

Internet Engineering Task Force (IETF)
Request for Comments: 7924
Category: Standards Track
ISSN: 2070-1721

S. Santesson
3xA Security AB
H. Tschofenig
ARM Ltd.
July 2016

Transport Layer Security (TLS) Cached Information Extension

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 chain (i.e., the certificates of intermediate CAs up to the root CA).

This document defines an extension that allows a TLS client to inform a server of cached information, thereby enabling the server to omit already available information.

Status of This Memo

This is an Internet Standards Track document.

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

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at <http://www.rfc-editor.org/info/rfc7924>.

Copyright Notice

Copyright (c) 2016 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	3
3. Cached Information Extension	3
4. Exchange Specification	5
4.1. Server Certificate Message	6
4.2. CertificateRequest Message	7
5. Fingerprint Calculation	7
6. Example	8
7. Security Considerations	10
8. IANA Considerations	10
8.1. New Entry to the TLS ExtensionType Registry	10
8.2. New Registry for CachedInformationType	11
9. References	11
9.1. Normative References	11
9.2. Informative References	12
Appendix A. Example	13
Acknowledgments	18
Authors' Addresses	19

1. Introduction

Reducing the amount of information exchanged during a Transport Layer Security handshake to a minimum helps to improve performance in environments where devices are connected to a network with a low bandwidth and lossy radio technology. With the Internet of Things, such environments exist, for example, when devices use IEEE 802.15.4, Bluetooth Low Energy, or low power wide area networks. 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 information cached in an earlier TLS handshake.

A typical example exchange may therefore look as follows. First, the client and the server execute the full TLS handshake. The client then caches the certificate provided by the server. When the TLS client connects to the TLS server some time in the future, without using session resumption, it then attaches the "cached_info" extension defined in this document to the ClientHello message to indicate that it has cached the certificate, and it provides the fingerprint of it. If the server's certificate has not changed, then the TLS server does not need to send its certificate and the corresponding certificate chain again. In case information has changed, which can be seen from the fingerprint provided by the client, the certificate payload is transmitted to the client to allow the client to update the cache.

2. Terminology

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

This document refers to the TLS protocol, but the description is equally applicable to Datagram Transport Layer Security (DTLS) as well.

3. Cached Information Extension

This document defines a new extension type (cached_info(25)), which is used in ClientHello and ServerHello messages. The extension type is specified as follows.

```
enum {
    cached_info(25), (65535)
} ExtensionType;
```

The `extension_data` field of this extension, when included in the `ClientHello`, MUST contain the `CachedInformation` structure. The client MAY send multiple `CachedObjects` of the same `CachedInformationType`. This may, for example, be the case when the client has cached multiple certificates from a server.

```
enum {
    cert(1), cert_req(2) (255)
} CachedInformationType;

struct {
    select (type) {
        case client:
            CachedInformationType type;
            opaque hash_value<1..255>;
        case server:
            CachedInformationType type;
    } body;
} CachedObject;

struct {
    CachedObject cached_info<1..2^16-1>;
} CachedInformation;
```

This document defines the following two types:

'cert' type for not sending the complete server certificate message:

With the `type` field set to 'cert', the client MUST include the fingerprint of the Certificate message in the `hash_value` field. For this type, the fingerprint MUST be calculated using the procedure described in Section 5 with the Certificate message as input data.

'cert_req' Type for not sending the complete CertificateRequest Message:

With the `type` set to 'cert_req', the client MUST include the fingerprint of the CertificateRequest message in the `hash_value` field. For this type, the fingerprint MUST be calculated using the procedure described in Section 5 with the CertificateRequest message as input data.

New cached info types can be added following the policy described in the IANA Considerations (Section 8). New message digest algorithms for use with these types can also be added by registering a new type that makes use of the updated message digest algorithm. For practical reasons, we recommend reusing hash algorithms already

available with TLS ciphersuites. To avoid additional code and to keep the collision probability low, new hash algorithms MUST NOT have a collision resistance worse than SHA-256.

4. Exchange Specification

Clients supporting this extension MAY include the "cached_info" extension in the (extended) ClientHello. If the client includes the extension, then it MUST contain one or more CachedObject attributes.

A server supporting this extension MAY include the "cached_info" extension in the (extended) ServerHello. By returning the "cached_info" extension, the server indicates that it supports the cached info types. For each indicated cached info type, the server MUST alter the transmission of respective payloads, according to the rules outlined with each type. If the server includes the extension, it MUST only include CachedObjects of a type also supported by the client (as expressed in the ClientHello). For example, if a client indicates support for 'cert' and 'cert_req', then the server cannot respond with a "cached_info" attribute containing support for ('foo-bar').

Since the client includes a fingerprint of information it cached (for each indicated type), the server is able to determine whether cached information is stale. If the server supports this specification and notices a mismatch between the data cached by the client and its own information, then the server MUST include the information in full and MUST NOT list the respective type in the "cached_info" extension.

Note: If a server is part of a hosting environment, then the client may have cached multiple data items for a single server. To allow the client to select the appropriate information from the cache, it is RECOMMENDED that the client utilizes the Server Name Indication (SNI) extension [RFC6066].

Following a successful exchange of the "cached_info" extension in the ClientHello and ServerHello, the server alters sending the corresponding handshake message. How information is altered from the handshake messages and for the types defined in this specification is defined in Sections 4.1 and 4.2, respectively.

Appendix A shows an example hash calculation, and Section 6 illustrates an example protocol exchange.

4.1. Server Certificate Message

When a ClientHello message contains the "cached_info" extension with a type set to 'cert', then the server MAY send the Certificate message shown in Figure 1 under the following conditions:

- o The server software implements the "cached_info" extension defined in this specification.
- o The 'cert' "cached_info" extension is enabled (for example, a policy allows the use of this extension).
- o The server compared the value in the hash_value field of the client-provided "cached_info" extension with the fingerprint of the Certificate message it normally sends to clients. This check ensures that the information cached by the client is current. The procedure for calculating the fingerprint is described in Section 5.

The original certificate handshake message syntax is defined in [RFC5246] and has been extended with [RFC7250]. RFC 7250 allows the certificate payload to contain only the SubjectPublicKeyInfo instead of the full information typically found in a certificate. Hence, when this specification is used in combination with [RFC7250] and the negotiated certificate type is a raw public key, then the TLS server omits sending a certificate payload that contains an ASN.1 certificate structure with the included SubjectPublicKeyInfo rather than the full certificate chain. As such, this extension is compatible with the raw public key extension defined in RFC 7250. Note: We assume that the server implementation is able to select the appropriate certificate or SubjectPublicKeyInfo from the received hash value. If the SNI extension is used by the client, then the server has additional information to guide the selection of the appropriate cached info.

When the cached info specification is used, then a modified version of the Certificate message is exchanged. The modified structure is shown in Figure 1.

```
struct {  
    opaque hash_value<1..255>;  
} Certificate;
```

Figure 1: Cached Info Certificate Message

4.2. CertificateRequest Message

When a fingerprint for an object of type 'cert_req' is provided in the ClientHello, the server MAY send the CertificateRequest message shown in Figure 2 under the following conditions:

- o The server software implements the "cached_info" extension defined in this specification.
- o The 'cert_req' "cached_info" extension is enabled (for example, a policy allows the use of this extension).
- o The server compared the value in the hash_value field of the client-provided "cached_info" extension with the fingerprint of the CertificateRequest message it normally sends to clients. This check ensures that the information cached by the client is current. The procedure for calculating the fingerprint is described in Section 5.
- o The server wants to request a certificate from the client.

The original CertificateRequest handshake message syntax is defined in [RFC5246]. The modified structure of the CertificateRequest message is shown in Figure 2.

```
struct {  
    opaque hash_value<1..255>;  
} CertificateRequest;
```

Figure 2: Cached Info CertificateRequest Message

The CertificateRequest payload is the input parameter to the fingerprint calculation described in Section 5.

5. Fingerprint Calculation

The fingerprint for the two cached info objects defined in this document MUST be computed as follows:

1. Compute the SHA-256 [RFC6234] hash of the input data. The input data depends on the cached info type. This document defines two cached info types, described in Sections 4.1 and in 4.2. Note that the computed hash only covers the input data structure (and not any type and length information of the record layer). Appendix A shows an example.
2. Use the output of the SHA-256 hash.

The purpose of the fingerprint provided by the client is to help the server select the correct information. For example, in case of a Certificate message, the fingerprint identifies the server certificate (and the corresponding private key) for use with the rest of the handshake. Servers may have more than one certificate, and therefore a hash needs to be long enough to keep the probability of hash collisions low. On the other hand, the cached info design aims to reduce the amount of data being exchanged. The security of the handshake depends on the private key and not on the size of the fingerprint. Hence, the fingerprint is a way to prevent the server from accidentally selecting the wrong information. If an attacker injects an incorrect fingerprint, then two outcomes are possible: (1) the fingerprint does not relate to any cached state and the server has to fall back to a full exchange, and (2) if the attacker manages to inject a fingerprint that refers to data the client has not cached, then the exchange will fail later when the client continues with the handshake and aims to verify the digital signature. The signature verification will fail since the public key cached by the client will not correspond to the private key that was used by the server to sign the message.

6. Example

In the regular, full TLS handshake exchange, shown in Figure 3, 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 Figure 4. The TLS client indicates support for this specification via the "cached_info" extension, see step (2), and indicates that it has stored the certificate from the earlier exchange (by indicating the 'cert' type). With step (3), the TLS server acknowledges the support of the 'cert' type and by including the value in the ServerHello, it informs the client that the content of the certificate payload contains the fingerprint of the certificate instead of the payload, defined in RFC 5246, of the Certificate message; see step (4).

```

ClientHello          ->
                    <- ServerHello
                      Certificate* // (1)
                      ServerKeyExchange*
                      CertificateRequest*
                      ServerHelloDone

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

```

Application Data <-----> Application Data

Figure 3: Example Message Exchange: Initial (Full) Exchange

```

ClientHello
cached_info=(cert)  -> // (2)
                    <- ServerHello
                      cached_info=(cert) (3)
                      Certificate (4)
                      ServerKeyExchange*
                      ServerHelloDone

ClientKeyExchange
CertificateVerify*
[ChangeCipherSpec]
Finished            ->
                    <- [ChangeCipherSpec]
                      Finished

```

Application Data <-----> Application Data

Figure 4: Example Message Exchange: TLS Cached Extension Usage

7. Security Considerations

This specification defines a mechanism to reference stored state using a fingerprint. Sending a fingerprint of cached information in an unencrypted handshake, as the ClientHello and ServerHello does, may allow an attacker or observer to correlate independent TLS exchanges. While some information elements used in this specification, such as server certificates, are public objects and usually do not contain sensitive information, other types that are not yet defined may. Those who implement and deploy this specification should therefore make an informed decision whether the cached information is in line with their security and privacy goals. In case of concerns, it is advised to avoid sending the fingerprint of the data objects in clear.

The use of the "cached_info" extension allows the server to send significantly smaller TLS messages. Consequently, these omitted parts of the messages are not included in the transcript of the handshake in the TLS Finish message. However, since the client and the server communicate the hash values of the cached data in the initial handshake messages, the fingerprints are included in the TLS Finish message.

Clients MUST ensure that they only cache information from legitimate sources. For example, when the client populates the cache from a TLS exchange, then it must only cache information after the successful completion of a TLS exchange to ensure that an attacker does not inject incorrect information into the cache. Failure to do so allows for man-in-the-middle attacks.

Security considerations for the fingerprint calculation are discussed in Section 5.

8. IANA Considerations

8.1. New Entry to the TLS ExtensionType Registry

IANA has added an entry to the existing TLS "ExtensionType Values" registry, defined in [RFC5246], for cached_info(25) defined in this document.

8.2. New Registry for CachedInformationType

IANA has established a registry titled "TLS CachedInformationType Values". The entries in the registry are:

Value	Description
0	Reserved
1	cert
2	cert_req
224-255	Reserved for Private Use

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

- o 0-63 (decimal): Standards Action
- o 64-223 (decimal): Specification Required

9. References

9.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<http://www.rfc-editor.org/info/rfc5246>>.
- [RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <<http://www.rfc-editor.org/info/rfc6066>>.
- [RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <<http://www.rfc-editor.org/info/rfc6234>>.

9.2. Informative References

- [ASN.1-Dump]
Gutmann, P., "ASN.1 Object Dump Program", November 2010,
<<http://manpages.ubuntu.com/manpages/precise/man1/dumpasn1.1.html>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.
- [RFC6574] Tschofenig, H. and J. Arkko, "Report from the Smart Object Workshop", RFC 6574, DOI 10.17487/RFC6574, April 2012, <<http://www.rfc-editor.org/info/rfc6574>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<http://www.rfc-editor.org/info/rfc7250>>.

Appendix A. Example

Consider a certificate containing a NIST P256 elliptic curve public key displayed using Peter Gutmann's ASN.1 decoder [ASN.1-Dump] in Figure 5.

```

0 556: SEQUENCE {
  4 434: SEQUENCE {
    8 3: [0] {
10 1: INTEGER 2
   : }
13 1: INTEGER 13
16 10: SEQUENCE {
18 8: OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
   : }
28 62: SEQUENCE {
30 11: SET {
32 9: SEQUENCE {
34 3: OBJECT IDENTIFIER countryName (2 5 4 6)
39 2: PrintableString 'NL'
   : }
   : }
43 17: SET {
45 15: SEQUENCE {
47 3: OBJECT IDENTIFIER organizationName (2 5 4 10)
52 8: PrintableString 'PolarSSL'
   : }
   : }
62 28: SET {
64 26: SEQUENCE {
66 3: OBJECT IDENTIFIER commonName (2 5 4 3)
71 19: PrintableString 'Polarssl Test EC CA'
   : }
   : }
92 30: SEQUENCE {
94 13: UTCTime 24/09/2013 15:52:04 GMT
109 13: UTCTime 22/09/2023 15:52:04 GMT
   : }
124 65: SEQUENCE {
126 11: SET {
128 9: SEQUENCE {
130 3: OBJECT IDENTIFIER countryName (2 5 4 6)
135 2: PrintableString 'NL'
   : }
   : }

```

```

139 17:      SET {
141 15:      SEQUENCE {
143  3:      OBJECT IDENTIFIER organizationName (2 5 4 10)
148  8:      PrintableString 'PolarSSL'
          :      }
          :      }
158 31:      SET {
160 29:      SEQUENCE {
162  3:      OBJECT IDENTIFIER commonName (2 5 4 3)
167 22:      PrintableString 'PolarSSL Test Client 2'
          :      }
          :      }
          :      }
191 89:      SEQUENCE {
193 19:      SEQUENCE {
195  7:      OBJECT IDENTIFIER ecPublicKey (1 2 840 10045 2 1)
204  8:      OBJECT IDENTIFIER prime256v1 (1 2 840 10045 3 1 7)
          :      }
214 66:      BIT STRING
          :      04 57 E5 AE B1 73 DF D3 AC BB 93 B8 81 FF 12 AE
          :      EE E6 53 AC CE 55 53 F6 34 0E CC 2E E3 63 25 0B
          :      DF 98 E2 F3 5C 60 36 96 C0 D5 18 14 70 E5 7F 9F
          :      D5 4B 45 18 E5 B0 6C D5 5C F8 96 8F 87 70 A3 E4
          :      C7
          :      }
282 157:     [3] {
285 154:     SEQUENCE {
288  9:     SEQUENCE {
290  3:     OBJECT IDENTIFIER basicConstraints (2 5 29 19)
295  2:     OCTET STRING, encapsulates {
297  0:     SEQUENCE {}
          :     }
          :     }
299 29:     SEQUENCE {
301  3:     OBJECT IDENTIFIER subjectKeyIdentifier (2 5 29 14)
306 22:     OCTET STRING, encapsulates {
308 20:     OCTET STRING
          :     7A 00 5F 86 64 FC E0 5D E5 11 10 3B B2 E6 3B C4
          :     26 3F CF E2
          :     }
          :     }
330 110:    SEQUENCE {
332  3:     OBJECT IDENTIFIER authorityKeyIdentifier (2 5 29 35)
337 103:    OCTET STRING, encapsulates {
339 101:    SEQUENCE {

```

```

341 20:          [0]
      :          9D 6D 20 24 49 01 3F 2B CB 78 B5 19 BC 7E 24
      :          C9 DB FB 36 7C
363 66:          [1] {
365 64:            [4] {
367 62:              SEQUENCE {
369 11:                SET {
371  9:                  SEQUENCE {
373  3:                    OBJECT IDENTIFIER countryName (2 5 4 6)
378  2:                    PrintableString 'NL'
      :                }
      :            }
382 17:          SET {
384 15:            SEQUENCE {
386  3:              OBJECT IDENTIFIER organizationName
      :                (2 5 4 10)
391  8:              PrintableString 'PolarSSL'
      :            }
      :          }
401 28:          SET {
403 26:            SEQUENCE {
405  3:              OBJECT IDENTIFIER commonName (2 5 4 3)
410 19:              PrintableString 'Polarssl Test EC CA'
      :            }
      :          }
      :        }
      :      }
431  9:          [2] 00 C1 43 E2 7E 62 43 CC E8
      :          }
      :        }
      :      }
      :    }
442 10:  SEQUENCE {
444  8:    OBJECT IDENTIFIER ecdsaWithSHA256 (1 2 840 10045 4 3 2)
      :  }
454 104: BIT STRING, encapsulates {
457 101:   SEQUENCE {
459  48:     INTEGER
      :     4A 65 0D 7B 20 83 A2 99 B9 A8 0F FC 8D EE 8F 3D
      :     BB 70 4C 96 03 AC 8E 78 70 DD F2 0E A0 B2 16 CB
      :     65 8E 1A C9 3F 2C 61 7E F8 3C EF AD 1C EE 36 20

```

```

509 49:    INTEGER
      :    00 9D F2 27 A6 D5 74 B8 24 AE E1 6A 3F 31 A1 CA
      :    54 2F 08 D0 8D EE 4F 0C 61 DF 77 78 7D B4 FD FC
      :    42 49 EE E5 B2 6A C2 CD 26 77 62 8E 28 7C 9E 57
      :    45
      :    }
      :  }
      : }

```

Figure 5: ASN.1-Based Certificate: Example

To include the certificate shown in Figure 5 in a TLS/DTLS Certificate message, it is prepended with a message header. This Certificate message header in our example is 0b 00 02 36 00 02 33 00 02 00 02 30, which indicates:

Message Type: 0b -- 1-byte type field indicating a Certificate message

Length: 00 02 36 -- 3-byte length field indicating a 566-byte payload

Certificates Length: 00 02 33 -- 3-byte length field indicating 563 bytes for the entire certificates_list structure, which may contain multiple certificates. In our example, only one certificate is included.

Certificate Length: 00 02 30 -- 3-byte length field indicating 560 bytes of the actual certificate following immediately afterwards. In our example, this is the certificate content with 30 82 02 9E 57 45 shown in Figure 6.

The hex encoding of the ASN.1-encoded certificate payload shown in Figure 5 leads to the following encoding.

```

30 82 02 2C 30 82 01 B2 A0 03 02 01 02 02 01 0D
30 0A 06 08 2A 86 48 CE 3D 04 03 02 30 3E 31 0B
30 09 06 03 55 04 06 13 02 4E 4C 31 11 30 0F 06
03 55 04 0A 13 08 50 6F 6C 61 72 53 53 4C 31 1C
30 1A 06 03 55 04 03 13 13 50 6F 6C 61 72 73 73
6C 20 54 65 73 74 20 45 43 20 43 41 30 1E 17 0D
31 33 30 39 32 34 31 35 35 32 30 34 5A 17 0D 32
33 30 39 32 32 31 35 35 32 30 34 5A 30 41 31 0B
30 09 06 03 55 04 06 13 02 4E 4C 31 11 30 0F 06
03 55 04 0A 13 08 50 6F 6C 61 72 53 53 4C 31 1F
30 1D 06 03 55 04 03 13 16 50 6F 6C 61 72 53 53
4C 20 54 65 73 74 20 43 6C 69 65 6E 74 20 32 30
59 30 13 06 07 2A 86 48 CE 3D 02 01 06 08 2A 86
48 CE 3D 03 01 07 03 42 00 04 57 E5 AE B1 73 DF
D3 AC BB 93 B8 81 FF 12 AE EE E6 53 AC CE 55 53
F6 34 0E CC 2E E3 63 25 0B DF 98 E2 F3 5C 60 36
96 C0 D5 18 14 70 E5 7F 9F D5 4B 45 18 E5 B0 6C
D5 5C F8 96 8F 87 70 A3 E4 C7 A3 81 9D 30 81 9A
30 09 06 03 55 1D 13 04 02 30 00 30 1D 06 03 55
1D 0E 04 16 04 14 7A 00 5F 86 64 FC E0 5D E5 11
10 3B B2 E6 3B C4 26 3F CF E2 30 6E 06 03 55 1D
23 04 67 30 65 80 14 9D 6D 20 24 49 01 3F 2B CB
78 B5 19 BC 7E 24 C9 DB FB 36 7C A1 42 A4 40 30
3E 31 0B 30 09 06 03 55 04 06 13 02 4E 4C 31 11
30 0F 06 03 55 04 0A 13 08 50 6F 6C 61 72 53 53
4C 31 1C 30 1A 06 03 55 04 03 13 13 50 6F 6C 61
72 73 73 6C 20 54 65 73 74 20 45 43 20 43 41 82
09 00 C1 43 E2 7E 62 43 CC E8 30 0A 06 08 2A 86
48 CE 3D 04 03 02 03 68 00 30 65 02 30 4A 65 0D
7B 20 83 A2 99 B9 A8 0F FC 8D EE 8F 3D BB 70 4C
96 03 AC 8E 78 70 DD F2 0E A0 B2 16 CB 65 8E 1A
C9 3F 2C 61 7E F8 3C EF AD 1C EE 36 20 02 31 00
9D F2 27 A6 D5 74 B8 24 AE E1 6A 3F 31 A1 CA 54
2F 08 D0 8D EE 4F 0C 61 DF 77 78 7D B4 FD FC 42
49 EE E5 B2 6A C2 CD 26 77 62 8E 28 7C 9E 57 45

```

Figure 6: Hex Encoding of the Example Certificate

Applying the SHA-256 hash function to the Certificate message, which starts with 0b 00 02 and ends with 9E 57 45, produces 0x086eefb4859adfe977defac494fff6b73033b4celf86b8f2a9fc0c6bf98605af.

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 (March 2012)
- o Ilari Liusvaara, Adam Langley, and Eric Rescorla (July 2014)
- o Sean Turner (August 2014)
- o Martin Thomson (August 2015)
- o Jouni Korhonen (November 2015)
- o Dave Garrett (December 2015)
- o Matt Miller (December 2015)
- o Anirudh Ramachandran (March 2016)

We would also to thank Martin Thomson, Karthikeyan Bhargavan, Sankalp Bagaria, and Eric Rescorla for their feedback regarding the fingerprint calculation.

Finally, we would like to thank the TLS working group chairs, Sean Turner and Joe Salowey, as well as the responsible Security Area Director, Stephen Farrell, for their support and their reviews.

Authors' Addresses

Stefan Santesson
3xA Security AB
Forskningsbyn Ideon
Lund 223 70
Sweden

Email: sts@aaa-sec.com

Hannes Tschofenig
ARM Ltd.
Hall in Tirol 6060
Austria

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

