TLS Working Group
Internet-Draft
Category: Standards Track
<draft-funk-tls-inner-application-extension-02.txt>

Paul Funk
Juniper Networks
Simon Blake-Wilson
Basic Commerce &
Industries, Inc.
Ned Smith
Intel Corp.
Hannes Tschofenig
Siemens AG
Thomas Hardjono
VeriSign Inc.
March 2006

TLS Inner Application Extension (TLS/IA)

Status of this Memo

Copyright Notice

Abstract

This document defines a new TLS extension called "Inner

(TLS/IA), additional messages are exchanged after completion of the TLS handshake, in effect providing an extended handshake prior to the start of upper layer data communications. Each TLS/IA message contains an encrypted sequence of Attribute-Value-Pairs (AVPs) from the RADIUS/Diameter namespace. Hence, the AVPs defined in RADIUS and Diameter have the same meaning in TLS/AI; that is, each attribute code point refers to the same logical attribute in any of these protocols. Arbitrary "applications" may be implemented using the AVP exchange. Possible applications include EAP or other forms of user authentication, client integrity checking, provisioning of additional tunnels, and the like. Use of the RADIUS/Diameter namespace provides natural compatibility between TLS/IA applications and widely deployed AAA infrastructures.

It is anticipated that TLS/IA will be used with and without subsequent protected data communication within the tunnel established by the handshake. For example, TLS/IA may be used to secure an HTTP data connection, allowing more robust password-based user authentication to occur than would otherwise be possible using mechanisms available in HTTP. TLS/IA may also be used for its handshake portion alone; for example, EAP-TTLSv1 encapsulates a TLS/IA handshake in EAP as a means to mutually authenticate a client and server and establish keys for a separate data connection.

Table of Contents

1 Introduction

This specification defines the TLS "Inner Application" extension. The term "TLS/IA" refers to the TLS protocol when used with the Inner Application extension.

In TLS/IA, the setup portion of TLS is extended to allow an arbitrary exchange of information between client and server within a protected tunnel established during the TLS handshake and prior to the start of upper layer TLS data communications. The TLS handshake itself is unchanged; the subsequent Inner Application exchange is conducted under the confidentiality and integrity protection that is afforded by the TLS handshake.

The primary motivation for providing this facility is to allow robust user authentication to occur as part of an "extended" handshake, in particular, user authentication that is based on password credentials, which is best conducted under the protection of an encrypted tunnel to preclude dictionary attack by

eavesdroppers. For example, the Extensible Authentication Protocol (EAP) may be used for authentication using any of a wide variety of methods as part of this extended handshake. The multi-layer approach of TLS/IA, in which a strong authentication, typically based on a server certificate, is used to protected a password-based authentication, distinguishes it from other TLS variants that rely entirely on a pre-shared key or password for security (such as [TLS-PSK]).

The protected exchange accommodates any type of client-server application, not just authentication, though authentication may often be the prerequisite for other applications to proceed. For example, TLS/IA may be used to set up HTTP connections, establish IPsec security associations (as an alternative to IKE), obtain

credentials for single sign-on, provide client integrity verification, and so on.

The new messages that are exchanged between client and server are encoded as sequences of Attribute-Value-Pairs (AVPs) from the RADIUS/Diameter namespace. Use of the RADIUS/Diameter namespace provides natural compatibility between TLS/IA applications and widely deployed AAA infrastructures. This namespace is extensible, allowing new AVPs and, thus, new applications to be defined as needed, either by standards bodies or by vendors wishing to define proprietary applications.

The TLS/IA exchange comprises one or more "phases", each of which consists of an arbitrary number of AVP exchanges followed by a confirmation exchange. Authentications occurring in any phase must be confirmed prior to continuing to the next phase. This allows applications to implement security dependencies in which particular assurances are required prior to the exchange of additional information.

1.1 A Bit of History

The TLS protocol has its roots in the Netscape SSL protocol, which was originally intended to protect HTTP traffic. It provides either one-way or mutual certificate-based authentication of client and server. In its most typical use in HTTP, the client authenticates

the server based on the server's certificate and establishes a tunnel through which HTTP traffic is passed.

For the server to authenticate the client within the TLS handshake, the client must have its own certificate. In cases where the client must be authenticated without a certificate, HTTP, not TLS, mechanisms would have to be employed. For example, HTTP headers have been defined to perform user authentications. However, these mechanisms are primitive compared to other mechanisms, most notably EAP, that have been defined for contexts other than HTTP. Furthermore, any mechanisms defined for HTTP cannot be utilized when TLS is used to protect non-HTTP traffic.

The TLS protocol has also found an important use in authentication for network access, originally within PPP for dial-up access and later for wireless and wired 802.1X access. Several EAP types have been defined that utilize TLS to perform mutual client-server authentication. The first to appear, EAP-TLS, uses the TLS handshake to authenticate both client and server based on their certificates.

Subsequently proposed protocols, such EAP-TTLSv0 and EAP-PEAP, utilize the TLS handshake to allow the client to authenticate the server based on the latter's certificate, and then use the protected channel established by the TLS handshake to perform user authentication, typically based on a password. Such protocols are

called "tunneled" EAP protocols. The authentication mechanism used inside the tunnel may itself be EAP, and the tunnel may also be used to convey additional information between client and server.

While tunneled authentication would be useful in other contexts besides EAP, the tunneled protocols mentioned above cannot be employed in a more general use of TLS, since the outermost protocol is EAP, not TLS. Furthermore, these protocols use the TLS tunnel to carry authentication exchanges, and thus preclude use of the TLS tunnel for other purposes such as carrying HTTP traffic.

TLS/IA provides a means to perform user authentication and other message exchanges between client and server strictly within TLS. TLS/IA can thus be used both for flexible user authentication within a TLS session and as a basis for tunneled authentication within EAP.

The TLS/IA approach is to insert an additional message exchange between the TLS handshake and the subsequent data communications phase. This message exchange is carried in a new record type, which is distinct from the record type that carries upper layer data. Thus, the data portion of the TLS exchange becomes available for HTTP or another protocol that needs to be secured.

1.2 TLS With or Without Upper Layer Data Communications

It is anticipated that TLS/IA will be used with and without subsequent protected data communication within the tunnel established by the handshake.

For example, TLS/IA may be used to protect an HTTP connection, allowing more robust password-based user authentication to occur within the TLS/IA extended handshake than would otherwise be possible using mechanisms available in HTTP.

TLS/IA may also be used for its handshake portion alone. For example, EAP-TTLSv1 encapsulates a TLS/IA extended handshake in EAP as a means to mutually authenticate a client and server and establish keys for a separate data connection; no subsequent TLS data portion is required. Another example might be the use of TLS/IA directly over TCP in order to provide a user with credentials for single sign-on.

2 The Inner Application Extension to TLS

The Inner Application extension to TLS follows the guidelines of [RFC3546].

A new extension type is defined for negotiating use of TLS/IA:

- The InnerApplicationExtension extension type. The client proposes use of this extension by including a InnerApplicationExtension message in its ClientHello handshake message, and the server confirms its use by including a InnerApplicationExtension message in its ServerHello handshake message.

A new record type (ContentType) is defined for use in TLS/IA:

- The InnerApplication record type. This record type carries all messages that are exchanged after the TLS handshake and prior to exchange of data.

A new message type is defined for use within the InnerApplication record type:

- The InnerApplication message. This message may encapsulate any of the three following subtypes:

  - The ApplicationPayload message. This message is used to carry AVP (Attribute-Value Pair) sequences within the TLS/IA extended handshake, in support of client-server applications such as authentication.

  - The IntermediatePhaseFinished message. This message confirms session keys established during the current TLS/IA phase, and indicates that at least one additional phase is to follow.

  - The FinalPhaseFinished message. This message confirms session keys established during the current TLS/IA phase, and indicates that no further phases are to follow.

Two new alert codes are defined for use in TLS/IA:

- The InnerApplicationFailure alert. This error alert allows either party to terminate the TLS/IA extended handshake due to a failure in an application implemented via AVP sequences carried in ApplicationPayload messages.

- The InnerApplicationVerification alert. This error alert allows either party to terminate the TLS/IA extended handshake due to incorrect verification data in a received IntermediatePhaseFinished or FinalPhaseFinished message.

The following new assigned numbers are used in TLS/IA:

- "InnerApplicationExtension" extension type: 37703

- "InnerApplication" record type: 24

- "InnerApplicationFailure" alert code: 208

- "InnerApplicationVerification" alert code: 209

[Editor's note: I have not checked these types yet against types defined in RFCs or drafts. The TLS RFC specifies that new record types use the next number after ones already defined; hence I used 24, though I don't know if that is already taken.]

2.1 TLS/IA Message Exchange

In TLS/IA, zero or more "application phases are inserted after the TLS handshake and prior to ordinary data exchange. The last such application phase is called the "final phase"; any application phases prior to the final phase are called "intermediate phases".

Intermediate phases are only necessary if interim confirmation of session keys generated during an application phase is desired.

Each application phase consists of ApplicationPayload handshake messages exchanged by client and server to implement applications such as authentication, plus concluding messages for cryptographic confirmation. These messages are encapsulated in records with ContentType of InnerApplication.

All application phases prior to the final phase conclude with an exchange of IntermediatePhaseFinished messages, or conclude with a FinalPhaseFinished message from the server and an IntermediatePhaseFinished message from the client, by which the client indicates its desire to keep the handshake open for one or more additional phases. The final phase concludes with an exchange of FinalPhaseFinished messages.

Application phases may be omitted entirely only when session resumption is used, provided both client and server agree that no application phase is required. The client indicates in its ClientHello whether it is willing to omit application phases in a resumed session, and the server indicates in its ServerHello whether any application phases are to ensue.

In each application phase, the client sends the first ApplicationPayload message. ApplicationPayload messages are traded one at a time between client and server, until the server concludes the phase by sending, in response to an ApplicationPayload message from the client, an IntermediatePhaseFinished or FinalPhaseFinished sequence to conclude the phase. The client then responds with its own IntermediatePhaseFinished or FinalPhaseFinished message.

The server determines which type of concluding message it wants to use, either IntermediatePhaseFinished or FinalPhaseFinished. If the server sent an IntermediatePhaseFinished, the client MUST respond

with an IntermediatePhaseFinished. If the server sent a FinalPhaseFinished, the client MAY respond with a FinalPhaseFinished to complete the handshake, or MAY respond with an IntermediatePhaseFinished to cause the handshake to continue. Thus,

conclusion of the entire handshake occurs only when both client and server have been satisfied.

Note that the server MUST NOT send an IntermediatePhaseFinished or FinalPhaseFinished message immediately after sending an ApplicationPayload message. It must allow the client to send an ApplicationPayload message prior to concluding the phase. Thus, within any application phase, there will be one more ApplicationPayload message sent by the client than sent by the server.

At the start of each application phase, the server MUST wait for the client's opening ApplicationPayload message before it sends its own ApplicationPayload message to the client. The client MUST NOT initiate conclusion of an application phase by sending the first IntermediatePhaseFinished or FinalPhaseFinished message; it MUST allow the server to initiate the conclusion of the phase.

Each IntermediatePhaseFinished or FinalPhaseFinished message provides cryptographic confirmation of any session keys generated during the current and any prior applications phases.

Each ApplicationPayload message contains opaque data interpreted as an AVP (Attribute-Value Pair) sequence. Each AVP in the sequence contains a typed data element. The exchanged AVPs allow client and server to implement "applications" within a secure tunnel. An application may be any procedure that someone may usefully define. A typical application might be authentication; for example, the server may authenticate the client based on password credentials using EAP. Other possible applications include distribution of keys, validating client integrity, setting up IPsec parameters, setting up SSL VPNs, and so on.

Note that it is perfectly acceptable for either client or server to send an ApplicationPayload message containing no AVPs. The client, for example, may have no AVPs to send in its first or last ApplicationPayload message during an application phase.

An "inner secret" is computed during each application phase that cryptographically combines the TLS master secret with any session keys that have been generated during the current and any previous application phases. At the conclusion of each application phase, a new inner secret is computed and is used to create verification data that is exchanged via the IntermediatePhaseFinished or FinalPhaseFinished messages. By mixing session keys of inner authentications with the TLS master secret, certain man-in-the-middle attacks are thwarted [MITM].

2.2 Inner Secret

The inner secret is a 48-octet value used to confirm that the endpoints of the TLS handshake are the same entities as the endpoints of the inner authentications that may have been performed during each application phase.

The inner secret is initialized to the master secret at the conclusion of the TLS handshake. At the conclusion of each application phase, prior to computing verification data for inclusion in the IntermediatePhaseFinished or FinalPhaseFinished message, each party permutes the inner secret using a PRF that includes session keys produced during the current application phase. The value that results replaces the current inner secret and is used to compute the verification data.

```
inner_secret = PRF(inner_secret,
                   "inner secret permutation",
                   SecurityParameters.server_random +
                   SecurityParameters.client_random +
                    session_key_material) [0..48];
```

session_key_material is the concatenation of session_key vectors, one for each session key generated during the current phase, where:

```
opaque session_key<1..2^16-1>;
```

In other words, each session key is prefixed by a 2-octet length to produce the session_key vector.

Since multiple session keys may be produced during a single application phase, the following method is used to determine the order of concatenation: Each session key is treated as an unsigned big-endian numeric value, and the set of session keys is ordered from lowest to highest. The session keys are then converted to session_key vectors and concatenated in the determined order to form session_key_material.

If no session keys were generated during the current phase, session_key_material will be null.

Note that session_key_material itself is not a vector and therefore not prefixed with the length of the entire collection of session_key vectors.

Note that, within TLS itself, the inner secret is used for verification only, not for encryption. However, the inner secret resulting from the final application phase may be exported for use as a key from which additional session keys may be derived for arbitrary purposes, including encryption of data communications separate from TLS.

An exported inner secret should not be used directly for any cryptographic purpose. Instead, additional keys should be derived from the inner secret, for example by using a PRF. This ensures cryptographic separation between use of the inner secret for session key confirmation and additional use of the inner secret outside TLS/IA.

2.2.1 Application Session Key Material

Many authentication protocols used today generate session keys that are bound to the authentication. Such keying material is normally intended for use in a subsequent data connection for encryption and validation. For example, EAP-TLS, MS-CHAP-V2, and EAP-MS-CHAP-V2 generate session keys.

Any session keys generated during an application phase MUST be used to permute the TLS/IA inner secret between one phase and the next,

and MUST NOT be used for any other purpose.

Each authentication protocol may define how the session key it generates is mapped to an octet sequence of some length for the purpose of TLS/IA mixing. However, for protocols which do not specify this (including the multitude of protocols that pre-date TLS/IA) the following rules are defined. The first rule that applies SHALL be the method for determining the session key.

- If the authentication protocol produces an MSK (as defined in [RFC3784]), the MSK is used as the session key. Note that an MSK is 64 octets.

- If the authentication protocol maps its keying material to the RADIUS attributes MS-MPPE-Recv-Key and MS-MPPE-Send-Key [RFC2548], then the keying material for those attributes are concatenated, with MS-MPPE-Recv-Key first (Note that this rule applies to MS-CHAP-V2 and EAP-MS-CHAP-V2.)

- If the authentication protocol uses a pseudo-random function to generate keying material, that function is used to generate 64 octets for use as keying material.

Providing verification of the binding of session keys to the TLS master secret is necessary to preclude man-in-the-middle attacks against tunneled authentication protocols, as described in [MITM]. In such an attack, an unsuspecting client is induced to perform an untunneled authentication with an attacker posing as a server; the attacker then introduces the authentication protocol into a tunneled authentication protocol, fooling an authentic server into believing that the attacker is the authentic user.

By mixing both the TLS master secret and session keys generated during application phase authentication into the inner secret used

for application phase verification, such attacks are thwarted, as it guarantees that the same client acted as the endpoint for both the TLS handshake and the application phase authentication. Note that the session keys generated during authentication must be cryptographically bound to the authentication and not derivable from data exchanged during authentication in order for the keying material to be useful in thwarting such attacks.

In addition, the fact that the inner secret cryptographically incorporates session keys from application phase authentications provides additional protection when the inner secret is exported for the purpose of generating additional keys for use outside of the TLS exchange. If such an exported secret did not include keying material from inner authentications, an eavesdropper who somehow knew the server's private key could, in an RSA-based handshake, determine the exported secret and hence would be able to compute the additional keys that are based on it. When inner authentication keying material, unknown to the attacker, is incorporated into the exported secret, such an attack becomes infeasible.

2.3 Session Resumption

A TLS/IA initial handshake phase may be resumed using standard mechanisms defined in [RFC2246]. When the TLS session is resumed, client and server may not deem it necessary to exchange AVPs in one or more additional application phases, as the resumption itself may provide the necessary security.

The client indicates within the InnerApplicationExtension message in ClientHello whether it requires AVP exchange when session resumption occurs. If it indicates that it does not, then the server may at its option omit application phases and the two parties proceed to upper layer data communications immediately upon completion of the TLS handshake. The server indicates whether application phases are to follow the TLS handshake in its InnerApplication extension message in ServerHello.

Note that [RFC3546] specifically states that when session resumption is used, the server MUST ignore any extensions in the ClientHello. However, it is not possible to comply with this requirement for the Inner Application extension, since even in a resumed session it may be necessary to include application phases, and whether they must be included is negotiated in the extension message itself. Therefore, the [RFC3546] provision is explicitly overridden for the single case of the Inner Application extension, which is considered an exception to this rule.

A TLS/IA session MAY NOT be resumed if an application phase resulted in failure, even though the TLS handshake itself succeeded. Both client and server MUST NOT save session state for possible future

resumption unless the TLS handshake and all subsequent application phases have been successfully executed.

2.4 Error Termination

The TLS/IA handshake may be terminated by either party sending a fatal alert, following standard TLS procedures.

2.5 Negotiating the Inner Application Extension

Use of the InnerApplication extension follows [RFC3546]. The client proposes use of this extension by including the InnerApplicationExtension message in the client_hello_extension_list of the extended ClientHello. If this message is included in the ClientHello, the server MAY accept the proposal by including the InnerApplicationExtension message in the server_hello_extension_list of the extended ServerHello. If use of this extension is either not proposed by the client or not confirmed by the server, the InnerApplication record type MUST NOT be used.

2.6 InnerApplication Protocol

All specifications of TLS/IA messages follow the usage defined in [RFC2246].

2.6.1 InnerApplicationExtension

```
enum {
   no(0), yes(1), (255)
} AppPhaseOnResumption;

struct {
   AppPhaseOnResumption app_phase_on_resumption;
} InnerApplicationExtension;
```

If the client wishes to propose use of the Inner Application extension, it must include the InnerApplicationExtension message in the extension_data vector in the Extension structure in its extended ClientHello message.

If the server wishes to confirm use of the Inner Application extension that has been proposed by the client, it must include the InnerApplicationExtension message in the extension_data vector in the Extension structure in its extended ServerHello message.

The AppPhaseOnResumption enumeration allow client and server to negotiate an abbreviated, single-phase handshake when session

resumption is employed. If the client sets app_phase_on_resumption to "no", and if the server resumes the previous session, then the server MAY set app_phase_on_resumption to "no" in the InnerApplication message it sends to the client. If the server sets

app_phase_on_resumption to "no", no application phases occur and the TLS connection proceeds to upper layer data exchange immediately upon conclusion of the TLS handshake.

The server MUST set app_phase_on_resumption to "yes" if the client set app_phase_on_resumption to "yes" or if the server does not resume the session. The server MAY set app_phase_on_resumption to "yes" for a resumed session even if the client set app_phase_on_resumption to "no", as the server may have reason to proceed with one or more application phases.

If the server sets app_phase_on_resumption to "yes" for a resumed session, then the client MUST initiate an application phase at the conclusion of the TLS handshake.

The value of app_phase_on_resumption applies to the current handshake only; that is, it is possible for app_phase_on_resumption to have different values in two handshakes that are both resumed from the same original TLS session.

2.6.2 InnerApplication Message

```
enum {
   application_payload(0), intermediate_phase_finished(1),
   final_phase_finished(2), (255)
} InnerApplicationType;

struct {
   InnerApplicationType msg_type;
   uint24 length;
   select (InnerApplicationType) {
      case application_payload:       ApplicationPayload;
      case intermediate_phase_finished:
   IntermediatePhaseFinished;
      case final_phase_finished:      FinalPhaseFinished;
      } body;
   } InnerApplication;
```

The InnerApplication message carries any of the message types defined for the InnerApplication protocol.

2.6.3 IntermediatePhaseFinished and FinalPhaseFinished Messages

```
struct {
   opaque verify_data[12];
} PhaseFinished;

PhaseFinished IntermediatePhaseFinished;

PhaseFinished FinalPhaseFinished;
```



```
verify_data
   PRF(inner_secret, finished_label) [0..11];

finished_label
   when sent by the client, the string "client phase finished"
   when sent by the server, the string "server phase finished"
```

The IntermediatePhaseFinished and FinalPhaseFinished messages have the same structure and include verification data based on the current inner secret. IntermediatePhaseFinished is sent by the server and echoed by the client to conclude an intermediate application phase, and FinalPhaseFinished is used in the same manner to conclude the final application phase.

2.6.4 The ApplicationPayload Message

The ApplicationPayload message carries an AVP sequence during an application handshake phase. It is defined as follows:

```
struct {
   opaque avps[InnerApplication.length];
} ApplicationPayload;

avps
   The AVP sequence, treated as an opaque sequence of octets.

InnerApplication.length
```

The length field in the encapsulating InnerApplication message.

Note that the "avps" element has its length defined in square bracket rather than angle bracket notation, implying a fixed rather than variable length vector. This avoids having the length of the AVP sequence specified redundantly both in the encapsulating InnerApplication message and as a length prefix in the avps element itself.

2.7 Alerts

Two new alert codes are defined for use during an application phase. The AlertLevel for either of these alert codes MUST be set to "fatal".

InnerApplicationFailure
: An InnerApplicationFailure error alert may be sent by either party during an application phase. This indicates that the sending party considers the negotiation to have failed due to an application carried in the AVP sequences, for example, a failed authentication.

InnerApplicationVerification
: An InnerApplicationVerification error alert is sent by either party during an application phase to indicate that the received IntermediatePhaseFinished or FinalPhaseFinished is invalid.

Note that other alerts are possible during an application phase; for example, decrypt_error. The InnerApplicationFailure alert relates specifically to the failure of an application implemented via AVP sequences; for example, failure of an EAP or other authentication method, or information passed within the AVP sequence that is found unsatisfactory.

3 Encapsulation of AVPs within ApplicationPayload Messages

During application phases of the TLS handshake, information is exchanged between client and server through the use of attribute-value pairs (AVPs). This data is encrypted using the current cipher

state.

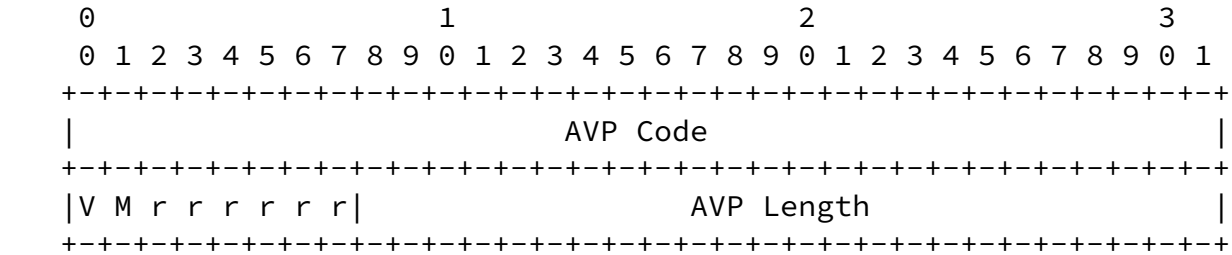
The AVP format chosen for TLS/IA is compatible with the Diameter AVP format. This does not in any way represent a requirement that Diameter be supported by any of the devices or servers participating in the TLS/IA conversation, whether directly as client or server or indirectly as a backend authenticator. Use of this format is merely a convenience. Diameter is a superset of RADIUS and includes the RADIUS attribute namespace by definition, though it does not limit the size of an AVP as does RADIUS. RADIUS, in turn, is a widely deployed AAA protocol and attribute definitions exist for the encapsulation of EAP as well as all commonly used non-EAP password authentication protocols.

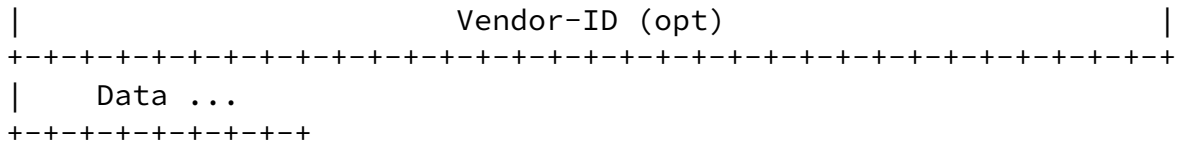
Thus, Diameter is not considered normative except as specified in this document. Specifically, the AVP Codes used in TLS/IA are semantically equivalent to those defined for Diameter, and, by extension, RADIUS.

Use of the RADIUS/Diameter namespace allows a TLS/IA server to translate between AVPs it uses to communicate with clients and the protocol requirements of AAA servers that are widely deployed. Additionally, it provides a well-understood mechanism to allow vendors to extend that namespace for their particular requirements.

3.1 AVP Format

The format of an AVP is shown below. All items are in network, or big-endian, order; that is, they have most significant octet first.





AVP Code

The AVP Code is four octets and, combined with the Vendor-ID field if present, identifies the attribute uniquely. The first 256 AVP numbers represent attributes defined in RADIUS. AVP numbers 256 and above are defined in Diameter.

AVP Flags

The AVP Flags field is one octet, and provides the receiver with information necessary to interpret the AVP.

The 'V' (Vendor-Specific) bit indicates whether the optional Vendor-ID field is present. When set to 1, the Vendor-ID field is present and the AVP Code is interpreted according to the namespace defined by the vendor indicated in the Vendor-ID field.

The 'M' (Mandatory) bit indicates whether support of the AVP is required. When set to 0, this indicates that the AVP may be safely ignored if the receiving party does not understand or support it. When set to 1, if the receiving party does not understand or support the AVP it MUST fail the negotiation by sending an InnerApplicationFailure error alert.

The 'r' (reserved) bits are unused and must be set to 0.

AVP Length

The AVP Length field is three octets, and indicates the length of this AVP including the AVP Code, AVP Length, AVP Flags, Vendor-ID (if present) and Data.

Vendor-ID

The Vendor-ID field is present if and only if the 'V' bit is set in the AVP Flags field. It is four octets, and contains the vendor's IANA-assigned "SMI Network Management Private Enterprise Codes" [RFC1700] value. Vendors defining their own AVPs must

maintain a consistent namespace for use of those AVPs within RADIUS, Diameter and TLS/IA.

A Vendor-ID value of zero is semantically equivalent to absence of the Vendor-ID field altogether.

3.2 AVP Sequences

Data encapsulated within the TLS Record Layer must consist entirely of a sequence of zero or more AVPs. Each AVP must begin on a 4-octet boundary relative to the first AVP in the sequence. If an AVP is not a multiple of 4 octets, it must be padded with 0s to the next 4-octet boundary.

Note that the AVP Length does not include the padding.

3.3 Guidelines for Maximum Compatibility with AAA Servers

When maximum compatibility with AAA servers is desired, the following guidelines for AVP usage are suggested:

- Non-vendor-specific AVPs should be selected from the set of attributes defined for RADIUS; that is, attributes with codes less than 256. This provides compatibility with both RADIUS and Diameter.

- Vendor-specific AVPs should be defined in terms of RADIUS. Vendor-specific RADIUS attributes translate to Diameter automatically; the reverse is not true. RADIUS vendor-specific attributes use RADIUS attribute 26 and include vendor ID, vendor-specific attribute code and length; see [RFC2865] for details.

4 Tunneled Authentication within Application Phases

TLS/IA permits user authentication information to be tunneled within an application phase between client and server, protecting the authentication information against active and passive attack.

Any type of authentication method may be tunneled. Also, multiple tunneled authentications may be performed. Normally, tunneled authentication is used when the TLS handshake provides only one-way authentication of the server to the client; however, in certain cases it may be desirable to perform certificate authentication of the client during the initial handshake phase as well as tunneled user authentication in a subsequent application phase.

This section establishes rules for using well known authentication

mechanisms within TLS/IA. Any new authentication mechanism should, in general, be covered by these rules if it is defined as an EAP type. Authentication mechanisms whose use within TLS/IA is not covered within this specification may require separate

standardization, preferably within the standard that describes the authentication mechanism in question.

4.1 Implicit challenge

Certain authentication protocols that use a challenge/response mechanism rely on challenge material that is not generated by the authentication server, and therefore require special handling.

In PPP protocols such CHAP, MS-CHAP and MS-CHAP-V2, for example, the Network Access Server (NAS) issues a challenge to the client, the client then hashes the challenge with the password and forwards the response to the NAS. The NAS then forwards both challenge and response to a AAA server. But because the AAA server did not itself generate the challenge, such protocols are susceptible to replay attack.

Since within TLS/IA the client also plays the role of NAS, the replay problem is exacerbated. If the client were able to create both challenge and response, anyone able to observe a CHAP or MS-CHAP exchange could pose as that user by replaying that challenge and response into a TLS/IA conversation.

To make these protocols secure in TLS/IA, it is necessary to provide a mechanism that produces a challenge that the client cannot control or predict.

When a challenge-based authentication mechanism is used, both client and server use the TLS PRF function to generate as many octets as are required for the challenge, using the constant string "inner application challenge", based on the master secret and random values established during the TLS handshake, as follows.

```
IA_challenge = PRF(SecurityParameters.master_secret,
                   "inner application challenge",
                   SecurityParameters.server_random +
                   SecurityParameters.client_random);
```

4.2 Tunneled Authentication Protocols

This section describes the rules for tunneling specific authentication protocols within TLS/IA.

For each protocol, the RADIUS RFC that defines the relevant attribute formats is cited. Note that these attributes are encapsulated as described in section 3.1; that is, as Diameter attributes, not as RADIUS attributes. In other words, the AVP Code, Length, Flags and optional Vendor-ID are formatted as described in section 3.1, while the Data is formatted as described by the cited RADIUS RFC.

All tunneled authentication protocols except EAP must be initiated by the client in the first ApplicationPayload message of an application phase. EAP may be initiated by the client in the first ApplicationPayload message of an application phase; it may also be initiated by the server in any ApplicationPayload message.

The authentication protocols described below may be performed directly by the TLS/IA server or may be forwarded to a backend AAA server. For authentication protocols that generate session keys, the backend server must return those session keys to the TLS/IA server in order to allow the protocol to succeed within TLS/IA. RADIUS or Diameter servers are suitable backend AAA servers for this purpose. RADIUS servers typically return session keys in MS-MPPE-Recv-Key and MS-MPPE-Send-Key attributes [RFC2548]; Diameter servers return session keys in the EAP-Master-Session-Key AVP [AAA-EAP].

4.2.1 EAP

EAP is described in [RFC3784]; RADIUS attribute formats are described in [RFC3579].

When EAP is the tunneled authentication protocol, each tunneled EAP packet between the client and server is encapsulated in an EAP-Message AVP.

Either the client or the server may initiate EAP.

The client is the first to transmit within any application phase, and it may include an EAP-Response/Identity AVP in its ApplicationPayload message to begin an EAP conversation. Alternatively, if the client does not initiate EAP the server may, by including an EAP-Request/Identity AVP in its ApplicationPayload message.

The client's EAP-Response/Identity provides the username, which MUST be a Network Access Identifier (NAI) [RFC2486]; that is, it MUST be in the following format:

```
username@realm
```

The @realm portion is optional, and is used to allow the server to forward the EAP message sequence to the appropriate server in the AAA infrastructure when necessary.

The EAP authentication between client and server proceeds normally, as described in [RFC3784]. However, upon completion the server does not send an EAP-Success or EAP-Failure AVP. Instead, the server signals success when it concludes the application phase by issuing a Finished or PhaseFinished message, or it signals failure by issuing an InnerApplicationFailure alert.

Note that the client may also issue an InnerApplicationFailure alert, for example, when authentication of the server fails in a method providing mutual authentication.

4.2.2 CHAP

The CHAP algorithm is described in [RFC1994]; RADIUS attribute formats are described in [RFC2865].

Both client and server generate 17 octets of challenge material, using the constant string "inner application challenge" as described above. These octets are used as follows:

```
CHAP-Challenge    [16 octets]
CHAP Identifier   [1 octet]
```

The client initiates CHAP by including User-Name, CHAP-Challenge and

CHAP-Password AVPs in the first ApplicationPayload message in any application phase. The CHAP-Challenge value is taken from the challenge material. The CHAP-Password consists of CHAP Identifier, taken from the challenge material; and CHAP response, computed according to the CHAP algorithm.

Upon receipt of these AVPs from the client, the server must verify that the value of the CHAP-Challenge AVP and the value of the CHAP Identifier in the CHAP-Password AVP are equal to the values generated as challenge material. If either item does not match, the server must reject the client. Otherwise, it validates the CHAP-Challenge to determine the result of the authentication.

4.2.3 MS-CHAP

The MS-CHAP algorithm is described in [RFC2433]; RADIUS attribute formats are described in [RFC2548].

Both client and server generate 9 octets of challenge material, using the constant string "inner application challenge" as described above. These octets are used as follows:

```
MS-CHAP-Challenge [8 octets]
Ident              [1 octet]
```

The client initiates MS-CHAP by including User-Name, MS-CHAP-Challenge and MS-CHAP-Response AVPs in the first ApplicationPayload message in any application phase. The MS-CHAP-Challenge value is taken from the challenge material. The MS-CHAP-Response consists of Ident, taken from the challenge material; Flags, set according the client preferences; and LM-Response and NT-Response, computed according to the MS-CHAP algorithm.

Upon receipt of these AVPs from the client, the server must verify that the value of the MS-CHAP-Challenge AVP and the value of the Ident in the client's MS-CHAP-Response AVP are equal to the values generated as challenge material. If either item does not match exactly, the server must reject the client. Otherwise, it validates the MS-CHAP-Challenge to determine the result of the authentication.

4.2.4 MS-CHAP-V2

The MS-CHAP-V2 algorithm is described in [RFC2759]; RADIUS attribute formats are described in [RFC2548].

Both client and server generate 17 octets of challenge material, using the constant string "inner application challenge" as described above. These octets are used as follows:

```
MS-CHAP-Challenge [16 octets]
Ident              [1 octet]
```

The client initiates MS-CHAP-V2 by including User-Name, MS-CHAP-Challenge and MS-CHAP2-Response AVPs in the first ApplicationPayload message in any application phase. The MS-CHAP-Challenge value is taken from the challenge material. The MS-CHAP2-Response consists of Ident, taken from the challenge material; Flags, set to 0; Peer-Challenge, set to a random value; and Response, computed according to the MS-CHAP-V2 algorithm.

Upon receipt of these AVPs from the client, the server must verify that the value of the MS-CHAP-Challenge AVP and the value of the Ident in the client's MS-CHAP2-Response AVP are equal to the values generated as challenge material. If either item does not match exactly, the server must reject the client. Otherwise, it validates the MS-CHAP2-Challenge.

If the MS-CHAP2-Challenge received from the client is correct, the server tunnels the MS-CHAP2-Success AVP to the client.

Upon receipt of the MS-CHAP2-Success AVP, the client is able to authenticate the server. In its next InnerApplicationPayload message to the server, the client does not include any MS-CHAP-V2 AVPs. (This may result in an empty InnerApplicationPayload if no other AVPs need to be sent.)

If the MS-CHAP2-Challenge received from the client is not correct, the server tunnels an MS-CHAP2-Error AVP to the client. This AVP contains a new Ident and a string with additional information such as error reason and whether a retry is allowed. If the error reason is an expired password and a retry is allowed, the client may proceed to change the user's password. If the error reason is not an expired password or if the client does not wish to change the user's password, it issues an InnerApplicationFailure alert.

If the client does wish to change the password, it tunnels MS-CHAP-NT-Enc-PW, MS-CHAP2-CPW, and MS-CHAP-Challenge AVPs to the server. The MS-CHAP2-CPW AVP is derived from the new Ident and Challenge received in the MS-CHAP2-Error AVP. The MS-CHAP-Challenge AVP simply echoes the new Challenge.

Upon receipt of these AVPs from the client, the server must verify that the value of the MS-CHAP-Challenge AVP and the value of the Ident in the client's MS-CHAP2-CPW AVP match the values it sent in the MS-CHAP2-Error AVP. If either item does not match exactly, the server must reject the client. Otherwise, it validates the MS-CHAP2-CPW AVP.

If the MS-CHAP2-CPW AVP received from the client is correct, and the server is able to change the user's password, the server tunnels the MS-CHAP2-Success AVP to the client and the negotiation proceeds as described above.

Note that additional AVPs associated with MS-CHAP-V2 may be sent by the server; for example, MS-CHAP-Domain. The server must tunnel such authentication-related AVPs along with the MS-CHAP2-Success.

4.2.5 PAP

PAP RADIUS attribute formats are described in [RFC2865].

The client initiates PAP by including User-Name and User-Password AVPs in the first ApplicationPayload message in any application phase.

In RADIUS, User-Password is padded with nulls to a multiple of 16 octets, then encrypted using a shared secret and other packet information.

A TLS/IA, however, does not RADIUS-encrypt the password since all application phase data is already encrypted. The client SHOULD, however, null-pad the password to a multiple of 16 octets, to obfuscate its length.

Upon receipt of these AVPs from the client, the server may be able to decide whether to authenticate the client immediately, or it may need to challenge the client for more information.

If the server wishes to issue a challenge to the client, it MUST tunnel the Reply-Message AVP to the client; this AVP normally contains a challenge prompt of some kind. It may also tunnel additional AVPs if necessary, such the Prompt AVP. Upon receipt of

the Reply-Message AVPs, the client tunnels User-Name and User-Password AVPs again, with the User-Password AVP containing new information in response to the challenge. This process continues

until the server determines the authentication has succeeded or failed.

4.3 Performing Multiple Authentications

In some cases, it is desirable to perform multiple user authentications. For example, a server may want first to authenticate the user by password, then by a hardware token.

The server may perform any number of additional user authentications using EAP, simply by issuing a EAP-Request with a new protocol type once the previous authentication has completed.

For example, a server wishing to perform MD5-Challenge followed by Generic Token Card would first issue an EAP-Request/MD5-Challenge AVP and receive a response. If the response is satisfactory, it would then issue EAP-Request/Generic Token Card AVP and receive a response. If that response were also satisfactory, it would consider the user authenticated.

5 Example Message Sequences

This section presents a variety of possible TLS/IA message sequences. These examples are not meant to exhaustively depict all possible scenarios.

Parentheses indicate optional TLS messages. Brackets indicate optional message exchanges. An ellipsis (. . .) indicates optional repetition of preceding messages.

5.1 Full Initial Handshake with Intermediate and Final Application Phases

The diagram below depicts a full initial handshake phase followed by two application phases.

Note that the client concludes the intermediate phase and starts the

final phase in an uninterrupted sequence of three messages: ChangeCipherSpec and PhaseFinished belong to the intermediate phase, and ApplicationPayload belongs to the final phase.

```
      Client                                               Server
      ------                                               ------

*** TLS Handshake:
      ClientHello                  -------->
                                                      ServerHello
                                                     (Certificate)
                                                ServerKeyExchange
                                               (CertificateRequest)
                                   <--------      ServerHelloDone


```



```
      (Certificate)
      ClientKeyExchange
      (CertificateVerify)
      ChangeCipherSpec
      Finished                     -------->
                                                 ChangeCipherSpec
                                   <--------        Finished

*** Intermediate Phase:
      ApplicationPayload           -------->

    [
                                   <--------   ApplicationPayload

      ApplicationPayload           -------->

                                      ...
    ]
                                   <--------
IntermediatePhaseFinished
      IntermediatePhaseFinished
*** Final Phase:
      ApplicationPayload           -------->

    [
                                   <--------   ApplicationPayload
```

```
      ApplicationPayload            -------->

                                        ...
    ]
                                    <--------   FinalPhaseFinished

      FinalPhaseFinished            -------->
```

5.2 Resumed Session with Single Application Phase

The diagram below depicts a resumed session followed by a single application phase.

Note that the client concludes the initial phase and starts the final phase in an uninterrupted sequence of three messages: ChangeCipherSpec and PhaseFinished belong to the initial phase, and ApplicationPayload belongs to the final phase.

```
      Client                                                Server
      ------                                                ------

*** TLS Handshake:
      ClientHello                   -------->
                                                      ServerHello
```

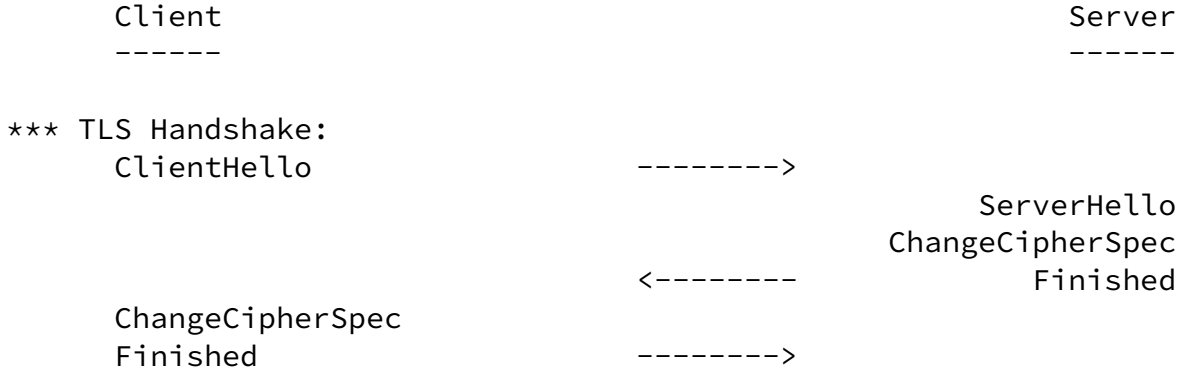


```
                                                  ChangeCipherSpec
                                    <--------             Finished
      ChangeCipherSpec
      Finished
*** Final Phase:
      ApplicationPayload            -------->

    [
                                    <--------   ApplicationPayload

      ApplicationPayload            -------->

                                        ...
    ]
                                    <--------   FinalPhaseFinished

      FinalPhaseFinished            -------->
```

5.3 Resumed Session with No Application Phase

The diagram below depicts a resumed session without any subsequent application phase. This will occur if the client indicates in its ClientInnerApplication message that no application phase is required and the server concurs.

Note that this message sequence is identical to that of a standard TLS resumed session.



6 Security Considerations

This document introduces a new TLS extension called "Inner Application". When TLS is used with the Inner Application extension (TLS/IA), additional messages are exchanged during the TLS handshake. Hence a number of security issues need to be taken into consideration. Since the security heavily depends on the information (called "applications") which are exchanged between the TLS client and the TLS server as part of the TLS/IA extension we try to classify them into two categories: The first category considers the case where the exchange results in the generation of keying material. This is, for example, the case with certain EAP methods.

EAP is one of the envisioned main "applications". The second category focuses on cases where no session key is generated. The security treatment of the latter category is discouraged since it is subject to man-in-the-middle attacks if the two sessions cannot be bound to each other as suggested in [MITM].

In the following, we investigate a number of security issues:

- Architecture and Trust Model

  For many of the use cases in this document we assume that three functional entities participate in the protocol exchange: TLS client, TLS server and a AAA infrastructure (typically consisting of a AAA server and possibly a AAA broker). The protocol exchange described in this document takes place between the TLS client and the TLS server. The interaction between the AAA client (which corresponds to the TLS server) and the AAA server is described in the respective AAA protocol documents and therefore outside the scope of this document. The trust model behind this architecture with respect to the authentication, authorization, session key establishment and key transport within the AAA infrastructure is discussed in [KEYING].

- Authentication

  This document assumes that the TLS server is authenticated to the TLS client as part of the authentication procedure of the initial TLS Handshake. This approach is similar to the one chosen with the EAP support in IKEv2 (see [IKEv2]). Typically, public key based server authentication is used for this purpose. More interesting is the client authentication property whereby information exchanged as part of the Inner Application is used to authenticate (or authorize) the client. For example, if EAP is used as an inner application then EAP methods are used to perform authentication and key agreement between the EAP peer (most likely the TLS client) and the EAP server (i.e., AAA server).

- Authorization

  Throughout this document it is assumed that the TLS server can be authorized by the TLS client as a legitimate server as part of the authentication procedure of the initial TLS Handshake. The entity acting as TLS client can be authorized either by the TLS server or by the AAA server (if the authorization decision is offloaded). Typically, the authenticated identity is used to compute the authorization decision but credential-based authorization mechanisms may be used as well.

- Man-in-the-Middle Attack

Man-in-the-middle attacks have become a concern with tunneled authentication protocols because of the discovered vulnerabilities (see [MITM]) of a missing cryptographic binding between the independent protocol sessions. This document also proposes a tunneling protocol, namely individual inner application sessions are tunneled within a previously executed session. The first protocol session in this exchange is the initial TLS Handshake. To avoid man-in-the-middle attacks, Section 2.2 addresses how to establish such a cryptographic binding.

- User Identity Confidentiality

  The TLS/IA extension allows splitting the authentication of the TLS server from the TLS client into two separate sessions. As one of the advantages, this provides active user identity confidentiality since the TLS client is able to authenticate the TLS server and to establish a unilateral authenticated and confidentiality-protected channel prior to starting the client-side authentication.

- Session Key Establishment

  TLS [RFC2246] defines how session key material produced during the TLS Handshake is generated with the help of a pseudo-random function to expand it to keying material of the desired length for later usage in the TLS Record Layer. Section 2.2 gives some guidelines with regard to the master key generation. Since the TLS/IA extension supports multiple exchanges whereby each phase concludes with a generated keying material. In addition to the keying material established as part of TLS itself, most inner applications will produce their keying material. For example, keying material established as part of an EAP method must be carried from the AAA server to the AAA client. Details are subject to the specific AAA protocol (for example, EAP usage in Diameter [AAA-EAP].

- Denial of Service Attacks

  This document does not modify the initial TLS Handshake and as such, does not introduce new vulnerabilities with regard to DoS attacks. Since the TLS/IA extension allows to postpone the client-side authentication to a later stage in the protocol phase. As such, it allows malicious TLS clients to initiate a number of exchanges while remaining anonymous. As a consequence, state at the server is allocated and computational efforts are required at the server side. Since the TLS client cannot be

stateless this is not strictly a DoS attack.

- Confidentiality Protection and Dictionary Attack Resistance

Similar to the user identity confidentiality property the usage of the TLS/IA extension allows to establish a unilateral authenticated tunnel which is confidentiality protected. This tunnel protects the inner application information elements to be protected against active adversaries and therefore provides resistance against dictionary attacks when password-based authentication protocols are used inside the tunnel. In general, information exchanged inside the tunnel experiences confidentiality protection.

- Downgrading Attacks

This document defines a new extension. The TLS client and the TLS server indicate the capability to support the TLS/IA extension as part of the client_hello_extension_list and the server_hello_extension_list payload. More details can be found in Section 2.5. To avoid downgrading attacks whereby an adversary removes a capability from the list is avoided by the usage of the IntermediatePhaseFinished or FinalPhaseFinished message as described in Section 2.1.

7 References

7.1 Normative References

[RFC1700] Reynolds, J., and J. Postel, "Assigned Numbers", RFC 1700, October 1994.

[RFC1994] Simpson, W., "PPP Challenge Handshake Authentication Protocol (CHAP)", RFC 1994, August 1996.

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

[RFC2246] Dierks, T., and C. Allen, "The TLS Protocol Version 1.0", RFC 2246, November 1998.

[RFC2433] Zorn, G., and S. Cobb, "Microsoft PPP CHAP Extensions", RFC 2433, October 1998.

[RFC2486] Aboba, B., and M. Beadles, "The Network Access Identifier", RFC 2486, January 1999.

[RFC2548] Zorn, G., "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, March 1999.

[RFC2759] Zorn, G., "Microsoft PPP CHAP Extensions, Version 2", RFC 2759, January 2000.

[RFC2865] Rigney, C., Willens, S., Rubens, A., and W. Simpson, "Remote Authentication Dial In User Service (RADIUS)", RFC 2865, June 2000.

[RFC3546] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 3546, June 2003.

[RFC3579] Aboba, B., and P.Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.

[RFC3588] Calhoun, P., Loughney, J., Guttman, E., Zorn, G., and J. Arkko, "Diameter Base Protocol", RFC 3588, July 2003.

[RFC3784] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "PPP Extensible Authentication Protocol (EAP)", RFC 3784, June 2004.

7.2 Informative References

[RFC1661] Simpson, W. (Editor), "The Point-to-Point Protocol (PPP)", STD 51, RFC 1661, July 1994.

[RFC2716] Aboba, B., and D. Simon, "PPP EAP TLS Authentication

Protocol", RFC 2716, October 1999.

[EAP-TTLS] Funk, P., and S. Blake-Wilson, " EAP Tunneled TLS Authentication Protocol (EAP-TTLS)", draft-ietf-pppext-eap-ttls-05.txt, July 2004.

[EAP-PEAP] Palekar, A., Simon, D., Salowey, J., Zhou, H., Zorn, G., and S. Josefsson, "Protected EAP Protocol (PEAP) Version 2", draft-josefsson-pppext-eap-tls-eap-08.txt, July 2004.

[TLS-PSK] Eronen, P., and H. Tschofenig, "Pre-Shared Key Ciphersuites for Transport Layer Security (TLS)", draft-ietf-tls-psk-01.txt, August 2004.

[802.1X] IEEE Standards for Local and Metropolitan Area Networks: Port based Network Access Control, IEEE Std 802.1X-2001, June 2001.

[MITM] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle in Tunneled Authentication", http://www.saunalahti.fi/~asokan/research/mitm.html, Nokia Research Center, Finland, October 24 2002.

[KEYING] Aboba, B., Simon, D., Arkko, J. and H. Levkowetz, "EAP Key Management Framework", draft-ietf-eap-keying-01.txt (work in progress), October 2003.

[IKEv2] C.Kaufman, "Internet Key Exchange (IKEv2) Protocol", draft-ietf-ipsec-ikev2-16.txt (work in progress), September 2004.

[AAA-EAP] Eronen, P., Hiller, T. and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", draft-ietf-aaa-eap-03.txt (work in progress), October 2003.

8 Authors' Addresses

Questions about this memo can be directed to:

Paul Funk

Juniper Networks
222 Third Street
Cambridge, MA 02142
USA
Phone: +1 617 497-6339
E-mail: pfunk@juniper.net

Simon Blake-Wilson
Basic Commerce & Industries, Inc.
96 Spadina Ave, Unit 606
Toronto, Ontario M5V 2J6
Canada
Phone: +1 416 214-5961
E-mail: sblakewilson@bcisse.com

Ned Smith
Intel Corporation
MS: JF1-229
2111 N.E. 25th Ave.
Hillsboro, OR 97124
USA
Phone: +1 503 264-2692
E-mail: ned.smith@intel.com

Hannes Tschofenig
Siemens
Otto-Hahn-Ring 6
Munich, Bayern 81739\
Germany
Phone: +49 89 636 40390
E-mail: Hannes.Tschofenig@siemens.com

Thomas Hardjono
VeriSign Inc.

487 East Middlefield Road
M/S MV6-2-1
Mountain View, CA 94043
USA
Phone: +1 650 426-3204
E-mail: thardjono@verisign.com

9 Intellectual Property Statement

The IETF takes no position regarding the validity or scope of any Intellectual Property Rights or other rights that might be claimed to pertain to the implementation or use of the technology described in this document or the extent to which any license under such rights might or might not be available; nor does it represent that it has made any independent effort to identify any such rights. Information on the procedures with respect to rights in RFC documents can be found in BCP 78 and BCP 79.

Copies of IPR disclosures made to the IETF Secretariat and any assurances of licenses to be made available, or the result of an attempt made to obtain a general license or permission for the use of such proprietary rights by implementers or users of this specification can be obtained from the IETF on-line IPR repository at http://www.ietf.org/ipr.

The IETF invites any interested party to bring to its attention any copyrights, patents or patent applications, or other proprietary rights that may cover technology that may be required to implement this standard. Please address the information to the IETF at ietf-ipr@ietf.org.

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.