Network Working Group
Request for Comments: 5077
Obsoletes: 4507
Category: Standards Track

J. Salowey
H. Zhou
Cisco Systems
P. Eronen
Nokia
H. Tschofenig
Nokia Siemens Networks

January 2008

# Transport Layer Security (TLS) Session Resumption without Server-Side State

## Status of This Memo



## Abstract

This document describes a mechanism that enables the Transport Layer Security (TLS) server to resume sessions and avoid keeping per-client session state. The TLS server encapsulates the session state into a ticket and forwards it to the client. The client can subsequently resume a session using the obtained ticket. This document obsoletes RFC 4507.

Table of Contents

## 1. Introduction

This document defines a way to resume a Transport Layer Security (TLS) session without requiring session-specific state at the TLS server. This mechanism may be used with any TLS ciphersuite. This document applies to both TLS 1.0 defined in [RFC2246], and TLS 1.1 defined in [RFC4346]. The mechanism makes use of TLS extensions defined in [RFC4366] and defines a new TLS message type.

This mechanism is useful in the following situations:

1. servers that handle a large number of transactions from different users

2. servers that desire to cache sessions for a long time

3. ability to load balance requests across servers

4. embedded servers with little memory

This document obsoletes RFC 4507 [RFC4507] to correct an error in the encoding that caused the specification to differ from deployed implementations. At the time of this writing, there are no known implementations that follow the encoding specified in RFC 4507. This update to RFC 4507 aligns the document with currently deployed implementations. More details of the change are given in Appendix A.

## 2. Terminology

Within this document, the term 'ticket' refers to a cryptographically protected data structure that is created and consumed by the server to rebuild session-specific state.

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].

## 3. Protocol

This specification describes a mechanism to distribute encrypted session-state information to the client in the form of a ticket and a mechanism to present the ticket back to the server. The ticket is created by a TLS server and sent to a TLS client. The TLS client presents the ticket to the TLS server to resume a session. Implementations of this specification are expected to support both mechanisms. Other specifications can take advantage of the session tickets, perhaps specifying alternative means for distribution or selection. For example, a separate specification may describe an

alternate way to distribute a ticket and use the TLS extension in this document to resume the session. This behavior is beyond the scope of the document and would need to be described in a separate specification.

## 3.1. Overview

The client indicates that it supports this mechanism by including a SessionTicket TLS extension in the ClientHello message. The extension will be empty if the client does not already possess a ticket for the server. The server sends an empty SessionTicket extension to indicate that it will send a new session ticket using the NewSessionTicket handshake message. The extension is described in Section 3.2.

If the server wants to use this mechanism, it stores its session state (such as ciphersuite and master secret) to a ticket that is encrypted and integrity-protected by a key known only to the server. The ticket is distributed to the client using the NewSessionTicket TLS handshake message described in Section 3.3. This message is sent during the TLS handshake before the ChangeCipherSpec message, after the server has successfully verified the client's Finished message.

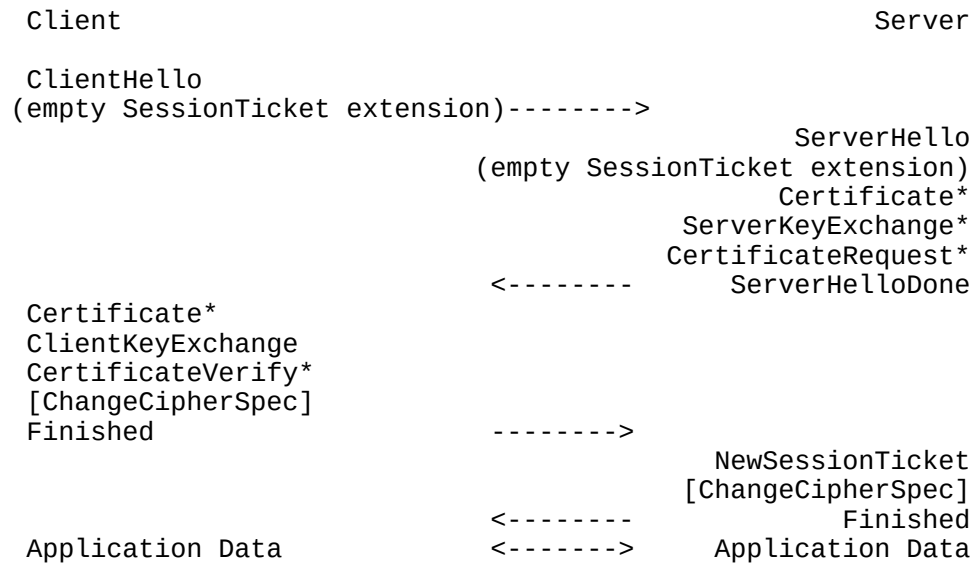


Figure 1: Message Flow for Full Handshake Issuing New Session Ticket

The client caches this ticket along with the master secret and other parameters associated with the current session. When the client wishes to resume the session, it includes the ticket in the SessionTicket extension within the ClientHello message. Appendix A provides a detailed description of the encoding of the extension and changes from RFC 4507. The server then decrypts the received ticket, verifies the ticket's validity, retrieves the session state from the contents of the ticket, and uses this state to resume the session. The interaction with the TLS Session ID is described in Section 3.4. If the server successfully verifies the client's ticket, then it may renew the ticket by including a NewSessionTicket handshake message after the ServerHello.

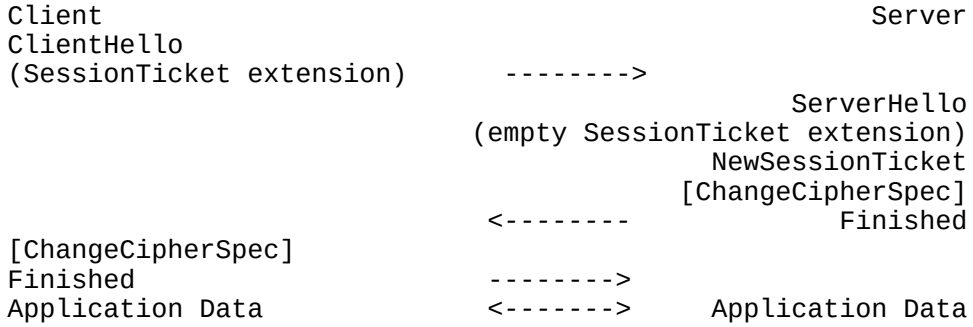


Figure 2: Message Flow for Abbreviated Handshake Using New Session Ticket

A recommended ticket format is given in Section 4.

If the server cannot or does not want to honor the ticket, then it can initiate a full handshake with the client.

In the case that the server does not wish to issue a new ticket at this time, it just completes the handshake without including a SessionTicket extension or NewSessionTicket handshake message. This is shown below (this flow is identical to Figure 1 in RFC 4346, except for the SessionTicket extension in the first message):

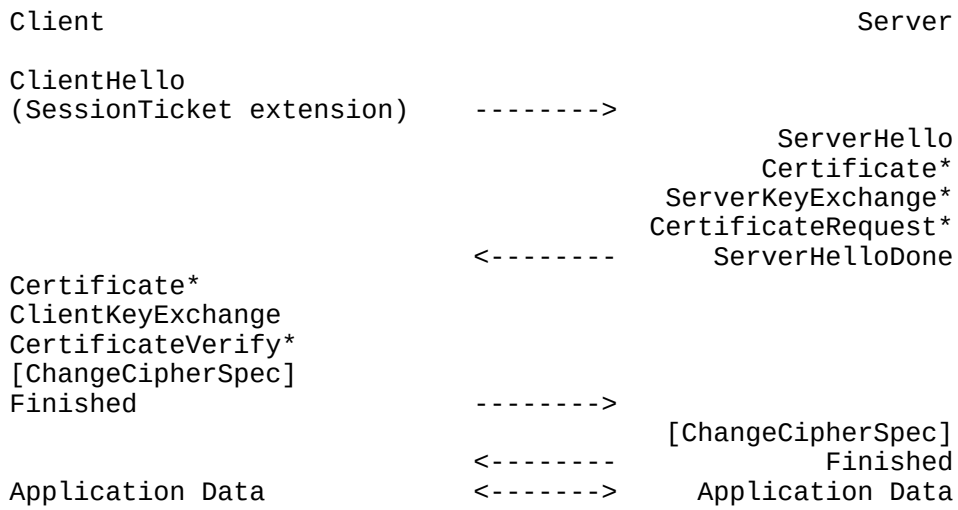


Figure 3: Message Flow for Server Completing Full Handshake Without Issuing New Session Ticket

It is also permissible to have an exchange similar to Figure 3 using the abbreviated handshake defined in Figure 2 of RFC 4346, where the client uses the SessionTicket extension to resume the session, but the server does not wish to issue a new ticket, and therefore does not send a SessionTicket extension.

If the server rejects the ticket, it may still wish to issue a new ticket after performing the full handshake as shown below (this flow is identical to Figure 1, except the SessionTicket extension in the ClientHello is not empty):

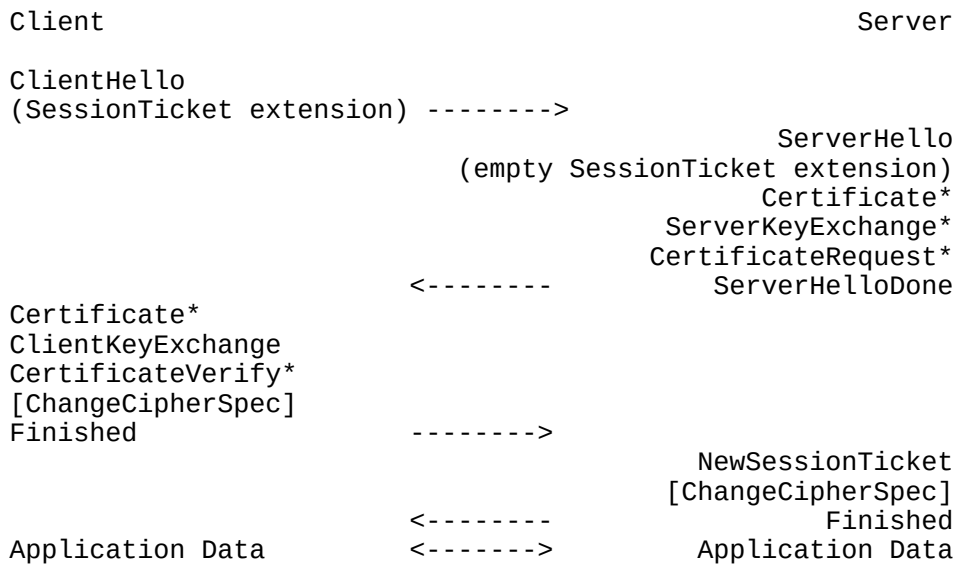


Figure 4: Message Flow for Server Rejecting Ticket, Performing Full Handshake, and Issuing New Session Ticket

## 3.2. SessionTicket TLS Extension

The SessionTicket TLS extension is based on [RFC4366]. The format of the ticket is an opaque structure used to carry session-specific state information. This extension may be sent in the ClientHello and ServerHello.

If the client possesses a ticket that it wants to use to resume a session, then it includes the ticket in the SessionTicket extension in the ClientHello. If the client does not have a ticket and is prepared to receive one in the NewSessionTicket handshake message, then it MUST include a zero-length ticket in the SessionTicket extension. If the client is not prepared to receive a ticket in the NewSessionTicket handshake message, then it MUST NOT include a SessionTicket extension unless it is sending a non-empty ticket it received through some other means from the server.

The server uses a zero-length SessionTicket extension to indicate to the client that it will send a new session ticket using the NewSessionTicket handshake message described in Section 3.3. The server MUST send this extension in the ServerHello if it wishes to issue a new ticket to the client using the NewSessionTicket handshake message. The server MUST NOT send this extension if it does not receive one in the ClientHello.

If the server fails to verify the ticket, then it falls back to performing a full handshake. If the ticket is accepted by the server but the handshake fails, the client SHOULD delete the ticket.

The SessionTicket extension has been assigned the number 35. The extension_data field of SessionTicket extension contains the ticket.

### 3.3. NewSessionTicket Handshake Message

This message is sent by the server during the TLS handshake before the ChangeCipherSpec message. This message MUST be sent if the server included a SessionTicket extension in the ServerHello. This message MUST NOT be sent if the server did not include a SessionTicket extension in the ServerHello. This message is included in the hash used to create and verify the Finished message. In the case of a full handshake, the server MUST verify the client's Finished message before sending the ticket. The client MUST NOT treat the ticket as valid until it has verified the server's Finished message. If the server determines that it does not want to include a ticket after it has included the SessionTicket extension in the ServerHello, then it sends a zero-length ticket in the NewSessionTicket handshake message.

If the server successfully verifies the client's ticket, then it MAY renew the ticket by including a NewSessionTicket handshake message after the ServerHello in the abbreviated handshake. The client should start using the new ticket as soon as possible after it verifies the server's Finished message for new connections. Note that since the updated ticket is issued before the handshake completes, it is possible that the client may not put the new ticket into use before it initiates new connections. The server MUST NOT assume that the client actually received the updated ticket until it successfully verifies the client's Finished message.

The NewSessionTicket handshake message has been assigned the number 4 and its definition is given at the end of this section. The ticket_lifetime_hint field contains a hint from the server about how long the ticket should be stored. The value indicates the lifetime in seconds as a 32-bit unsigned integer in network byte order relative to when the ticket is received. A value of zero is reserved to indicate that the lifetime of the ticket is unspecified. A client SHOULD delete the ticket and associated state when the time expires. It MAY delete the ticket earlier based on local policy. A server MAY treat a ticket as valid for a shorter or longer period of time than what is stated in the ticket_lifetime_hint.

```
struct {
    HandshakeType msg_type;
    uint24 length;
    select (HandshakeType) {
        case hello_request:       HelloRequest;
        case client_hello:        ClientHello;
        case server_hello:        ServerHello;
        case certificate:         Certificate;
        case server_key_exchange: ServerKeyExchange;
        case certificate_request: CertificateRequest;
        case server_hello_done:   ServerHelloDone;
        case certificate_verify:  CertificateVerify;
        case client_key_exchange: ClientKeyExchange;
        case finished:            Finished;
        case session_ticket:      NewSessionTicket; /* NEW */
    } body;
} Handshake;


struct {
    uint32 ticket_lifetime_hint;
    opaque ticket<0..2^16-1>;
} NewSessionTicket;
```

### 3.4. Interaction with TLS Session ID

If a server is planning on issuing a session ticket to a client that does not present one, it SHOULD include an empty Session ID in the ServerHello. If the server rejects the ticket and falls back to the full handshake then it may include a non-empty Session ID to indicate its support for stateful session resumption. If the client receives a session ticket from the server, then it discards any Session ID that was sent in the ServerHello.

When presenting a ticket, the client MAY generate and include a Session ID in the TLS ClientHello. If the server accepts the ticket and the Session ID is not empty, then it MUST respond with the same Session ID present in the ClientHello. This allows the client to easily differentiate when the server is resuming a session from when it is falling back to a full handshake. Since the client generates a Session ID, the server MUST NOT rely upon the Session ID having a particular value when validating the ticket. If a ticket is presented by the client, the server MUST NOT attempt to use the Session ID in the ClientHello for stateful session resumption. Alternatively, the client MAY include an empty Session ID in the ClientHello. In this case, the client ignores the Session ID sent in the ServerHello and determines if the server is resuming a session by the subsequent handshake messages.

## 4. Recommended Ticket Construction

This section describes a recommended format and protection for the ticket. Note that the ticket is opaque to the client, so the structure is not subject to interoperability concerns, and implementations may diverge from this format. If implementations do diverge from this format, they must take security concerns seriously. Clients MUST NOT examine the ticket under the assumption that it complies with this document.

The server uses two different keys: one 128-bit key for Advanced Encryption Standard (AES) [AES] in Cipher Block Chaining (CBC) mode [CBC] encryption and one 256-bit key for HMAC-SHA-256 [RFC4634].

The ticket is structured as follows:

```
struct {
    opaque key_name[16];
    opaque iv[16];
    opaque encrypted_state<0..2^16-1>;
    opaque mac[32];
} ticket;
```

Here, key_name serves to identify a particular set of keys used to protect the ticket. It enables the server to easily recognize tickets it has issued. The key_name should be randomly generated to avoid collisions between servers. One possibility is to generate new random keys and key_name every time the server is started.

The actual state information in encrypted_state is encrypted using 128-bit AES in CBC mode with the given IV. The Message Authentication Code (MAC) is calculated using HMAC-SHA-256 over key_name (16 octets) and IV (16 octets), followed by the length of the encrypted_state field (2 octets) and its contents (variable length).

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

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

      struct {
          ClientAuthenticationType client_authentication_type;
          select (ClientAuthenticationType) {
              case anonymous: struct {};
              case certificate_based:
                  ASN.1Cert certificate_list<0..2^24-1>;
              case psk:
                  opaque psk_identity<0..2^16-1>;   /* from [RFC4279] */
          };
       } ClientIdentity;
```

The structure StatePlaintext stores the TLS session state including the master_secret. The timestamp within this structure allows the TLS server to expire tickets. To cover the authentication and key exchange protocols provided by TLS, the ClientIdentity structure contains the authentication type of the client used in the initial exchange (see ClientAuthenticationType). To offer the TLS server with the same capabilities for authentication and authorization, a certificate list is included in case of public-key-based authentication. The TLS server is therefore able to inspect a number of different attributes within these certificates. A specific implementation might choose to store a subset of this information or additional information. Other authentication mechanisms, such as Kerberos [RFC2712], would require different client identity data. Other TLS extensions may require the inclusion of additional data in the StatePlaintext structure.

## 5. Security Considerations

This section addresses security issues related to the usage of a ticket. Tickets must be authenticated and encrypted to prevent modification or eavesdropping by an attacker. Several attacks described below will be possible if this is not carefully done.

Implementations should take care to ensure that the processing of tickets does not increase the chance of denial of service as described below.

### 5.1. Invalidating Sessions

The TLS specification requires that TLS sessions be invalidated when errors occur. [CSSC] discusses the security implications of this in detail. In the analysis within this paper, failure to invalidate sessions does not pose a security risk. This is because the TLS handshake uses a non-reversible function to derive keys for a session so information about one session does not provide an advantage to attack the master secret or a different session. If a session invalidation scheme is used, the implementation should verify the integrity of the ticket before using the contents to invalidate a session to ensure that an attacker cannot invalidate a chosen session.

### 5.2. Stolen Tickets

An eavesdropper or man-in-the-middle may obtain the ticket and attempt to use it to establish a session with the server; however, since the ticket is encrypted and the attacker does not know the secret key, a stolen ticket does not help an attacker resume a session. A TLS server MUST use strong encryption and integrity protection for the ticket to prevent an attacker from using a brute force mechanism to obtain the ticket's contents.

### 5.3. Forged Tickets

A malicious user could forge or alter a ticket in order to resume a session, to extend its lifetime, to impersonate another user, or to gain additional privileges. This attack is not possible if the ticket is protected using a strong integrity protection algorithm such as a keyed HMAC-SHA-256.

### 5.4. Denial of Service Attacks

The key_name field defined in the recommended ticket format helps the server efficiently reject tickets that it did not issue. However, an adversary could store or generate a large number of tickets to send

to the TLS server for verification. To minimize the possibility of a denial of service, the verification of the ticket should be lightweight (e.g., using efficient symmetric key cryptographic algorithms).

### 5.5. Ticket Protection Key Management

A full description of the management of the keys used to protect the ticket is beyond the scope of this document. A list of RECOMMENDED practices is given below.

- The keys should be generated securely following the randomness recommendations in [RFC4086].
- The keys and cryptographic protection algorithms should be at least 128 bits in strength. Some ciphersuites and applications may require cryptographic protection greater than 128 bits in strength.
- The keys should not be used for any purpose other than generating and verifying tickets.
- The keys should be changed regularly.
- The keys should be changed if the ticket format or cryptographic protection algorithms change.

### 5.6. Ticket Lifetime

The TLS server controls the lifetime of the ticket. Servers determine the acceptable lifetime based on the operational and security requirements of the environments in which they are deployed. The ticket lifetime may be longer than the 24-hour lifetime recommended in [RFC4346]. TLS clients may be given a hint of the lifetime of the ticket. Since the lifetime of a ticket may be unspecified, a client has its own local policy that determines when it discards tickets.

### 5.7. Alternate Ticket Formats and Distribution Schemes

If the ticket format or distribution scheme defined in this document is not used, then great care must be taken in analyzing the security of the solution. In particular, if confidential information, such as a secret key, is transferred to the client, it MUST be done using secure communication so as to prevent attackers from obtaining or modifying the key. Also, the ticket MUST have its integrity and confidentiality protected with strong cryptographic techniques to prevent a breach in the security of the system.

### 5.8. Identity Privacy, Anonymity, and Unlinkability

This document mandates that the content of the ticket is confidentiality protected in order to avoid leakage of its content, such as user-relevant information. As such, it prevents disclosure of potentially sensitive information carried within the ticket.

The initial handshake exchange, which was used to obtain the ticket, might not provide identity confidentiality of the client based on the properties of TLS. Another relevant security threat is the ability for an on-path adversary to observe multiple TLS handshakes where the same ticket is used, therefore concluding they belong to the same communication endpoints. Application designers that use the ticket mechanism described in this document should consider that unlinkability [ANON] is not necessarily provided.

While a full discussion of these topics is beyond the scope of this document, it should be noted that it is possible to issue a ticket using a TLS renegotiation handshake that occurs after a secure tunnel has been established by a previous handshake. This may help address some privacy and unlinkability issues in some environments.

## 6. Acknowledgements

The authors would like to thank the following people for their help with preparing and reviewing this document: Eric Rescorla, Mohamad Badra, Tim Dierks, Nelson Bolyard, Nancy Cam-Winget, David McGrew, Rob Dugal, Russ Housley, Amir Herzberg, Bernard Aboba, and members of the TLS working group.

[CSSC] describes a solution that is very similar to the one described in this document and gives a detailed analysis of the security considerations involved. [RFC2712] describes a mechanism for using Kerberos [RFC4120] in TLS ciphersuites, which helped inspire the use of tickets to avoid server state. [RFC4851] makes use of a similar mechanism to avoid maintaining server state for the cryptographic tunnel. [SC97] also investigates the concept of stateless sessions.

The authors would also like to thank Jan Nordqvist, who found the encoding error in RFC 4507, corrected by this document. In addition Nagendra Modadugu, Wan-Teh Chang, and Michael D'Errico provided useful feedback during the review of this document.

## 7. IANA Considerations

IANA has assigned a TLS extension number of 35 to the SessionTicket TLS extension from the TLS registry of ExtensionType values defined in [RFC4366].

IANA has assigned a TLS HandshakeType number 4 to the NewSessionTicket handshake type from the TLS registry of HandshakeType values defined in [RFC4346].

This document does not require any actions or assignments from IANA.

## 8. References

### 8.1. Normative References

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

[RFC2246] Dierks, T. and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[RFC4507] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 4507, May 2006.

### 8.2. Informative References

[AES] National Institute of Standards and Technology, "Advanced Encryption Standard (AES)", Federal Information Processing Standards (FIPS) Publication 197, November 2001.

[ANON] Pfitzmann, A. and M. Hansen, "Anonymity, Unlinkability, Unobservability, Pseudonymity, and Identity Management - A Consolidated Proposal for Terminology", http://dud.inf.tu-dresden.de/literatur/Anon_Terminology_v0.26-1.pdf Version 0.26, December 2005.

[CBC] National Institute of Standards and Technology, "Recommendation for Block Cipher Modes of Operation - Methods and Techniques", NIST Special Publication 800-38A, December 2001.

[CSSC] Shacham, H., Boneh, D., and E. Rescorla, "Client-side caching for TLS", Transactions on Information and System Security (TISSEC) , Volume 7, Issue 4, November 2004.

[RFC2712] Medvinsky, A. and M. Hur, "Addition of Kerberos Cipher Suites to Transport Layer Security (TLS)", RFC 2712, October 1999.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

[RFC4279] Eronen, P. and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", RFC 4279, December 2005.

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

[RFC4851] Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

[SC97] Aura, T. and P. Nikander, "Stateless Connections", Proceedings of the First International Conference on Information and Communication Security (ICICS '97) , 1997.

## Appendix A. Discussion of Changes to RFC 4507

RFC 4507 [RFC4507] defines a mechanism to resume a TLS session without maintaining server side state by specifying an encrypted ticket that is maintained on the client. The client presents this ticket to the server in a SessionTicket hello extension. The encoding in RFC 4507 used the XDR style encoding specified in TLS [RFC4346].

An error in the encoding caused the specification to differ from deployed implementations. At the time of this writing there are no known implementations that follow the encoding specified in RFC 4507. This update to RFC 4507 aligns the document with these currently deployed implementations.

Erroneous encoding in RFC 4507 resulted in two length fields; one for the extension contents and one for the ticket itself. Hence, for a ticket that is 256 bytes long and begins with the hex value FF FF, the encoding of the extension would be as follows according to RFC 4507:

```
00 23          Ticket Extension type 35
01 02          Length of extension contents
01 00          Length of ticket
FF FF .. ..    Actual ticket
```

The update proposed in this document reflects what implementations actually encode, namely it removes the redundant length field. So, for a ticket that is 256 bytes long and begins with the hex value FF FF, the encoding of the extension would be as follows according to this update:

```
00 23          Extension type 35
01 00          Length of extension contents (ticket)
FF FF .. ..    Actual ticket
```

A server implemented according to RFC 4507 receiving a ticket extension from a client conforming to this document would interpret the first two bytes of the ticket as the length of this ticket. This will result in either an inconsistent length field or in the processing of a ticket missing the first two bytes. In the first case, the server should reject the request based on a malformed length. In the second case, the server should reject the ticket based on a malformed ticket, incorrect key version, or failed decryption. A server implementation based on this update receiving an RFC 4507 extension would interpret the first length field as the

length of the ticket and include the second two length bytes as the first bytes in the ticket, resulting in the ticket being rejected based on a malformed ticket, incorrect key version, or failed decryption.

Note that the encoding of an empty SessionTicket extension was ambiguous in RFC 4507. An RFC 4507 implementation may have encoded it as:

```
00 23      Extension type 35
00 02      Length of extension contents
00 00      Length of ticket
```

or it may have encoded it the same way as this update:

```
00 23      Extension type 35
00 00      Length of extension contents
```

A server wishing to support RFC 4507 clients should respond to an empty SessionTicket extension encoded the same way as it received it.

A server implementation can construct tickets such that it can detect an RFC 4507 implementation, if one existed, by including a cookie at the beginning of the tickets that can be differentiated from a valid length. For example, if an implementation constructed tickets to start with the hex values FF FF, then it could determine where the ticket begins and determine the length correctly from the type of length fields present.

This document makes a few additional changes to RFC 4507 listed below.

- Clarifying that the server can allow session resumption using a ticket without issuing a new ticket in Section 3.1.
- Clarifying that the lifetime is relative to when the ticket is received in section 3.3.
- Clarifying that the NewSessionTicket handshake message is included in the hash generated for the Finished messages in Section 3.3.
- Clarifying the interaction with TLS Session ID in Section 3.4.
- Recommending the use of SHA-256 for the integrity protection of the ticket in Section 4.
- Clarifying that additional data can be included in the StatePlaintext structure in Section 4.

## Authors' Addresses

Joseph Salowey
Cisco Systems
2901 3rd Ave
Seattle, WA 98121
US

EMail: jsalowey@cisco.com

Hao Zhou
Cisco Systems
4125 Highlander Parkway
Richfield, OH 44286
US

EMail: hzhou@cisco.com

Pasi Eronen
Nokia Research Center
P.O. Box 407
FIN-00045 Nokia Group
Finland

EMail: pasi.eronen@nokia.com

Hannes Tschofenig
Nokia Siemens Networks
Otto-Hahn-Ring 6
Munich, Bayern 81739
Germany

EMail: Hannes.Tschofenig@nsn.com

## Full Copyright Statement



## Intellectual Property