inactivity Process                 Inactivity Process
                   |                                  |
         Disconnection Process                        |
                   |                                  |
              Send Disc  --------------------> Receive DISC
             Receive DM  <-------------------      Send DM
                   |                                  |
```

Figure 4. Overview of Process Operations

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.

2.2 Inactivity Process

When the LLCP layer detects an inactivity period greater that a given timeout value (see figure 5), it generates a SYMM PDU. Therefore each time a LLCP 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.

```
                  Initiator                       Target
                      |                             |
    +------>  LLCP inactivity                       + <-------------+
    |                 |                             |               |
    |      +----------+-----------+    +------------+-----------+   |
    |      +  Inactivity Timeout  +    + Waiting for a LLCP PDU +   |
    |      +----------+-----------+    +------------+-----------+   |
    |                 |                             |               |
    |          Send SYMM PDU      ---->    Reception of a PDU       |
    |                 |                    |               |        |
    |                 |                    |SYMM           |Other   |
    |       Reception of a PDU    <----    |Send SYMM PDU  |PDU     |
    |             |      |                    |            |        |
    |         SYMM|      |Other       SYMM-Ct-t++          |        |
    |  SYMM-Ct-i++|      |PDU                 |            |        |
    +-------------+   +--+                    +------------|--------+
                      |                                    |
            End Of LLCP Inactivity                  Send a LLCP PDU
```

Figure 5. Inactivity Process

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

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. 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 LLCP session management.

2.3.1 Initiator side

The Initiator MUST transmit a SYMM LLCP PDU.

The Initiator MUST receive a CONNECT PDU, with DSAP=1, including the option SN, 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

```
              Initiator                              Target
                  |                                     |
         socket= acceptllcp()                 socket=connectllcp()
                  |                                     |
              Send SYMMM      ------------>       Receive SYMM
                  |                                     |
            Receive CONNECT   <-------------  Send CONNECT, DSAP=1
               Check SN                       SN = "com.ietf.tls"
                  |                                     |
               Send CC        -------------->     Receive CC
        Allocate Ephemeral SAP                          |
                  |                                     |
             Receive SYMM     <--------------     Send SYMM
                  |                                     |
                 Done                                  Done
```

Figure 6. Connection Choreography

2.4 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.

2.4.1 Disconnection initiated by the Initiator

The Initiator MUST sends a DISC PDU

The Target receives the DISC PDU

The Target MUST send the DM PDU.

The Initiator MUST receive a DM PDU

2.4.2 Disconnection initiated by the Target

The Target receives an LLCP PDU. If it receives DISC then it sends DM; else it sends the DISC PDU.

The target waits for an LLCP PDU. Upon reception of a LLCP PDU it MUST send SYMM.

2.4.3 Disconnection choreography

```
                   Initiator                            Target
                       |                                  |
                closellcp(socket)                         |
                       |                                  |
                   Send DISC      -------------->    Receive DISC
                       |                                  |
                   Receive DM     <--------------      Send DM
                       |                                  |
                     Done
```

Figure 7. Disconnection initiated by the Initiator

```
              Initiator                        Target
                  |                              |
                  |                      closellcp(socket)
                  |                              |
              Send SYMM   ------------->  Receive LLCP PDU
                  |                        |            |
                  |                        | DISC       |Other
                  |<-----------------------+ Send DM    |(SYMM)
                  |                        | Done       |
                  |<------------------------------------+Send DISC
            Receive LLCP PDU                            |
               |       |                                |
               [ DM    |DISC                     Receive LLCP PDU
               |       |Send DM                         |
                       +------------------------------->|
                       |                                |
                       |<-------------------------------+Send SYMM
                       |                                |
                       |                               Done
```

Figure 8. Disconnection initiated by the Target

2.5 Sending Process

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

2.5.1 Initiator side

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

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

```
        Initiator                                   Target
            |                                         |
     Sendllcp(buffer)                            recvllcp()
            |                                         |
   Send INFORMATION PDU -----------------> Receive INFORMATION PDU
         NS-i++                                       |
            |                                         |
       Receive RR <-------------------------- Send RR(NR-t)
            |                                         |
   Send INFORMATION PDU -----------------> Receive INFORMATION PDU
        NS-i++                                        |
            |                                         |
       Receive RR <-------------------------- Send RR(NR-t)
            |                                         |
      Buffer Empty                                    |
            |
         Done
```

Figure 9. Sending Process, Initiator side.

2.5.2 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.

```
                 Initiator                      Target
                    |                             |
                recvllcp()                  sendllcp(buffer)
                    |                             |
                Send SYMM        -------->  Receive SYMM
                    |                             |
        Receive INFORMATION PDU <------- Send INFORMATION PDU
                    |                          NS-t++
                    |                             |
             SEND RR(NR-i)       ------->     Receive RR
                    |                             |
        Receive INFORMATION PDU <------- Send INFORMATION PDU
                    |                          NS-t++
                    |                             |
             SEND RR(NR-i)       ------->     Receive RR
                    |                             |
                    |                       Buffer Empty
                    |                             |
              Receive SYMM       <-------     Send SYMM
                    |                             |
                                                 Done
```

Figure 10. Sending Process, Target side.

2.6 Receiving Process

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

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

B2) Else, while there is not enough data in the buffer, the following loop is performed
- Send RR PDU
- Receive INFORMATION 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, a RR PDU is sent to the Target. The PDU received from the Target is either an INFORMATION PDU or a SYMM PDU.

```
                     Initiator                      Target
                        |                             |
                  buffer empty                   sendllcp()
                        |                             |
  recvllcp()  ===>   Send SYMM       -------->  Receive SYMM
                        |                             |
            Receive INFORMATION PDU <------- Send INFORMATION PDU
                        |                          NS-t++
  enough data <===      |                             |
                        |                             |
  recvllcp()  ===>      |                             |
  enough data <===      |                             |
                  buffer empty                        |
                        |                             |
                  Send RR(NR-i)      ------->   Receive RR
                        |                             |
            Receive INFORMATION PDU <------- Send INFORMATION PDU
                        |                          NS-t++
                        |                             |
  recvllcp()  ===>  Send RR(NR-i)      ------->   Receive RR
                        |                             |
            Receive INFORMATION PDU <------- Send INFORMATION PDU
                        |                           NS-t++
  enough data <===      |                             |
                        |                             |
  recvllcp()  ===>      |                             |
                 Send RR(NR-i)      ------->     Receive RR
                        |                             |
            Receive INFORMATION PDU <------- Send INFORMATION PDU
                        |                           NS-t++
                        |                             |
              ===> Send RR(NR-i)      ------->     Receive RR
                        |                             |
            Receive INFORMATION PDU <------- Send INFORMATION PDU
                        |                           NS-t++
  enough data <===      |                             |
                  buffer empty                        |
                        |                             |
                 Send RR(NR-i)      ------->     Receive RR
                        |                             |
                        |                        buffer empty
                        |                             |
                  Receive SYMM      <-------     Send SYMM
                        |                             |
                       Done                          Done
```

Figure 11. Receiving Process, Initiator side.

2.6.2 Target side

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

B1) Else, while there is not enough data in the buffer, the following loop is performed
- Receive INFORMATION PDU
- Send RR PDU

```
        Initiator                   Target
           |                          |
     Sendllcp(buffer)            buffer empty
           |                          |
           |                          | <=== recvllcp()
           |                          |
   Send INFORMATION PDU ----> Receive INFORMATION PDU
         NS-i++                       |
           |                          |
       Receive RR <------------ Send RR(NR-t)
           |                          |
           |                          | ===> enough data
           |                          |
           |                          | <=== recvllcp()
           |                          | ===> enough data
           |                          |
           |                     buffer empty
           |                          |
           |                          | <=== recvllcp()
           |                          |
   Send INFORMATION PDU --> Receive INFORMATION PDU
        NS-i++                        |
           |                          |
       Receive RR <------------ Send RR(NR-t)
           |                          |
   Send INFORMATION PDU --> Receive INFORMATION PDU
        NS-i++                        |
           |                          |
       Receive RR <------------ Send RR(NR-t)
           |                          |
      buffer empty                    | ===> enough data
           |                    buffer empty
         Done                         |
                                    Done
```

Figure 12. Receiving Process, Initiator side.

3 Example of LLCPS session

3.1 Protocol Activation and Parameters Selection

3.1.1 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)

3.1.2 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)

3.2 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

3.3 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
```

3.4 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
```

3.5 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
```

3.6 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
```

3.7 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
```

3.8 Exchanging Data

3.8.1 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
```

3.8.2 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
```

3.9 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 Security Considerations

To be done.

4 IANA Considerations

6 References

6.1 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

6.2 Informative References

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

7 Authors' Addresses

Pascal Urien
Telecom ParisTech
23 avenue d' Italie
75013 Paris Phone: NA
France Email: Pascal.Urien@telecom-paristech.fr