

EMU Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 12, 2012

S. Hartman, Ed.
Painless Security
T. Clancy
Electrical and Computer
Engineering
K. Hooper
Motorola, Inc.
July 11, 2011

Channel Binding Support for EAP Methods
draft-ietf-emu-chbind-08.txt

Abstract

This document defines how to implement channel bindings for Extensible Authentication Protocol (EAP) methods to address the lying NAS as well as the lying provider problem.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on January 12, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

| | |
|--|----|
| 1. Introduction | 5 |
| 2. Terminology | 6 |
| 3. Problem Statement | 6 |
| 4. Channel Bindings | 8 |
| 4.1. Types of EAP Channel Bindings | 8 |
| 4.2. Channel Bindings in the Secure Association Protocol | 9 |
| 4.3. Channel Bindings Scope | 10 |
| 5. Channel Binding Process | 12 |
| 5.1. Protocol Operation | 12 |
| 5.2. Channel Binding Consistency Check | 14 |
| 5.3. EAP Protocol | 16 |
| 5.3.1. Channel Binding Codes | 17 |
| 5.3.2. Namespace Identifiers | 17 |
| 5.3.3. Radius Namespace | 17 |
| 6. System Requirements | 18 |
| 6.1. General Transport Protocol Requirements | 18 |
| 6.2. EAP Method Requirements | 18 |
| 7. Channel Binding TLV | 19 |
| 7.1. Requirements for Lower-Layer Bindings | 19 |
| 7.2. General Attributes | 19 |
| 7.3. IEEE 802.11 | 19 |
| 7.3.1. IEEE 802.11r | 20 |
| 8. AAA-Layer Bindings | 20 |
| 9. Security Considerations | 21 |
| 9.1. Trust Model | 21 |
| 9.2. Consequences of Trust Violation | 22 |
| 9.3. Privacy Violations | 22 |
| 10. Operations and Management Considerations | 23 |
| 11. IANA Considerations | 23 |
| 12. Acknowledgements | 24 |
| 13. References | 24 |
| 13.1. Normative References | 24 |
| 13.2. Informative References | 24 |

| | |
|---|----|
| Appendix A. Attacks Prevented by Channel Bindings | 25 |
| A.1. Enterprise Subnetwork Masquerading | 25 |
| A.2. Forced Roaming | 26 |
| A.3. Downgrading attacks | 26 |
| A.4. Bogus Beacons in IEEE 802.11r | 27 |
| A.5. Forcing false authorization in IEEE 802.11i | 27 |
| Appendix B. Change History | 27 |
| B.1. Changes since Version 06 | 27 |
| B.2. Changes since version 04 | 27 |
| Authors' Addresses | 28 |

1. Introduction

The so-called "lying NAS" problem is a well-documented problem with the current Extensible Authentication Protocol (EAP) architecture [RFC3748] when used in pass-through authenticator mode. Here, a Network Access Server (NAS), or pass-through authenticator, may represent one set of information (e.g. network identity, capabilities, configuration, etc) to the backend Authentication, Authorization, and Accounting (AAA) infrastructure, while representing contrary information to EAP peers. Another possibility is that the same false information could be provided to both the EAP peer and EAP server by the NAS. A "lying" entity can also be located anywhere on the AAA path between the NAS and the EAP server.

This problem results when the same credentials are used to access multiple services that differ in some interesting property. The EAP server learns which client credentials are in use. The client knows which EAP credentials are used, but cannot distinguish between servers that use those credentials.

As a concrete example, consider an organization with two different IEEE 802.11 wireless networks. One is a relatively low-security network for reading e-mail while the other has access to valuable confidential information. An access point on the e-mail network could act as a lying NAS, sending the SSID of the confidential network in its beacons. This access point could gain an advantage by doing so if it tricks clients intending to connect to the confidential network to connect to it and disclose confidential information.

A similar problem can be observed in the context of roaming. Here, the lying entity is located in a visited service provider network, e.g. attempting to lure peers to connect to the network based on false advertized roaming rates. This is referred to as "lying provider" problem in the remainder of this document. The lying entity's motivation often is financial; the entity may be paid whenever peers roam to its service. However a lying entity in a provider network can gain access to traffic that it might not otherwise see.

This document defines and implements EAP channel bindings to solve the lying NAS and the lying provider problems, using a process in which the EAP peer provides information about the characteristics of the service provided by the authenticator to the AAA server protected within the EAP method. This allows the server to verify the authenticator is providing information to the peer that is consistent with the information received from this authenticator as well as the information stored about this authenticator. "AAA Payloads" defined

in [I-D.clancy-emu-aaapay] served as the starting point for the mechanism proposed in this specification to carry this information.

2. Terminology

In this document, several words are used to signify the requirements of the specification. These words are often capitalized. 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. Problem Statement

In a [RFC4017] compliant EAP authentication, the EAP peer and EAP server mutually authenticate each other, and derive keying material. However, when operating in pass-through mode, the EAP server can be far removed from the authenticator. A malicious or compromised authenticator may represent incorrect information about the network to the peer in an effort to affect its operation in some way. Additionally, while an authenticator may not be compromised, other compromised elements in the network (such as proxies) could provide false information to the authenticator that it could simply be relaying to EAP peers. Hence, the goal must be to ensure that the authenticator is providing correct information to the EAP peer during the initial network discovery, selection, and authentication.

There are two different types of networks to consider: enterprise networks and service provider networks. In enterprise networks, assuming a single administrative domain, it is feasible for an EAP server to have information about all the authenticators in the network. In service provider networks, global knowledge is infeasible due to indirection via roaming. When a peer is outside its home administrative domain, the goal is to ensure that the level of service received by the peer is consistent with the contractual agreement between the two service providers. The same EAP server may need to support both types of networks. For example an enterprise may have a roaming agreement permitting its users to use the networks of third-party service providers. In these situations, the EAP server may authenticate for an enterprise and provider network.

The following are example attacks possible by presenting false network information to peers.

- o Enterprise Network: A corporate network may have multiple virtual Lads (VLANs) running throughout their campus network, and have IEEE 802.11 access points connected to each VLAN. Assume one VLAN

connects users to the firewalled corporate network, while the other connects users to a public guest network. The corporate network is assumed to be free of adversarial elements, while the guest network is assumed to possibly have malicious elements. Access Points on both VLANs are serviced by the same EAP server, but broadcast different SSIDs to differentiate. A compromised access point connected to the guest network but not the corporate network could advertise the SSID of the corporate network in an effort to lure peers to connect to a network with a false sense of security regarding their traffic. Conditions and further details of this attack can be found in the Appendix.

- o Enterprise network: The EAP GSS-API mechanism [I-D.ietf-abfab-gss-eap] mechanism provides a way to use EAP to authenticate to mail servers, instant messaging servers and other non-network services. Without EAP channel binding, an attacker could trick the user into connecting to a relatively untrusted service instead of a relatively trusted service.
- o Service Provider Network: An EAP-enabled mobile phone provider could advertize very competitive flat rates but send per minute rates to the home server, thus, luring peers to connect to their network and overcharging them. In more elaborate attacks, peers can be tricked into roaming without their knowledge. For example, a mobile phone provider operating along a geo-political boundary could boost their cell towers' transmission power and advertise the network identity of the neighboring country's indigenous provider. This would cause unknowing handsets to associate with an unintended operator, and consequently be subject to high roaming fees without realizing they had roamed off their home provider's network. These types of scenarios can be considered as "lying provider" problem, because here the provider configures its NAS to broadcast false information. For the purpose of channel bindings as defined in this draft, it does not matter which local entity (or entities) is "lying" in a service provider network (local NAS, local authentication server and/or local proxies), because the only information received from the visited network that is verified by channel bindings is the information the home authentication server received from the last hop in the communication chain. In other words, channel bindings enable the detection of inconsistencies in the information from a visited network, but cannot determine which entity is lying. Naturally, channel bindings for EAP methods can only verify the endpoints and, if desirable, intermediate hops need to be protected by the employed AAA protocol.

- o Enterprise and provider networks: In a situation where an enterprise has roaming agreements with providers, a compromised access point in a provider network could masquerade as the enterprise network in an attempt to gain confidential information. Today this could potentially be solved by using different credentials for internal and external access. Depending on the type of credential this may introduce usability or man-in-the-middle security issues.

To address these problems, a mechanism is required to validate unauthenticated information advertised by EAP authenticators.

4. Channel Bindings

EAP channel bindings seek to authenticate previously unauthenticated information provided by the authenticator to the EAP peer, by allowing the peer and server to compare their perception of network properties in a secure channel.

It should be noted that the definition of EAP channel bindings differs somewhat from channel bindings documented in [RFC5056], which seek to securely bind together the end points of a multi-layer protocol, allowing lower layers to protect data from higher layers. Unlike [RFC5056], EAP channel bindings do not ensure the binding of different layers of a session but rather the information advertised to EAP peer by an authenticator acting as pass-through device during an EAP execution. The term channel bindings was independently adopted by these two related concepts; by the time the conflict was discovered, a wide body of literature existed for each usage. EAP channel bindings could be used to provide RFC 5056 channel bindings. In particular, an inner EAP method could be bound to an outer method by including the RFC 5056 channel binding data for the outer channel in the inner EAP method's channel bindings. Doing so would provide a facility similar to EAP cryptographic binding, except that a man-in-the-middle could not extract the inner method from the tunnel. This specification does not weigh the advantages of doing so nor specify how to do so; the example is provided only to illustrate how EAP channel binding and RFC 5056 channel binding overlap.

4.1. Types of EAP Channel Bindings

There are two categories of approach to EAP channel bindings:

- o After keys have been derived during an EAP execution, the peer and server can, in an integrity-protected channel, exchange plaintext information about the network with each other, and verify consistency and correctness.

- o The peer and server can both uniquely encode their respective view of the network information without exchanging it, resulting into an opaque blob that can be included directly into the derivation of EAP session keys.

Both approaches are only applicable to key deriving EAP methods and both have advantages and disadvantages. Various hybrid approaches are also possible. Advantages of exchanging plaintext information include:

- o It allows for policy-based comparisons of network properties, rather than requiring precise matches for every field, which achieves a policy-defined consistency, rather than bitwise equality. This allows network operators to define which properties are important and even verifiable in their network.
- o EAP methods that support extensible, integrity-protected channels can easily include support for exchanging this network information. In contrast, direct inclusion into the key derivation would require more extensive revisions to existing EAP methods or a wrapper EAP method.
- o Given it doesn't affect the key derivation, this approach facilitates debugging, incremental deployment, backward compatibility and a logging mode in which verification results are recorded but do not have an effect on the remainder of the EAP execution. The exact use of the verification results can be subject to the network policy. Additionally, consistent information canonicalization and formatting for the key derivation approach would likely cause significant deployment problems.

The following are advantages of directly including channel binding information in the key derivation:

- o EAP methods not supporting extensible, integrity-protected channels could still be supported, either by revising their key derivation, revising EAP, or wrapping them in a universal method that supports channel binding.
- o It can guarantee proper channel information, since subsequent communication would be impossible if differences in channel information yielded different session keys on the EAP peer and server.

4.2. Channel Bindings in the Secure Association Protocol

This document describes channel bindings performed by transporting channel binding information as part of an integrity-protected

exchange within an EAP method. Alternatively, some future document could specify a mechanism for transporting channel bindings within the lower layer's secure association protocol. Such a specification would need to describe how channel bindings are exchanged over the lower layer protocol between the peer and authenticator. In addition, since the EAP exchange concludes before the secure association protocol begins, a mechanism for transporting the channel bindings from the authenticator to the EAP server needs to be specified. A mechanism for transporting a protected result from the EAP server, through the authenticator, back to the peer needs to be specified.

The channel bindings MUST be transported with integrity protection based on a key known only to the peer and EAP server. The channel bindings SHOULD be confidentiality protected using a key known only to the peer and EAP server. For the system to function, the EAP server or AAA server needs access to the channel binding information from the peer as well as the AAA attributes and a local database described later in this document.

The primary advantage of sending channel bindings as part of the secure association protocol is that EAP methods need not be changed. The disadvantage is that a new AAA exchange is required, and secure association protocols need to be changed. As the result of the secure association protocol change, every NAS needs to be upgraded to support channel bindings within the secure association protocol.

For many deployments, changing all the NASes is expensive and adding channel binding support to enough EAP methods to meet the goals of the deployment will be cheaper. However for deployment of new equipment, or especially deployment of a new lower layer technology, changing the NASes may be cheaper than changing EAP methods. Especially if such a deployment needed to support a large number of EAP methods, sending channel bindings in the secure association protocol might make sense.

If channel bindings using a secure association protocol is specified, semantics as well as the set of information that peers exchange can be shared with the mechanism described in this document.

4.3. Channel Bindings Scope

The scope of EAP channel bindings differs somewhat depending on the type of deployment in which they are being used. In enterprise networks, they can be used to authenticate very specific properties of the authenticator (e.g. MAC address, supported link types and data rates, etc), while in service provider networks they can generally only authenticate broader information about a roaming

partner's network (e.g. network name, roaming information, link security requirements, etc). The reason for the difference has to do with the amount of information about the authenticator and/or network to which the peer is connected the home EAP server is expected to have access to. In roaming cases, the home server is likely to only have access to information contained in their roaming agreements.

With any multi-hop AAA infrastructure, many of the NAS-specific AAA attributes are obscured by the AAA proxy that's decrypting, reframing, and retransmitting the underlying AAA messages. Especially service provider networks are affected by this and the AAA information received from the last hop may not contain much verifiable information any longer. For example, information carried in AAA attributes such as the NAS IP address may have been lost in transition and are thus not known to the EAP server. This affects the ability of the EAP server to verify specific NAS properties. However, often verification of the MAC or IP address of the NAS is not useful for improving the overall security posture of a network. More often it is useful to make policy decisions about services being offered to peers. For example, in an IEEE 802.11 network, the EAP server may wish to ensure that peers connecting to the corporate intranet are using secure link-layer encryption, while link-layer security requirements for peers connecting to the guest network could be less stringent. These types of policy decisions can be made without knowing or being able to verify the IP address of the NAS through which the peer is connecting.

The properties of the network that the peer wishes to validate depend on the specific deployment. In a mobile phone network, peers generally don't care what the name of the network is, as long as they can make their phone call and are charged the expected amount for the call. However, in an enterprise network the administrators of a peer may be more concerned with specifics of where their network traffic is being routed and what VLAN is in use. To establish policies surrounding these requirements administrators would capture some attribute such as SSID to describe the properties of the network they care about. Channel bindings could validate the SSID. The administrator would need to make sure that the network guarantees that when an authenticator trusted by the AAA infrastructure to offer a particular SSID to clients does offer this SSID, that network has the intended properties. Generally it is not possible for channel bindings to detect lying NAS behavior when the NAS is authorized to claim a particular service. That is, if the same physical authenticator is permitted to advertize two networks, the AAA infrastructure is unlikely to be able to determine when this authenticator lies.

As discussed in the next section, some of the most important

information to verify cannot come from AAA attributes but instead comes from local configuration. For example in the mobile phone case, the expected roaming rate cannot come from the roaming provider without being verified against the contract between the two providers. Similarly, in an enterprise, the SSID a particular access point is expected to advertize is a matter of configuration rather than something that can be trusted because it is included in an AAA exchange.

Channel bindings can be important for forming pockets of trust, especially when provider networks are involved, and exact information is not available to the EAP server. Without channel bindings, all entities in the system need to be held to the standards of the most trusted entity that could be accessed using the EAP credential. Otherwise, a less trusted entity can impersonate a more trusted entity. However when channel bindings are used, the EAP server can use information supplied by the peer, AAA protocols and local database to distinguish less trusted entities from more trusted entities. One possible deployment involves being able to verify a number of characteristics about relatively trusted entities while for other entities simply verifying that they are less trusted.

Any deployment of channel bindings should take into consideration both what information the EAP server is likely to know or have access to, and also what type of network information the peer would want and need authenticated.

5. Channel Binding Process

This section defines the process for verifying channel binding information during an EAP authentication. The protocol uses the approach where plaintext data is exchanged, since it allows channel bindings to be used more flexibly in varied deployment models (see Section 4.1). In the first subsection, the general communication infrastructure is outlined, the messages used for channel binding verifications are specified, and the protocol flows are defined. The second subsection explores the difficulties of checking the different pieces of information that are exchanged during the channel binding protocol for consistency. The third subsection describes the information carried in the EAP exchange.

5.1. Protocol Operation

Channel bindings are always provided between two communication endpoints, here the EAP peer and the EAP server, who communicate through an authenticator typically in pass-through mode. For the channel binding protocol presented in this draft to work, the EAP

server needs to be able to access information from the AAA server that is utilized during the EAP session and a local database. For example, the EAP server and the local database can be co-located with the AAA server, as illustrated in Figure 1. An alternate architecture would be to provide a mechanism for the EAP server to inform the AAA server what channel binding attributes were supplied and the AAA server to inform the EAP server about what channel binding attributes it considered when making its decision.

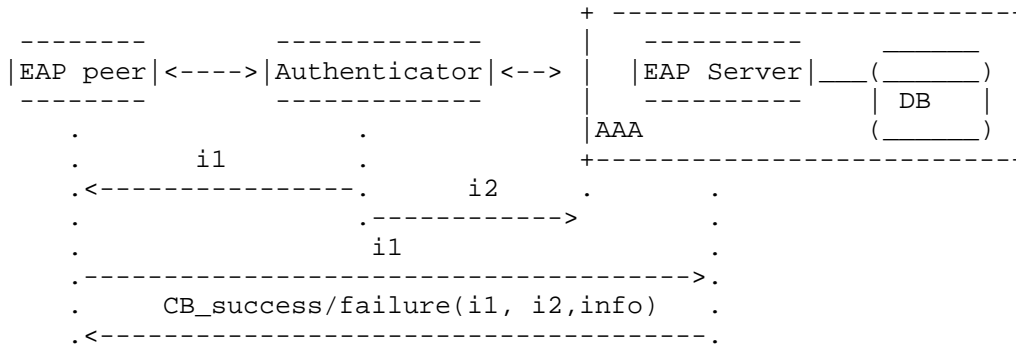


Figure 1: Overview of Channel Binding Protocol

During network advertisement, selection, and authentication, the authenticator presents unauthenticated information, labeled i1, about the network to the peer. Message i1 could include an authenticator identifier and the identity of the network it represents, in addition to advertised network information such as offered services and roaming information. Information may be communicated implicitly in i1, such as the type of media in use. As there is no established trust relationship between the peer and authenticator, there is no way for the peer to validate this information.

Additionally, during the transaction the authenticator presents a number of information properties in form of AAA attributes about itself and the current request to the AAA infrastructure which may or may not be valid. This information is labeled i2. Message i2 is the information the AAA server receives from the last hop in the AAA proxy chain which is not necessarily the authenticator.

AAA hops between the authenticator and AAA server can validate some of i2. Whether the AAA server will be able to depend on this depends significantly on the business relationship executed with these proxies and on the structure of the AAA network.

The local database is perhaps the most important part of this system.

In order for the EAP server or AAA server to know whether i1 and i2 are correct, they need access to trustworthy information, since an authenticator could include false information in both i1 and i2. Additional reasons why such a database is necessary for channel bindings to work are discussed in the next subsection. The information contained within the database could involve wildcards. For example, this could be used to check whether WiFi access points on a particular IP subnet all use a specific SSID. The exact IP address is immaterial, provided it is on the correct subnet.

During an EAP method execution with channel bindings, the peer sends i1 to the EAP server using the mechanism described in Section 5.3. The EAP server verifies the consistency of i1 provided by the peer, i2 provided by the authenticator, and the information in the local database. Upon the check, the EAP server sends a message to the peer indicating whether the channel binding validation check succeeded or failed and includes the attributes that were used in the check. The message flow is illustrated in Figure 1.

Above, the EAP server is described as performing the channel binding validation. In most deployments, this will be a necessary implementation constraint. The EAP exchange needs to include an indication of channel binding success or failure. Most existing implementations do not have a way to have an exchange between the EAP server and another AAA entity during the EAP server's processing of a single EAP message. However another AAA entity can provide information to the EAP server to make its decision.

If the compliance of i1 or i2 information with the authoritative policy source is mandatory and a consistency check failed, then after sending a protected indication of failed consistency, the EAP server MUST send an EAP-Failure message to terminate the session. If the EAP server is otherwise configured, it MUST allow the EAP session to complete normally, and leave the decision about network access up to the peer's policy. If i1 or i2 does not comply with policy, the EAP server MUST not list information that failed to comply in the set of information used to perform channel binding. In this case the EAP server SHOULD indicate channel binding failure; this requirement may be upgraded to a MUST in the future.

5.2. Channel Binding Consistency Check

The validation check that is the core of the channel binding protocol described in the previous subsection, consists of two parts in which the server checks whether:

1. the authenticator is lying to the peer, i.e. i1 contains false information,
2. the authenticator or any entity on the AAA path to the AAA server provides false information in form of AAA attributes, i.e. i2 contains false information,

These checks enable the EAP server to detect lying NAS/authenticator in enterprise networks and lying providers in service provider networks.

Checking the consistency of i1 and i2 is nontrivial, as has been pointed out already in [HC07]. First, i1 can contain any type of information propagated by the authenticator, whereas i2 is restricted to information that can be carried in AAA attributes. Second, because the authenticator typically communicates over different link layers with the peer and the AAA infrastructure, different type of identifiers and addresses may have been presented to both communication endpoints. Whether these different identifiers and addresses belong to the same device cannot be directly checked by the EAP server or AAA server without additional information. Finally, i2 may be different from the original information sent by the authenticator because of en route processing or malicious modifications. As a result, in the service provider model, typically the i1 information available to the EAP server can only be verified against the last-hop portion of i2, or values propagated by proxy servers. In addition, checking the consistency of i1 and i2 alone is insufficient because an authenticator could lie to both, the peer and the EAP server, i.e. i1 and i2 may be consistent but both contain false information.

A local database is required to leverage the above mentioned shortcomings and support the consistency and validation checks. In particular, information stored for each NAS/authenticator (enterprise scenario) or each roaming partner (service provider scenario) enables a comparison of any information received in i1 with AAA attributes in i2 as well as additionally stored AAA attributes that might have gone lost in transition. Furthermore, only such a database enables the EAP server and AAA server to check the received information against trusted information about the network including roaming agreements.

Section 7 describes lower-layer specific properties that can be exchanged as a part of i1. Section 8 describes specific AAA attributes that can be included and evaluated in i2. The EAP server reports back the results from the channel binding validation check that compares the consistency of all the values with those in the local database. The challenges of setting up such a local database are discussed in Section 10.

5.3. EAP Protocol

EAP methods supporting channel binding consistent with this specification provide a mechanism for carrying channel binding data from the peer to the EAP server and a channel binding response from the EAP server to the peer. The specifics of this mechanism are dependent on the method, although the content of the channel binding data and channel binding response are defined by this section.

Typically the lower layer will communicate a set of attributes to the EAP implementation on the peer that should be part of channel binding. The EAP implementation may need to indicate to the lower layer that channel binding information cannot be sent. Reasons for failing to send channel binding information include an EAP method that does not support channel binding is selected, or channel binding data is too big for the EAP method selected. Peers SHOULD provide appropriate policy controls to select channel binding or mandate its success.

The EAP server receives the channel binding data and performs the validation. The EAP method provides a way to return a response; the channel binding response uses the same basic format as the channel binding data.

XXX Nice figure goes here. Both the channel binding data and response use the following format. The protocol starts with a one byte code; see Section 5.3.1. Then for each type of attributes contained in the channel binding data, the following information is encoded:

NSID: One octet describing the namespace from which the attributes are drawn. See Section 5.3.3 for a description of how to encode RADIUS attributes in channel binding data and responses. RADIUS uses a namespace identifier of 0.

len: two octets of length in network byte order, not including the length or the namespace ID.

NSSPECIFIC: The encoding of the attributes in a manner specific to the type of attribute.

A given NSID MUST NOT appear more than once in a channel binding data or channel binding response.

In channel binding data, the code is set to 0 (channel binding data) and the full attributes and values are included. In a channel binding response, the server selects the code; see Section 5.3.1. The server includes any attributes that were considered in making the channel binding decision. The server SHOULD only include the attributes but not their values. In some cases, a server may not have information sufficient to know where the attribute stops and the

value starts; as an example, consider attribute containers such as the RADIUS vendor-specific attribute. For this reason and for future extensibility, clients MUST ignore any values sent in the channel binding response.

5.3.1. Channel Binding Codes

| Code | Meaning |
|------|----------------------------------|
| 0 | Channel Binding data from client |
| 1 | Channel binding success |
| 2 | Channel binding failure |

5.3.2. Namespace Identifiers

| ID | Namespace | Reference |
|-----|-------------|---------------|
| 0 | RADIUS | Section 5.3.3 |
| 255 | Private Use | |

5.3.3. Radius Namespace

RADIUS attribute-value pairs (AVPs) are encoded with a one-octet attribute type followed by a one-octet length followed by the value of the RADIUS attribute being encoded. The length includes the type and length octets; the minimum legal length is 2 for an attribute with no value. Attributes are concatenated to form the namespace specific portion of the packet.

XXX nice figure goes here.

The full value of an attribute is included in the channel binding data. For most attributes, none of the value is included in a channel binding response. However for attributes such as the vendor-specific attribute that include sub-attributes, enough information SHOULD be included to identify the sub-attribute. This means that the internal lengths SHOULD be 0 but the attribute information is included. Some RADIUS container specifications forbid sending attributes with no value. This prohibition notwithstanding, these attributes SHOULD be sent with no value in channel binding responses. XXX example figure of a RFC-style VSA in a channel binding response.

6. System Requirements

This section defines requirements on components used to implement the channel bindings protocol.

The channel binding protocol defined in this document **MUST** be transported after keying material has been derived between the EAP peer and server, and before the peer would suffer adverse affects from joining an adversarial network. This document describes a protocol for performing channel binding within EAP methods. As discussed in Section 4.2, an alternative approach for meeting this requirement is to perform channel bindings during the secure association protocol of the lower layer.

6.1. General Transport Protocol Requirements

The transport protocol for carrying channel binding information **MUST** support end-to-end (i.e. between the EAP peer and server) message integrity protection to prevent the adversarial NAS or AAA device from manipulating the transported data. The transport protocol **SHOULD** provide confidentiality. The motivation for this that the channel bindings could contain private information, including peer identities, which **SHOULD** be protected. If confidentiality cannot be provided, private information **MUST NOT** be sent as part of the channel binding information.

Any transport needs to be careful not to exceed the MTU for its lower-layer medium. In particular, if channel binding information is exchanged within protected EAP method channels, these methods may or may not support fragmentation. In order to work with all methods, the channel binding messages must fit within the available payload. For example, if the EAP MTU is 1020 octets, and EAP-GPSK is used as the authentication method, and maximal-length identities are used, a maximum of 384 octets are available for conveying channel binding information. Other methods, such as EAP-TTLS, support fragmentation and could carry significantly longer payloads.

6.2. EAP Method Requirements

If transporting data directly within an EAP method, it **MUST** be able to carry integrity protected data from the EAP peer to server. EAP methods **SHOULD** provide a mechanism to carry protected data from server to peer. EAP methods **MUST** exchange channel binding data with the AAA subsystem hosting the EAP server. EAP methods **MUST** be able to import channel binding data from the lower layer on the EAP peer.

7. Channel Binding TLV

This section defines some channel binding TLVs. While message `il` is not limited to AAA attributes, for the sake of tangible attributes that are already in place, this section discusses AAA AVPs that are appropriate for carrying channel bindings (i.e. data from `il` in Section 5). In particular, attributes for IEEE 802.11 are provided, which can be used as a template for developing bindings for other EAP lower-layer protocols.

For any lower-layer protocol, network information of interest to the peer and server can be encapsulated in AVPs or other defined payload containers. The appropriate AVPs depend on the lower layer protocol as well as on the network type (i.e. enterprise network or service provider network) and its application. Additional TLV types can be defined beyond AAA AVPs. For example it may be useful to define TLVs that can carry 802.11 information elements.

7.1. Requirements for Lower-Layer Bindings

Lower-layer protocols MUST support EAP in order to support EAP channel bindings. These lower layers MUST support EAP methods that derive keying material, as otherwise no integrity-protected channel would be available to execute the channel bindings protocol. Lower-layer protocols need not support traffic encryption, since this is independent of the authentication phase.

The data conveyed within the AVP type MUST NOT conflict with the externally-defined usage of the AVP. Additional TLV types SHOULD be defined for values that are not communicated within AAA attributes.

7.2. General Attributes

This section lists AAA AVPs useful to all link-layers. The peer SHOULD transmit to the server the following fields, encapsulated within the appropriate AVPs:

NAS-Port-Type: Indicates the underlying link-layer technology used to connect (e.g. IEEE 802.11, PPP, etc), and SHOULD be included by the EAP peer, and SHOULD be verified against the database and NAS-Port-Type received from the NAS.

7.3. IEEE 802.11

The peer SHOULD transmit to the server the following fields, encapsulated within the appropriate AVPs:

Called-Station-Id: contains BSSID and SSID and SHOULD be verified against the database and Called-Station-Id received from the NAS

7.3.1. IEEE 802.11r

In addition to the AVPs for IEEE 802.11, an IEEE 802.11r client SHOULD transmit the following additional field:

Mobility-Domain-Id: contains the identity of the mobility domain and SHOULD be verified against the database and Mobility-Domain-Id received from the NAS [I-D.aboba-radext-wlan]

8. AAA-Layer Bindings

This section discusses which AAA attributes in a AAA Accept-Request messages can and should be validated by a AAA server (i.e. data from i2 in Section 5). As noted before, this data can be manipulated by AAA proxies either to enable functionality (e.g. removing realm information after messages have been proxied) or maliciously (e.g. in the case of a lying provider). As such, this data cannot always be easily validated. However as thorough of a validation as possible should be conducted in an effort to detect possible attacks.

User-Name: This value should be checked for consistency with the database and any method-specific user information. If EAP method identity protection is employed, this value typically contains a pseudonym or keyword.

NAS-IP-Address: This value is typically the IP address of the authenticator, but in a proxied connection it likely will not match the source IP address of an Access-Request. A consistency check MAY verify the subnet of the IP address was correct based on the last-hop proxy.

NAS-IPv6-Address: This value is typically the IPv6 address of the authenticator, but in a proxied connection it likely will not match the source IPv6 address of an Access-Request. A consistency check MAY verify the subnet of the IPv6 address was correct based on the last-hop proxy.

Called-Station-Id: This is typically the MAC address of the NAS. On an enterprise network, it MAY be validated against the MAC address is one that has been provisioned on the network.

Calling-Station-Id: This is typically the MAC address of the EAP peer, and verification of this is likely difficult, unless EAP credentials have been provisioned on a per-host basis to specific L2 addresses. It SHOULD be validated against the database in an enterprise deployment.

NAS-Identifier: This is an identifier populated by the NAS, and could be related to the MAC address, and should be validated similarly to the Called-Station-Id.

NAS-Port-Type: This specifies the underlying link technology. It SHOULD be validated against the value received from the peer in the information exchange, and against a database of authorized link-layer technologies.

9. Security Considerations

This section discusses security considerations surrounding the use of EAP channel bindings.

9.1. Trust Model

In the considered trust model, EAP peer and authentication server are honest while the authenticator is maliciously sending false information to peer and/or server. In the model, the peer and server trust each other, which is not an unreasonable assumption, considering they already have a trust relationship. The following are the trust relationships:

- o The server trusts that the channel binding information received from the peer is the information that the peer received from the authenticator.
- o The peer trusts the channel binding result received from the server.
- o The server trusts the information contained within its local database.

In order to establish the first two trust relationships during an EAP execution, an EAP method needs to provide the following:

- o mutual authentication between peer and server
- o derivation of keying material including a key for integrity protection of channel binding messages
- o sending i1 from peer to server over an integrity-protected channel
- o sending the result and optionally i2 from server to peer over an integrity-protected channel

9.2. Consequences of Trust Violation

If any of the trust relationships listed in Section 9.1 are violated, channel binding cannot be provided. In other words, if mutual authentication with key establishment as part of the EAP method as well as protected database access are not provided, then achieving channel binding is not feasible.

Dishonest peers can only manipulate the first message *i1* of the channel binding protocol. In this scenario, a peer sends *i1'* to the server. If *i1'* is invalid, the channel binding validation will fail. On the other hand if *i1'* passes the validation, either the original *i1* was wrong and *i1'* corrected the problem or both *i1* and *i1'* constitute valid information. A peer could potentially gain an advantage in auditing or charging if both are valid and information from *i1* is used for auditing or charging. Such peers can be detected by including the information in *i2* and checking *i1* against *i2*.

Dishonest servers can send EAP-Failure messages and abort the EAP authentication even if the received *i1* is valid. However, servers can always abort any EAP session independent of whether channel binding is offered or not. On the other hand, dishonest servers can claim a successful validation even if *i1* contains invalid information. This can be seen as collaboration of authenticator and server. Channel binding can neither prevent nor detect such attacks. In general such attacks cannot be prevented by cryptographic means and should be addressed using policies making servers liable for their provided information and services.

Additional network entities (such as proxies) might be on the communication path between peer and server and may attempt to manipulate the channel binding protocol. If these entities do not possess the keying material used for integrity protection of the channel binding messages, the same threat analysis applies as for the dishonest authenticators. Hence, such entities can neither manipulate single channel binding messages nor the outcome. On the other hand, entities with access to the keying material must be treated like a server in a threat analysis. Hence such entities are able to manipulate the channel binding protocol without being detected. However, the required knowledge of keying material is unlikely since channel binding is executed before the EAP method is completed, and thus before keying material is typically transported to other entities.

9.3. Privacy Violations

While the channel binding information exchanged between EAP peer and EAP server (i.e. *i1* and the optional result message) must always be

integrity-protected it may not be encrypted. In the case that these messages contain identifiers of peer and/or network entities, the privacy property of the executed EAP method may be violated. Hence, in order to maintain the privacy of an EAP method, the exchanged channel binding information must be encrypted. If encryption is not available, private information is not sent as part of the channel binding information, as described in Section 6.1.

10. Operations and Management Considerations

As with any extension to existing protocols, there will be an impact on existing systems. Typically the goal is to develop an extension that minimizes the impact on both development and deployment of the new system, subject to the system requirements. This section discusses the impact on existing devices that currently utilize EAP, assuming the channel binding information is transported within the EAP method execution.

The EAP peer will need an API between the EAP lower layer and the EAP method that exposes the necessary information from the NAS to be validated to the EAP peer, which can then feed that information into the EAP methods for transport. For example, an IEEE 802.11 system would need to make available the various information elements that require validation to the EAP peer which would properly format them and pass them to the EAP method. Additionally, the EAP peer will require updated EAP methods that support transporting channel binding information. While most method documents are written modularly to allow incorporating arbitrary protected information, implementations of those methods would need to be revised to support these extensions. Driver updates are also required so methods can access the required information.

No changes to the pass-through authenticator would be required.

The EAP server would need an API between the database storing NAS information and the individual EAP server. The database may already exist on the AAA server in which case the EAP server passes the parameters to the AAA server for validation. The EAP methods need to be able to export received channel binding information to the EAP server so it can be validated.

11. IANA Considerations

A new top level registry is created for "EAP Channel Binding Parameters." This registry consists of several sub registries.

The "Channel Binding Codes" registry defines values for the code field in the channel binding data and channel binding response packet. See the table in Section 5.3.1 for initial registrations. This registry requires standards action [RFC5226] for new registrations. Early allocation is allowed. An additional reference column should be added to the table for the registry, pointing all codes in the initial registration to this specification.

The "Channel Binding Namespaces" sub-registry contains registrations for the NSID field in the channel binding data and channel binding response. Initial registrations are found in the table in Section 5.3.2. Registrations in this registry require IETF review.

12. Acknowledgements

The authors and editor would like to thank Bernard Aboba, Glen Zorn, Joe Salowey, and Klaas Wierenga for their valuable inputs that helped to improve and shape this document over the time.

Sam hartman's work on this specification is funded by JANET(UK).

13. References

13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowitz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

13.2. Informative References

- [I-D.aboba-radext-wlan]
Aboba, B., Malinen, J., Congdon, P., and J. Salowey,
"RADIUS Attributes for IEEE 802 Networks",
draft-aboba-radext-wlan-13 (work in progress),
February 2010.
- [I-D.clancy-emu-aaapay]
Clancy, T., Lior, A., and G. Zorn, Ed., "EAP Method

Support for Transporting AAA Payloads", Internet Draft draft-clancy-emu-aaapay-02, May 2009.

[RFC4006] Hakala, H., Mattila, L., Koskinen, J-P., Stura, M., and J. Loughney, "Diameter Credit-Control Application", RFC 4006, August 2005.

[RFC4017] Stanley, D., Walker, J., and B. Aboba, "Extensible Authentication Protocol (EAP) Method Requirements for Wireless LANs", RFC 4017, March 2005.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.

[HC07] Hoeper, K. and L. Chen, "Where EAP Security Claims Fail", ICST QShine, August 2007.

[80211U-D4.01]
"Information technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications - Amendment 7: Interworking with External Networks", IEEE Draft Standard 802.11u, November 2008.

[I-D.ietf-abfab-gss-eap]
Hartman, S. and J. Howlett, "A GSS-API Mechanism for the Extensible Authentication Protocol", draft-ietf-abfab-gss-eap-01 (work in progress), February 2011.

Appendix A. Attacks Prevented by Channel Bindings

In the following it is demonstrated how the presented channel bindings can prevent attacks by malicious authenticators (representing the lying NAS problem) as well as malicious visited networks (representing the lying provider problem).

A.1. Enterprise Subnetwork Masquerading

As outlined in Section 3, an enterprise network may have multiple VLANs providing different levels of security. In an attack, a malicious NAS connecting to a guest network with lesser security protection could broadcast the SSID of a subnetwork with higher protection. This could lead peers to believe that they are accessing the network over secure connections, and, e.g., transmit confidential

information that they normally would not send over a weakly protected connection. This attack works under the conditions that peers use the same set of credentials to authenticate to the different kinds of VLANs and that the VLANs support at least one common EAP method. If these conditions are not met, the EAP server would not authorize the peers to connect to the guest network, because the peers used credentials and/or an EAP method that is associated with the corporate network.

A.2. Forced Roaming

Mobile phone providers boosting their cell tower's transmission power to get more users to use their networks have occurred in the past. The increased transmission range combined with a NAS sending a false network identity lures users to connect to the network without being aware of that they are roaming.

Channel bindings would detect the bogus network identifier because the network identifier send to the authentication server in i1 will neither match information i2 nor the stored data. The verification fails because the info in i1 claims to come from the peer's home network while the home authentication server knows that the connection is through a visited network outside the home domain. In the same context, channel bindings can be utilized to provide a "home zone" feature that notifies users every time they are about to connect to a NAS outside their home domain.

A.3. Downgrading attacks

A malicious authenticator could modify the set of offered EAP methods in its Beacon to force the peer to choose from only the weakest EAP method(s) accepted by the authentication server. For instance, instead of having a choice between EAP-MD5-CHAP, EAP-FAST and some other methods, the authenticator reduces the choice for the peer to the weaker EAP-MD5-CHAP method. Assuming that weak EAP methods are supported by the authentication server, such a downgrading attack can enable the authenticator to attack the integrity and confidentiality of the remaining EAP execution and/or break the authentication and key exchange. The presented channel bindings prevent such downgrading attacks, because peers submit the offered EAP method selection that they have received in the beacon as part of i1 to the authentication server. As a result, the authentication server recognizes the modification when comparing the information to the respective information in its policy database.

A.4. Bogus Beacons in IEEE 802.11r

In IEEE 802.11r, the SSID is bound to the TSK calculations, so that the TSK needs to be consistent with the SSID advertised in an authenticator's Beacon. While this prevents outsiders from spoofing a Beacon it does not stop a "lying NAS" from sending a bogus Beacon and calculating the TSK accordingly.

By implementing channel bindings, as described in this draft, in IEEE 802.11r, the verification by the authentication server would detect the inconsistencies between the information the authenticator has sent to the peer and the information the server received from the authenticator and stores in the policy database.

A.5. Forcing false authorization in IEEE 802.11i

In IEEE 802.11i a malicious NAS can modify the beacon to make the peer believe it is connected to a network different from the one the peer is actually connected to.

In addition, a malicious NAS can force an authentication server into authorizing access by sending an incorrect Called-Station-ID that belongs to an authorized NAS in the network. This could cause the authentication server to believe it had granted access to a different network or even provider than the one the peer got access to.

Both attacks can be prevented by implementing channel bindings, because the server can compare the information that was sent to the peer, with information it received from the authenticator during the AAA communication as well as the information stored in the policy database.

Appendix B. Change History

RFC editor, remove this section prior to publication.

B.1. Changes since Version 06

The purpose of this revision is to provide a specific candidate protocol for channel binding data and channel binding responses.

B.2. Changes since version 04

- o Clarify examples in introduction.
- o In problem statement note that one EAP server may deal with both enterprise and provider networks.

- o Update discussion of the architecture. Talk about channel bindings as a mechanism to introduce levels of trust.
- o Indicate that this document is focusing on EAP channel bindings within methods while trying to do a better job of describing the SAP approach in more detail.
- o Claim that we're using the encoding from draft-clancy-emu-aaapay. The WG almost certainly doesn't have consensus on this, but in the interest of actually describing what the protocol might be like, it is a good straw-man proposal.
- o Update protocol description.

Authors' Addresses

Sam Hartman (editor)
Painless Security
356 Abbott ST
North Andover, MA 01845
USA

Email: hartmans-ietf@mit.edu

T. Charles Clancy
Electrical and Computer Engineering
Virginia Tech
Arlington, VA 22203
USA

Email: tcc@vt.edu

Katrin Hoeper
Motorola, Inc.
1301 E. Algonquin Road
Schaumburg, IL 60196
USA

Email: khoeper@motorola.com

EMU Working Group
Internet-Draft
Intended status: Standards Track
Expires: November 14, 2011

H. Zhou
N. Cam-Winget
J. Salowey
Cisco Systems
S. Hanna
Juniper Networks
May 13, 2011

Flexible Authentication via Secure Tunneling Extensible Authentication
Protocol (EAP-FAST) Version 2
draft-ietf-emu-eap-tunnel-method-00.txt

Abstract

This document defines the Extensible Authentication Protocol (EAP) based Flexible Authentication via Secure Tunneling (EAP-FAST) protocol version 2. EAP-FAST is an EAP method that enables secure communication between a peer and a server by using the Transport Layer Security (TLS) to establish a mutually authenticated tunnel. Within the tunnel, Type-Length-Value (TLV) objects are used to convey authentication related data between the EAP peer and the EAP server.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on November 14, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

| | |
|---|----|
| 1. Introduction | 5 |
| 1.1. Specification Requirements | 5 |
| 1.2. Major Differences from Version 1 | 5 |
| 1.3. Design Goals | 5 |
| 1.4. Terminology | 7 |
| 2. Protocol Overview | 7 |
| 2.1. Architectural Model | 8 |
| 2.2. Protocol Layering Model | 9 |
| 3. EAP-FAST Protocol | 9 |
| 3.1. Version Negotiation | 10 |
| 3.2. EAP-FAST Authentication Phase 1: Tunnel Establishment | 11 |
| 3.2.1. TLS Session Resume Using Server State | 12 |
| 3.2.2. TLS Session Resume Using a PAC | 12 |
| 3.2.3. Transition between Abbreviated and Full TLS Handshake | 14 |
| 3.3. EAP-FAST Authentication Phase 2: Tunneled Authentication | 14 |
| 3.3.1. EAP Sequences | 15 |
| 3.3.2. Optional Password Authentication | 15 |
| 3.3.3. Protected Termination and Acknowledged Result Indication | 16 |
| 3.4. Determining Peer-Id and Server-Id | 17 |
| 3.5. EAP-FAST Session Identifier | 17 |
| 3.6. Error Handling | 17 |
| 3.6.1. TLS Layer Errors | 18 |
| 3.6.2. Phase 2 Errors | 18 |
| 3.7. Fragmentation | 19 |
| 4. Message Formats | 20 |
| 4.1. EAP-FAST Message Format | 20 |
| 4.1.1. Authority ID Data | 22 |
| 4.2. EAP-FAST TLV Format and Support | 23 |
| 4.2.1. General TLV Format | 24 |
| 4.2.2. Result TLV | 25 |
| 4.2.3. NAK TLV | 26 |
| 4.2.4. Error TLV | 28 |
| 4.2.5. Vendor-Specific TLV | 29 |
| 4.2.6. EAP-Payload TLV | 30 |
| 4.2.7. Intermediate-Result TLV | 32 |
| 4.2.8. Crypto-Binding TLV | 33 |

| | |
|---|----|
| 4.2.9. Request-Action TLV | 35 |
| 4.2.10. Channel-Binding TLV | 36 |
| 4.2.11. Identity-Type TLV | 37 |
| 4.2.12. Basic-Password-Auth-Req TLV | 38 |
| 4.2.13. Basic-Password-Auth-Resp TLV | 39 |
| 4.3. Table of TLVs | 41 |
| 5. Cryptographic Calculations | 42 |
| 5.1. EAP-FAST Authentication Phase 1: Key Derivations | 42 |
| 5.2. Intermediate Compound Key Derivations | 42 |
| 5.3. Computing the Compound MAC | 43 |
| 5.4. EAP Master Session Key Generation | 43 |
| 6. IANA Considerations | 44 |
| 7. Security Considerations | 46 |
| 7.1. Mutual Authentication and Integrity Protection | 46 |
| 7.2. Method Negotiation | 46 |
| 7.3. Separation of Phase 1 and Phase 2 Servers | 47 |
| 7.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies | 47 |
| 7.4.1. User Identity Protection and Verification | 48 |
| 7.4.2. Dictionary Attack Resistance | 49 |
| 7.4.3. Protection against Man-in-the-Middle Attacks | 49 |
| 7.4.4. PAC Binding to User Identity | 50 |
| 7.5. Protecting against Forged Clear Text EAP Packets | 50 |
| 7.6. Server Certificate Validation | 51 |
| 7.7. Tunnel PAC Considerations | 51 |
| 7.8. Security Claims | 51 |
| 8. Acknowledgements | 53 |
| 9. References | 53 |
| 9.1. Normative References | 53 |
| 9.2. Informative References | 54 |
| Appendix A. Evaluation Against Tunnel Based EAP Method Requirements | 56 |
| A.1. Requirement 4.1.1 RFC Compliance | 56 |
| A.2. Requirement 4.2.1 TLS Requirements | 56 |
| A.3. Requirement 4.2.1.1.1 Cipher Suite Negotiation | 56 |
| A.4. Requirement 4.2.1.1.2 Tunnel Data Protection Algorithms | 57 |
| A.5. Requirement 4.2.1.1.3 Tunnel Authentication and Key Establishment | 57 |
| A.6. Requirement 4.2.1.2 Tunnel Replay Protection | 57 |
| A.7. Requirement 4.2.1.3 TLS Extensions | 57 |
| A.8. Requirement 4.2.1.4 Peer Identity Privacy | 57 |
| A.9. Requirement 4.2.1.5 Session Resumption | 57 |
| A.10. Requirement 4.2.2 Fragmentation | 58 |
| A.11. Requirement 4.2.3 Protection of Data External to Tunnel | 58 |
| A.12. Requirement 4.3.1 Extensible Attribute Types | 58 |
| A.13. Requirement 4.3.2 Request/Challenge Response Operation | 58 |
| A.14. Requirement 4.3.3 Indicating Criticality of Attributes | 58 |
| A.15. Requirement 4.3.4 Vendor Specific Support | 58 |

| | |
|--|----|
| A.16. Requirement 4.3.5 Result Indication | 58 |
| A.17. Requirement 4.3.6 Internationalization of Display Strings | 58 |
| A.18. Requirement 4.4 EAP Channel Binding Requirements | 59 |
| A.19. Requirement 4.5.1.1 Confidentiality and Integrity | 59 |
| A.20. Requirement 4.5.1.2 Authentication of Server | 59 |
| A.21. Requirement 4.5.1.3 Server Certificate Revocation Checking | 59 |
| A.22. Requirement 4.5.2 Internationalization | 59 |
| A.23. Requirement 4.5.3 Meta-data | 59 |
| A.24. Requirement 4.5.4 Password Change | 59 |
| A.25. Requirement 4.6.1 Method Negotiation | 59 |
| A.26. Requirement 4.6.2 Chained Methods | 60 |
| A.27. Requirement 4.6.3 Cryptographic Binding with the TLS Tunnel | 60 |
| A.28. Requirement 4.6.4 Peer Initiated | 60 |
| A.29. Requirement 4.6.5 Method Meta-data | 60 |
| Appendix B. Examples | 60 |
| B.1. Successful Authentication | 60 |
| B.2. Failed Authentication | 62 |
| B.3. Full TLS Handshake using Certificate-based Cipher Suite | 63 |
| B.4. Client authentication during Phase 1 with identity privacy | 65 |
| B.5. Fragmentation and Reassembly | 66 |
| B.6. Sequence of EAP Methods | 68 |
| B.7. Failed Crypto-binding | 70 |
| B.8. Sequence of EAP Method with Vendor-Specific TLV Exchange | 72 |

1. Introduction

Since the introduction of EAP-FAST version 1 [RFC4851] a few years ago, it has been widely adopted in variety of devices and platforms due to its strong security, flexibility and ease of deployment. This document describes EAP-FAST Version 2, with some minor changes from Version 1, to meet the requirements outlined in [I-D.ietf-emu-eaptunnel-req] for a standard tunnel based EAP method.

1.1. Specification Requirements

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] .

1.2. Major Differences from Version 1

This document is a revision of the EAP-FAST version 1 [RFC4851] which contains improved flexibility, particularly for negotiation of cryptographic algorithms. The major changes are:

1. Version number has been changed from 1 to 2.
2. This version of EAP-FAST MUST support TLS 1.2 [RFC5246].
3. The key derivation now makes use of TLS keying material exporters [RFC5705] and the PRF and hash function negotiated in TLS. This is to simplify implementation and better support cryptographic algorithm agility.
4. EAP-FASTv2 is in full conformance with TLS Ticket extension [RFC5077] as described in Section 3.2.2.
5. Basic password authentication on the TLV level has been added in addition to the existing inner EAP method.
6. Additional TLV types have been defined to support EAP channel binding and meta-data. They are Identity Type TLV and Channel-Binding TLVs, defined in Section 4.2.

1.3. Design Goals

Network access solutions requiring user friendly and easily deployable secure authentication mechanisms highlight the need for strong mutual authentication protocols that enable the use of weaker user credentials. This document defines an Extensible Authentication Protocol (EAP) which consists of establishing a Transport Layer Security (TLS) tunnel using TLS 1.2 [RFC5246] or a successor version

of TLS, using the latest version supported by both parties. Once the tunnel is established, the protocol further exchanges data in the form of type, length, value objects (TLV) to perform further authentication. EAP-FAST supports the TLS extension defined in [RFC5077] to support fast re-establishment of the secure tunnel without having to maintain per-session state on the server.

EAP-FAST's design motivations included:

- o Mutual authentication: an EAP server must be able to verify the identity and authenticity of the peer, and the peer must be able to verify the authenticity of the EAP server.
- o Immunity to passive dictionary attacks: many authentication protocols require a password to be explicitly provided (either as cleartext or hashed) by the peer to the EAP server; at minimum, the communication of the weak credential (e.g., password) must be immune from eavesdropping.
- o Immunity to man-in-the-middle (MitM) attacks: in establishing a mutually authenticated protected tunnel, the protocol must prevent adversaries from successfully interjecting information into the conversation between the peer and the EAP server.
- o Flexibility to enable support for most password authentication interfaces: as many different password interfaces (e.g., Microsoft Challenge Handshake Authentication Protocol (MS-CHAP), Lightweight Directory Access Protocol (LDAP), One-Time Password (OTP), etc.) exist to authenticate a peer, the protocol must provide this support seamlessly.
- o Efficiency: specifically when using wireless media, peers will be limited in computational and power resources. The protocol must enable the network access communication to be computationally lightweight.

With these motivational goals defined, further secondary design criteria are imposed:

- o Flexibility to extend the communications inside the tunnel: with the growing complexity in network infrastructures, the need to gain authentication, authorization, and accounting is also evolving. For instance, there may be instances in which multiple existing authentication protocols are required to achieve mutual

authentication. Similarly, different protected conversations may be required to achieve the proper authorization once a peer has successfully authenticated.

- o Minimize the authentication server's per user authentication state requirements: with large deployments, it is typical to have many servers acting as the authentication servers for many peers. It is also highly desirable for a peer to use the same shared secret to secure a tunnel much the same way it uses the username and password to gain access to the network. The protocol must facilitate the use of a single strong shared secret by the peer while enabling the servers to minimize the per user and device state it must cache and manage.

1.4. Terminology

Much of the terminology in this document comes from [RFC3748]. Additional terms are defined below:

Protected Access Credential (PAC)

Credentials distributed to a peer for future optimized network authentication. The PAC consists of, at most, three components: a shared secret, an opaque element, and optionally other information. The shared secret component contains the pre-shared key between the peer and the authentication server. The opaque part is provided to the peer and is presented to the authentication server when the peer wishes to obtain access to network resources. Finally, a PAC may optionally include other information that may be useful to the peer. The opaque part of the PAC is the same type of data as the ticket in [RFC5077] and the shared secret is used to derive the TLS master secret.

2. Protocol Overview

EAP-FAST is an authentication protocol similar to EAP-TLS [RFC5216] that enables mutual authentication and cryptographic context establishment by using the TLS [RFC5246] handshake protocol. EAP-FAST allows for the established TLS tunnel to be used for further authentication exchanges. EAP-FAST makes use of TLVs to carry out the inner authentication exchanges. The tunnel is then used to protect weaker inner authentication methods, which may be based on passwords, and to communicate the results of the authentication.

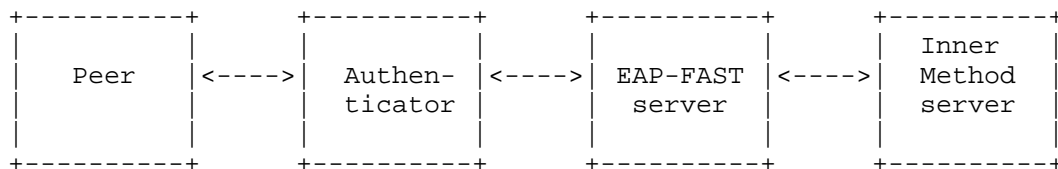
EAP-FAST makes use of the TLS enhancements in [RFC5077] to enable an optimized TLS tunnel session resume while minimizing server state. The secret key used in EAP-FAST is referred to as the Protected

Access Credential key (or PAC-Key); the PAC-Key is used to mutually authenticate the peer and the server when securing a tunnel. The ticket is referred to as the Protected Access Credential opaque data (or PAC-Opaque). The secret key and ticket used to establish the tunnel may be provisioned through mechanisms that do not involve the TLS handshake. It is RECOMMENDED that implementations support the capability to distribute the ticket and secret key within the EAP-FAST tunnel as specified in [RFC5422].

The EAP-FAST conversation is used to establish or resume an existing session to typically establish network connectivity between a peer and the network. Upon successful execution of EAP-FAST, both EAP peer and EAP server derive strong session key material that can then be communicated to the network access server (NAS) for use in establishing a link layer security association.

2.1. Architectural Model

The network architectural model for EAP-FAST usage is shown below:

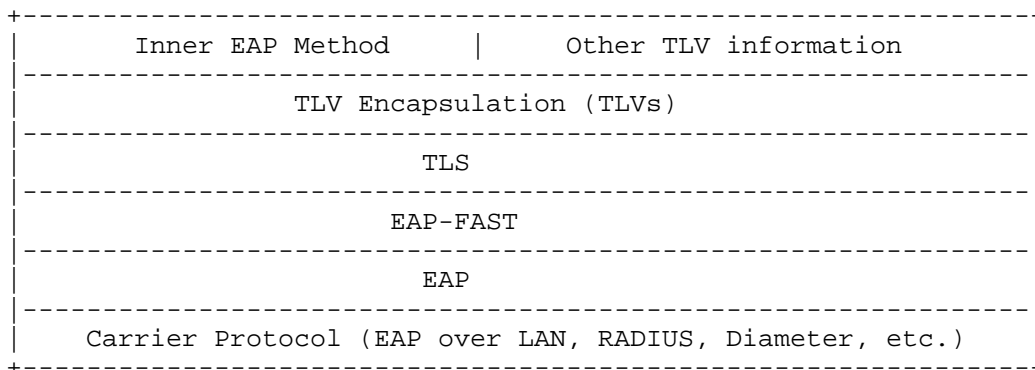


EAP-FAST Architectural Model

The entities depicted above are logical entities and may or may not correspond to separate network components. For example, the EAP-FAST server and inner method server might be a single entity; or the authenticator and EAP-FAST server might be a single entity; or the functions of the authenticator, EAP-FAST server, and inner method server might be combined into a single physical device. For example, typical 802.11 deployments place the Authenticator in an access point (AP) while a Radius server may provide the EAP-FAST and inner method server components. The above diagram illustrates the division of labor among entities in a general manner and shows how a distributed system might be constructed; however, actual systems might be realized more simply. The security considerations Section 7.3 provides an additional discussion of the implications of separating the EAP-FAST server from the inner method server.

2.2. Protocol Layering Model

EAP-FAST packets are encapsulated within EAP; EAP in turn requires a carrier protocol for transport. EAP-FAST packets encapsulate TLS, which is then used to encapsulate user authentication information. Thus, EAP-FAST messaging can be described using a layered model, where each layer encapsulates the layer above it. The following diagram clarifies the relationship between protocols:



Protocol Layering Model

The TLV layer is a payload with Type-Length-Value (TLV) Objects defined in Section 4.2. The TLV objects are used to carry arbitrary parameters between an EAP peer and an EAP server. All conversations in the EAP-FAST protected tunnel must be encapsulated in a TLV layer.

Methods for encapsulating EAP within carrier protocols are already defined. For example, IEEE 802.1X [IEEE.802-1X.2004] may be used to transport EAP between the peer and the authenticator; RADIUS [RFC3579] or Diameter [RFC4072] may be used to transport EAP between the authenticator and the EAP-FAST server.

3. EAP-FAST Protocol

EAP-FAST authentication occurs in two phases. In the first phase, EAP-FAST employs the TLS handshake to provide an authenticated key exchange and to establish a protected tunnel. Once the tunnel is established the second phase begins with the peer and server engaging in further conversations to establish the required authentication and authorization policies. The operation of the protocol, including Phase 1 and Phase 2, is the topic of this section. The format of EAP-FAST messages is given in Section 4 and the cryptographic

calculations are given in Section 5.

3.1. Version Negotiation

EAP-FAST packets contain a 3-bit version field, following the TLS Flags field, which enables EAP-FAST implementations to be backward compatible with previous versions of the protocol. This specification documents the EAP-FAST version 2 protocol; implementations of this specification MUST use a version field set to 2.

Version negotiation proceeds as follows:

In the first EAP-Request sent with EAP type=EAP-FAST, the EAP server must set the version field to the highest supported version number.

If the EAP peer supports this version of the protocol, it MUST respond with an EAP-Response of EAP type=EAP-FAST, and the version number proposed by the EAP-FAST server.

If the EAP-FAST peer does not support this version, it responds with an EAP-Response of EAP type=EAP-FAST and the highest supported version number that's less than the number proposed by the EAP-FAST server.

If the EAP-FAST server does not support the version number proposed by the EAP-FAST peer, it terminates the conversation. Otherwise the EAP-FAST conversation continues.

The version negotiation procedure guarantees that the EAP-FAST peer and server will agree to the latest version supported by both parties. If version negotiation fails, then use of EAP-FAST will not be possible, and another mutually acceptable EAP method will need to be negotiated if authentication is to proceed.

The EAP-FAST version is not protected by TLS; and hence can be modified in transit. In order to detect a modification of the EAP-FAST version, the peers MUST exchange the EAP-FAST version number received during version negotiation using the Crypto-Binding TLV described in Section 4.2.8. The receiver of the Crypto-Binding TLV MUST verify that the version received in the Crypto-Binding TLV matches the version sent by the receiver in the EAP-FAST version negotiation.

3.2. EAP-FAST Authentication Phase 1: Tunnel Establishment

EAP-FAST is based on the TLS handshake [RFC5246] to establish an authenticated and protected tunnel. The TLS version offered by the peer and server MUST be TLS version 1.2 [RFC5246] or later. This version of the EAP-FAST implementation MUST support the following TLS ciphersuites:

TLS_RSA_WITH_AES_128_CBC_SHA [RFC3268]

TLS_DHE_RSA_WITH_AES_128_CBC_SHA [RFC3268]

Other ciphersuites MAY be supported. It is RECOMMENDED that anonymous ciphersuites such as TLS_DH_anon_WITH_AES_128_CBC_SHA only be used in the context of the provisioning described in [RFC5422]. Care must be taken to address potential man-in-the-middle attacks when ciphersuites that do not provide authenticated tunnel establishment are used. During the EAP-FAST Phase 1 conversation the EAP-FAST endpoints MAY negotiate TLS compression. During TLS tunnel establishment, TLS extensions MAY be used. For instance, Certificate Status Request extension [RFC6066] MAY be used to leverage a certificate-status protocol such as OCSP [RFC2560] to check the validity of server certificates. TLS renegotiation indications defined in RFC 5746 [RFC5746] MUST be supported.

The EAP server initiates the EAP-FAST conversation with an EAP request containing an EAP-FAST/Start packet. This packet includes a set Start (S) bit, the EAP-FAST version as specified in Section 3.1, and an authority identity. The TLS payload in the initial packet is empty. The authority identity (A-ID) is used to provide the peer a hint of the server's identity that may be useful in helping the peer select the appropriate credential to use. Assuming that the peer supports EAP-FAST the conversation continues with the peer sending an EAP-Response packet with EAP type of EAP-FAST with the Start (S) bit clear and the version as specified in Section 3.1. This message encapsulates one or more TLS records containing the TLS handshake messages. If the EAP-FAST version negotiation is successful then the EAP-FAST conversation continues until the EAP server and EAP peer are ready to enter Phase 2. When the full TLS handshake is performed, then the first payload of EAP-FAST Phase 2 MAY be sent along with server-finished handshake message to reduce the number of round trips.

After the TLS session is established, another EAP exchange MAY occur within the tunnel to authenticate the EAP peer. EAP-FAST implementations MUST support client authentication during tunnel establishment using the TLS ciphersuites specified in Section 3.2. EAP-FAST implementations SHOULD also support the immediate

renegotiation of a TLS session to initiate a new handshake message exchange under the protection of the current ciphersuite. This allows support for protection of the peer's identity. Note that the EAP peer does not need to authenticate as part of the TLS exchange, but can alternatively be authenticated through additional EAP exchanges carried out in Phase 2.

The EAP-FAST tunnel protects peer identity information from disclosure outside the tunnel. Implementations that wish to provide identity privacy for the peer identity must carefully consider what information is disclosed outside the tunnel.

The following sections describe resuming a TLS session based on server-side or client-side state.

3.2.1. TLS Session Resume Using Server State

EAP-FAST session resumption is achieved in the same manner TLS achieves session resume. To support session resumption, the server and peer must minimally cache the Session ID, master secret, and ciphersuite. The peer attempts to resume a session by including a valid Session ID from a previous handshake in its ClientHello message. If the server finds a match for the Session ID and is willing to establish a new connection using the specified session state, the server will respond with the same Session ID and proceed with the EAP-FAST Authentication Phase 1 tunnel establishment based on a TLS abbreviated handshake. After a successful conclusion of the EAP-FAST Authentication Phase 1 conversation, the conversation then continues on to Phase 2.

3.2.2. TLS Session Resume Using a PAC

EAP-FAST supports the resumption of sessions based on client-side state using TLS SessionTicket extension techniques described in [RFC5077]. This version of EAP-FAST supports the provisioning of a ticket called a Protected Access Credential (PAC) through the use of the NewSessionTicket handshake described in [RFC5077], as well as provisioning of a PAC as described in [RFC5422]. Implementations may provide additional ways to provision the PAC, such as manual configuration. Since the PAC mentioned here is used for establishing the TLS Tunnel, it is more specifically referred to as the Tunnel PAC. The Tunnel PAC is a security credential provided by the EAP server to a peer and comprised of:

1. PAC-Key: this is a 32-octet key used by the peer to establish the EAP-FAST Phase 1 tunnel. This key is used to derive the TLS premaster secret as described in Section 5.1. The PAC-Key is randomly generated by the EAP server to produce a strong entropy

32-octet key. The PAC-Key is a secret and MUST be treated accordingly. For example, as the PAC-Key is a separate component provisioned by the server to establish a secure tunnel, the server may deliver this component protected by a secure channel, and it must be stored securely by the peer.

2. PAC-Opaque: this is a variable length field that is sent to the EAP server during the EAP-FAST Phase 1 tunnel establishment. The PAC-Opaque can only be interpreted by the EAP server to recover the required information for the server to validate the peer's identity and authentication. For example, the PAC-Opaque includes the PAC-Key and may contain the PAC's peer identity. The PAC-Opaque format and contents are specific to the PAC issuing server. The PAC-Opaque may be presented in the clear, so an attacker MUST NOT be able to gain useful information from the PAC-Opaque itself. The server issuing the PAC-Opaque must ensure it is protected with strong cryptographic keys and algorithms.
3. PAC-Info: this is a variable length field used to provide, at a minimum, the authority identity of the PAC issuer. Other useful but not mandatory information, such as the PAC-Key lifetime, may also be conveyed by the PAC issuing server to the peer during PAC provisioning or refreshment.

The use of the PAC is based on the SessionTicket extension defined in [RFC5077]. The EAP server initiates the EAP-FAST conversation as normal. Upon receiving the A-ID from the server, the peer checks to see if it has an existing valid PAC-Key and PAC-Opaque for the server. If it does, then it obtains the PAC-Opaque and puts it in the SessionTicket extension in the ClientHello. It is RECOMMENDED in EAP-FAST that the peer include an empty Session ID in a ClientHello containing a PAC-Opaque. This version of EAP-FAST supports the NewSessionTicket Handshake message as described in [RFC5077] for distribution of a new PAC, as well as the provisioning mechanism described in [RFC5422]. If the PAC-Opaque included in the SessionTicket extension is valid and the EAP server permits the abbreviated TLS handshake, it will select the ciphersuite allowed to be used from information within the PAC and finish with the abbreviated TLS handshake. If the server receives a Session ID and a PAC-Opaque in the SessionTicket extension in a ClientHello, it should place the same Session ID in the ServerHello if it is resuming a session based on the PAC-Opaque. The conversation then proceeds as described in [RFC5077] until the handshake completes or a fatal error occurs. After the abbreviated handshake completes, the peer and the server are ready to commence Phase 2. Note that when a PAC is used, the TLS master secret is calculated from the PAC-Key, client random,

and server random as described in Section 5.1.

Specific details for the Tunnel PAC format, provisioning and security considerations are best described in [RFC5422].

3.2.3. Transition between Abbreviated and Full TLS Handshake

If session resumption based on server-side or client-side state fails, the server can gracefully fall back to a full TLS handshake. If the ServerHello received by the peer contains an empty Session ID or a Session ID that is different than in the ClientHello, the server may be falling back to a full handshake. The peer can distinguish the server's intent of negotiating full or abbreviated TLS handshake by checking the next TLS handshake messages in the server response to the ClientHello. If ChangeCipherSpec follows the ServerHello in response to the ClientHello, then the server has accepted the session resumption and intends to negotiate the abbreviated handshake. Otherwise, the server intends to negotiate the full TLS handshake. A peer can request for a new PAC to be provisioned after the full TLS handshake and mutual authentication of the peer and the server. In order to facilitate the fallback to a full handshake the peer SHOULD include ciphersuites that allow for a full handshake and possibly PAC provisioning so the server can select one of these in case session resumption fails. An example of the transition is shown in Appendix B.

3.3. EAP-FAST Authentication Phase 2: Tunneled Authentication

The second portion of the EAP-FAST Authentication occurs immediately after successful completion of Phase 1. Phase 2 occurs even if both peer and authenticator are authenticated in the Phase 1 TLS negotiation. Phase 2 MUST NOT occur if the Phase 1 TLS handshake fails. Phase 2 consists of a series of requests and responses encapsulated in TLV objects defined in Section 4.2. Phase 2 MUST always end with a protected termination exchange described in Section 3.3.3. The TLV exchange may include the execution of zero or more EAP methods within the protected tunnel as described in Section 3.3.1. A server MAY proceed directly to the protected termination exchange if it does not wish to request further authentication from the peer. However, the peer and server must not assume that either will skip inner EAP methods or other TLV exchanges. The peer may have roamed to a network that requires conformance with a different authentication policy or the peer may request the server take additional action through the use of the Request-Action TLV.

3.3.1. EAP Sequences

EAP [RFC3748] prohibits use of multiple authentication methods within a single EAP conversation in order to limit vulnerabilities to man-in-the-middle attacks. EAP-FAST addresses man-in-the-middle attacks through support for cryptographic protection of the inner EAP exchange and cryptographic binding of the inner authentication method(s) to the protected tunnel. EAP methods are executed serially in a sequence. This version of EAP-FAST does not support initiating multiple EAP methods simultaneously in parallel. The methods need not be distinct. For example, EAP-TLS could be run twice as an inner method, first using machine credentials followed by a second instance using user credentials.

EAP method messages are carried within EAP-Payload TLVs defined in Section 4.2.6. If more than one method is going to be executed in the tunnel, then upon method completion of a method, the server MUST send an Intermediate-Result TLV indicating the result. The peer MUST respond to the Intermediate-Result TLV indicating its result. If the result indicates success, the Intermediate-Result TLV MUST be accompanied by a Crypto-Binding TLV. The Crypto-Binding TLV is further discussed in Section 4.2.8 and Section 5.3. The Intermediate-Result TLVs can be included with other TLVs such as EAP-Payload TLVs starting a new EAP conversation or with the Result TLV used in the protected termination exchange. In the case where only one EAP method is executed in the tunnel, the Intermediate-Result TLV MUST NOT be sent with the Result TLV. In this case, the status of the inner EAP method is represented by the final Result TLV, which also represents the result of the whole EAP-FAST conversation. This is to maintain backward compatibility with existing implementations.

If both peer and server indicate success, then the method is considered complete. If either indicates failure, then the method is considered failed. The result of failure of an EAP method does not always imply a failure of the overall authentication. If one authentication method fails, the server may attempt to authenticate the peer with a different method.

3.3.2. Optional Password Authentication

The use of EAP-GTC as defined in RFC 5421 [RFC5421] is not recommended with EAP-FASTv2. Implementations should instead make use of the password authentication TLVs defined in this specification. The authentication server initiates password authentication by sending a Basic-Password-Auth-Req TLV defined in Section 4.2.12. If the peer wishes to participate in password authentication then it responds with a Basic-Password-Auth-Resp TLV as defined in Section 4.2.13 that contains the username and password. If it does

not wish to perform password authentication then it responds with a NAK TLV indicating the rejection of the Basic-Password-Auth-Req TLV. Upon receiving the response the server indicates the success or failure of the exchange using an Intermediate-Result TLV. Multiple roundtrips of password authentication requests and responses MAY be used to support some "housecleaning" functions such as password change, change pin, etc. before a user is authenticated.

3.3.3. Protected Termination and Acknowledged Result Indication

A successful EAP-FAST Phase 2 conversation MUST always end in a successful Result TLV exchange. An EAP-FAST server may initiate the Result TLV exchange without initiating any EAP conversation in EAP-FAST Phase 2. After the final Result TLV exchange, the TLS tunnel is terminated and a clear text EAP-Success or EAP-Failure is sent by the server. The format of the Result TLV is described in Section 4.2.2.

A server initiates a successful protected termination exchange by sending a Result TLV indicating success. The server may send the Result TLV along with an Intermediate-Result TLV and a Crypto-Binding TLV. If the peer requires nothing more from the server it will respond with a Result TLV indicating success accompanied by an Intermediate-Result TLV and Crypto-Binding TLV if necessary. The server then tears down the tunnel and sends a clear text EAP-Success.

If the peer receives a Result TLV indicating success from the server, but its authentication policies are not satisfied (for example it requires a particular authentication mechanism be run or it wants to request a PAC), it may request further action from the server using the Request-Action TLV. The Request-Action TLV is sent along with the Result TLV indicating what EAP Success/Failure result the peer would expect if the requested action is not granted. The value of the Request-Action TLV indicates what the peer would like to do next. The format and values for the Request-Action TLV are defined in Section 4.2.9.

Upon receiving the Request-Action TLV the server may process the request or ignore it, based on its policy. If the server ignores the request, it proceeds with termination of the tunnel and send the clear text EAP Success or Failure message based on the value of the peer's result TLV. If the server honors and processes the request, it continues with the requested action. The conversation completes with a Result TLV exchange. The Result TLV may be included with the TLV that completes the requested action.

Error handling for Phase 2 is discussed in Section 3.6.2.

3.4. Determining Peer-Id and Server-Id

The Peer-Id and Server-Id may be determined based on the types of credentials used during either the EAP-FAST tunnel creation or authentication.

When X.509 certificates are used for peer authentication, the Peer-Id is determined by the subject or subjectAltName fields in the peer certificate. As noted in [RFC3280] (updated by [RFC4630]):

The subject field identifies the entity associated with the public key stored in the subject public key field. The subject name MAY be carried in the subject field and/or the subjectAltName extension.... If subject naming information is present only in the subjectAltName extension (e.g., a key bound only to an email address or URI), then the subject name MUST be an empty sequence and the subjectAltName extension MUST be critical.

Where it is non-empty, the subject field MUST contain an X.500 distinguished name (DN).

If an inner EAP method is run, then the Peer-Id is obtained from the inner method.

When the server uses an X.509 certificate to establish the TLS tunnel, the Server-Id is determined in a similar fashion as stated above for the Peer-Id; e.g., the subject or subjectAltName field in the server certificate defines the Server-Id.

3.5. EAP-FAST Session Identifier

The EAP session identifier is constructed using the random values provided by the peer and server during the TLS tunnel establishment. The Session-Id is defined as follows:

Session-Id = 0x2B || client_random || server_random)

client_random = 32 byte nonce generated by the peer

server_random = 32 byte nonce generated by the server

3.6. Error Handling

EAP-FAST uses the following error handling rules summarized below:

1. Errors in the TLS layer are communicated via TLS alert messages in all phases of EAP-FAST.

2. The Intermediate-Result TLVs carry success or failure indications of the individual EAP methods in EAP-FAST Phase 2. Errors within the EAP conversation in Phase 2 are expected to be handled by individual EAP methods.
3. Violations of the TLV rules are handled using Result TLVs together with Error TLVs.
4. Tunnel compromised errors (errors caused by Crypto-Binding failed or missing) are handled using Result TLVs and Error TLVs.

3.6.1. TLS Layer Errors

If the EAP-FAST server detects an error at any point in the TLS Handshake or the TLS layer, the server SHOULD send an EAP-FAST request encapsulating a TLS record containing the appropriate TLS alert message rather than immediately terminating the conversation so as to allow the peer to inform the user of the cause of the failure and possibly allow for a restart of the conversation. The peer MUST send an EAP-FAST response to an alert message. The EAP-Response packet sent by the peer may encapsulate a TLS ClientHello handshake message, in which case the EAP-FAST server MAY allow the EAP-FAST conversation to be restarted, or it MAY contain an EAP-FAST response with a zero-length message, in which case the server MUST terminate the conversation with an EAP-Failure packet. It is up to the EAP-FAST server whether to allow restarts, and if so, how many times the conversation can be restarted. An EAP-FAST server implementing restart capability SHOULD impose a limit on the number of restarts, so as to protect against denial-of-service attacks.

If the EAP-FAST peer detects an error at any point in the TLS layer, the EAP-FAST peer should send an EAP-FAST response encapsulating a TLS record containing the appropriate TLS alert message. The server may restart the conversation by sending an EAP-FAST request packet encapsulating the TLS HelloRequest handshake message. The peer may allow the EAP-FAST conversation to be restarted or it may terminate the conversation by sending an EAP-FAST response with a zero-length message.

3.6.2. Phase 2 Errors

Any time the peer or the server finds a fatal error outside of the TLS layer during Phase 2 TLV processing, it MUST send a Result TLV of failure and an Error TLV with the appropriate error code. For errors involving the processing of the sequence of exchanges, such as a violation of TLV rules (e.g., multiple EAP-Payload TLVs), the error code is `Unexpected_TLVs_Exchanged`. For errors involving a tunnel compromise, the error-code is `Tunnel_Compromise_Error`. Upon sending

a Result TLV with a fatal Error TLV the sender terminates the TLS tunnel. Note that a server will still wait for a message from the peer after it sends a failure, however the server does not need to process the contents of the response message.

If a server receives a Result TLV of failure with a fatal Error TLV, it SHOULD send a clear text EAP-Failure. If a peer receives a Result TLV of failure, it MUST respond with a Result TLV indicating failure. If the server has sent a Result TLV of failure, it ignores the peer response, and it SHOULD send a clear text EAP-Failure.

3.7. Fragmentation

A single TLS record may be up to 16384 octets in length, but a TLS message may span multiple TLS records, and a TLS certificate message may in principle be as long as 16 MB. This is larger than the maximum size for a message on most media types, therefore it is desirable to support fragmentation. Note that in order to protect against reassembly lockup and denial-of-service attacks, it may be desirable for an implementation to set a maximum size for one such group of TLS messages. Since a typical certificate chain is rarely longer than a few thousand octets, and no other field is likely to be anywhere near as long, a reasonable choice of maximum acceptable message length might be 64 KB. This is still a fairly large message packet size so an EAP-FAST implementation MUST provide its own support for fragmentation and reassembly.

Since EAP is a lock-step protocol, fragmentation support can be added in a simple manner. In EAP, fragments that are lost or damaged in transit will be retransmitted, and since sequencing information is provided by the Identifier field in EAP, there is no need for a fragment offset field.

EAP-FAST fragmentation support is provided through the addition of flag bits within the EAP-Response and EAP-Request packets, as well as a TLS Message Length field of four octets. Flags include the Length included (L), More fragments (M), and EAP-FAST Start (S) bits. The L flag is set to indicate the presence of the four-octet TLS Message Length field, and MUST be set for the first fragment of a fragmented TLS message or set of messages. The M flag is set on all but the last fragment. The S flag is set only within the EAP-FAST start message sent from the EAP server to the peer. The TLS Message Length field is four octets, and provides the total length of the TLS message or set of messages that is being fragmented; this simplifies buffer allocation.

When an EAP-FAST peer receives an EAP-Request packet with the M bit set, it MUST respond with an EAP-Response with EAP-Type of EAP-FAST

and no data. This serves as a fragment ACK. The EAP server must wait until it receives the EAP-Response before sending another fragment. In order to prevent errors in processing of fragments, the EAP server MUST increment the Identifier field for each fragment contained within an EAP-Request, and the peer must include this Identifier value in the fragment ACK contained within the EAP-Response. Retransmitted fragments will contain the same Identifier value.

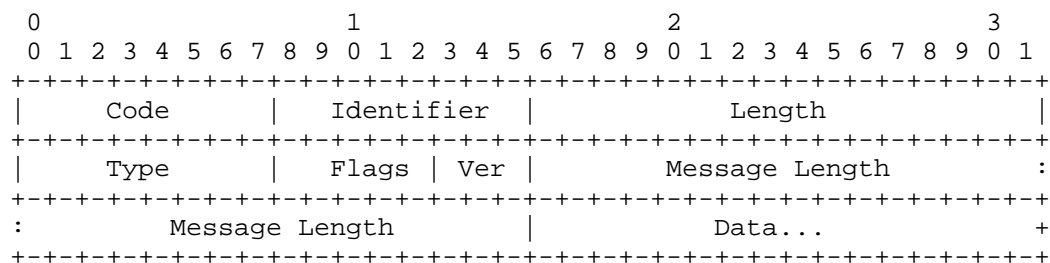
Similarly, when the EAP-FAST server receives an EAP-Response with the M bit set, it must respond with an EAP-Request with EAP-Type of EAP-FAST and no data. This serves as a fragment ACK. The EAP peer MUST wait until it receives the EAP-Request before sending another fragment. In order to prevent errors in the processing of fragments, the EAP server MUST increment the Identifier value for each fragment ACK contained within an EAP-Request, and the peer MUST include this Identifier value in the subsequent fragment contained within an EAP-Response.

4. Message Formats

The following sections describe the message formats used in EAP-FAST. The fields are transmitted from left to right in network byte order.

4.1. EAP-FAST Message Format

A summary of the EAP-FAST Request/Response packet format is shown below.



Code

The code field is one octet in length defined as follows:

- 1 Request
- 2 Response

Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field MUST be changed on each Request packet. The Identifier field in the Response packet MUST match the Identifier field from the corresponding request.

Length

The Length field is two octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, Flags, Ver, Message Length, and Data fields. Octets outside the range of the Length field should be treated as Data Link Layer padding and should be ignored on reception.

Type

43 for EAP-FAST

Flags

```
  0 1 2 3 4
+---+---+---+
|L M S R R|
+---+---+---+
```

- L Length included; set to indicate the presence of the four octet Message Length field
- M More fragments; set on all but the last fragment
- S EAP-FAST start; set in an EAP-FAST Start message
- R Reserved (must be zero)

Ver

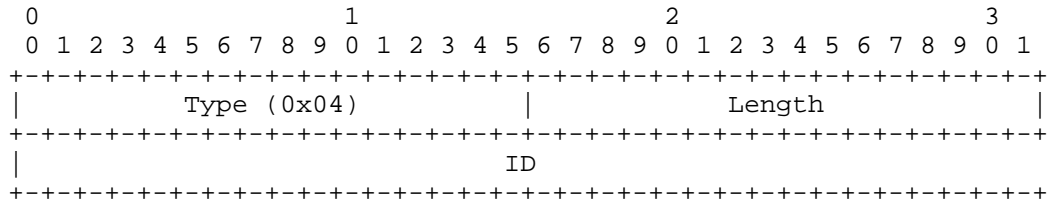
This field contains the version of the protocol. This document describes version 2 (010 in binary) of EAP-FAST.

Message Length

The Message Length field is four octets, and is present only if the L bit is set. This field provides the total length of the message that may be fragmented over the data fields of multiple packets.

Data

In the case of an EAP-FAST Start request (i.e., when the S bit is set) the Data field consists of the A-ID described in Section 4.1.1. In other cases, when the Data field is present, it consists of an encapsulated TLS packet in TLS record format. An EAP-FAST packet with Flags and Version fields, but with zero length data field, is used to indicate EAP-FAST acknowledgement for either a fragmented message, a TLS Alert message or a TLS Finished message.

4.1.1. Authority ID Data**Type**

The Type field is two octets. It is set to 0x0004 for Authority ID

Length

The Length field is two octets, which contains the length of the ID field in octets.

ID

Hint of the identity of the server. It should be unique across the deployment.

4.2. EAP-FAST TLV Format and Support

The TLVs defined here are standard Type-Length-Value (TLV) objects. The TLV objects could be used to carry arbitrary parameters between EAP peer and EAP server within the protected TLS tunnel.

The EAP peer may not necessarily implement all the TLVs supported by the EAP server. To allow for interoperability, TLVs are designed to allow an EAP server to discover if a TLV is supported by the EAP peer, using the NAK TLV. The mandatory bit in a TLV indicates whether support of the TLV is required. If the peer or server does not support a TLV marked mandatory, then it MUST send a NAK TLV in the response, and all the other TLVs in the message MUST be ignored. If an EAP peer or server finds an unsupported TLV that is marked as optional, it can ignore the unsupported TLV. It MUST NOT send an NAK TLV for a TLV that is not marked mandatory.

Note that a peer or server may support a TLV with the mandatory bit set, but may not understand the contents. The appropriate response to a supported TLV with content that is not understood is defined by the individual TLV specification.

EAP implementations compliant with this specification MUST support TLV exchanges, as well as the processing of mandatory/optional settings on the TLV. Implementations conforming to this specification MUST support the following TLVs:

Result TLV

NAK TLV

Error TLV

EAP-Payload TLV

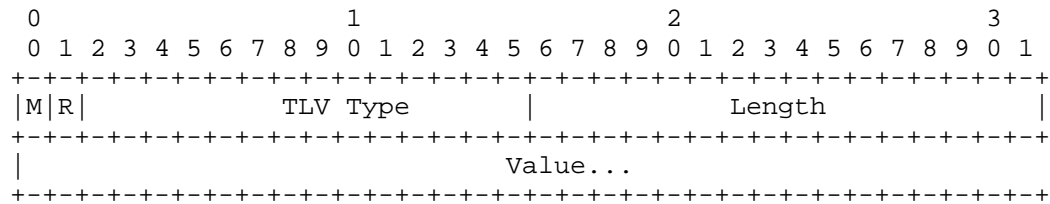
Intermediate-Result TLV

Crypto-Binding TLV

Request-Action TLV

4.2.1. General TLV Format

TLVs are defined as described below. The fields are transmitted from left to right.



M

0 Optional TLV

1 Mandatory TLV

R

Reserved, set to zero (0)

TLV Type

A 14-bit field, denoting the TLV type. Allocated Types include:

0 Reserved

1 Reserved

2 Reserved

3 Result TLV (Section 4.2.2)

4 NAK TLV (Section 4.2.3)

- 5 Error TLV (Section 4.2.4)
- 7 Vendor-Specific TLV (Section 4.2.5)
- 9 EAP-Payload TLV (Section 4.2.6)
- 10 Intermediate-Result TLV (Section 4.2.7)
- 11 PAC TLV [RFC5422]
- 12 Crypto-Binding TLV (Section 4.2.8)
- 18 Server-Trusted-Root TLV [RFC5422]
- 19 Request-Action TLV (Section 4.2.9)
- 20 PKCS#7 TLV [RFC5422]
- TBD Channel-Binding TLV (Section 4.2.10)
- TBD Identity-Type TLV (Section 4.2.11)
- TBD Basic-Password-Auth-Req TLV (Section 4.2.12)
- TBD Basic-Password-Auth-Resp TLV (Section 4.2.13)

Length

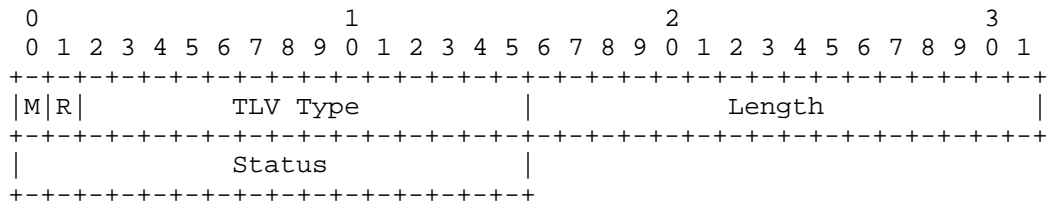
The length of the Value field in octets.

Value

The value of the TLV.

4.2.2. Result TLV

The Result TLV provides support for acknowledged success and failure messages for protected termination within EAP-FAST. If the Status field does not contain one of the known values, then the peer or EAP server MUST treat this as a fatal error of Unexpected_TLVs_Exchanged. The behavior of the Result TLV is further discussed in Section 3.3.3 and Section 3.6.2. A Result TLV indicating failure MUST NOT be accompanied by the following TLVs: NAK, EAP-Payload TLV, or Crypto-Binding TLV. The Result TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

3 for Result TLV

Length

2

Status

The Status field is two octets. Values include:

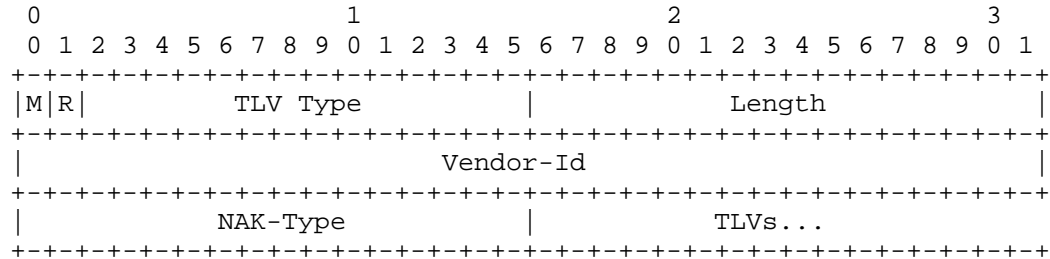
1 Success

2 Failure

4.2.3. NAK TLV

The NAK TLV allows a peer to detect TLVs that are not supported by the other peer. An EAP-FAST packet can contain 0 or more NAK TLVs. A NAK TLV should not be accompanied by other TLVs. A NAK TLV MUST NOT be sent in response to a message containing a Result TLV, instead a Result TLV of failure should be sent indicating failure and an Error TLV of Unexpected_TLVs_Exchanged. The NAK TLV is defined as

follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

4 for NAK TLV

Length

>=6

Vendor-Id

The Vendor-Id field is four octets, and contains the Vendor-Id of the TLV that was not supported. The high-order octet is 0 and the low-order three octets are the Structure of Management Information (SMI) Network Management Private Enterprise Code of the Vendor in network byte order. The Vendor-Id field MUST be zero for TLVs that are not Vendor-Specific TLVs.

NAK-Type

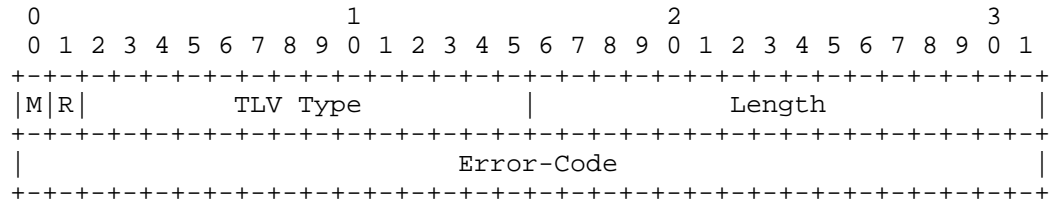
The NAK-Type field is two octets. The field contains the Type of the TLV that was not supported. A TLV of this Type MUST have been included in the previous packet.

TLVs

This field contains a list of zero or more TLVs, each of which MUST NOT have the mandatory bit set. These optional TLVs are for future extensibility to communicate why the offending TLV was determined to be unsupported.

4.2.4. Error TLV

The Error TLV allows an EAP peer or server to indicate errors to the other party. An EAP-FAST packet can contain 0 or more Error TLVs. The Error-Code field describes the type of error. Error Codes 1-999 represent successful outcomes (informative messages), 1000-1999 represent warnings, and codes 2000-2999 represent fatal errors. A fatal Error TLV MUST be accompanied by a Result TLV indicating failure and the conversation must be terminated as described in Section 3.6.2. The Error TLV is defined as follows:



M

Mandatory, set to one (1)

R

Reserved, set to zero (0)

TLV Type

5 for Error TLV

Length

4

Error-Code

The Error-Code field is four octets. Currently defined values for Error-Code include:

2001 Tunnel_Compromise_Error

2002 Unexpected_TLVs_Exchanged

4.2.5. Vendor-Specific TLV

The Vendor-Specific TLV is available to allow vendors to support their own extended attributes not suitable for general usage. A Vendor-Specific TLV attribute can contain one or more TLVs, referred to as Vendor TLVs. The TLV-type of a Vendor-TLV is defined by the vendor. All the Vendor TLVs inside a single Vendor-Specific TLV belong to the same vendor. There can be multiple Vendor-Specific TLVs from different vendors in the same message.

Vendor TLVs may be optional or mandatory. Vendor TLVs sent with Result TLVs MUST be marked as optional.

The Vendor-Specific TLV is defined as follows:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R|          TLV Type          |          Length          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          Vendor-Id          |
+-----+-----+-----+-----+-----+-----+-----+-----+
|          Vendor TLVs....
+-----+-----+-----+-----+-----+-----+-----+-----+

```


M

0 or 1

R

Reserved, set to zero (0)

TLV Type

7 for Vendor Specific TLV

Length

4 + cumulative length of all included Vendor TLVs

Vendor-Id

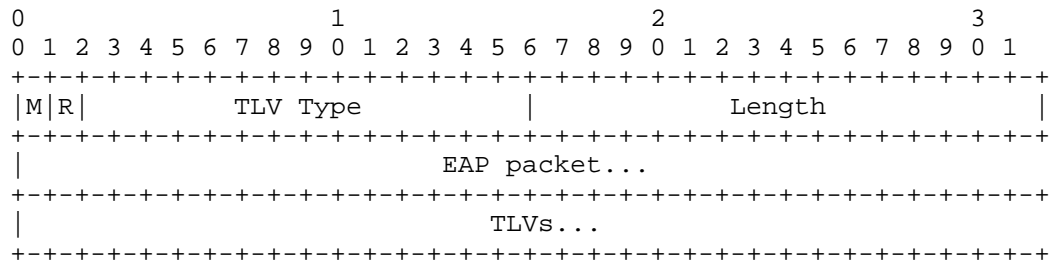
The Vendor-Id field is four octets, and contains the Vendor-Id of the TLV. The high-order octet is 0 and the low-order 3 octets are the SMI Network Management Private Enterprise Code of the Vendor in network byte order.

Vendor TLVs

This field is of indefinite length. It contains vendor-specific TLVs, in a format defined by the vendor.

4.2.6. EAP-Payload TLV

To allow piggybacking an EAP request or response with other TLVs, the EAP-Payload TLV is defined, which includes an encapsulated EAP packet and a list of optional TLVs. The optional TLVs are provided for future extensibility to provide hints about the current EAP authentication. Only one EAP-Payload TLV is allowed in a message. The EAP-Payload TLV is defined as follows:



M

Mandatory, set to (1)

R

Reserved, set to zero (0)

TLV Type

9 for EAP-Payload TLV

Length

length of embedded EAP packet + cumulative length of additional TLVs

EAP packet

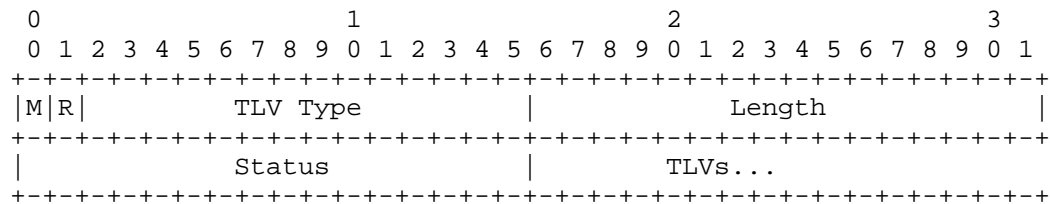
This field contains a complete EAP packet, including the EAP header (Code, Identifier, Length, Type) fields. The length of this field is determined by the Length field of the encapsulated EAP packet.

TLVs

This (optional) field contains a list of TLVs associated with the EAP packet field. The TLVs MUST NOT have the mandatory bit set. The total length of this field is equal to the Length field of the EAP-Payload TLV, minus the Length field in the EAP header of the EAP packet field.

4.2.7. Intermediate-Result TLV

The Intermediate-Result TLV provides support for acknowledged intermediate Success and Failure messages between multiple inner EAP methods within EAP. An Intermediate-Result TLV indicating success MUST be accompanied by a Crypto-Binding TLV. The optional TLVs associated with this TLV are provided for future extensibility to provide hints about the current result. The Intermediate-Result TLV is defined as follows:



M

Mandatory, set to (1)

R

Reserved, set to zero (0)

TLV Type

10 for Intermediate-Result TLV

Length

2 + cumulative length of the embedded associated TLVs

Status

The Status field is two octets. Values include:

- 1 Success
- 2 Failure

TLVs

This field is of indeterminate length, and contains zero or more of the TLVs associated with the Intermediate Result TLV. The TLVs in this field MUST NOT have the mandatory bit set.

4.2.8. Crypto-Binding TLV

The Crypto-Binding TLV is used to prove that both the peer and server participated in the tunnel establishment and sequence of authentications. It also provides verification of the EAP-FAST version negotiated before TLS tunnel establishment, see Section 3.1.

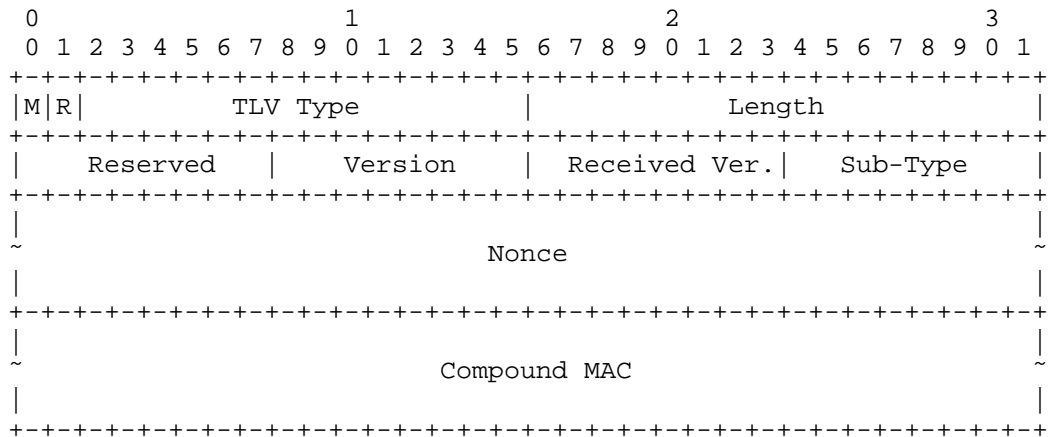
The Crypto-Binding TLV MUST be included with the Intermediate-Result TLV to perform Cryptographic Binding after each successful EAP method in a sequence of EAP methods. The Crypto-Binding TLV can be issued at other times as well.

The Crypto-Binding TLV is valid only if the following checks pass:

- o The Crypto-Binding TLV version is supported
- o The MAC verifies correctly
- o The received version in the Crypto-Binding TLV matches the version sent by the receiver during the EAP version negotiation
- o The subtype is set to the correct value

If any of the above checks fails, then the TLV is invalid. An invalid Crypto-Binding TLV is a fatal error and is handled as described in Section 3.6.2

The Crypto-Binding TLV is defined as follows:



M

Mandatory, set to (1)

R

Reserved, set to zero (0)

TLV Type

12 for Crypto-Binding TLV

Length

56

Reserved

Reserved, set to zero (0)

Version

The Version field is a single octet, which is set to the version of Crypto-Binding TLV the EAP method is using. For an implementation compliant with this version of EAP-FAST, the

version number MUST be set to 1.

Received Version

The Received Version field is a single octet and MUST be set to the EAP version number received during version negotiation. Note that this field only provides protection against downgrade attacks, where a version of EAP requiring support for this TLV is required on both sides.

Sub-Type

The Sub-Type field is one octet. Defined values include

0 Binding Request

1 Binding Response

Nonce

The Nonce field is 32 octets. It contains a 256-bit nonce that is temporally unique, used for compound MAC key derivation at each end. The nonce in a request MUST have its least significant bit set to 0 and the nonce in a response MUST have the same value as the request nonce except the least significant bit MUST be set to 1.

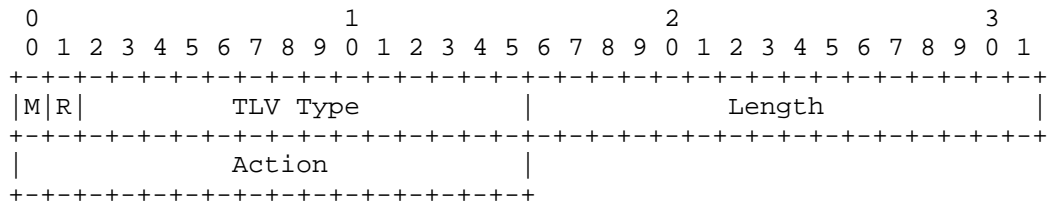
Compound MAC

The Compound MAC field is 20 octets. This can be the Server MAC (B1_MAC) or the Client MAC (B2_MAC). The computation of the MAC is described in Section 5.3.

4.2.9. Request-Action TLV

The Request-Action TLV MAY be sent by the peer along with a Result TLV in response to a server's successful Result TLV. It allows the peer to request the EAP server to negotiate additional EAP methods or process TLVs specified in the response packet. The server MAY ignore this TLV.

The Request-Action TLV is defined as follows:



M

Mandatory set to one (1)

R

Reserved, set to zero (0)

TLV Type

19 for Request-Action TLV

Length

2

Action

The Action field is two octets. Values include:

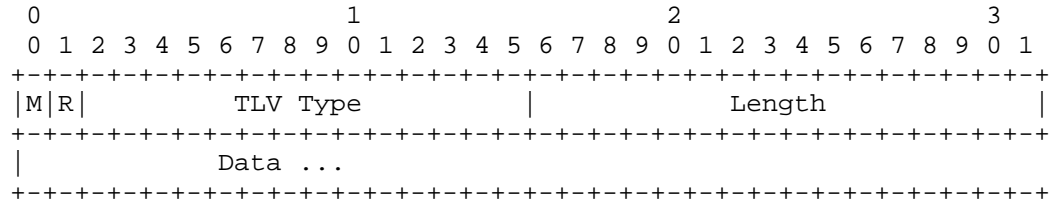
1 Process-TLV

2 Negotiate-EAP

4.2.10. Channel-Binding TLV

The Channel-Binding TLV allows an EAP-peer to send channel binding data to the EAP-server as described in [I-D.ietf-emu-chbind]. EAP-FASTv2 implementations MAY support this TLV, which cannot be responded to with a NAK TLV. If the Channel-Binding data field does not contain one of the known values or if the EAP server does not support this TLV, then the server MUST ignore the value. The

Channel-Binding TLV is defined as follows:



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

TBD for Channel-Binding TLV

Length

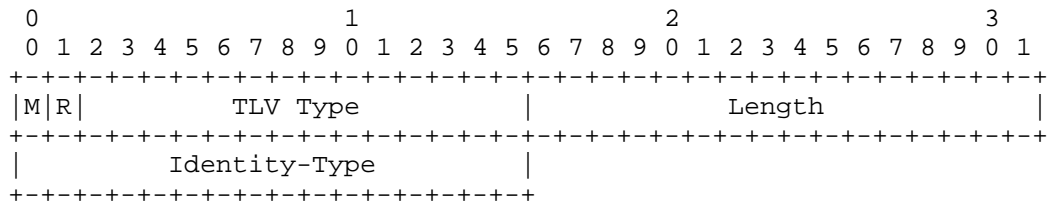
variable

Data

The data field contains channel binding data defined in [I-D.ietf-emu-chbind].

4.2.11. Identity-Type TLV

The Identity-Type TLV allows an EAP server to send a hint to help the EAP peer select the right type of identity; for example; user or machine. EAP-FASTv2 implementations MUST support this TLV. If the Identity-Type field does not contain one of the known values or if the EAP peer does not have an identity corresponding to the identity-type, then the peer SHOULD respond with a NAK TLV. The Identity-Type TLV is defined as follows:



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

TBD for Identity-Type TLV

Length

2

Identity-Type

The Identity-Type field is two octets. Values include:

1 User

2 Machine

4.2.12. Basic-Password-Auth-Req TLV

The Basic-Password-Auth-Req TLV is used by the authentication server to request a username and password from the peer. It contains an optional user prompt message for the request. The peer is expected to obtain the username and password and send them in a Basic-Password-Auth-Resp TLV.

The Basic-Password-Auth-Req TLV is defined as follows:

```

      0               1               2               3
      0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+-----+-----+-----+-----+-----+
|M|R|           TLV Type           |           Length           |
+-----+-----+-----+-----+-----+-----+-----+-----+
|   Prompt ....
+-----+-----+-----+-----+-----+-----+-----+-----+

```

M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

TBD for Basic-Password-Auth-Req TLV

Length

variable

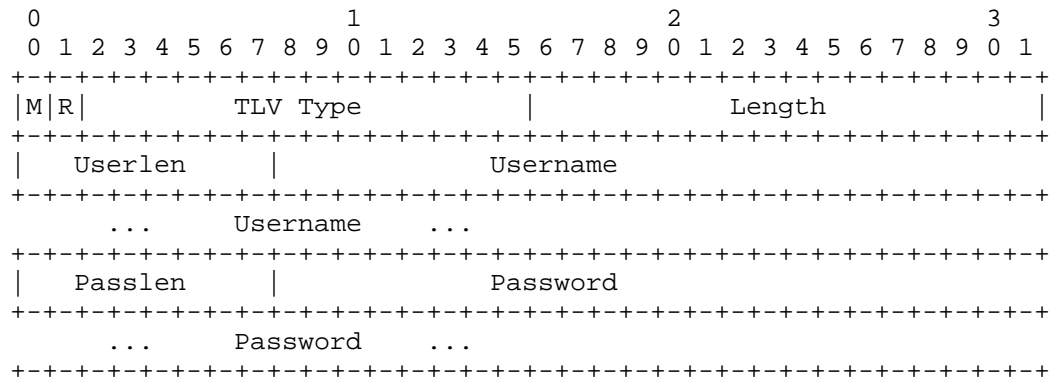
Prompt

optional user prompt message in UTF-8 format

4.2.13. Basic-Password-Auth-Resp TLV

The Basic-Password-Auth-Resp TLV is used by the peer to respond to a Basic-Password-Auth-Req TLV with a username and password. The TLV contains a username and password. The username and password are in UTF-8 format and prepared as defined in SASLprep [RFC4013].

The Basic-Password-Auth-Resp TLV is defined as follows:



M

0 (Optional)

R

Reserved, set to zero (0)

TLV Type

TBD for Basic-Password-Auth-Resp TLV

Length

variable

Userlen

Length of Username field in octets

Username

Username in UTF-8 format

Passlen

Length of Password field in octets

Password

Password in UTF-8 format

4.3. Table of TLVs

The following table provides a guide to which TLVs may be found in which kinds of messages, and in what quantity. The messages are as follows: Request is an EAP-FAST Request, Response is an EAP-FAST Response, Success is a message containing a successful Result TLV, and Failure is a message containing a failed Result TLV.

| Request | Response | Success | Failure | TLVs |
|---------|----------|---------|---------|--------------------------|
| 0-1 | 0-1 | 0-1 | 0-1 | Intermediate-Result |
| 0-1 | 0-1 | 0 | 0 | EAP-Payload |
| 0-1 | 0-1 | 1 | 1 | Result |
| 0-1 | 0-1 | 0-1 | 0-1 | Crypto-Binding |
| 0+ | 0+ | 0+ | 0+ | Error |
| 0+ | 0+ | 0 | 0 | NAK |
| 0+ | 0+ | 0+ | 0+ | Vendor-Specific [NOTE1] |
| 0 | 0-1 | 0-1 | 0-1 | Request-Action |
| 0 | 0-1 | 0 | 0 | Channel-Binding |
| 0-1 | 0 | 0 | 0 | Identity-Type |
| 0-1 | 0 | 0 | 0 | Basic-Password-Auth-Req |
| 0 | 0-1 | 0 | 0 | Basic-Password-Auth-Resp |

[NOTE1] Vendor TLVs (included in Vendor-Specific TLVs) sent with a Result TLV MUST be marked as optional.

The following table defines the meaning of the table entries in the sections below:

0 This TLV MUST NOT be present in the message.

0+ Zero or more instances of this TLV MAY be present in the message.

0-1 Zero or one instance of this TLV MAY be present in the message.

1 Exactly one instance of this TLV MUST be present in the message.

5. Cryptographic Calculations

5.1. EAP-FAST Authentication Phase 1: Key Derivations

The EAP-FAST Authentication tunnel key is calculated similarly to the TLS key calculation with an additional 40 octets (referred to as the `session_key_seed`) generated. The additional `session_key_seed` is used in the Session Key calculation in the EAP-FAST Tunneled Authentication conversation.

With EAP-FASTv2 the master secret is generated as specified in TLS. If a PAC is used then the master secret is obtained as described in RFC 5077 [RFC5077].

EAP-FASTv2 makes use of the TLS Keying Material Exporters defined in RFC 5705 [RFC5705] to derive the `session_key_seed`. The Label used in the derivation is "EXPORTER-EAPFASTv2-SKS". The length of the additional key material is 40 bytes. No context data is used in the export process.

The `session_key_seed` is used by the EAP-FAST Authentication Phase 2 conversation to both cryptographically bind the inner method(s) to the tunnel as well as generate the resulting EAP-FAST session keys. The other quantities are used as they are defined in [RFC5246].

5.2. Intermediate Compound Key Derivations

The `session_key_seed` derived as part of EAP-FAST Phase 2 is used in EAP-FAST Phase 2 to generate an Intermediate Compound Key (IMCK) used to verify the integrity of the TLS tunnel after each successful inner authentication and in the generation of Master Session Key (MSK) and Extended Master Session Key (EMSK) defined in [RFC3748]. Note that the IMCK must be recalculated after each successful inner EAP method.

The first step in these calculations is the generation of the base compound key, `IMCK[n]` from the `session_key_seed` and any session keys derived from the successful execution of `n` inner EAP methods. The inner EAP method(s) may provide Master Session Keys, `MSK1..MSKn`, corresponding to inner methods 1 through `n`. The MSK is truncated at 32 octets if it is longer than 32 octets or padded to a length of 32 octets with zeros if it is less than 32 octets. If the `i`th inner method does not generate an MSK, then `MSKi` is set to zero (e.g., `MSKi` = 32 octets of 0x00s). If an inner method fails, then it is not included in this calculation. The derivations of S-IMCK is as follows:


```
S-IMCK[0] = session_key_seed
For j = 1 to n-1 do
    IMCK[j] = TLS-PRF(S-IMCK[j-1], "Inner Methods Compound Keys",
        MSK[j], 60)
    S-IMCK[j] = first 40 octets of IMCK[j]
    CMK[j] = last 20 octets of IMCK[j]
```

where TLS-PRF is the PRF negotiated as part of TLS handshake [RFC5246].

5.3. Computing the Compound MAC

For authentication methods that generate keying material, further protection against man-in-the-middle attacks is provided through cryptographically binding keying material established by both EAP-FAST Phase 1 and EAP-FAST Phase 2 conversations. After each successful inner EAP authentication, EAP MSKs are cryptographically combined with key material from EAP-FAST Phase 1 to generate a compound session key, CMK. The CMK is used to calculate the Compound MAC as part of the Crypto-Binding TLV described in Section 4.2.8, which helps provide assurance that the same entities are involved in all communications in EAP-FAST. During the calculation of the Compound-MAC the MAC field is filled with zeros.

The Compound MAC computation is as follows:

```
CMK = CMK[j]
Compound-MAC = HMAC-HASH( CMK, Crypto-Binding TLV )
```

where j is the number of the last successfully executed inner EAP method, and HASH is the default hash function or the alternative hash function negotiated in TLS 1.2 [RFC5246].

5.4. EAP Master Session Key Generation

EAP-FAST Authentication assures the master session key (MSK) and Extended Master Session Key (EMSK) output from the EAP method are the result of all authentication conversations by generating an Intermediate Compound Key (IMCK). The IMCK is mutually derived by the peer and the server as described in Section 5.2 by combining the MSKs from inner EAP methods with key material from EAP-FAST Phase 1. The resulting MSK and EMSK are generated as part of the IMCKn key hierarchy as follows:

```
MSK = TLS-PRF(S-IMCK[j], "Session Key Generating Function", 64)
EMSK = TLS-PRF(S-IMCK[j],
    "Extended Session Key Generating Function", 64)
```


where j is the number of the last successfully executed inner EAP method.

The EMSK is typically only known to the EAP-FAST peer and server and is not provided to a third party. The derivation of additional keys and transportation of these keys to a third party is outside the scope of this document.

If no EAP methods have been negotiated inside the tunnel or no EAP methods have been successfully completed inside the tunnel, the MSK and EMSK will be generated directly from the session_key_seed meaning $S-IMCK = session_key_seed$.

6. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the EAP-FAST protocol, in accordance with BCP 26, [RFC2434].

EAP-FAST has already been assigned the EAP Method Type number 43.

The document defines a registry for EAP-FAST TLV types, which may be assigned by Specification Required as defined in [RFC2434]. Section 4.2 defines the TLV types that initially populate the registry. A summary of the EAP-FAST TLV types is given below:

- 0 Reserved
- 1 Reserved
- 2 Reserved
- 3 Result TLV
- 4 NAK TLV
- 5 Error TLV
- 7 Vendor-Specific TLV
- 9 EAP-Payload TLV

- 10 Intermediate-Result TLV
- 11 PAC TLV [RFC5422]
- 12 Crypto-Binding TLV
- 18 Server-Trusted-Root TLV [RFC5422]
- 19 Request-Action TLV
- 20 PKCS#7 TLV [RFC5422]

The following TLV type values need to be assigned:

Channel-Binding TLV

Identity-Type TLV

Basic-Password-Auth-Req TLV

Basic-Password-Auth-Resp TLV

The Error-TLV defined in Section 4.2.4 requires an error-code. EAP-FAST Error-TLV error-codes are assigned based on Specification Required as defined in [RFC2434]. The initial list of error codes is as follows:

2001 Tunnel_Compromise_Error

2002 Unexpected_TLVs_Exchanged

The Request-Action TLV defined in section Section 4.2.9 contains an action code which is assigned on a Specification Required basis as defined in [RFC2434]. The initial actions defined are:

- 1 Process-TLV
- 2 Negotiate-EAP

The various values under Vendor-Specific TLV are assigned by Private Use and do not need to be assigned by IANA.

7. Security Considerations

EAP-FAST is designed with a focus on wireless media, where the medium itself is inherent to eavesdropping. Whereas in wired media, an attacker would have to gain physical access to the wired medium; wireless media enables anyone to capture information as it is transmitted over the air, enabling passive attacks. Thus, physical security can not be assumed and security vulnerabilities are far greater. The threat model used for the security evaluation of EAP-FAST is defined in the EAP [RFC3748].

7.1. Mutual Authentication and Integrity Protection

EAP-FAST as a whole, provides message and integrity protection by establishing a secure tunnel for protecting the authentication method(s). The confidentiality and integrity protection is defined by TLS and provides the same security strengths afforded by TLS employing a strong entropy shared master secret. The integrity of the key generating authentication methods executed within the EAP-FAST tunnel is verified through the calculation of the Crypto-Binding TLV. This ensures that the tunnel endpoints are the same as the inner method endpoints.

The Result TLV is protected and conveys the true Success or Failure of EAP-FAST, and should be used as the indicator of its success or failure respectively. However, as EAP must terminate with a clear text EAP Success or Failure, a peer will also receive a clear text EAP Success or Failure. The received clear text EAP Success or Failure must match that received in the Result TLV; the peer SHOULD silently discard those clear text EAP success or failure messages that do not coincide with the status sent in the protected Result TLV.

7.2. Method Negotiation

As is true for any negotiated EAP protocol, NAK packets used to suggest an alternate authentication method are sent unprotected and as such, are subject to spoofing. During unprotected EAP method negotiation, NAK packets may be interjected as active attacks to negotiate down to a weaker form of authentication, such as EAP-MD5 (which only provides one-way authentication and does not derive a key). Both the peer and server should have a method selection policy that prevents them from negotiating down to weaker methods. Inner method negotiation resists attacks because it is protected by the mutually authenticated TLS tunnel established. Selection of EAP-FAST as an authentication method does not limit the potential inner authentication methods, so EAP-FAST should be selected when available.

An attacker cannot readily determine the inner EAP method used, except perhaps by traffic analysis. It is also important that peer implementations limit the use of credentials with an unauthenticated or unauthorized server.

7.3. Separation of Phase 1 and Phase 2 Servers

Separation of the EAP-FAST Phase 1 from the Phase 2 conversation is not recommended. Allowing the Phase 1 conversation to be terminated at a different server than the Phase 2 conversation can introduce vulnerabilities if there is not a proper trust relationship and protection for the protocol between the two servers. Some vulnerabilities include:

- o Loss of identity protection
- o Offline dictionary attacks
- o Lack of policy enforcement

There may be cases where a trust relationship exists between the Phase 1 and Phase 2 servers, such as on a campus or between two offices within the same company, where there is no danger in revealing the inner identity and credentials of the peer to entities between the two servers. In these cases, using a proxy solution without end-to-end protection of EAP-FAST MAY be used. The EAP-FAST encrypting/decrypting gateway SHOULD, at a minimum, provide support for IPsec or similar protection in order to provide confidentiality for the portion of the conversation between the gateway and the EAP server.

7.4. Mitigation of Known Vulnerabilities and Protocol Deficiencies

EAP-FAST addresses the known deficiencies and weaknesses in the EAP method. By employing a shared secret between the peer and server to establish a secured tunnel, EAP-FAST enables:

- o Per packet confidentiality and integrity protection
- o User identity protection
- o Better support for notification messages
- o Protected EAP inner method negotiation
- o Sequencing of EAP methods

- o Strong mutually derived master session keys
- o Acknowledged success/failure indication
- o Faster re-authentications through session resumption
- o Mitigation of dictionary attacks
- o Mitigation of man-in-the-middle attacks
- o Mitigation of some denial-of-service attacks

It should be noted that EAP-FAST, as in many other authentication protocols, a denial-of-service attack can be mounted by adversaries sending erroneous traffic to disrupt the protocol. This is a problem in many authentication or key agreement protocols and is therefore noted for EAP-FAST as well.

EAP-FAST was designed with a focus on protected authentication methods that typically rely on weak credentials, such as password-based secrets. To that extent, the EAP-FAST Authentication mitigates several vulnerabilities, such as dictionary attacks, by protecting the weak credential-based authentication method. The protection is based on strong cryptographic algorithms in TLS to provide message confidentiality and integrity. The keys derived for the protection relies on strong random challenges provided by both peer and server as well as an established key with strong entropy. Implementations should follow the recommendation in [RFC4086] when generating random numbers.

7.4.1. User Identity Protection and Verification

The initial identity request response exchange is sent in cleartext outside the protection of EAP-FAST. Typically the Network Access Identifier (NAI) [RFC4282] in the identity response is useful only for the realm information that is used to route the authentication requests to the right EAP server. This means that the identity response may contain an anonymous identity and just contain realm information. In other cases, the identity exchange may be eliminated altogether if there are other means for establishing the destination realm of the request. In no case should an intermediary place any trust in the identity information in the identity response since it is unauthenticated and may not have any relevance to the authenticated identity. EAP-FAST implementations should not attempt to compare any identity disclosed in the initial cleartext EAP Identity response packet with those Identities authenticated in Phase 2.

Identity request-response exchanges sent after the EAP-FAST tunnel is established are protected from modification and eavesdropping by attackers.

Note that since TLS client certificates are sent in the clear, if identity protection is required, then it is possible for the TLS authentication to be re-negotiated after the first server authentication. To accomplish this, the server will typically not request a certificate in the `server_hello`, then after the `server_finished` message is sent, and before EAP-FAST Phase 2, the server MAY send a `TLS hello_request`. This allows the client to perform client authentication by sending a `client_hello` if it wants to, or send a `no_renegotiation` alert to the server indicating that it wants to continue with EAP-FAST Phase 2 instead. Assuming that the client permits renegotiation by sending a `client_hello`, then the server will respond with `server_hello`, a certificate and `certificate_request` messages. The client replies with `certificate`, `client_key_exchange` and `certificate_verify` messages. Since this re-negotiation occurs within the encrypted TLS channel, it does not reveal client certificate details. It is possible to perform certificate authentication using an EAP method (for example: EAP-TLS) within the TLS session in EAP-FAST Phase 2 instead of using TLS handshake renegotiation.

7.4.2. Dictionary Attack Resistance

EAP-FAST was designed with a focus on protected authentication methods that typically rely on weak credentials, such as password-based secrets. EAP-FAST mitigates dictionary attacks by allowing the establishment of a mutually authenticated encrypted TLS tunnel providing confidentiality and integrity to protect the weak credential based authentication method.

7.4.3. Protection against Man-in-the-Middle Attacks

Allowing methods to be executed both with and without the protection of a secure tunnel opens up a possibility of a man-in-the-middle attack. To avoid man-in-the-middle attacks it is recommended to always deploy authentication methods with protection of EAP-FAST. EAP-FAST provides protection from man-in-the-middle attacks even if a deployment chooses to execute inner EAP methods both with and without EAP-FAST protection, EAP-FAST prevents this attack in two ways:

1. By using the PAC-Key to mutually authenticate the peer and server during EAP-FAST Authentication Phase 1 establishment of a secure tunnel.

2. By using the keys generated by the inner authentication method (if the inner methods are key generating) in the crypto-binding exchange and in the generation of the key material exported by the EAP method described in Section 5.

7.4.4. PAC Binding to User Identity

A PAC may be bound to a user identity. A compliant implementation of EAP-FAST MUST validate that an identity obtained in the PAC-Opaque field matches at minimum one of the identities provided in the EAP-FAST Phase 2 authentication method. This validation provides another binding to ensure that the intended peer (based on identity) has successfully completed the EAP-FAST Phase 1 and proved identity in the Phase 2 conversations.

7.5. Protecting against Forged Clear Text EAP Packets

EAP Success and EAP Failure packets are, in general, sent in clear text and may be forged by an attacker without detection. Forged EAP Failure packets can be used to attempt to convince an EAP peer to disconnect. Forged EAP Success packets may be used to attempt to convince a peer that authentication has succeeded, even though the authenticator has not authenticated itself to the peer.

By providing message confidentiality and integrity, EAP-FAST provides protection against these attacks. Once the peer and AS initiate the EAP-FAST Authentication Phase 2, compliant EAP-FAST implementations must silently discard all clear text EAP messages, unless both the EAP-FAST peer and server have indicated success or failure using a protected mechanism. Protected mechanisms include TLS alert mechanism and the protected termination mechanism described in Section 3.3.3.

The success/failure decisions within the EAP-FAST tunnel indicate the final decision of the EAP-FAST authentication conversation. After a success/failure result has been indicated by a protected mechanism, the EAP-FAST peer can process unprotected EAP Success and EAP Failure messages; however the peer MUST ignore any unprotected EAP success or failure messages where the result does not match the result of the protected mechanism.

To abide by [RFC3748], the server must send a clear text EAP Success or EAP Failure packet to terminate the EAP conversation. However, since EAP Success and EAP Failure packets are not retransmitted, the final packet may be lost. While an EAP-FAST protected EAP Success or EAP Failure packet should not be a final packet in an EAP-FAST conversation, it may occur based on the conditions stated above, so an EAP peer should not rely upon the unprotected EAP success and

failure messages.

7.6. Server Certificate Validation

As part of the TLS negotiation, the server presents a certificate to the peer. The peer **MUST** verify the validity of the EAP server certificate, and **SHOULD** also examine the EAP server name presented in the certificate, in order to determine whether the EAP server can be trusted. Please note that in the case where the EAP authentication is remoted, the EAP server will not reside on the same machine as the authenticator, and therefore the name in the EAP server's certificate cannot be expected to match that of the intended destination. In this case, a more appropriate test might be whether the EAP server's certificate is signed by a CA controlling the intended domain and whether the authenticator can be authorized by a server in that domain.

7.7. Tunnel PAC Considerations

Since the Tunnel PAC is stored by the peer, special care should be given to the overall security of the peer. The Tunnel PAC must be securely stored by the peer to prevent theft or forgery of any of the Tunnel PAC components. In particular, the peer must securely store the PAC-Key and protect it from disclosure or modification. Disclosure of the PAC-Key enables an attacker to establish the EAP-FAST tunnel; however, disclosure of the PAC-Key does not reveal the peer or server identity or compromise any other peer's PAC credentials. Modification of the PAC-Key or PAC-Opaque components of the Tunnel PAC may also lead to denial of service as the tunnel establishment will fail. The PAC-Opaque component is the effective TLS ticket extension used to establish the tunnel using the techniques of [RFC5077]. Thus, the security considerations defined by [RFC5077] also apply to the PAC- Opaque. The PAC-Info may contain information about the Tunnel PAC such as the identity of the PAC issuer and the Tunnel PAC lifetime for use in the management of the Tunnel PAC. The PAC-Info should be securely stored by the peer to protect it from disclosure and modification.

7.8. Security Claims

This section provides the needed security claim requirement for EAP [RFC3748].

| | |
|------------------|--|
| Auth. mechanism: | Certificate based, shared secret based and various tunneled authentication mechanisms. |
|------------------|--|

Ciphersuite negotiation: Yes

Mutual authentication: Yes

Integrity protection: Yes, Any method executed within the EAP-FAST tunnel is integrity protected. The cleartext EAP headers outside the tunnel are not integrity protected.

Replay protection: Yes

Confidentiality: Yes

Key derivation: Yes

Key strength: See Note 1 below.

Dictionary attack prot.: Yes

Fast reconnect: Yes

Cryptographic binding: Yes

Session independence: Yes

Fragmentation: Yes

Key Hierarchy: Yes

Channel binding: Yes

Notes

1. BCP 86 [RFC3766] offers advice on appropriate key sizes. The National Institute for Standards and Technology (NIST) also offers advice on appropriate key sizes in [NIST SP 800-57]. [RFC3766] Section 5 advises use of the following required RSA or DH module and DSA subgroup size in bits, for a given level of attack resistance in bits. Based on the table below, a 2048-bit RSA key is required to provide 128-bit equivalent key strength:

| Attack Resistance (bits) | RSA or DH Modulus size (bits) | DSA subgroup size (bits) |
|-----------------------------|----------------------------------|-----------------------------|
| ----- | ----- | ----- |
| 70 | 947 | 129 |
| 80 | 1228 | 148 |
| 90 | 1553 | 167 |
| 100 | 1926 | 186 |
| 150 | 4575 | 284 |
| 200 | 8719 | 383 |
| 250 | 14596 | 482 |

8. Acknowledgements

The EAP-FAST v1 design and protocol specification is based on the ideas and hard efforts of Pad Jakkahalli, Mark Krischer, Doug Smith, and Glen Zorn of Cisco Systems, Inc.

The TLV processing was inspired from work on the Protected Extensible Authentication Protocol version 2 (PEAPv2) with Ashwin Palekar, Dan Smith and Simon Josefsson. Helpful review comments were provided by Russ Housley, Jari Arkko, Ilan Frenkel and Jeremy Steiglitz.

9. References

9.1. Normative References

- [I-D.ietf-emu-chbind] Hartman, S., Clancy, C., and K. Hoeper, "Channel Binding Support for EAP Methods", draft-ietf-emu-chbind-06 (work in progress), October 2010.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 2434, October 1998.
- [RFC3268] Chown, P., "Advanced Encryption Standard (AES) Ciphersuites for Transport Layer Security (TLS)", RFC 3268, June 2002.
- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol

(EAP)", RFC 3748, June 2004.

- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.
- [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.
- [RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5422] Cam-Winget, N., McGrew, D., Salowey, J., and H. Zhou, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, March 2009.
- [RFC5705] Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010.
- [RFC5746] Rescorla, E., Ray, M., Dispensa, S., and N. Oskov, "Transport Layer Security (TLS) Renegotiation Indication Extension", RFC 5746, February 2010.

9.2. Informative References

- [I-D.ietf-emu-eaptunnel-req] Hoeper, K., Hanna, S., Zhou, H., and J. Salowey, "Requirements for a Tunnel Based EAP Method", draft-ietf-emu-eaptunnel-req-09 (work in progress), December 2010.
- [IEEE.802-1X.2004] "Local and Metropolitan Area Networks: Port-Based Network Access Control",

- IEEE Standard 802.1X, December 2004.
- [NIST SP 800-57] National Institute of Standards and Technology, "Recommendation for Key Management", NIST Special Publication 800-57, May 2006.
- [RFC2560] Myers, M., Ankney, R., Malpani, A., Galperin, S., and C. Adams, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP", RFC 2560, June 1999.
- [RFC3280] Housley, R., Polk, W., Ford, W., and D. Solo, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3280, April 2002.
- [RFC3579] Aboba, B. and P. Calhoun, "RADIUS (Remote Authentication Dial In User Service) Support For Extensible Authentication Protocol (EAP)", RFC 3579, September 2003.
- [RFC3766] Orman, H. and P. Hoffman, "Determining Strengths For Public Keys Used For Exchanging Symmetric Keys", BCP 86, RFC 3766, April 2004.
- [RFC4072] Eronen, P., Hiller, T., and G. Zorn, "Diameter Extensible Authentication Protocol (EAP) Application", RFC 4072, August 2005.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.
- [RFC4630] Housley, R. and S. Santesson, "Update to DirectoryString Processing in the Internet X.509 Public Key Infrastructure Certificate and

Certificate Revocation List (CRL) Profile", RFC 4630, August 2006.

[RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008.

[RFC5421] Cam-Winget, N. and H. Zhou, "Basic Password Exchange within the Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5421, March 2009.

[RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.

Appendix A. Evaluation Against Tunnel Based EAP Method Requirements

This section evaluates all tunnel based EAP method requirements described in [I-D.ietf-emu-eaptunnel-req] against EAP-FAST version 2.

A.1. Requirement 4.1.1 RFC Compliance

EAP-FAST v2 meets this requirement by being compliant to RFC 3748, RFC 4017, RFC 5247, and RFC 4962. It is also compliant with the "cryptographic algorithm agility" requirement by leveraging TLS 1.2 for all cryptographic algorithm negotiation.

A.2. Requirement 4.2.1 TLS Requirements

Requirement 4.2.1 states:

The tunnel based method MUST support TLS version 1.2 [RFC5246] and may support earlier versions greater than SSL 2.0 to enable the possibility of backwards compatibility.

EAP-FAST v2 meets this requirement by mandating TLS version 1.2 support as defined in Section 3.2.

A.3. Requirement 4.2.1.1.1 Cipher Suite Negotiation

Requirement 4.2.1.1.1 states:

Hence, the tunnel method MUST provide integrity protected cipher suite negotiation with secure integrity algorithms and integrity keys.

EAP-FAST v2 meets this requirement by using TLS to provide protected cipher suite negotiation.

A.4. Requirement 4.2.1.1.2 Tunnel Data Protection Algorithms

Requirement 4.2.1.1.2 states:

The tunnel method MUST provide at least one mandatory to implement cipher suite that provides the equivalent security of 128-bit AES for encryption and message authentication.

EAP-FAST v2 meets this requirement by mandating TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory to implement cipher suite as defined in Section 3.2.

A.5. Requirement 4.2.1.1.3 Tunnel Authentication and Key Establishment

EAP-FAST v2 meets this requirement by mandating TLS_RSA_WITH_AES_128_CBC_SHA as a mandatory to implement cipher suite which provides certificate-based authentication of the server and is approved by NIST. The mandatory to implement cipher suites only include cipher suites that use strong cryptographic algorithms. They do not include cipher suites providing mutually anonymous authentication or static Diffie-Hellman cipher suites as defined in Section 3.2.

A.6. Requirement 4.2.1.2 Tunnel Replay Protection

EAP-FAST v2 meets this requirement by using TLS to provide sufficient replay protection.

A.7. Requirement 4.2.1.3 TLS Extensions

EAP-FAST v2 meets this requirement by allowing TLS extensions, such as TLS Certificate Status Request extension [RFC6066] and SessionTicket extension [RFC5077] to be used during TLS tunnel establishment.

A.8. Requirement 4.2.1.4 Peer Identity Privacy

EAP-FAST v2 meets this requirement by establishment of the TLS tunnel and protection of inner method specific identities. In addition, the peer certificate can be sent confidentially (i.e. encrypted).

A.9. Requirement 4.2.1.5 Session Resumption

EAP-FAST v2 meets this requirement by mandating support of TLS session resumption as defined in Section 3.2.1 and TLS Session Resume

Using a PAC as defined in Section 3.2.2 .

A.10. Requirement 4.2.2 Fragmentation

EAP-FAST v2 meets this requirement by leveraging fragmentation support provided by TLS as defined in Section 3.7.

A.11. Requirement 4.2.3 Protection of Data External to Tunnel

EAP-FAST v2 meets this requirement by including EAP-FAST version number received in the computation of crypto-binding TLV as defined in Section 4.2.8.

A.12. Requirement 4.3.1 Extensible Attribute Types

EAP-FAST v2 meets this requirement by using an extensible TLV data layer inside the tunnel as defined in Section 4.2.

A.13. Requirement 4.3.2 Request/Challenge Response Operation

EAP-FAST v2 meets this requirement by allowing multiple TLVs to be sent in a single EAP request or response packet, while maintaining the half-duplex operation typical of EAP.

A.14. Requirement 4.3.3 Indicating Criticality of Attributes

EAP-FAST v2 meets this requirement by having a mandatory bit in TLV to indicate whether it is mandatory to support or not as defined in Section 4.2.

A.15. Requirement 4.3.4 Vendor Specific Support

EAP-FAST v2 meets this requirement by having a Vendor-Specific TLV to allow vendors to define their own attributes as defined in Section 4.2.5.

A.16. Requirement 4.3.5 Result Indication

EAP-FAST v2 meets this requirement by having a Result TLV to exchange the final result of the EAP authentication so both the peer and server have a synchronized state as defined in Section 4.2.2.

A.17. Requirement 4.3.6 Internationalization of Display Strings

EAP-FAST v2 meets this requirement by supporting UTF-8 format in Basic-Password-Auth-Req TLV as defined in Section 4.2.12 and Basic-Password-Auth-Resp TLV as defined in Section 4.2.13.

A.18. Requirement 4.4 EAP Channel Binding Requirements

EAP-FAST v2 meets this requirement by having a Channel-Binding TLV to exchange the EAP channel binding data as defined in Section 4.2.10.

A.19. Requirement 4.5.1.1 Confidentiality and Integrity

EAP-FAST v2 meets this requirement by running the password authentication inside a protected TLS tunnel.

A.20. Requirement 4.5.1.2 Authentication of Server

EAP-FAST v2 meets this requirement by mandating authentication of the server before establishment of the protected TLS and then running inner password authentication as defined in Section 3.2.

A.21. Requirement 4.5.1.3 Server Certificate Revocation Checking

EAP-FAST v2 meets this requirement by supporting TLS Certificate Status Request extension [RFC6066] during tunnel establishment.

A.22. Requirement 4.5.2 Internationalization

EAP-FAST v2 meets this requirement by supporting UTF-8 format in Basic-Password-Auth-Req TLV as defined in Section 4.2.12 and Basic-Password-Auth-Resp TLV as defined in Section 4.2.13.

A.23. Requirement 4.5.3 Meta-data

EAP-FAST v2 meets this requirement by supporting Identity-Type TLV as defined in Section 4.2.11 to indicate whether the authentication is for a user or a machine.

A.24. Requirement 4.5.4 Password Change

EAP-FAST v2 meets this requirement by supporting multiple Basic-Password-Auth-Req TLV and Basic-Password-Auth-Resp TLV exchanges within a single EAP authentication, which allows "housekeeping" functions such as password change.

A.25. Requirement 4.6.1 Method Negotiation

EAP-FAST v2 meets this requirement by supporting inner EAP method negotiation within the protected TLS tunnel.

A.26. Requirement 4.6.2 Chained Methods

EAP-FAST v2 meets this requirement by supporting inner EAP method chaining within protected TLS tunnel as defined in Section 3.3.1.

A.27. Requirement 4.6.3 Cryptographic Binding with the TLS Tunnel

EAP-FAST v2 meets this requirement by supporting cryptographic binding of the inner EAP method keys with the keys derived from the TLS tunnel as defined in Section 4.2.8.

A.28. Requirement 4.6.4 Peer Initiated

EAP-FAST v2 meets this requirement by supporting Request-Action TLV as defined in Section 4.2.9 to allow peer to initiate another inner EAP method.

A.29. Requirement 4.6.5 Method Meta-data

EAP-FAST v2 meets this requirement by supporting Identity-Type TLV as defined in Section 4.2.11 to indicate whether the authentication is for a user or a machine.

Appendix B. Examples

B.1. Successful Authentication

The following exchanges show a successful EAP-FAST authentication with optional PAC refreshment, the conversation will appear as follows:

| Authenticating Peer | Authenticator |
|---|--|
| ----- | ----- |
| | <- EAP-Request/ Identity |
| EAP-Response/ Identity (MyID1) -> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (EAP-FAST Start, S bit set, A-ID) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 (TLS client_hello with PAC-Opaque in SessionTicket extension)-> | |
| | <- EAP-Request/ |


```
EAP-Type=EAP-FAST, V=2
(TLS server_hello,
(TLS change_cipher_spec,
  TLS finished)
```

```
EAP-Response/
EAP-Type=EAP-FAST, V=2 ->
(TLS change_cipher_spec,
  TLS finished)
```

```
TLS channel established
(messages sent within the TLS channel)
```

```
<- EAP Payload TLV, EAP-Request,
    EAP-GTC, Challenge
```

```
EAP Payload TLV, EAP-Response,
EAP-GTC, Response with both
user name and password) ->
```

```
optional additional exchanges (new pin mode,
password change etc.) ...
```

```
<- Intermediate-Result TLV (Success)
Crypto-Binding TLV (Request)
```

```
Intermediate-Result TLV (Success)
Crypto-Binding TLV(Response) ->
```

```
<- Result TLV (Success)
    (Optional PAC TLV)
```

```
Result TLV (Success)
(PAC TLV Acknowledgment) ->
```

```
TLS channel torn down
(messages sent in clear text)
```

```
<- EAP-Success
```


B.2. Failed Authentication

The following exchanges show a failed EAP-FAST authentication due to wrong user credentials, the conversation will appear as follows:

```

Authenticating Peer      Authenticator
-----
                                <- EAP-Request/
                                Identity

EAP-Response/
Identity (MyID1) ->

                                <- EAP-Request/
                                EAP-Type=EAP-FAST, V=2
                                (EAP-FAST Start, S bit set, A-ID)

EAP-Response/
EAP-Type=EAP-FAST, V=2
(TLS client_hello with
PAC-Opaque in SessionTicket extension)->

                                <- EAP-Request/
                                EAP-Type=EAP-FAST, V=2
                                (TLS server_hello,
                                (TLS change_cipher_spec,
                                TLS finished)

EAP-Response/
EAP-Type=EAP-FAST, V=2 ->
(TLS change_cipher_spec,
TLS finished)

TLS channel established
(messages sent within the TLS channel)

                                <- EAP Payload TLV, EAP-Request,
                                EAP-GTC, Challenge

EAP Payload TLV, EAP-Response,
EAP-GTC, Response with both
user name and password) ->

                                <- EAP Payload TLV, EAP-Request,
                                EAP-GTC, error message

EAP Payload TLV, EAP-Response,
```


EAP-GTC, empty data packet to
 acknowledge unrecoverable error) ->

<- Result TLV (Failure)

Result TLV (Failure) ->

TLS channel torn down
 (messages sent in clear text)

<- EAP-Failure

B.3. Full TLS Handshake using Certificate-based Cipher Suite

In the case where an abbreviated TLS handshake is tried and failed and falls back to certificate based full TLS handshake occurs within EAP-FAST Phase 1, the conversation will appear as follows:

| Authenticating Peer ----- | Authenticator ----- |
|--|---|
| | <- EAP-Request/Identity |
| EAP-Response/ Identity (MyID1) -> | |
| <p>// Identity sent in the clear. May be a hint to help route the authentication request to EAP server, instead of the full user identity.</p> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (EAP-FAST Start, S bit set, A-ID) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 (TLS client_hello [PAC-Opaque extension])-> | |
| <p>// Peer sends PAC-Opaque of Tunnel PAC along with a list of ciphersuites supported. If the server rejects the PAC- Opaque, it falls through to the full TLS handshake</p> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (TLS server_hello, TLS certificate, [TLS server_key_exchange,] [TLS certificate_request,] TLS server_hello_done) |
| EAP-Response/ | |


```
EAP-Type=EAP-FAST, V=2
([TLS certificate,]
 TLS client_key_exchange,
 [TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->

                                <- EAP-Request/
EAP-Type=EAP-FAST, V=2
(TLS change_cipher_spec,
 TLS finished,
 EAP-Payload-TLV[EAP-Request/
 Identity])

// TLS channel established
(messages sent within the TLS channel)

// First EAP Payload TLV is piggybacked to the TLS Finished as
Application Data and protected by the TLS tunnel

EAP-Payload-TLV
[EAP-Response/Identity (MyID2)]->

// identity protected by TLS.

                                <- EAP-Payload-TLV
                                [EAP-Request/EAP-Type=X]

EAP-Payload-TLV
[EAP-Response/EAP-Type=X] ->

// Method X exchanges followed by Protected Termination

                                <- Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
                                Result TLV (Success)

Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
Result-TLV (Success) ->

// TLS channel torn down
(messages sent in clear text)

                                <- EAP-Success
```


B.4. Client authentication during Phase 1 with identity privacy

In the case where a certificate based TLS handshake occurs within EAP-FAST Phase 1, and client certificate authentication and identity privacy is desired, the conversation will appear as follows:

```

Authenticating Peer      Authenticator
-----
EAP-Response/
Identity (MyID1) ->

// Identity sent in the clear. May be a hint to help route
// the authentication request to EAP server, instead of the
// full user identity.

                                <- EAP-Request/Identity

EAP-Response/
EAP-Type=EAP-FAST, V=2
(TLS client_hello)->

                                <- EAP-Request/
                                EAP-Type=EAP-FAST, V=2
                                (EAP-FAST Start, S bit set, A-ID)

EAP-Response/
EAP-Type=EAP-FAST, V=2
(TLS client_key_exchange,
 TLS change_cipher_spec,
 TLS finished) ->

                                <- EAP-Request/
                                EAP-Type=EAP-FAST, V=2
                                (TLS server_hello,
                                 TLS certificate,
                                 [TLS server_key_exchange,]
                                 [TLS certificate_request,]
                                 TLS server_hello_done)

EAP-Response/
EAP-Type=EAP-FAST, V=2
(TLS client_key_exchange,
 TLS change_cipher_spec,
 TLS finished) ->

                                <- EAP-Request/
                                EAP-Type=EAP-FAST, V=2
                                (TLS change_cipher_spec,
                                 TLS finished,TLS Hello-Request)

// TLS channel established
// (messages sent within the TLS channel)

// TLS Hello-Request is piggybacked to the TLS Finished as
// Handshake Data and protected by the TLS tunnel

TLS client_hello ->

```



```

                                <- TLS server_hello,
                                TLS certificate,
                                [TLS server_key_exchange,]
                                [TLS certificate_request,]
                                TLS server_hello_done
[TLS certificate,]
  TLS client_key_exchange,
[TLS certificate_verify,]
  TLS change_cipher_spec,
  TLS finished ->

                                <- TLS change_cipher_spec,
                                TLS finished,
                                Result TLV (Success)

Result-TLV (Success)) ->

//TLS channel torn down
(messages sent in clear text)

                                <- EAP-Success

```

B.5. Fragmentation and Reassembly

In the case where EAP-FAST fragmentation is required, the conversation will appear as follows:

| Authenticating Peer ----- | Authenticator ----- |
|---|---|
| EAP-Response/ Identity (MyID) -> | <- EAP-Request/ Identity |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 (TLS client_hello)-> | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (EAP-FAST Start, S bit set, A-ID) |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (TLS server_hello, TLS certificate, [TLS server_key_exchange,] [TLS certificate_request,] TLS server_hello_done) |


```

                                (Fragment 1: L, M bits set)

EAP-Response/
EAP-Type=EAP-FAST, V=2 ->

                                <- EAP-Request/
                                    EAP-Type=EAP-FAST, V=2
                                (Fragment 2: M bit set)

EAP-Response/
EAP-Type=EAP-FAST, V=2 ->
                                <- EAP-Request/
                                    EAP-Type=EAP-FAST, V=2
                                (Fragment 3)

EAP-Response/
EAP-Type=EAP-FAST, V=2
([TLS certificate,]
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished)
(Fragment 1: L, M bits set)->

                                <- EAP-Request/
                                    EAP-Type=EAP-FAST, V=2

EAP-Response/
EAP-Type=EAP-FAST, V=2
(Fragment 2)->

                                <- EAP-Request/
                                    EAP-Type=EAP-FAST, V=2
                                    (TLS change_cipher_spec,
                                     TLS finished,
                                     [EAP-Payload-TLV[
                                     EAP-Request/Identity]])

// TLS channel established
  (messages sent within the TLS channel)

// First EAP Payload TLV is piggybacked to the TLS Finished as
  Application Data and protected by the TLS tunnel

EAP-Payload-TLV
[EAP-Response/Identity (MyID2)]->

// identity protected by TLS.

                                <- EAP-Payload-TLV
                                    [EAP-Request/EAP-Type=X]

```



```

EAP-Payload-TLV
[EAP-Response/EAP-Type=X] ->

// Method X exchanges followed by Protected Termination

                                <- Crypto-Binding TLV (Version=1,
                                EAP-FAST Version=2, Nonce,
                                CompoundMAC),
                                Result TLV (Success)

Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
Result-TLV (Success) ->

// TLS channel torn down
(messages sent in clear text)

                                <- EAP-Success

```

B.6. Sequence of EAP Methods

When EAP-FAST is negotiated, with a sequence of EAP method X followed by method Y, the conversation will occur as follows:

| Authenticating Peer | Authenticator |
|---|---|
| ----- | ----- |
| | <- EAP-Request/ Identity |
| EAP-Response/ Identity (MyID1) -> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (EAP-FAST Start, S bit set, A-ID) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 (TLS client_hello)-> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (TLS server_hello, TLS certificate, [TLS server_key_exchange,] [TLS certificate_request,] TLS server_hello_done) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 | |


```
([TLS certificate,]
 TLS client_key_exchange,
[TLS certificate_verify,]
 TLS change_cipher_spec,
 TLS finished) ->
    <- EAP-Request/
    EAP-Type=EAP-FAST, V=2
    (TLS change_cipher_spec,
     TLS finished,
     EAP-Payload-TLV[
     EAP-Request/Identity])

// TLS channel established
(messages sent within the TLS channel)

// First EAP Payload TLV is piggybacked to the TLS Finished as
Application Data and protected by the TLS tunnel

EAP-Payload-TLV
[EAP-Response/Identity] ->

    <- EAP-Payload-TLV
    [EAP-Request/EAP-Type=X]

EAP-Payload-TLV
[EAP-Response/EAP-Type=X] ->

    // Optional additional X Method exchanges...

    <- EAP-Payload-TLV
    [EAP-Request/EAP-Type=X]

EAP-Payload-TLV
[EAP-Response/EAP-Type=X]->

    <- Intermediate Result TLV (Success),
    Crypto-Binding TLV (Version=1
    EAP-FAST Version=2, Nonce,
    CompoundMAC),
    EAP Payload TLV [EAP-Type=Y],

// Next EAP conversation started after successful completion
of previous method X. The Intermediate-Result and Crypto-
Binding TLVs are sent in next packet to minimize round-
trips. In this example, identity request is not sent
before negotiating EAP-Type=Y.

// Compound MAC calculated using Keys generated from
```


EAP methods X and the TLS tunnel.

```
Intermediate Result TLV (Success),
Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
EAP-Payload-TLV [EAP-Type=Y] ->
```

```
// Optional additional Y Method exchanges...
```

```
<- EAP Payload TLV [
EAP-Type=Y]
```

```
EAP Payload TLV
[EAP-Type=Y] ->
```

```
<- Intermediate-Result-TLV (Success),
Crypto-Binding TLV (Version=1
EAP-FAST Version=2, Nonce,
CompoundMAC),
Result TLV (Success)
```

```
Intermediate-Result-TLV (Success),
Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
Result-TLV (Success) ->
```

```
// Compound MAC calculated using Keys generated from EAP
methods X and Y and the TLS tunnel. Compound Keys
generated using Keys generated from EAP methods X and Y;
and the TLS tunnel.
```

```
// TLS channel torn down (messages sent in clear text)
```

```
<- EAP-Success
```

B.7. Failed Crypto-binding

The following exchanges show a failed crypto-binding validation. The conversation will appear as follows:

| Authenticating Peer | Authenticator |
|--------------------------------------|-----------------------------|
| ----- | ----- |
| | <- EAP-Request/ Identity |
| EAP-Response/ Identity (MyID1) -> | |


```
<- EAP-Request/
EAP-Type=EAP-FAST, V=2
(EAP-FAST Start, S bit set, A-ID)

EAP-Response/
EAP-Type=EAP-FAST, V=2
(TLS client_hello without
PAC-Opaque extension)->

<- EAP-Request/
EAP-Type=EAP-FAST, V=2
(TLS Server Key Exchange
 TLS Server Hello Done)

EAP-Response/
EAP-Type=EAP-FAST, V=2 ->
(TLS Client Key Exchange
 TLS change_cipher_spec,
 TLS finished)

<- EAP-Request/
EAP-Type=EAP-FAST, V=2
(TLS change_cipher_spec
 TLS finished)
EAP-Payload-TLV[
 EAP-Request/Identity])

// TLS channel established
(messages sent within the TLS channel)

// First EAP Payload TLV is piggybacked to the TLS Finished as
Application Data and protected by the TLS tunnel

EAP-Payload TLV/
EAP Identity Response ->

<- EAP Payload TLV, EAP-Request,
(EAP-MSCHAPV2, Challenge)

EAP Payload TLV, EAP-Response,
(EAP-MSCHAPV2, Response) ->

<- EAP Payload TLV, EAP-Request,
(EAP-MSCHAPV2, Success Request)

EAP Payload TLV, EAP-Response,
(EAP-MSCHAPV2, Success Response) ->

<- Crypto-Binding TLV (Version=1,
 EAP-FAST Version=2, Nonce,
```



```

                                CompoundMAC),
                                Result TLV (Success)

Result TLV (Failure)
Error TLV with
(Error Code = 2001) ->

// TLS channel torn down
(messages sent in clear text)

<- EAP-Failure

```

B.8. Sequence of EAP Method with Vendor-Specific TLV Exchange

When EAP-FAST is negotiated, with a sequence of EAP method followed by Vendor-Specific TLV exchange, the conversation will occur as follows:

| Authenticating Peer ----- | Authenticator ----- |
|--|---|
| | <- EAP-Request/ Identity |
| EAP-Response/ Identity (MyID1) -> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (EAP-FAST Start, S bit set, A-ID) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 (TLS client_hello)-> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 (TLS server_hello, TLS certificate, [TLS server_key_exchange,] [TLS certificate_request,] TLS server_hello_done) |
| EAP-Response/ EAP-Type=EAP-FAST, V=2 ([TLS certificate,] TLS client_key_exchange, [TLS certificate_verify,] TLS change_cipher_spec, TLS finished) -> | |
| | <- EAP-Request/ EAP-Type=EAP-FAST, V=2 |


```
(TLS change_cipher_spec,
 TLS finished,
 EAP-Payload-TLV[
 EAP-Request/Identity])

// TLS channel established
(messages sent within the TLS channel)

// First EAP Payload TLV is piggybacked to the TLS Finished as
Application Data and protected by the TLS tunnel

EAP-Payload-TLV
[EAP-Response/Identity] ->

    <- EAP-Payload-TLV
    [EAP-Request/EAP-Type=X]

EAP-Payload-TLV
[EAP-Response/EAP-Type=X] ->

    <- EAP-Payload-TLV
    [EAP-Request/EAP-Type=X]

EAP-Payload-TLV
[EAP-Response/EAP-Type=X]->

    <- Intermediate Result TLV (Success),
    Crypto-Binding TLV (Version=1
    EAP-FAST Version=2, Nonce,
    CompoundMAC),
    Vendor-Specific TLV,

// Vendor Specific TLV exchange started after successful
completion of previous method X. The Intermediate-Result
and Crypto-Binding TLVs are sent with Vendor Specific TLV
in next packet to minimize round-trips.

// Compound MAC calculated using Keys generated from
EAP methods X and the TLS tunnel.

Intermediate Result TLV (Success),
Crypto-Binding TLV (Version=1,
EAP-FAST Version=2, Nonce,
CompoundMAC),
Vendor-Specific TLV ->

    // Optional additional Vendor-Specific TLV exchanges...
```



```
<- Vendor-Specific TLV

Vendor Specific TLV ->
    <- Result TLV (Success)

Result-TLV (Success) ->

// TLS channel torn down (messages sent in clear text)

<- EAP-Success
```

Authors' Addresses

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

EMail: hzhou@cisco.com

Nancy Cam-Winget
Cisco Systems
3625 Cisco Way
San Jose, CA 95134
US

EMail: ncamwing@cisco.com

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

EMail: jsalowey@cisco.com

Stephen Hanna
Juniper Networks
79 Parsons Street
Brighton, MA 02135
US

EMail: shanna@juniper.net

