TLS
Internet-Draft
Intended status: Standards Track
Expires: September 9, 2015

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

March 8, 2015

Transport Layer Security (TLS) Cached Information Extension
draft-ietf-tls-cached-info-18.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 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, allowing the server to omit 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 September 9, 2015.

Table of Contents

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 Internet of Things such environments exist, for example, when devices use IEEE 802.15.4 or Bluetooth Smart. 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 executes the usual TLS handshake. The client may, for example, decide to cache 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 client hello message to indicate that it had 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 list 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", "MUST", "MUST NOT", "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 DTLS as well.

3. Cached Information Extension

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

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

The extension_data field of this extension, when included in the client hello, MUST contain the CachedInformation structure. The client MUST NOT send multiple CachedObjects of the same CachedInformationType.

```
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 types:

Omitting the Server Certificate Message:

With the type field set to 'cert', the client MUST include the message digest of the Certificate message in the hash_value field. For this type the message digest MUST be calculated using SHA-256 [RFC4634].

Omitting the CertificateRequest Message

With the type set to 'cert_req', the client MUST include the message digest of the CertificateRequest message in the hash_value field. For this type the message digest MUST be calculated using SHA-256 [RFC4634].

Omitting the Certificate Status Information (OCSP Stapling and Multiple OCSP Stapling) Message

With the type set to 'cert_status', the client MUST include the message digest of the CertificateStatus message in the hash_value field. For this type the message digest MUST be calculated using SHA-256 [RFC4634].

New types can be added following the policy described in the IANA considerations section, see Section 7. Different message digest algorithms for use with these types can also be added by registering a new type that makes use of this updated message digest algorithm.

4. Exchange Specification

Clients supporting this extension MAY include the "cached_info" extension in the (extended) client hello. 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) server hello. 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 client hello). 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 'cert_status'.

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 extension [RFC6066].

Following a successful exchange of the "cached_info" extension in the client and server hello, the server alters sending the corresponding handshake message. How information is altered from the handshake messages is defined in Section 4.1, Section 4.2 and Section 4.3 for the types defined in this specification.

4.1. Omitting the Server Certificate Message

When a ClientHello message contains the "cached_info" extension with a type set to 'cert' then the server MAY omit the Certificate message under the following conditions:

- The server software implements the "cached_info" extension defined in this specification.

- The 'cert' cached info extension is enabled (for example, a policy allows the use of this extension).

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 original Certificate 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;
```

Certificate Message as defined in RFC 5246.

The fingerprint MUST be computed as follows: hash_value:=SHA-256(Certificate)

Note that RFC 7250 [RFC7250] 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. As such, this extension is compatible with the raw public key extension defined in RFC 7250.

4.2. Omitting the CertificateRequest Message

When a fingerprint for an object of type 'cert_req' is provided in the client hello, the server MAY omit the CertificateRequest message under the following conditions:

The server software implements the "cached_info" extension defined in this specification.

The 'cert_req' cached info extension is enabled (for example, a policy allows the use of this extension).

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 server wants to request a certificate from the client.

The original CertificateRequest 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;
```

The fingerprint MUST be computed as follows: hash_value:=SHA-256(CertificateRequest)

4.3. Omitting the Certificate Status Information (OCSP Stapling and Multi OCSP Stapling)

When a fingerprint for an object of type 'cert_status' is provided in the client hello, the server MAY omit the CertificateStatus message under the following conditions:

The server software implements the "cert_status" extension defined in this specification.

The 'cert_status' cached info extension is enabled (for example, a policy allows the use of this extension).

The server compared the value in the hash_value field of the client-provided "cached_info" extension with the fingerprint of the CertificateStatus message it normally sends to clients. This check ensures that the information cached by the client is current.

Both client and server support the use of OCSP Stapling and/or Multiple OCSP Stapling, as defined in RFC 6066 [RFC6066] and in [RFC6961].

The CertificateStatus message syntax, defined in [RFC6961], has the following structure:

```
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;
```

The fingerprint MUST be computed as follows: hash_value:=SHA-256(CertificateStatus)

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_info" extension, see [2], and indicates that it has stored the certificate from the earlier exchange (by indicating the 'cert' type). With [3] the TLS server acknowledges the supports of the 'cert' type and by including the value in the server hello informs the client that the certificate payload has been omitted.

```
   (A) Initial (full) Exchange

   ClientHello            ->
                          <-  ServerHello
                              Certificate* // [1]
                              ServerKeyExchange*
                              CertificateRequest*
                              ServerHelloDone

   Certificate*
   ClientKeyExchange
   CertificateVerify*
   [ChangeCipherSpec]
   Finished               ->

                          <- [ChangeCipherSpec]
                             Finished

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


   (B) TLS Cached Extension Usage

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

   ClientKeyExchange
   CertificateVerify*
   [ChangeCipherSpec]
   Finished               ->

                          <- [ChangeCipherSpec]
                             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. Sending a fingerprint of cached information in an unencrypted handshake, as the client and server hello is, 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 (not yet defined cached info types) may. Those who implement and deploy this specification should therefore make an informed decision whether the cached information is inline 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 obmit sending certain TLS messages. Consequently, these omitted messages are not included in the transcript of the handshake in the TLS Finish message per value. However, since the client communicates the hash values of the cached values in the initial handshake message 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.

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_info(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

- cert(1)
- cert_req(2)
- cert_status(3)

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

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

- 224-255 (decimal): reserved for Private Use

8. Acknowledgments

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

- Paul Wouters and Nikos Mavrogiannopoulos (December 2011)
- Rob Stradling (February 2012)
- Ondrej Mikle (in March 2012)
- Ilari Liusvaara, Adam Langley, and Eric Rescorla (in July 2014)
- Sean Turner (in August 2014)

Additionally, 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.

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.

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

[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.

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

9.2. Informative References

[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.

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

Appendix A. Example

The Wireshark trace of an example TLS exchange shown in Figure 2 illustrates the use of an ECC-based ciphersuite with a 256 bit key. ECC allows for a small certificate size compared to RSA with equivalent security strength. The Certificate message provided by the server is with 557 bytes (including the record layer header) one of the largest message even though it only contains a single certificate (i.e., no intermediate CA certificates). The client-provided Certificate message has a length of 570 bytes (also including the record layer header).

```
Client --> Server:

    TLSv1.2 Record Layer: Handshake Protocol: Client Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 121
        Handshake Protocol: Client Hello
            Handshake Type: Client Hello (1)
            Length: 117
            Version: TLS 1.2 (0x0303)
            Random
                gmt_unix_time: Jan 14, 2015 12:43:58.000000000 CET
                random_bytes: c61b966bba2781c50b07c3278c43f5892b3d...
            Session ID Length: 0
            Cipher Suites Length: 10
            Cipher Suites (5 suites)
            Compression Methods Length: 1
            Compression Methods (1 method)
            Extensions Length: 66
            Extension: server_name
            Extension: signature_algorithms
```

```
            Extension: elliptic_curves
            Extension: ec_point_formats

 Client <-- Server:

    TLSv1.2 Record Layer: Handshake Protocol: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 87
        Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 83
            Version: TLS 1.2 (0x0303)
            Random
                gmt_unix_time: Jan 14, 2015 12:43:58.000000000 CET
                random_bytes: 82d3d09b44149d738b7002da4ff5a986fe11...
            Session ID Length: 32
            Session ID: d069a74661088676b98db8346070278a7475b617a0...
            Cipher Suite: Unknown (0xc0ad)
            Compression Method: null (0)
            Extensions Length: 11
            Extension: renegotiation_info
            Extension: ec_point_formats

    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 557
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 553
            Certificates Length: 550
            Certificates (550 bytes)
                Certificate Length: 547
                Certificate (id-at-commonName=localhost,
                id-at-organizationName=PolarSSL,id-at-countryName=NL)
                    signedCertificate
                    algorithmIdentifier (iso.2.840.10045.4.3.2)
                    Padding: 0
                    encrypted: 30650231009a2c5cd7a6dba2e5640df0b94ed...

    TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 215
        Handshake Protocol: Server Key Exchange
            Handshake Type: Server Key Exchange (12)
            Length: 211
```

```
    TLSv1.2 Record Layer: Handshake Protocol: Certificate Request
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 78
        Handshake Protocol: Certificate Request
            Handshake Type: Certificate Request (13)
            Length: 74
            Certificate types count: 1
            Certificate types (1 type)
            Signature Hash Algorithms Length: 2
            Signature Hash Algorithms (1 algorithm)
            Distinguished Names Length: 66
            Distinguished Names (66 bytes)

    TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 4
        Handshake Protocol: Server Hello Done
            Handshake Type: Server Hello Done (14)
            Length: 0

 Client --> Server:

    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 570
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 566
            Certificates Length: 563
            Certificates (563 bytes)
                Certificate Length: 560
                Certificate (id-at-commonName=PolarSSL Test Client 2,
                id-at-organizationName=PolarSSL,id-at-countryName=NL)
                    signedCertificate
                    algorithmIdentifier (iso.2.840.10045.4.3.2)
                    Padding: 0
                    encrypted: 306502304a650d7b2083a299b9a80ffc8dee8...

    TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 138
        Handshake Protocol: Client Key Exchange
            Handshake Type: Client Key Exchange (16)
            Length: 134
```

```
    TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 80
        Handshake Protocol: Certificate Verify
            Handshake Type: Certificate Verify (15)
            Length: 76

    TLSv1.2 Record Layer: Change Cipher Spec Protocol
        Content Type: Change Cipher Spec (20)
        Version: TLS 1.2 (0x0303)
        Length: 1
        Change Cipher Spec Message

    TLSv1.2 Record Layer: Handshake Protocol:
        Encrypted Handshake Message (TLS Finished)
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 40
        Handshake Protocol: Encrypted Handshake Message

Client <-- Server:

    TLSv1.2 Record Layer: Change Cipher Spec Protocol
        Content Type: Change Cipher Spec (20)
        Version: TLS 1.2 (0x0303)
        Length: 1
        Change Cipher Spec Message

    TLSv1.2 Record Layer: Handshake Protocol
        Encrypted Handshake Message (TLS Finished)
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 40
        Handshake Protocol: Encrypted Handshake Message
```

Figure 2: Example TLS Exchange (without Cached Info Extension).

The total size of the TLS exchange shown in Figure 2 is 1932 bytes whereas the exchange shown in Figure 3 reduces the size to 1323 bytes by omitting the Certificate and the CertificateRequest messages. As it can be seen, the use of the cached info extension leads to an on-the-wire improvement of more than 600 bytes.

```
Client --> Server:

    TLSv1.2 Record Layer: Handshake Protocol: Client Hello
```

```
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 121 + 21
        Handshake Protocol: Client Hello
            Handshake Type: Client Hello (1)
            Length: 117 + 21
            Version: TLS 1.2 (0x0303)
            Random
                gmt_unix_time: Jan 14, 2015 12:43:58.000000000 CET
                random_bytes: c61b966bba2781c50b07c3278c43f5892b3d...
            Session ID Length: 0
            Cipher Suites Length: 10
            Cipher Suites (5 suites)
            Compression Methods Length: 1
            Compression Methods (1 method)
            Extensions Length: 66
            Extension: server_name
            Extension: signature_algorithms
            Extension: elliptic_curves
            Extension: ec_point_formats
            Extension: cached_info

 Client <-- Server:

    TLSv1.2 Record Layer: Handshake Protocol: Server Hello
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 87 + 5
        Handshake Protocol: Server Hello
            Handshake Type: Server Hello (2)
            Length: 83 + 5
            Version: TLS 1.2 (0x0303)
            Random
                gmt_unix_time: Jan 14, 2015 12:43:58.000000000 CET
                random_bytes: 82d3d09b44149d738b7002da4ff5a986fe11...
            Session ID Length: 32
            Session ID: d069a74661088676b98db8346070278a7475b617a0...
            Cipher Suite: Unknown (0xc0ad)
            Compression Method: null (0)
            Extensions Length: 11 + 5
            Extension: renegotiation_info
            Extension: ec_point_formats
            Extension: cached_info

    TLSv1.2 Record Layer: Handshake Protocol: Server Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 215
```

```
        Handshake Protocol: Server Key Exchange
            Handshake Type: Server Key Exchange (12)
            Length: 211


    TLSv1.2 Record Layer: Handshake Protocol: Server Hello Done
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 4
        Handshake Protocol: Server Hello Done
            Handshake Type: Server Hello Done (14)
            Length: 0

 Client --> Server:

    TLSv1.2 Record Layer: Handshake Protocol: Certificate
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 570
        Handshake Protocol: Certificate
            Handshake Type: Certificate (11)
            Length: 566
            Certificates Length: 563
            Certificates (563 bytes)
                Certificate Length: 560
                Certificate (id-at-commonName=PolarSSL Test Client 2,
                id-at-organizationName=PolarSSL,id-at-countryName=NL)
                    signedCertificate
                    algorithmIdentifier (iso.2.840.10045.4.3.2)
                    Padding: 0
                    encrypted: 306502304a650d7b2083a299b9a80ffc8dee8...

    TLSv1.2 Record Layer: Handshake Protocol: Client Key Exchange
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 138
        Handshake Protocol: Client Key Exchange
            Handshake Type: Client Key Exchange (16)
            Length: 134

    TLSv1.2 Record Layer: Handshake Protocol: Certificate Verify
        Content Type: Handshake (22)
        Version: TLS 1.2 (0x0303)
        Length: 80
        Handshake Protocol: Certificate Verify
            Handshake Type: Certificate Verify (15)
            Length: 76
```

```
     TLSv1.2 Record Layer: Change Cipher Spec Protocol
         Content Type: Change Cipher Spec (20)
         Version: TLS 1.2 (0x0303)
         Length: 1
         Change Cipher Spec Message

     TLSv1.2 Record Layer: Handshake Protocol:
         Encrypted Handshake Message (TLS Finished)
         Content Type: Handshake (22)
         Version: TLS 1.2 (0x0303)
         Length: 40
         Handshake Protocol: Encrypted Handshake Message

 Client <-- Server:

     TLSv1.2 Record Layer: Change Cipher Spec Protocol
         Content Type: Change Cipher Spec (20)
         Version: TLS 1.2 (0x0303)
         Length: 1
         Change Cipher Spec Message

     TLSv1.2 Record Layer: Handshake Protocol
         Encrypted Handshake Message (TLS Finished)
         Content Type: Handshake (22)
         Version: TLS 1.2 (0x0303)
         Length: 40
         Handshake Protocol: Encrypted Handshake Message
```

Figure 3: Example TLS Exchange (with Cached Info Extension).

Note: To accomplish further on-the-wire handshake size message reductions the Certificate message sent by the client can be reduced in size by using the Client Certificate URL extension.

Authors' Addresses

Stefan Santesson
3xA Security AB
Scheelev. 17
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