DICE
Internet-Draft
Updates: 5077, 5246 (if approved)
Intended status: Experimental
Expires: April 21, 2014

R. Hummen, Ed.
COMSYS, RWTH Aachen
J. Gilger
IT-Security, RWTH Aachen
H. Shafagh
ETH Zurich

October 18, 2013

Extended DTLS Session Resumption for Constrained Network Environments
draft-hummen-dtls-extended-session-resumption-01

Abstract

This draft defines two extensions for the existing session resumption mechanisms of TLS that specifically apply to Datagram TLS (DTLS) in constrained network environments. Session resumption type negotiation enables the client and the server to explicitly agree on the session resumption mechanism for subsequent handshakes, thus avoiding unnecessary overheads occurring with the existing specifications. Session resumption without client-side state additionally enables a constrained DTLS client to resume a session without the need to maintain state while the session is inactive. The extensions defined in this draft update [RFC5077] and [RFC5246].

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 [RFC2119].

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 April 21, 2014.

Table of Contents

1. Introduction

The complex processing of DTLS handshake packets and the non-negligible computational overhead of cryptographic handshake operations - especially in case of public-key cryptography - render the use of the DTLS protocol in constrained network environments challenging. One of the main goals of the DICE WG therefore is to reduce computation and transmission overheads by defining a lightweight DTLS profile that considers the special characteristics of constrained network environments.

In addition to these efforts that mainly target the properties of the base protocol, DTLS extensions afford a further adaptation of the protocol to constrained network environments. Session resumption as

defined in [RFC5077] and [RFC5246] denotes one of these extensions. Session resumption is useful in the following scenarios considering constrained environments:

- On-path soft-state middleboxes: Middleboxes such as stateful firewalls may require periodic keep-alive messages to allow for a bidirectional packet flow. If application data is transmitted in significantly larger time intervals than the keep-alive interval, session resumption allows to reduce the overall transmission overhead throughout the lifetime of a constrained device by tearing down a connection and resuming it when required.

- Short-lived server sessions: Especially large-scale Internet services often employ short-lived server sessions at the security layer to efficiently handle a multitude of clients in parallel. For periodic application data transfers, this implies that constrained clients need to perform the full DTLS handshake on a regular basis. With session resumption, constrained clients can leverage a less complex abbreviated handshake to resume a session at decreased computation and transmission cost.

- Limited server memory: A constrained server, e.g., a constrained CoAP server [I-D.ietf-core-coap], may be equipped with insufficient memory resources to handle connections for multiple clients in parallel. Session resumption allows to efficiently manage the limited memory for the per session security context by tearing down and resuming a session when required.

However, not surprisingly, the existing session resumption specifications have not specifically been designed with constrained devices (client and/or server) and networks in mind. More precisely, the abbreviated handshake in [RFC5246] requires both communication end-points to store session state across connections opportunistically. As a result of this opportunism, a constrained device may store its session state without a return on its memory investment if the DTLS peer did not maintain session state across connections as well. This is due to the lack of explicit session resumption signaling during the full handshake.

[RFC5077] enables a DTLS server to offload its state to the DTLS client for safe-keeping while the session is inactive. This mechanism largely supports the resource asymmetry when a constrained DTLS server communicates with an unconstrained DTLS client. However, it falls short for the reverse resource asymmetry, i.e., when a constrained DTLS client communicates with an unconstrained DTLS server. To leverage the vast resource difference between the DTLS client and the DTLS server in constrained network environments, there is the additional need for session resumption without client-side state.

Moreover, the roles of a DTLS client and a DTLS server may not always be readily apparent. For example, a CoAP server may not be restricted to the single role of a DTLS server, but may need to re-establish connections to other nodes due to asynchronous communication as provided by the CoAP Observe extension [I-D.ietf-core-observe]. In such situations, the CoAP server would act as a DTLS client. Hence, session resumption with state offloading also has to cover this interchangeability in roles at the DTLS layer. However, this is currently not possible when purely relying on session resumption as defined in [RFC5077].

Finally, the recommended ticket structure for stored session state as defined in [RFC5077] does not yet fully consider constrained network environments. As a result, especially certificate-based authentication leads to large ticket structures if the recommendations are followed. This in turn considerably increases transmission and memory overhead, thus requiring revised recommendations for constrained network environments.

To overcome the above shortcomings in constrained network environments, this document proposes two extensions for the existing session resumption mechanisms:

1. session resumption type negotiation, and

2. session resumption without client-side state.

Session resumption type negotiation enables the DTLS peers to explicitly negotiate the use and the type of the session resumption mechanism for the subsequent DTLS handshakes. As a result, opportunistic storing of session state is no longer required and an agreement for a specific state offloading type becomes possible. Moreover, this document specifies the required handshake signaling for session resumption without client-side state. This enables unconstrained DTLS servers to store session state on behalf of constrained DTLS clients. In combination with the existing session resumption extension specified in [RFC5077], this also allows for

session resumption when the client and server roles change at the DTLS layer.

Regarding the proposed protocol extensions, this document aims at keeping the changes to [RFC5077] minimal. To this end, the existing SessionTicket extension and the NewSessionTicket message are reused. Moreover, while this document only refers to the DTLS protocol, the defined extensions are similarly applicable to the TLS protocol.

2. Session Resumption Type Negotiation

Regarding session resumption with an abbreviated DTLS handshake as defined in [RFC5246], i.e., when both peers maintain session state across connections, DTLS currently neither provides a guarantee to the client nor to the server during the full handshake that the peer is in fact willing to store session state beyond the lifetime of the current connection. Specifically, the DTLS peers only discover during the subsequent handshake if both of them kept their session state for session resumption. However, this delayed signaling may lead to a constrained device needlessly occupying its constrained memory resources with state information while the session is inactive.

In case of session resumption without server-side state [RFC5077], the client already signals its support for this extension early during the initial full handshake by including the SessionTicket extension in the ClientHello message. The server acknowledges its own support by including the SessionTicket in the ServerHello message. Towards the end of the full handshake, the server then offloads its state to the client by means of the NewSessionTicket message. Due to this explicit negotiation in the current handshake, the client and the server do not store session state unnecessarily.

With the introduction of a third session resumption type in this document, i.e., session resumption without client-side state (see Section 3), this simple signaling mechanism introduced in [RFC5077] no longer suffices to clearly differentiate between the available session resumption types early during the Hello-phase of the DTLS handshake. Hence, additional signaling is required when reusing the SessionTicket extension for the signaling of session resumption without client-side state.

To explicitly signal the use of session resumption and to differentiate between the different state offloading types, this document defines a new session resumption type negotiation extension for the ClientHello and ServerHello messages, i.e., the ResumptionType extension. This ResumptionType extension enables the DTLS peers to clearly indicate which of the three available resumption types they support:

1. The regular abbreviated handshake (with client & server state),

2. session resumption without client-side state, and

3. session resumption without server-side state.

The integration of this extension in the DTLS handshake and the extension structure are defined in the following sections.

2.1. Protocol

The DTLS client and server use the ResumptionType extension in order to negotiate the session resumption type for the subsequent handshakes. The remaining handshake concludes as originally specified for the negotiated session resumption type. Hence, the session resumption type negotiation extends, but does not modify existing DTLS session resumption mechanisms.

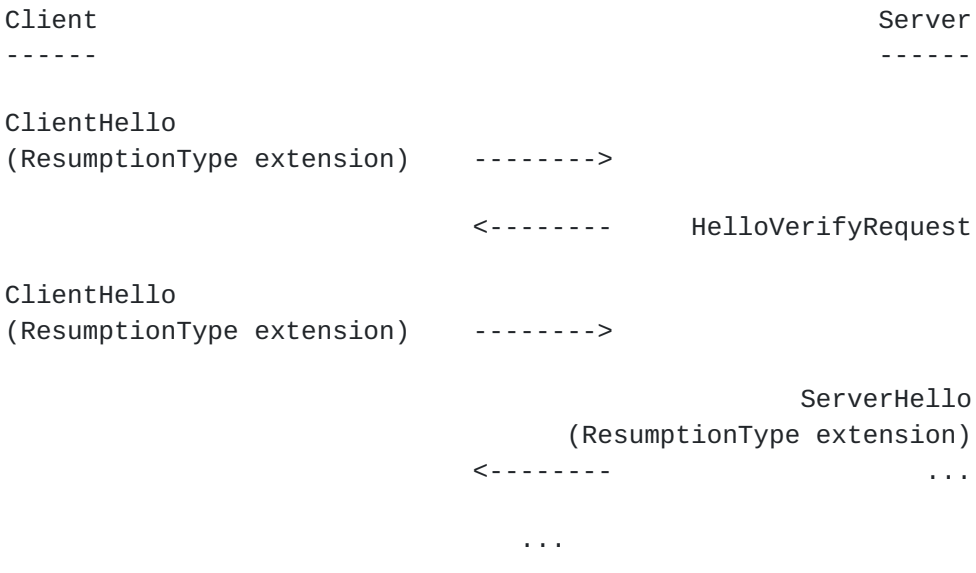

Figure 1: Message Flow for Negotiating the Session Resumption Type during a DTLS Handshake

The client adds the ResumptionType extension to its ClientHello message and indicates its supported session resumption types in the

order of preference. The server concludes the negotiation by selecting its preferred session resumption type considering the preference of the client. It signals the chosen session resumption type in the ResumptionType extension of the ServerHello message.

Each ResumptionType negotiation refers to the subsequent session resumptions. Hence, a session resumption handshake MAY omit the session resumption type negotiation. In this case, both client and server simply keep using the previously negotiated session resumption type, as long as the client and server roles have not changed. However, it is important to note that both, client and server, can resume the same DTLS session. Hence, if the roles of the client and the server have changed when the session is resumed, the ResumptionType implicitly adapts accordingly in order to keep storing session state at the same communication end-point as negotiated before. More precisely, in case of a negotiated session resumption without client-side state, state offloading follows the specified signaling of session resumption without server-side state. A negotiated session resumption without server-side state adapts vice versa. If both peers maintain session state with the regular abbreviated handshake, the change in roles does not impact this resumption type.

2.2. ResumptionType Extension

The ResumptionType extension is based on [RFC6066]. The "extension_data" field of this extension SHALL contain "ResumptionTypeList" where:

```
enum {
    abbreviated(0),
    without_client_state(1),
    without_server_state(2), (255)
} ResumptionType;

struct {
    ResumptionType resumption_type_list<1..3>
} ResumptionTypeList;
```

The ResumptionType extension may be sent in the ClientHello and ServerHello messages. The client adds the ResumptionType extension to the ClientHello message. It thereby orders the resumption types by preference. When receiving the ResumptionType extension, the server select its preferred session resumption type considering the indicated preference of the client. The server then signals the chosen session resumption type in the ResumptionType extension of the ServerHello message. Thus, the ResumptionType extension in the

ServerHello message MUST only contain a single session resumption type.

The ResumptionType extension has been assigned the number of "TBD".

3. Session Resumption Without Client-Side State

Traditional client-server communication protocols and architectures typically make the assumption of a number of clients opening connections to a single more powerful server. Scaling the system means to ensure that the server can handle the load of additional clients. With this mindset, [RFC5077] enables a DTLS server to remain stateless while the session is inactive by offloading its session state to the DTLS client.

However, in the domain of constrained network environments, not only do some devices have vastly different capabilities and resources, they regularly take the role of both client and server. In terms of higher-layer protocols such as CoAP, the distinction between client and server may still be intact while on the lower layers a device will have to accept inbound as well as establish outbound connections. This fact blurs the distinction between client and server roles at the DTLS layer.

For the communication of two devices with highly differing capabilities and resources, e.g., an unconstrained Internet host and a constrained device, enabling the constrained device to save scarce memory resources may actually help the overall system, regardless of whether it is acting as a server or a client. For example, a memory-constrained client may be able to maintain several connections sequentially, but not in parallel. Likewise, a CoAP server may take the role of a DTLS server during the initial session establishment, but re-establish the session as a DTLS client due to the asynchronous communication with CoAP Observe. To support these and other scenarios, this document introduces session resumption without client-side state in addition to the session resumption mechanisms defined in [RFC5077] and [RFC5246].

3.1. Protocol

For session resumption without client-side state, the DTLS client and server first agree on this session resumption type with a mandatory session resumption type negotiation in the full handshake. The client then sends its encrypted session state to the server.

Client	Server

Figure 2: Message Flow for Full Handshake Issuing New Session Ticket

In the full DTLS handshake, the ClientHello message contains a ResumptionType extension indicating the willingness of the client to perform session resumption without client-side state. The ClientHello message additionally contains an empty SessionTicket extension. This extension is defined in Section 3.2 of [RFC5077].

If supported and preferred by the server, the server echoes back this type in the ResumptionType extension of the ServerHello reply. The client then sends its encrypted session state to the server in the NewSessionTicket message of the fifth message flight. The ticket contains the necessary information for the client to resume the session at a later point in time. The NewSessionTicket message is defined in Section 3.3 of [RFC5077].

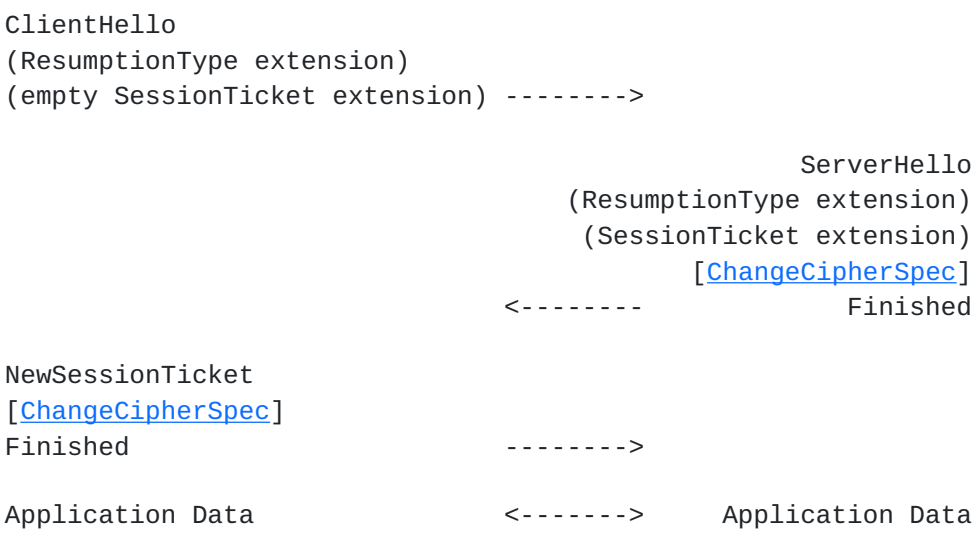

Figure 3: Message Flow for Abbreviated Handshake Using New Session Ticket

When the stateless client subsequently connects to the same server, it is oblivious of the previous full handshake. Hence, the ClientHello message in the abbreviated handshake is equal to the full handshake. On receipt of the ClientHello message, the server tries to re-identify the client (e.g. based on the source IP address or other identifying information) and searches for a matching session ticket. If it finds a matching ticket, it sends the stored session ticket to the client. To this end, the server adds the SessionTicket extension with the corresponding session ticket to its ServerHello reply.

If the client is able to authenticate and to decrypt the SessionTicket received by the server, it resumes the previous session. The client can additionally send its new session state in the NewSessionTicket message for the subsequent handshake.

4. Revised Recommended Ticket Construction

Section 4 of [RFC5077] recommends a ticket construction that may lead to an excessive ticket size for constrained network environments. This recommended ticket construction, for example, includes an entire certificate chain as the client identity in case of certificate-based authentication. The aim of this section is to provide revised recommendations for the ticket construction that take device and network constraints into account.

As defined in [RFC5077], the NewSessionTicket handshake message contains a lifetime value and a session ticket. The lifetime indicates the number of seconds until the ticket expires relative to

the time of ticket issuing. The ticket structure is opaque to the peer storing the ticket while the session is active. Only the ticket issuer needs to access the session ticket information. Hence, the specific structure of the ticket is not subject to interoperability concerns.

The revised session ticket has the following structure:

```
struct {
    opaque key_name[8];
    opaque iv[16];
    opaque encrypted_state<0..2^16-1>;
    opaque ccm_auth_tag[8];
} ticket;
```

Regarding the above structure, key_name refers to the key used by the ticket issuer to protect the confidentiality and integrity of the offloaded session state information. To allow for early detection of forged session tickets during the session resumption handshake, the key_name SHOULD be generated randomly. The ticket issuer MUST take care that it does not use the same key_name for different keys.

The session state information of the revised ticket is protected by AES CCM with an 8 byte authentication tag (see [RFC3610]). The integrity protection includes the key_name and the encrypted_state. The key_name and iv are transmitted in plain. The shorter authentication tag compared to the recommendation in [RFC5077] denotes a trade-off between a lower ticket expansion and a higher probability of forgery. Moreover, with AES CCM, the stateless peer only needs to maintain a single 128-bit key instead of one 128-bit key for encryption and one 256-bit key for authentication purposes.

The StatePlaintext structure describes the unencrypted session state information carried in a session ticket. In this document, we define a new structure for the peer_identity, which is called client_identity in [RFC5077]. The renaming was deemed necessary due to the fact that a ticket can now be generated by a client as well as a server.

```
struct {
    ProtocolVersion protocol_version;
    CipherSuite cipher_suite;
    CompressionMethod compression_method;
    opaque master_secret[48];
    PeerIdentity peer_identity;
    uint32 timestamp;
} StatePlaintext;
```

```
        enum {
           anonymous(0),
           certificate_based(1),
           psk(2)
        } PeerAuthenticationType;

        struct {
            PeerAuthenticationType peer_authentication_type;
            select (PeerAuthenticationType) {
                case anonymous: struct {};
                case certificate_based:
                    uint32 certificate_lifetime_hint;
                case psk:
                    opaque psk_identity<0..2^16-1>;
            };
        } PeerIdentity;
```

Here, the certificate_lifetime_hint indicates how long the validated certificate chain remains valid. To this end, the certificate_lifetime_hint holds the minimum lifetime for all certificates in a chain in seconds. If the time indicated in the lifetime hint is exceeded, a full handshake MUST be performed. Additional information may need to be added to the ticket structure in future revisions of this document in order to enable a state-offloading peer to validate the certificate status via a Certificate Revocation List (CRL) or the Online Certificate Status Protocol (OCSP) during the session resumption handshake.

5. Security Considerations

Session resumption without client-side state as defined in this document is strongly based on [RFC5077]. As such, the security considerations discussed in Section 5 of [RFC5077] apply here as well. Additional security considerations stem from the introduction of the new ResumptionType extension.

5.1. Session Resumption Type Negotiation

The ResumptionType extension is part of the regular DTLS handshake and thus covered by the hash in the Finished message. Hence, an on-path attacker cannot enforce a particular session resumption type without the peers noticing.

6. IANA Considerations

This document specifies the new ResumptionType extension for DTLS. The corresponding IANA considerations will be addressed in a future version of this document.

7. Acknowledgements

The authors would like to thank Shahid Raza for the discussion and comments regarding the extensions defined in this document. We especially acknowledge the prototyping and implementation efforts of Hossein Shafagh that confirm the feasibility of the proposed extensions in constrained network environments. Finally, the authors appreciate the feedback and suggestions of Sandeep Kumar. This work is funded by the DFG Cluster of Excellence on Ultra High- Speed Mobile Information and Communication (UMIC).

8. Changelog

8.1. Version 1

- Add scenarios where session resumption is beneficial

- Add section on ticket construction

- Minor editorial changes

8.2. Version 0

- Initial version

9. Informative References

[I-D.ietf-core-coap]
Shelby, Z., Hartke, K., and C. Bormann, "Constrained Application Protocol (CoAP)", draft-ietf-core-coap-18 (work in progress), June 2013.

[I-D.ietf-core-observe]
Hartke, K., "Observing Resources in CoAP", draft-ietf-core-observe-08 (work in progress), February 2013.

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

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

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

[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.

Authors' Addresses

Rene Hummen (editor)
Chair of Communication and Distributed Systems, RWTH Aachen
Ahornstrasse 55
Aachen 52074
Germany

Email: hummen@comsys.rwth-aachen.de
URI: http://www.comsys.rwth-aachen.de/team/rene-hummen/

Johannes Gilger
Research Group IT-Security, RWTH Aachen
Mies-van-der-Rohe Strasse 15
Aachen 52074
Germany

Email: gilger@itsec.rwth-aachen.de
URI: http://itsec.rwth-aachen.de/people/johannes-gilger/

Hossein Shafagh
ETH Zurich
Universitaetstrasse 6
Zurich 8092
Switzerland

Email: shafgah@inf.ethz.ch
URI: http://www.inf.ethz.ch/~mshafagh/