

EMU
Internet-Draft
Intended status: Standards Track
Expires: May 19, 2022

M. Chen, Ed.
Li. Su
China Mobile
H. Wang
Huawei International Pte. Ltd.
November 15, 2021

Use Identity as Raw Public Key in EAP-TLS
draft-chen-emu-eap-tls-ibs-03

Abstract

This document specifies the use of identity as a raw public key in EAP-TLS, EAP-TLS for TLS1.2 is defined in RFC 5216 and EAP-TLS for TLS1.3 is defined in the draft draft-ietf-emu-eap-tls13 and draft-ietf-tls-dtls13. The procedures of EAP-TLS-IBS will consistent with EAP-TLS's interactive process, Identity-based signature will be extended to support EAP-TLS's signature algorithms.

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 <https://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 May 19, 2022.

Copyright Notice

Copyright (c) 2021 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 (<https://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	2
2. Terminology	4
3. Use case of the EAP-TLS-IBS	4
4. Structure of the Raw Public Key Extension	5
5. EAP-TLS using raw public keys	6
5.1. EAP TLS1.2 Client and Server Handshake Behavior	6
5.1.1. raw public keys TLS exchange	6
5.1.2. EAP-TLS handshake in TLS1.2	7
5.1.3. raw public keys EAP-TLS exchange	8
5.1.4. EAP-TLS1.2-IBS example	10
5.2. EAP TLS1.3 Client and Server Handshake Behavior	12
5.2.1. TLS1.3 handshake	12
5.2.2. EAP-TLS1.3 handshake procedure	13
5.2.3. raw public keys EAP-TLS1.3 exchange	14
5.2.4. EAP-TLS1.3-IBS example	16
6. Security Considerations	18
7. IANA Considerations	18
8. Acknowledgement	18
9. References	18
9.1. Normative References	18
9.2. Informative references	19
Authors' Addresses	19

1. Introduction

The Extensible Authentication Protocol (EAP) defined in RFC 3748 [RFC3748] can provide support for multiple authentication methods. Transport Layer Security (TLS) provides for mutual authentication, integrity-protected ciphersuite negotiation, and exchange between two endpoints. The EAP-TLS defined in RFC 5216 [RFC5216] which combines EAP and TLS that apply EAP method to load TLS procedures.

Traditionally, TLS client and server public keys are obtained in PKIX containers in-band as part of the TLS handshake procedure and are validated using trust anchors based on a PKIX certification authority (CA). But there is another method, Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS) are defined in RFC 7250 [RFC7250], the document defines two TLS extensions `client_certificate_type` and `server_certificate_type`, which can be used as part of an extended TLS handshake when raw public keys are used. In the draft `draft-ietf-emu-eap-tls13` reads certificates

can be of any type supported by TLS including raw public keys. In RFC7250[RFC7250] it assuming that an out-of-band mechanism is used to bind the public key to the entity presenting the key.

Digital signatures provide the functions of Sender reliability and Message integrity. A chain of trust for such signatures is usually provided by certificates, but in low-bandwidth and resource-constrained environments, the use of certificates might be undesirable. In comparison with the original certificate, the raw public key is fairly small. This document describes a signature algorithm using identity as a raw public key in EAP-TLS, instead of transmitting a full certificate in the EAP-TLS message, only public keys are exchanged between client and server, also known as EAP-TLS-IBS.

With the existing raw public key scheme, a public key and identity mapping table is required at server. This table usually established with offline method and may require additional efforts for establishment and maintenance, especially when the number of devices are huge. On the other hand, with IBS signature algorithm, it not only can take the advantage of raw public key, but also eliminates the efforts for the mapping table establishment and maintenance at the server side. Instead, a small table for CRL is enough for exclude revoked identity from accessing the network. A number of IBE and IBS algorithms have been standardized, such as ECCSI defined in RFC 6507[RFC6507].

IBC was first proposed by Adi Shamir in 1984. For an IBC system, a Key Management System (KMS) is required to generate keys for devices. The KMS choose its KMS Secret Authentication Key(KSAK) as the root of trust. A public parameter, KMS Public Authentication Key (KPAK) is derived from this secrete key and is used by others in verifying the signature. The signatures are generated by an entity with private keys obtained from the KMS. KMS is a trusted third party, users or devices can obtain private key using their identities from KMS. In IBS the private key is also known as Secret Signing Key(SSK). A sender can sign a message using SSK. The receiver can verify the signature with sender's identity and the KPAK.

This document is organized as follows: the second section defines the terms used in the text; the third section gives a brief overview of the IBS algorithms; the fourth section presents the example message flow and message format for EAP-TLS-IBS and follows by security consideration and IANA cosideration etc.

2. Terminology

The readers should be familiar with the terms defined in.

In addition, this document makes use of the following terms:

IBC: Identity-Based Cryptograph, it is an asymmetric public key cryptosystem.

IBS: Identity-based Signature, such as ECCSI.

PKI: Public Key Infrastructure, an infrastructure built with a public-key mechanism.

authenticator: The entity initiating EAP authentication.

peer: The entity that responds to the authenticator.

backend authenticator server: A backend authentication server is an entity that provides an authentication service to an authenticator. When used, this server typically executes EAP methods for the authenticator.

EAP server: The entity that terminates the EAP authentication method with the peer. In the case where no backend authentication server is used, the EAP server is part of the authenticator. In the case where the authenticator operates in pass-through mode, the EAP server is located on the backend authentication server.

3. Use case of the EAP-TLS-IBS

Used for authentication of Internet of Things devices: due to the limited processing power of IoT devices, resources for secure identity authentication are limited, especially passive, long life cycle devices, however, the traditional certificate authentication based on PKI X509, because of the complexity of certificate processing and certificate chain authentication, not very suitable for the Internet of Things scenario. Internet of Things devices really need a more lightweight authentication method, and EAP-TLS-IBS as one of the candidates.

Used for systems that do not support CA certificates: an internal system with network security boundaries that can self-operate the Key Management System(KMS) secret key distribution center, EAP-TLS-IBS can be used between internal subsystems.

4. Structure of the Raw Public Key Extension

To support the negotiation of using raw public between client and server, a new certificate structure is defined in RFC 7250[RFC7250]. It is used by the client and server in the hello messages to indicate the types of certificates supported by each side. When RawPublicKey type is selected for authentication, SubjectPublicKeyInfo which is a data structure is used to carry the raw public key and its cryptographic algorithm.

The SubjectPublicKeyInfo structure is defined in Section 4.1 of RFC 5280 [PKIX][RFC5280] and not only contains the raw keys, such as the public exponent and the modulus of an RSA public key, but also an algorithm identifier. The algorithm identifier can also include parameters. The structure of SubjectPublicKeyInfo is shown in Figure 1:

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm             AlgorithmIdentifier,
    subjectPublicKey       BIT STRING }

AlgorithmIdentifier ::= SEQUENCE {
    algorithm             OBJECT IDENTIFIER,
    parameters            ANY DEFINED BY algorithm OPTIONAL }
```

Figure 1: SubjectPublicKeyInfo ASN.1 Structure

The algorithms identifiers are Object Identifier(OIDs), AlgorithmIdentifier is also data structure with two fields, OID represent the cryptographic algorithm used with raw public key, such as ECCSI, parameters are the necessary parameters associated with the algorithm.

In the case of IBS algorithm, the User's identity is the raw public key which can be represented by "subjectPublicKey", when ECCSI is used as the Identity-based signature algorithm, then "algorithm" is for ECCSI, and "parameters" is the parameters needed in ECCSI.

So far, IBS has the following four algorithms, the following table is the corresponding table of Key type and OID.

Key Type	Document	OID
ISO/IEC 14888-3 IBS-1	ISO/IEC 14888-3: IBS-1 mechanism	1.0.14888.3.0.7
ISO/IEC 14888-3 IBS-2	ISO/IEC 14888-3: IBS-2 mechanism	1.0.14888.3.0.8
ISO/IEC 14888-3 ChineseIBS(SM9)	ISO/IEC 14888-3: ChineseIBS mechanism	1.2.156.10197.1.302.1
Elliptic Curve-Based Signatureless For Identity-based Encryption (ECCSI)	Section 5.2 in RFC 6507	1.3.6.1.5.5.7.6.29

Table 1: Algorithm Object Identifiers

In the draft draft-wang-tls-raw-public-key-with-ibc, there extend signature scheme with IBS algorithm which indicated in the client's "signature_algorithms" extension. The SignatureScheme data structure also keep pace with the section 4.

5. EAP-TLS using raw public keys

This section describes EAP-TLS-IBS both in the case of TLS1.2 and TLS1.3, each section contains EAP-TLS and EAP-TLS using raw public keys full message authentication, and finally give the example when using IBS.

5.1. EAP TLS1.2 Client and Server Handshake Behavior

5.1.1. raw public keys TLS exchange

As described in [RFC7250][RFC7250], the document intrudoces the use of raw public keys in TLS/DTLS, the basic raw public key TLS exchange will appear as follows, Figure 2 shows the client_certificate_type and server_certificate_type extensions added to the client and server hello messages. An extension type MUST NOT appear in the ServerHello unless the same extension type appeared in the corresponding ClientHello, defined in RFC5246[RFC5246].

The `server_certificate_type` extension in the client hello indicates the types of certificates the client is able to process when provided by the server in a subsequent certificate payload.

The `client_certificate_type` and `server_certificate_type` extensions sent in the client hello each carry a list of supported certificate types, sorted by client preference. When the client supports only one certificate type, it is a list containing a single element. Many types of certificates can be used, such as `RawPublicKey`, `X.509` and `OpenPGP`.



Figure 2: Basic Raw Public Key TLS Exchange

5.1.2. EAP-TLS handshake in TLS1.2

As described in [RFC3748] [RFC3748], the EAP-TLS conversation will typically begin with the authenticator and the peer negotiating EAP. The authenticator will then typically send an EAP-Request/Identity packet to the peer, and the peer will respond with an EAP-Response/Identity packet to the authenticator, containing the peer's user-Id. The authenticator MAY act as a pass-through device, with the EAP packets received from the peer being encapsulated for transmission to a backend authentication server.

In the case where the EAP-TLS mutual authentication is successful, defined in RFC5216 [RFC5216], the conversation will appear as follows:

Authenticating Peer	Authenticator
-----	-----
	<- EAP-Request/ Identity
EAP-Response/ Identity (MyID) ->	
	<- EAP-Request/ EAP-Type=EAP-TLS (TLS Start)
EAP-Response/ EAP-Type=EAP-TLS (TLS client_hello)->	
	<- EAP-Request/ EAP-Type=EAP-TLS (TLS server_hello, TLS certificate, TLS server_key_exchange, TLS certificate_request, TLS server_hello_done)
EAP-Response/ EAP-Type=EAP-TLS (TLS certificate, TLS client_key_exchange, TLS certificate_verify, TLS change_cipher_spec, TLS finished) ->	
	<- EAP-Request/ EAP-Type=EAP-TLS (TLS change_cipher_spec, TLS finished)
EAP-Response/ EAP-Type=EAP-TLS ->	
	<- EAP-Success

Figure 3: EAP-TLS authentication procedure with TLS1.2 handshake

5.1.3. raw public keys EAP-TLS exchange

This section describes EAP-TLS extend using raw public keys, the procedures is as follows, In the discussion, we will use the term "EAP server" to denote the ultimate endpoint conversing with the peer.

Authenticating Peer ----- EAP-Response/ Identity (MyID) -> EAP-Response/ EAP-Type=EAP-TLS (TLS client_hello +signature_algorithm server_certificate_type, client_certificate_type)->	EAP server ----- <- EAP-Request/ Identity <- EAP-Request/ EAP-Type=EAP-TLS (TLS Start) <- EAP-Request/ EAP-Type=EAP-TLS (TLS server_hello, {client_certificate_type} {server_certificate_type} {TLS certificate} {TLS server_key_exchange} {TLS certificate_request} {TLS server_hello_done}) EAP-Response/ EAP-Type=EAP-TLS (TLS certificate, TLS client_key_exchange, TLS certificate_verify, TLS change_cipher_spec, TLS finished) -> <- EAP-Request/ EAP-Type=EAP-TLS (TLS change_cipher_spec, TLS finished) EAP-Response/ EAP-Type=EAP-TLS -> <- EAP-Success
---	--

Figure 4: EAP-TLS extend raw public keys authentication procedure with TLS1.2 handshake

5.1.4. EAP-TLS1.2-IBS example

In this example, both the TLS client and server use ECCSI for authentication, and they are restricted in that they can only process ECCSI signature algorithm. As a result, the TLS client sets both the `server_certificate_type` and the `client_certificate_type` extensions to be raw public key; in addition, the client sets the signature algorithm in the client hello message to be `eccsi_sha256`.

Authenticating Peer	EAP server
EAP-Response/ Identity (MyID) ->	<- EAP-Request/ Identity
EAP-Response/ EAP-Type=EAP-TLS (TLS client_hello signature_algorithm = (eccsi_sha256) server_certificate_type = (RawPublicKey,...) client_certificate_type = (RawPublicKey,...)) ->	<- EAP-Request/ EAP-Type=EAP-TLS (TLS Start)
	<- EAP-Request/ EAP-Type=EAP-TLS (TLS server_hello, {client_certificate_type = RawPublicKey} {server_certificate_type = RawPublicKey} {certificate = (1.3.6.1.5.5.7.6.29, hash value of ECCSIPublicParameters), serverID}) {certificate_request = (eccsi_sha256)} {server_hello_done})
EAP-Response/ EAP-Type=EAP-TLS (({certificate = ((1.3.6.1.5.5.7.6.29, hash value of ECCSIPublicParameters), ClientID})), {certificate_verify = (ECCSI-Sig-Value)}, {finished}) ->	
EAP-Response/ EAP-Type=EAP-TLS ->	<- EAP-Request/ EAP-Type=EAP-TLS (TLS finished)
	<- EAP-Success

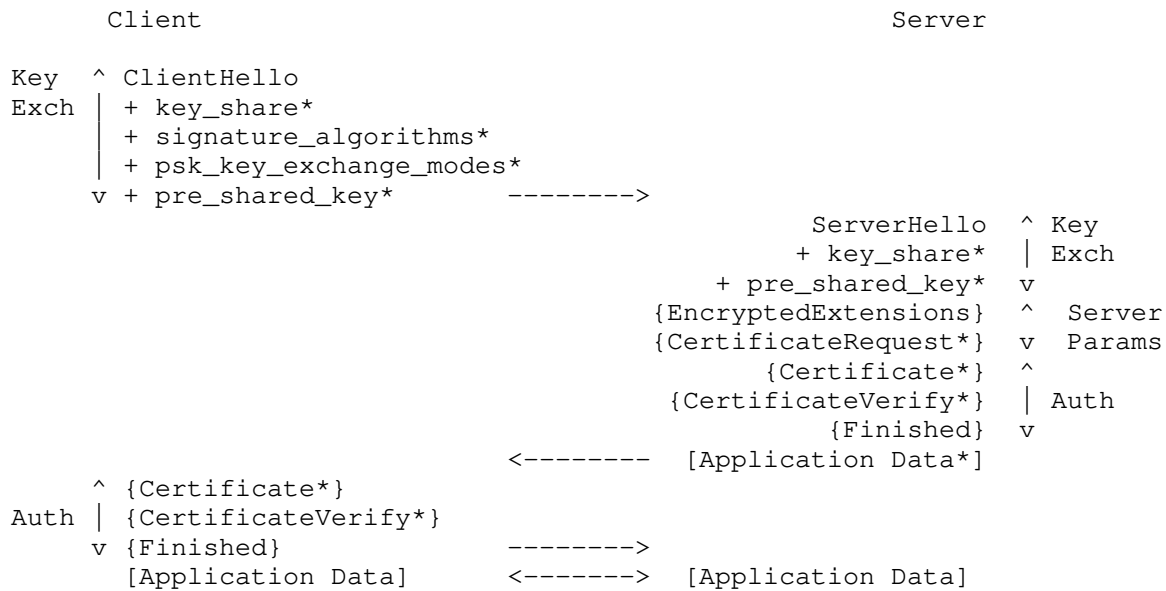
Figure 5: EAP-TLS1.2-IBS example

5.2. EAP TLS1.3 Client and Server Handshake Behavior

TLS1.3 defined in RFC8446, as TLS 1.3 is not directly compatible with previous versions, all versions of TLS incorporate a versioning mechanism which allows clients and servers to interoperably negotiate a common version if one is supported by both peers. When making the discussion on EAP-TLS using raw public keys we also make a difference with TLS1.2, This section is for EAP-TLS1.3 handshake behavior using raw public keys and give example for EAP-TLS-IBS.

5.2.1. TLS1.3 handshake

TLS1.3 is more secure than TLS1.2 in preventing eavesdropping, tampering, and message forgery. The handshake can be thought of having three phases: Key Exchange, Server Parameters and Authentication. The message flow for full TLS handshake is as follows.



+ Indicates noteworthy extensions sent in the previously noted message.

* Indicates optional or situation-dependent messages/extensions that are not always sent.

{ } Indicates messages protected using keys derived from a [sender]_handshake_traffic_secret.

[] Indicates messages protected using keys derived from [sender]_application_traffic_secret_N.

Figure 6: Message Flow for Full TLS1.3 Handshake

5.2.2. EAP-TLS1.3 handshake procedure

EAP-TLS mutual authentication in the case of TLS1.3. defined in the draft-ietf-emu-eap-tls13. TLS 1.3 provides significantly improved security, privacy, and reduced latency when compared to earlier versions of TLS. EAP-TLS with TLS 1.3 further improves security and privacy by mandating use of privacy and revocation checking.

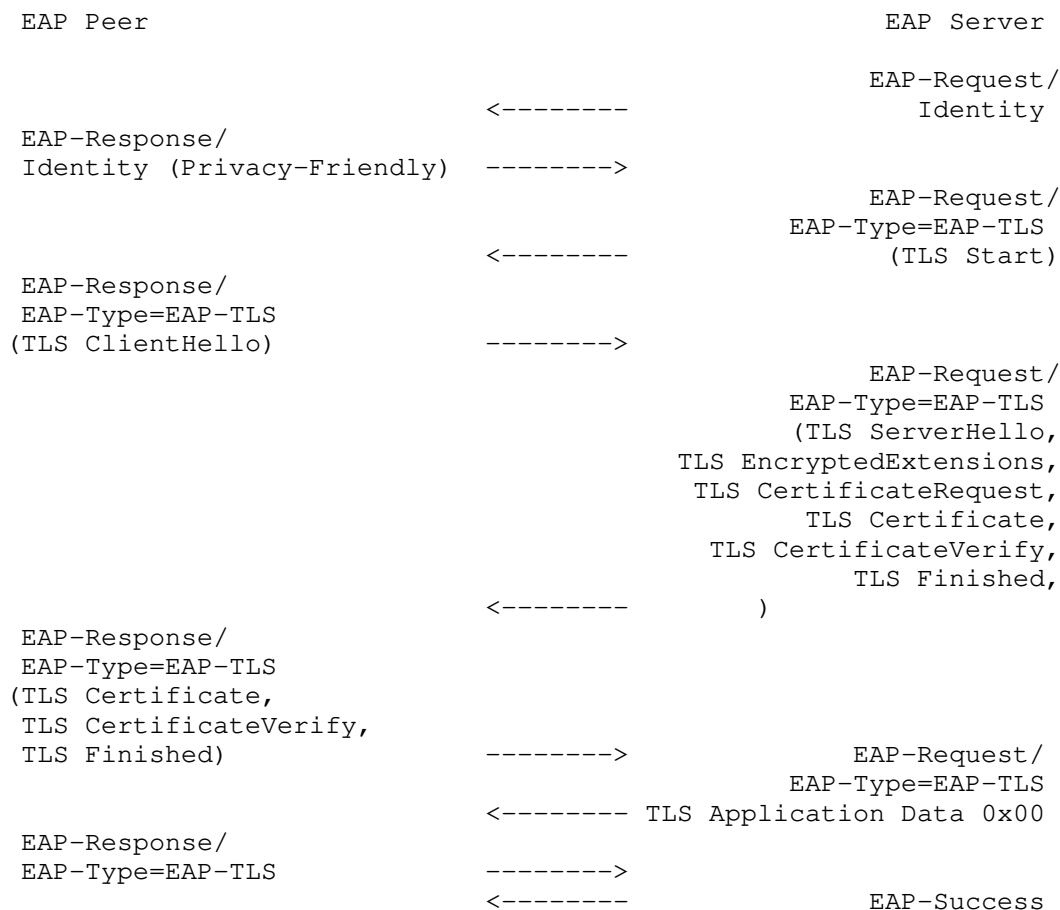


Figure 7: EAP-TLS mutual authentication with TLS1.3 handshake

5.2.3. raw public keys EAP-TLS1.3 exchange

This section describes EAP-TLS1.3 extend using raw public keys, the procedures is as follows, both client and server have the extension "key_share", the "key_share" extension contains the endpoint's cryptographic parameters. the "signature_algorithm" extension contains the signature algorithm and hash algorithms the client and server can support for the new signature algorithms specific to the IBS algorithms. When IBS is chosen as signature algorithm, the server need to indicated the required IBS signature algorithms int the signature_algorithm extension within the CertificateRequest.

<p>Authenticating Peer</p> <p>-----</p> <p>EAP-Response/ Identity (MyID) -></p> <p>EAP-Response/ EAP-Type=EAP-TLS (TLS client_hello +key_share +signature_algorithm server_certificate_type, client_certificate_type)-></p> <p>EAP-Response/ EAP-Type=EAP-TLS ({certificate} {CertificateVerify} {Finished}) -></p> <p>EAP-Response/ EAP-Type=EAP-TLS--></p>	<p>EAP server</p> <p>-----</p> <p><- EAP-Request/ Identity</p> <p><- EAP-Request/ EAP-Type=EAP-TLS (TLS Start)</p> <p><- EAP-Request/ EAP-Type=EAP-TLS (TLS server_hello, +key_share {EncryptedExtensions} {client_certificate_type} {server_certificate_type} {certificate} {CertificateVerify} {certificateRequest} {Finished})</p> <p>EAP-Request/ EAP-Type=EAP-TLS <--TLS Application Data 0x00</p> <p><- EAP-Success</p>
---	---

Figure 8: EAP-TLS1.3 authentication procedure with raw public keys

5.2.4. EAP-TLS1.3-IBS example

When the EAP server receives the client hello, it processes the message. Since it has an ECCSI raw public key from the KMS, it indicates that it agrees to use ECCSI and provides an ECCSI key by placing the SubjectPublicKeyInfo structure into the Certificate payload back to the client, including the OID, the identity of server, ServerID, which is the public key of server also, and hash value of KMS public parameters. The client_certificate_type indicates that the TLS server accepts raw public key. The TLS server demands client authentication, and therefore includes a certificate_request, which requires the client to use eccsi_sha256 for signature. A signature value based on the eccsi_sha256 algorithm is carried in the CertificateVerify. The client, which has an ECCSI key, returns its ECCSI public key in the Certificate payload to the server, which includes an OID for the ECCSI signature. The example of EAP-TLS1.3-IBS is as follows:


```

Authenticating Peer
-----

EAP-Response/
Identity (MyID) ->

EAP-Response/
EAP-Type=EAP-TLS
(TLS client_hello
signature_algorithm = (eccsi_sha256)
server_certificate_type = (RawPublicKey)
client_certificate_type = (RawPublicKey))->

EAP-Response/
EAP-Type=EAP-TLS
(TLS server_hello,
+key_share
{client_certificate_type = RawPublicKey}
{server_certificate_type = RawPublicKey}
{certificate = (1.3.6.1.5.5.7.6.29, hash
value of ECCSIPublicParameters,
serverID)})
{certificate_request = (eccsi_sha256)}
{certificate_verify = {ECCSI-Sig-Value}}
{Finished}

)

EAP-Response/
EAP-Type=EAP-TLS
({certificate = ((1.3.6.1.5.5.7.6.29,
hash value of ECCSIPublicParameters),
ClientID)},
{certificate_verify = (ECCSI-Sig-Value)},
{Finished})
---->

EAP-Request/
EAP-Type=EAP-TLS
<----TLS Application Data 0x00)

EAP-Response/
EAP-Type=EAP-TLS----->

<---- EAP-Success

```

Figure 9: EAP-TLS1.3-IBS example

6. Security Considerations

TBD

7. IANA Considerations

This document registers the following item in the "Method Types" registry under the "extensible Authentication Protocol (EAP) Registry" heading.

Value	Description
TBD	EAP-TLS-IBS

8. Acknowledgement

TBD

9. References

9.1. Normative References

- [RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP)", RFC 3748, DOI 10.17487/RFC3748, June 2004, <<https://www.rfc-editor.org/info/rfc3748>>.
- [RFC5216] Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, DOI 10.17487/RFC5216, March 2008, <<https://www.rfc-editor.org/info/rfc5216>>.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, DOI 10.17487/RFC5246, August 2008, <<https://www.rfc-editor.org/info/rfc5246>>.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.

[RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.

9.2. Informative references

[RFC6507] Groves, M., "Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)", RFC 6507, DOI 10.17487/RFC6507, February 2012, <<https://www.rfc-editor.org/info/rfc6507>>.

Authors' Addresses

Meiling Chen (editor)
China Mobile
32, Xuanwumen West
BeiJing, BeiJing 100053
China

Email: chenmeiling@chinamobile.com

Li Su
China Mobile
32, Xuanwumen West
BeiJing 100053
China

Email: suli@chinamobile.com

Haiguang Wang
Huawei International Pte. Ltd.
11 North Buona Vista Dr, #17-08
Singapore 138589
SG

Phone: +65 6825 4200
Email: wang.haiguang1@huawei.com

Network Working Group
INTERNET-DRAFT
Updates: 7585
Category: Standards Track
Expires: September 05, 2022

DeKok, Alan
Network RADIUS
5 March 2022

EAP Usability
draft-dekok-emu-eap-usability-01.txt

Abstract

This document defines methods which enable simpler deployment of TLS-based EAP methods. It defines new certificate fields, and uses existing certificate fields in order describe new methods for bootstrapping security. The methods defined here change TLS-based EAP supplicant configuration from a complex and insecure process to one that is automated, and is essentially trivial. These methods are still, however, compatible with existing standards and practices.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on January 12, 2022.

Copyright Notice

Copyright (c) 2022 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. Requirements Language	6
2. Historical Problems	7
2.1. Supplicant Changes over Time	7
2.1.1. Phone Vendor One	7
2.1.2. Phone Vendor Two	8
2.1.3. Operating System Vendor One	9
2.1.4. Operating System Vendor Two	9
2.1.5. Operating System Vendor Three	9
2.2. Problems with Certificates	10
2.2.1. Problematic Use of Certificate Stores	10
2.2.2. Problematic use of key purpose fields	11
2.2.3. Problematic Use of Certificates	12
2.2.4. Obtaining Certificates with the new OIDs	13
3. Principles and Guidelines	15
3.1. Network Configuration Guidelines	15
4. New Recommendations for Certificates with EAP	17
4.1. Comparison to HTTPS	17
4.2. Additional Information Required	18
5. How It Works in Practice	19
5.1. A worked example for EAP-TLS	19
5.1.1. The Problem Statement	19
5.1.2. Obtaining the Certificates	20
5.1.3. Configuring the end user device	21
5.1.4. Obtaining Network Access	22
5.2. Other TLS-based EAP methods	23
5.3. EAP methods which do not use TLS	24
5.4. Other Methods of Provisioning	24
5.5. Trust on First Use Can be Secure	25
5.6. Additional Considerations	26
6. Extending the Solution	28
6.1. Bootstrapping via EST	28
6.1.1. Closing the loop	29
6.2. Bootstrapping via DNS	30
6.2.1. CERT records	30
6.2.2. CERT record labels for Server Certificates	32
6.2.3. CERT record labels for CA Certificates	32
7. Related issues	33
7.1. Provisioning Issues	33
7.1.1. Bootstrapping via a Separate Network	33
7.1.2. Configuration Change is just Refresh	34
7.1.3. Secure versus Insecure Provisioning	35
7.2. Issues related to Security	36
7.2.1. Why id-on-naiRealm	36
7.2.2. Resumption	36
7.2.3. Choosing EAP Types	37

7.2.4. User Experience	37
7.3. Issues related to Certificates	38
7.3.1. Public CA versus Private CA	38
7.3.2. Limitations of public CAs	39
7.3.3. CA Chains	40
7.3.4. Delegated Authentication	41
7.3.5. Identification of Networks	42
7.4. Anti-solutions	42
7.4.1. MDM Products Are not the Solution	42
7.4.2. EST and similar protocols do not solve all of th	44
7.4.3. Captive Portals and Hotspots	45
7.4.4. Fully Anonymous Network Access and Provisioning	45
7.5. id-kp-eapOverLAN May not be sufficient	46
7.6. Guest Networks	46
7.7. Using TLS with protocols other than EAP	48
8. Moving to the new methods	49
8.1. Using the new OIDs	49
8.2. Recommendations for EAP peers and authenticators	50
8.3. Principles and Guidelines	52
9. Security Considerations	53
9.1. On Identities and Service Discovery	53
9.2. Password Hashing and Storage	53
10. IANA Considerations	54
10.1. Key Purpose OIDs	55
10.2. Underscored and Globally Scoped DNS Node Names	55
11. References	55
11.1. Normative References	55
11.2. Informative References	56

1. Introduction

TLS [RFC8446] has been widely deployed, and is used with EAP [RFC3748] and with RADIUS [RFC2865]. Historically, these specifications have been written to define the protocols "on the wire", with minimal description of use-cases and usability. The success of these specifications has been that perhaps a billion devices use EAP. The failure of these specifications is that EAP can be still difficult to use, both for administrators and for end users.

Even with a clear standard, implementations do not always follow the specifications exactly. In some cases implementations do less than what is recommended, which can cause security and inter-operability issues. In other cases, implementors do more than what is recommended, as they have found the specifications insufficient to address practical requirements. In other cases, there is no standard, so implementors make individual choices as to how their implementations work. Even worse, implementors change their implementations over time, to solve problems which are not addressed in the standards.

All of these issues lead to confusion for end users and administrators. These issues lead to decreased security of the protocols, and decreased trust in the protocols.

The result of the above problems is software where many critical aspects of its operation are vendor-defined. This wide variation gives a poor experience for all parties involved, and contributes to decreased security.

This document therefore defines method to enable the simple and easy deployment of TLS-based EAP methods. It defines new certificate fields, and describes how those fields can be used to gain network access easily and securely. The processes it defines are clear and straightforward. The end user experience is understandable, and difficult to get wrong.

That is, this specification removes the need to rely on end users to make security decisions. History shows us that such reliance is misplaced. Instead, we rely on the global certificate system which has proven to work well, along with a few changes to the behavior of EAP systems.

These ideas are not new. [RFC4334] Section 1 says:

Automated selection of client certificates for use with PPP and IEEE 802.1X is highly desirable. By using certificate extensions to identify the intended environment for a particular certificate,

the need for user input is minimized.

We extend the above statements to include server certificates, and to further define automated processes by which TLS-based EAP methods can be configured. In addition, where [RFC4334] describes the "automated selection of client certificates", we invert that use-case to show how certificates can be used to automate network configuration, via a set of simple and clearly defined processes.

The requirements on the client device are simple:

- * it has some kind of network connectivity,
- * it has a root certificates for the web,
- * it knows the domain for which it wishes to authenticate

e.g. "example.com",

- * it knows a username and password at that domain.

With that information, the configuration process can be automated, with essentially zero user input.

The document begins with an overview of the current situation. We describe the problems which have motivated this document. First by showing how the behavior of multiple vendor implementations has changed over time. We then discuss problems with the ways that certificate are handled. We discuss how the standards contradict each other, and how current practices contradict, ignore, or extend the standards in incompatible ways.

The document begins with a worked example, initially just for EAP-TLS, and then showing how the processes described for EAP-TLS can be easily applied to other EAP methods.

We extend the given solution by using DNS and HTTPS to perform initial bootstrapping of the supplicant configuration. We show how supplicants can be configured securely by leveraging the existing trust in the web PKI. This bootstrapping requires no changes to supplicant code or behavior.

We finally summarize the work by giving a set of specific recommendations.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Historical Problems

There have historically been a number of issues with configuring devices for EAP authentication. The overly positive description of this history is that it has resulted in a wide variety of tools and products available to configure EAP on end-user devices. In addition to the wide variety of configuration products, the behavior of native supplicants has also varied widely over time.

These issues point to an underlying problem which has, as yet, been unresolved. Each vendor of configuration products or devices to be configured has largely been performing searches by "trial and error" in order to find the best user experience. The result, of course, has been a frustrating experience for both users and for vendors.

We do not discuss Mobile Device Management (MDM) vendors or vendors of other supplicant configuration products here. Their products are largely tools which use the APIs presented by supplicant vendors, and attempt to hide the complexity from end users.

We also do not discuss the behavior of EAP servers (authenticators) or RADIUS servers. The issues seen by supplicants are largely related to user experience, and have little effect on the "on the wire" protocol. As such, RADIUS servers have been required to make correspondingly fewer changes to their implementations. In addition, RADIUS servers are generally designed to be complex, with complex policies. These policies enable their administrators to change the behavior of the software without resorting to product upgrades.

As a result, we focus initially on supplicants, how their behavior has changed over time, and the publicly visible effects of those changes.

2.1. Supplicant Changes over Time

In this section we discuss how the behavior of multiple supplicants has changed over time. We do not name the vendors of these supplicants, as there is no need to blame them for being unable to solve an industry-wide problem.

2.1.1. Phone Vendor One

This vendor has been gradually disabling the supplicant configuration API. Instead, configuration product vendors are able to influence the user interface with suggested prompts at different points in the authentication framework.

It appears that the goal is to give the user control over the

network. A related goal is to require the user's consent before making changes to the network.

The system also treats SSIDs as defining a location, even though SSIDs are not inherently location-specific. This mislabelling means that users' are shown prompts about location, when in fact the operation being performed is connecting to an SSID.

The effect of these issues is that the user is unable to meaningfully consent. There may insufficient information available, the available information may be meaningless to the end user, or the information given to the end user is simply wrong.

This vendor has changed how manual connections are managed over time. The user is not always prompted, but the systems behavior has gone through the following changes to connection requirements:

- * do not perform certificate validation
- * validate that the root CA is in the system certificate store
- * validate that the root CA is in the WiFi certificate store
- * require a DNS name for the RADIUS server.

None of these solutions are optimal.

2.1.2. Phone Vendor Two

This vendor has a standard format for WiFi configuration files. The user can manually install the configuration, but that configuration is not active until an additional manual step is performed to enable it.

A standard configuration is useful, but the configuration file is not typically signed, even though that is supported. It appears that the the manual enablement step is an attempt to work around the lack of authentication for the configuration files.

As was seen in the previous section, the end user has no meaningful information about the configuration. This lack of information means that the user is conditioned to simply accept the configuration and enable it, without paying attention to its contents.

This vendor does, however, provide a robust API. This API permits a rich ecosystem of MDM products which automate the configuration of end-user devices.

2.1.3. Operating System Vendor One

This vendor has been gradually disabling their WLAN API. The remaining APIs permit MDM solutions to associate a user identity with a particular network. However, there is no ability to set a root CA or a RADIUS server DNS host name.

When the user connects to the network, a prompt is shown which asks the user if the server is valid. The server certificate is presented to the user, but the user has no information about which root CA is acceptable or not. Instead, the user is shown fields from an unknown certificate, and is asked to validate that the certificate is acceptable.

Again, the user has insufficient information to meaningfully consent. Which again means that the only course of action is to mindlessly click on "accept".

2.1.4. Operating System Vendor Two

This vendor retains a rich WLAN API, but has removed the ability to configure specific users. Instead, only system-wide profiles can be set.

This vendor provides for easy installation of additional root CAs, but those root CAs are permitted to be used for any purpose. Which means that a malicious private CA can issue HTTPS certificates for any domain. These fake certificates will be silently accepted by the systems browser as being valid. The user will be unable to distinguish malicious sites presenting those fake certificates from the genuine domains.

It is difficult to overstate the negative security impact of that process.

An MDM product adding a private CA generally requires a privileged account to install the CA. A user can install a CA manually, but the operating system will show the user large amounts of text in order to warn the user about the security issues of this process. The user, of course, has no way of understanding many of these warnings, and is left again to mindlessly click on "accept".

2.1.5. Operating System Vendor Three

This vendor retains a rich WLAN API, and a number of tools by which network configuration can be performed. These tools are widely used by MDM vendors to automate the configuration of networks.

However, the end user experience is still complex. The user still must manually select each individual parameter from multiple options. This capability gives the user a substantial amount of control over the process, but does not provide any more information than is available in other operating systems.

This process of allowing the end user to configure everything is useful for experienced and knowledgeable users. However, it leaves the average user with a bewildering set of choices, most of which are meaningless or opaque. The user is then left to mindlessly follow online guides, which may or may not work, and which do not give the user enough information to give informed consent for the actions that are being taken.

2.2. Problems with Certificates

The widely (and wildly) changing behavior of supplicant software is not the end of the story, however. There are a large number of problems related to the use and abuse of certificates. These issues are discussed in more detail in this section.

2.2.1. Problematic Use of Certificate Stores

Some EAP peers use a different certificate store for EAP than for other (e.g. web) applications. In practice, the use-case of "downloading video from a known source" is substantially different from the use-case of "sending authentication credentials to a known destination". As such, the certificate stores should be different for these two use-cases.

When a CA is allowed to be used for EAP, then the implication is that all certificates signed by that CA are allowed to be used for EAP. This result is not secure. It permits attackers to get a valid server certificate from a public CA, and then to set up an EAP server. Naive EAP peers will then send user credentials to the malicious server. Worse, there is no general way for any third-party to detect that this impersonation has happened. It is only visible to EAP peers who are in a small geographic area.

Tests have shown that in a university environment, up to fifty percent of EAP peers will connect to a malicious SSID without checking the CA or server certificate. In effect, these peers will send authentication credentials to anyone who asks.

The security problems associated with such behavior cannot be overstated.

However, not all EAP peers uses separate certificate stores. Using one certificate store is less of an issue when "self signed" or

"private" CAs are used. The use of private CAs for EAP means that the EAP system is now more secure. However, the addition of private CAs to a global certificate store means that those private CAs can now issue certificates for well-known public web sites. The possibility of such forgery has made it difficult for MDM vendors or site administrators to create and use private CAs.

As such, we reiterate that the certificate stores SHOULD be different for these each application or use-case. Where the system cannot tolerate multiple different stores, it SHOULD at least mark each CA certificate with an annotation as to its intended purpose. While there is a "key purpose" field defined for certificates, we will see that this field is not always suitable for differentiating certificate purposes.

2.2.2. Problematic use of key purpose fields

[RFC5216] Section 5.3 makes the following recommendations about the certificate used by the EAP-TLS server:

In the case of the EAP-TLS peer, this involves ensuring that the certificate presented by the EAP-TLS server was intended to be used as a server certificate. Implementations SHOULD use the Extended Key Usage (see Section 4.2.1.13 of RFC3280) extension and ensure that at least one of the following is true:

- 1) The certificate issuer included no Extended Key Usage identifiers in the certificate.
- 2) The issuer included the anyExtendedKeyUsage identifier in the certificate ...
- 3) The issuer included the id-kp-serverAuth identifier in the certificate ...

These recommendations have also been used in EAP-TLS [EAPTLS], EAP-TTLS [RFC5281], PEAP [PEAP] and [MSPEAP], EAP-FAST [RFC4851], and TEAP [RFC7170].

We first note that this document extends and strengthens the suggestion that systems ensure "that the certificate presented by the EAP-TLS server was intended to be used as a server certificate." We also extend this recommendation to client certificates, further strengthening the security around TLS-based EAP methods.

However there is an issue with the [RFC5216] recommendations. These recommendations appear to be in direct conflict with the definition of id-kp-serverAuth from [RFC5280] 4.2.1.12, and of the requirements for its usage:

If the extension is present, then the certificate MUST only be

used for one of the purposes indicated. If multiple purposes are indicated the application need not recognize all purposes indicated, as long as the intended purpose is present.

... If a certificate contains both a key usage extension and an extended key usage extension, then both extensions MUST be processed independently and the certificate MUST only be used for a purpose consistent with both extensions. If there is no purpose consistent with both extensions, then the certificate MUST NOT be used for any purpose.

```
...
id-kp-serverAuth          OBJECT IDENTIFIER ::= { id-kp 1 }
-- TLS WWW server authentication
```

This definition shows that id-kp-serverAuth is intended for "WWW server" usage, which is in conflict with how it is defined in [RFC5216]. Similar issues exist for the id-kp-clientAuth OID, in that it is intended for "WWW client", which is not correct for EAP.

It appears that the recommendations made in [RFC5216] were taken from [RFC2716], and were made for entirely practical reasons. The desire was for users of EAP to be able to obtain certificates from public root CAs. However, those root CAs could not (or would not) issue certificates which contained OIDs other than id-kp-serverAuth. Therefore as work around, [RFC5216] Section 5.3 allowed for a wide variety of EKUs to be used in server certificates. These certificates could then come from private CAs, or from publicly known root CAs.

We believe that the long-term correct solution is to define and use additional key purpose OIDs. These key purpose OIDs can initially be used by EAP implementations along with private CAs. As support for these OIDs becomes more widely available, it may be possible for public CAs to issue purpose-specific certificates.

The problematic use of id-kp-serverAuth has had a number of impacts, past the issue of contradictory specifications. These impacts result in certificates being used in problematic ways, which we discuss below.

2.2.3. Problematic Use of Certificates

The current workarounds to the contradictions in the specifications are two-fold. One, is simply to get certificates with id-kp-serverAuth from a public CA, and hope that using it for EAP either is acceptable, or that it is not noticed. Another is to use a self-signed CA. Both work-arounds have problems.

Many people prefer to use public CAs, as they are seen as "better"

than self-signed CAs. However, using a public CA likely means violating the terms of use of that CA. Which means that the network continues to work so long as this mis-use is not reported.

It can be useful instead to use private CA. A private CA can add id-kp-serverAuth without violating any terms of use, or it can omit the key purpose OIDs, or it can add custom key purpose OIDs.

However, in addition to the problems noted in the earlier section, private CAs are not installed by default in client devices. This limitation means that these CAs must be provisioned somehow. As seen above, these provisioning methods can be complex and prone to failure.

As such, there is no simple, easy, way for administrators to both obtain and provision certificates for use with EAP.

We also note that these OIDs are used not only for EAP, but that they are also used for other public-facing TLS services such as XMPP, SMTPS, LDAPS, etc. Those protocols may have similar issues with alleged mis-use of these OIDs. If these use-cases are forbidden user CAB guidelines, then this would seem to be a serious problem with the global certificate framework.

We leave the solution of these issues as a point of discussion for the wider Internet community.

2.2.4. Obtaining Certificates with the new OIDs

Most CAs currently offer limited support for non-"WWW" OIDs in certificates. In many cases, the Certificate Signing Request (CSR) supplied by the customer is (in practice) used only as a vague suggestion. While the CA generally does not add any fields, it may drop fields that it does not recognize or support. Or, the CA may discard the CSR entirely.

[CAB] Section 7.1.2.3 makes it difficult for existing CAs to issue client certificates which contain the new OIDs:

Either the value id-kp-serverAuth [RFC5280] or id-kp-clientAuth [RFC5280] or both values MUST be present. id-kp-emailProtection [RFC5280] MAY be present. Other values SHOULD NOT be present. The value anyExtendedKeyUsage MUST NOT be present.

Further, the requirements of [CAB] Section 7.1.2.4 essentially forbids CAs from signing certificates which are intended for use with EAP:

CAs SHALL NOT issue a Certificate with:

- a. Extensions that do not apply in the context of the public Internet (such as an extKeyUsage value for a service that is only valid in the context of a privately managed network), unless:

None of the reasons listed after "unless" allow for CAs to issue certificates for use with EAP in a privately managed network.

This behavior by CAs makes it difficult in practice, if not impossible, to obtain non-"WWW" certificates from a public CA.

The suggestion given here is to simply use self-signed CAs. This suggestion is not always practical.

It is possible to define CAs for "walled gardens" with a private CA. One example is certificates used internally in an organization, or in a group such as Eduroam [RFC7593] and [EDUROAM]. In those situations, the members requesting certificates have already validated, and there is already a legal framework in place to protect the parties.

Other suggestions have been that it is relatively simple to set up a new CA, with new procedures and requirements. Given the regulatory requirements around CAs, it appears that new public-facing CAs have to be well funded. i.e. requiring many millions of dollars. It is therefore difficult, if not impossible, for small public-facing CAs to be created.

The goal of this document is to permit better behavior for EAP peers and authenticators. If this specification is widely deployed, then there may be sufficient demand for CAs to offer new certificates which are marked as fit for their intended purpose.

3. Principles and Guidelines

After analysis of the historical practices and standards for EAP, we came to a set of guidelines which are outlined in this section. Application of these guidelines drove the rest of the specification which we define herein.

3.1. Network Configuration Guidelines

It is RECOMMENDED that the guidelines given below are followed when developing new network configuration standards and methods:

- * Automated provisioning is strongly preferred to manual provisioning. We define "automated provisioning" as provisioning which is performed via software, with little or no user intervention. i.e. "zero touch" for end users. Automation minimizes the possibility for end users to create broken or insecure configurations.
- * Manual provisioning should be limited to "Trust on first use" (ToFU), and cached or "pinned" after that. That is, manual provisioning should be limited to allowing a user to approve validation decisions which have been made by the system.
- * Relying on end users to manually configure complex systems is strongly discouraged. Complex systems are difficult to configure, and improperly configured systems create many issues related to security, usability, and network access.
- * Configuration should be "pinned" in order to permit systems to detect and prevent unauthorized changes, and to detect malicious networks which claim to be updated versions of the true network.
- * The identity and role of both parties should be exchanged, and verified. In practice, this suggestion often means that TLS-based EAP methods are preferred to ones which only do name / password credential verification. TLS allows both parties to exchange certificates which demonstrate their identities,
- * The previous requirement usually means that the both parties know which RFC 7542 NAI realm is being used. This realm serves a similar purpose to the the DNS host name used in other TLS-based protocols such as HTTPS. As such, similar methods can be used to validate certificate authenticity. This NAI realm is contained in an id-on-nairealm field, as defined in [RFC7585] Section 2.2
- * For TLS-based EAP methods, trust should be based on a certification authority (CA), which signs certificates for a

particular realm. If the CA is trusted, then everything derived from that CA can be trusted. If the CA is not trusted, then it is impossible to trust anything derived from an untrusted CA.

- * CAs should also be associated with permitted uses. For example, a root CA which is trusted for web surfing is not necessarily trusted for use with EAP authentication. In practice this means either having separate certificate stores for different purposes, or annotating root certificates with their permitted uses. Again, this process should be automated, because end users have no way to verify which CA is valid for what purpose.

We believe that these recommendations are correct, simple, practical, and will improve security and usability for all participants in EAP. We show that there is a clear upgrade path from current behavior to better behavior. Each step of that upgrade path is simple, and involves minimal change for end users or administrators.

4. New Recommendations for Certificates with EAP

The first step towards a complete solution is to define new OIDs. These OIDs indicate that certificates are intended for use with an EAP server, or an EAP peer.

The following key usage purposes are defined within id-kp:

```
id-kp OBJECT IDENTIFIER ::= { id-pkix 3 }
```

```
id-kp-eapServer          OBJECT IDENTIFIER ::= { id-kp TBD-1 }  
-- TLS EAP server authentication  
-- Key usage bits that may be consistent: digitalSignature,  
-- keyEncipherment or keyAgreement
```

```
id-kp-eapClient          OBJECT IDENTIFIER ::= { id-kp TBD-2 }  
-- TLS EAP peer authentication  
-- Key usage bits that may be consistent: digitalSignature,  
-- and/or keyAgreement
```

These EKU fields mirror id-kp-serverAuth, and id-kp-clientAuth, respectively.

We also rely on id-on-naiRealm, as defined in [RFC7585] Section 2.2. This field contains the NAI realm [RFC7542] in which the user has an identity, and for which the EAP server is performing authentication.

4.1. Comparison to HTTPS

We can further explain how these fields help EAP by comparison with how certificates are used for HTTPS [RFC2818]:

- * HTTPS uses id-kp-serverAuth to indicate that a certificate is permitted to be used with an HTTPS server. We define id-kp-eapServer which indicates that a certificate is permitted to be used with an EAP server.
- * HTTPS uses id-kp-clientAuth to indicate that a certificate is permitted to be used with an HTTPS client. We define id-kp-eapClient which indicates that a certificate is permitted to be used with an EAP client.
- * HTTPS uses id-ce-subjectAltName with dNSName [RFC5280] to contain the DNS name of the server to which the client is connecting. We use id-on-naiRealm [RFC7585] to indicate the NAI realm of the server to which the

client is authenticating.

4.2. Additional Information Required

When combined with a clearly defined process, the above definitions allow devices to use TLS-based EAP methods with no more complexity than is seen when browsing the web. That is, in many situations, all the end device needs is the following:

* network access (trusted or not) * an account in a domain, e.g.
"user@example.com" * one or more trusted root CAs from the web PKI.

This information can be used to securely obtain network access. The procedures outlined here work with both public CAs and private CAs.

We will first describe how these fields can be used to make EAP authentication easier to use. Once we have described a worked example using these fields, we will show how to extend the solution to solve the remaining open issues.

5. How It Works in Practice

In this section we provide a worked example for EAP-TLS. This discussion uses EAP-TLS as an example, but the methods discussed here are not limited to that use-case. Describing a specific EAP method allows us to discuss every aspect of the proposal, without worrying about how similar methods are applicable to different situations.

We expand the discussion later in this section to show how these methods are applicable to other TLS-based EAP methods.

5.1. A worked example for EAP-TLS

We explain how this specification works via an example using EAP-TLS. We start off with the problem statement, then how the certificates might be obtained, then how the EAP peer is configured using information in the certificates, and finally how the device obtains network access.

5.1.1. The Problem Statement

For the initial worked example, we assume that we are trying to solve the limited problem of an end user who has a WiFi enabled device such as a laptop. The user wishes to use that device to get online, via the simplest possible method. There is an administrator who also wishes to get that user online, and wishes to configure the end user device to do EAP-TLS.

We also presume that there are some additional pieces of the solution, as follows:

- * There may be a Wireless LAN (WLAN) System Service identifier (SSID) which is used for WiFi authentication. This information can be realized in the certificate field `id-pe-wlanSSID`, as defined in [RFC4334] Section 3.
- * All parties know which NAI realm [RFC7542] is being being used. For example, an individual may work for a company "example.com". This information is placed into the certificate field `id-on-naiRealm`, as defined in [RFC7585] Section 2.2.
- * We use the new EKU field `id-kp-eapServer`. This field indicates that a certificate is intended to be used as a server certificate within EAP.
- * We use the new EKU field `id-kp-eapClient`. This field indicates that a certificate is intended to be used as a client certificate within EAP.

Knowing the name of the SSID is necessary, but perhaps not sufficient. Some environments may have additional security requirements, such as mandating that only WPA3-192-bit connections may be used. This information should likely go into another field of the certificate. For the purposes of this document, we will assume that knowing the SSID name is acceptable.

EAP methods are also used to authenticate users when SSIDs are not available (e.g. wired 802.1X), the use of `id-pe-wlanSSID` is recommended, but is not required. For the purpose of this section, we assume that WiFi access is being configured. Later discussion will show how the methods outlined here can be applied to other forms of network access.

As we will see below, this information is sufficient to configure EAP-TLS securely, and with minimal effort by the end user.

5.1.2. Obtaining the Certificates

The administrator begins by obtaining a server certificate from a root CA. This CA can be public or private. The only requirement is that the CA is willing to sign certificates which contain an `id-on-naiRealm` field, and which also contain the `id-kp-eapServer` field which indicates that this certificate is suitable for use with EAP. The rest of the fields, and the validation process for those fields, can be identical to the processes used today.

The use of the new EKU fields here is intended to illustrate the end goal of simplifying deployments. As we will see later, there are intermediate steps which do not require the new EKU fields.

The administrator also obtains a client certificate, which will be given to the end user for installation on the device. This client certificate is issued via a hierarchy which ends in a known root CA. The rest of the certificate hierarchy here is not important to this example. We only presume that it exists; that it is valid; and therefore that it is trusted.

The client certificate contains an `id-on-naiRealm` field, which has the same NAI realm as seen in the server certificate. The client certificate also contains the `id-kp-eapClient` field which indicates that this certificate is suitable for use with EAP. The client certificate may also contain a `id-pe-wlanSSID` field, though this is not required.

We presume that the client certificate also has an associated private key, and that this key is encrypted using a password.

We now have enough information to configure the end user device.

5.1.3. Configuring the end user device

The administrator configures the end device with the client certificate, along with its associated private key and password. For the purposes of this example, it is not important how the device obtains that information. As noted earlier, the distribution problem will be solved later in this document by extending the solution.

The configuration may also include the entire certificate chain up to, and including, the root CA. Not all of these certificates are always required, as they can be exchanged in the initial EAP-TLS session.

We note here that the configuration of the end user device is not embodied in a downloadable executable, or in a vendor-specific configuration file. Instead, the configuration is given in the form of standardized certificates which have been marked up to express their intended usage.

When the end user receives the client certificate (and possibly certificate chain), it can be installed on the device immediately. When the certificate is installed, it causes a number of additional steps to be taken by the device which is using that certificate. These steps are new to EAP peers, and are not performed today.

First, the device determines that this certificate contains `id-kp-eapClient`, which indicates that this certificate is intended to be used for EAP. If there is a certificate chain, then those certificates can be saved. If the client certificate contains `id-pe-wlanSSID`, then the device uses that information to configure itself so that it will connect to the named SSID, and to perform EAP-TLS authentication using this client certificate.

This process is similar to that outlined in [RFC4334]. The changes we make over that specification are new extensions to the certificates, and additional steps which mandate new behavior.

Finally, if there is a root CA in the certificate chain, that the root CA is installed. The device also annotates this root CA (pre-existing or otherwise) as being trusted to issue certificates for use with EAP. As we will see, the device can also require the EAP server certificate to contain `id-kp-eapServer`, along with an `id-on-naiRealm` value which matches the `id-on-naiRealm` which is in the client certificate.

At this point, the end user device is fully configured for using EAP-

TLS with a particular SSID.

5.1.4. Obtaining Network Access

When the end user device wishes to obtain network access, it can for the most part follow the methods used prior to the publication of this specification. There are, of course, a few changes which simplify the process and make it more secure.

For privacy, the device uses an anonymous identifier in the EAP Response Identity field. This identifier is the NAI realm which is taken from the id-on-naiRealm field of the client certificate. Taking the NAI realm from the client certificate means that there is no need for the user to configure the publicly visible EAP Response Identity. This usage also provides for the anonymity required in [EAPTLS].

In order to provide for privacy of the client certificate, TLS 1.3 is used. Older versions of TLS are NOT RECOMMENDED.

When the EAP-TLS connection is established, the device verifies that the server certificate which is presented also contains a id-on-naiRealm field, which matches the value in the client certificate. This validation is similar to the validation of DNS names performed by web browsers when accessing HTTPS sites. However, as DNS is not available during EAP authentication, the id-on-naiRealm field is used instead to validate the server certificate.

For clarity, we repeat the instructions in [RFC7585] Section 2.2, for matching the NAI realm:

The comparison of an NAIRealm to the NAI realm as derived from user input with this algorithm is a byte-by-byte comparison, except for the optional leftmost dot-separated part of the value whose content is a single "*" character; such labels match all strings in the same dot-separated part of the NAI realm. If at least one of the sAN:otherName:NAIRealm values match the NAI realm, the server is considered authorized; if none match, the server is considered unauthorized.

Since multiple names and multiple name forms may occur in the subjectAltName extension, an arbitrary number of NAIRealms can be specified in a certificate.

The device also verifies that the server certificate also contains the id-kp-eapServer field. It verifies that the certificate is signed by a root CA which is annotated as being permitted for use with an EAP server. If these verification steps fail, then the

client stops the authentication process, as it has determined that the network is not trusted.

If all of these verification steps pass, then the end user device can trust the EAP server, and be authenticated to the network.

We note here that from the perspective of the end user, the only actions which have been performed have been (1) to install a certificate, and (2) to enter a password for that certificate. This process is substantially simpler than most WiFi configuration processes used today. This process is also likely to be easy to follow for most users.

5.2. Other TLS-based EAP methods

While the above example discusses EAP-TLS, it is easily extensible to any other TLS-based EAP methods. Instead of distributing a client certificate to end users, the administrator can distribute a server certificate and/or a root CA which is intended for use with EAP. For simplicity, the server certificate should also contain id-pe-wlanSSID, which informs the client as to which SSID(s) should be used for authentication.

We reiterate that the public EAP Response Identity used should always be in the form "@realm", as per [EAPTLS] Section 2.1.7. The user's full identity should only be sent inside of the TLS tunnel. We also recommend that the inner authentication methods use the full identity of "user@realm", and not just the "user" portion.

The end device then follows the same process to configure the SSID for authentication, to mark up the SSID as being used with a particular NAI realm, and to annotate the root CA as being permitted for use with EAP. Again, having an SSID here simply makes this example clearer, this specification does not mandate its use, and this specification is applicable to any type of network access which uses EAP.

When the device connects, it does not need to verify that the server certificate being presented is the same as was used for configuration. Instead, the device simply has to "pin" the combination of SSID, NAI realm, and root CA. This pinning allows for flexibility in accepting other server certificates, while preventing down-grade attacks which attempt to supply different root CAs for that NAI realm. This pinning means that the device associates the SSID and NAI realm with a particular root CA, and then does not permit that NAI realm to authenticate to an EAP server which does not use the same permitted root CA.

The only requirement on the new server certificate is that it has to match the same criteria as outlined in the previous section. That is, the certificate must contain `id-kp-eapServer`, it must have a matching `id-on-naiRealm`, and it must be signed by a root CA which the supplicant has permitted to be used with EAP.

The device can then safely send authentication credentials inside of the TLS tunnel. This process is substantially similar to that used to log into an HTTPS enabled web site. The only difference here is that the device must associate the user's credentials with EAP and an NAI realm, instead of with the web and a DNS host name.

5.3. EAP methods which do not use TLS

Unfortunately, the methods outlined here apply only to TLS-based EAP methods. This limitation is because we are leveraging the TLS certificate format in order to both define additional permitted uses for those certificates, and to inform devices how non-TLS systems should be configured.

This extra information is simply not possible to add to other EAP methods such as EAP-PWD [RFC5931]. Those methods typically authenticate users, but do not provide for carrying additional information. Those methods also generally do not provide for the mutual exchange of identities, and for mutual authentication.

Authentication protocols such as EAP-PWD are simple enough that it is both impossible, and generally unnecessary, for them to use the methods outlined in this specification. That is, the cryptographic guarantees in EAP-PWD ensure that it is always safe to perform EAP-PWD with unknown EAP servers, as there is no possibility for leakage of user credentials. As such, there is less need to verify the identity of the EAP-PWD server.

5.4. Other Methods of Provisioning

The EAP-TLS example described how provisioning was done via an administrator sending certificates to an end user. This process is not always necessary.

For EAP-TLS, [EAPTLS] Section 2.1.5 provides for the protocol to be used without peer authentication. This capability can be leveraged to perform provisioning. All that is needed on the device is for the EAP peer to have a pre-configured root CA, and to know the NAI realm which it belongs to, and which is being used for provisioning.

For the purposes of this section, we assume that the root CA known to the device is willing to issue and sign server certificates which

contain the id-kp-eapServer and id-on-naiRealm fields. As we will see below, this assumption may be difficult to achieve in practice.

When the device connects to a network, it can perform the verification steps outlined above. That is, the server certificate presented has a matching id-on-naiRealm field; the server certificate contains id-kp-eapServer; and that the server certificate is signed by a known root CA. The root CA does not necessarily have to be annotated as being permitted for EAP.

If all of those requirements are satisfied, then the device can obtain limited network access. The device can then leverage normal networking protocols to download provisioning information, which is then used to configure the device. As noted above, this provisioning information needs to be little more than a client certificate.

For example, the device could use Enrolment over Secure Transport (EST) [RFC7030]. It could also use vendor-specific methods.

This process works because every device capable of doing TLS is shipped with a set of known root CAs, which are intended for use with the web. In addition, every end user wishing to connect to a known network is aware of the identity of that network (e.g. "example.com"), and their identity in that network (e.g. "user@example.com").

If the device does not have a root CA configured, as we will see below, it can use the limited authorization network with other protocols such as DNS. The device could use DNS to query a pre-defined SRV record (as with [RFC7585] Section 3). The results of that record could be a "self signed" root CA. Certificates can therefore be obtained over DNS, such as via the methods outlined in [RFC4398].

The only requirement here is that the DNS record be obtained securely (DNSSec or DNS over TLS), otherwise an attacker could forge the response, or replace the root CA in transit.

Other methods are also possible, though not discussed here.

5.5. Trust on First Use Can be Secure

Similar provisioning methods can be used for other TLS-based EAP methods. In those methods, when a device connects to the network, it could prompt the user for a username (with an NAI realm) and password. Then, it could use that information to derive the NAI Realm, and perform the verification steps described previously. The device simply needs to know that it is trying to authenticate to a

specific NAI Realm before verifying the server certificate, and needs to verify that server certificate prior to sending any user identity or authentication credentials to the EAP server.

That is, a device which knows that it is trying to authenticate to a realm "example.com", can then verify that the server certificate contains id-on-naiRealm which matches "example.com". This process is similar to a web browser that wishes to connect to a web site for "example.com". The client already knows that "example.com" is the desired destination, which then means that the client must verify that the site which it connects to has a certificate matching "example.com".

If the device is not configured with any realm, then it has no way of determining whether or not it should trust any EAP server. As such, the use of the NAI realm is a critical component of this specification.

This process is close to "Trust on First Use" (ToFU) provisioning, with minimal knowledge required, and with a high degree of security. From the point of view of the end user, the only actions which have been taken are to select an SSID, and then to enter a name and password. The process outlined here ensures that the user is authenticated to a known and trusted network, and that the EAP peer sends the identity and authentication credentials only to known and trusted networks.

Some EAP methods such as TEAP [RFC7170] support provisioning of end user devices. Since this provisioning information is automatic, it can include additional information not discussed here. The process, however, remains substantially similar. The client can download one or more certificates, and then perform the validation and configuration steps outlined above.

5.6. Additional Considerations

Note that there is no requirement that the device use only the SSID given in the id-pe-wlanSSID field of a certificate. If the device sees an authorized server certificate on a different SSID, then it should proceed with authentication as discussed previously.

However, the EAP server may not permit the client to be authenticated via other SSIDs. It is therefore RECOMMENDED that channel bindings [RFC6677] are used in all EAP methods, in order to inform the server about the clients local environment. Channel bindings also solve problems with supplicants that do MAC address randomization: The real MAC address is sent inside of the TLS tunnel, as part of the channel binding exchange.

Similarly, there is no requirement for the device to "pin" a particular server certificate. If the presented server certificate meets the criteria of known root CA; containing id-kp-eapServer; and matching id-on-naiRealm, then the connection can be trusted. In fact, there is no need to cache the server certificate at all.

6. Extending the Solution

The process described above greatly simplifies the usability of EAP, and its security. We can, however, do better.

The process described above requires changes to both supplicants, and to the systems which issue certificates. These changes are useful, but are not always trivial. Further, the processes still have a bootstrapping problem, which was waved away in Section 2.2.3 during the discussion of the worked example. The bootstrapping problem was somewhat addressed by the use of ToFU provisioning in Section 2.6, but there are still open issues with respect to security and provisioning.

In this section, we describe a few ways in which the remaining issues may be addressed, in order to come up with a complete solution to the problem. We first describe protocols such as EST [RFC7030], and then later how DNS may also be used.

6.1. Bootstrapping via EST

EST [RFC7030] can be used to distribute previously created certificates for CAs and servers. It can also be used by clients to request new client certificates. As there is no distinction in EST between public CAs and private CAs, either can be used. This feature enables EST-capable systems to use the new OIDs defined here.

The use of EST therefore solves the certificate distribution problem which was described earlier in the worked example for EAP-TLS.

However, the requirement here is that the client implements EST, and not all currently do. Another requirement is that EST uses the ".well-known" relative URL from [RFC5785], and both specifications assume implicitly that the base domain is rooted both in the web, and in the top-level subdomain. For example the base URL for "example.com" is given as "www.example.com". While the ".well-known" prefix is capable of being used with other portions of the domain name tree, there is no standard way for a client and server to agree on its location. The location is simply implicit in the protocol.

This limitation is an issue mainly because we wish to use automatic enrollment schemes in "captive portal" or "walled gardens", which have limited network access. It may be possible for a system with limited network access to reach a URI such as "https://www.example.com/.well-known/". However, there is no guarantee that the administrator of the main web server system for a domain is the same as the administrator of the captive portal system.

As a result, we need a way for both the client to discover the URI to use, and for that URI to point to a web sever which is controlled by the administrator of the captive portal system. The solution here is to use DNS.

We define a DNS SRV record which points to the EST server for this NAI realm. In order to provide for the separation of responsibilities, we make this record specific to EAP.

The format of the SRV record is as follows:

```
_est._eap.<naiRealm>
```

An EAP client system can query for this record, and then connect to the ".well-known" service at the provided host, in order to perform EST. This record may point to the main EST server for a domain, or it may point to a separate EST server which is specifically used for EAP. We believe that it is important to make provisions for separation of services such as these, even if this separation is not always used.

We note that the DNS resource record type here is SRV and not URI, as we only need to define the hostname and port for the EST server. The rest of the URI is defined by EST.

We discuss the use of DNS in more detail in the next section.

6.1.1. Closing the loop

An EAP peer which has a username, password, and NAI realm can leverage EST to securely provision client certificates for use with TLS-based EAP methods.

The device first discovers the location of the EST server via the DNS lookup above. It can then download any necessary CAs, using the methods outlined in [RFC7030] Section 4.1 The device then uses the username and password with HTTP basic authentication in order to authenticate itself to the EST server. Finally, the device can request that the EST server create and sign a client certificate, using the methods outlined in [RFC7030] Section 4.2.

One benefit of this method is that the key associated with the client certificate can be generated automatically by the client device, and to some extent hidden from the user. This secrecy ensures that the client certificate is associated with a particular device, and that the user is prevented from copying the certificate to multiple devices.

6.2. Bootstrapping via DNS

We have seen above that the EAP configuration problem can be largely reduced to getting a properly formed certificate onto the device. We show here how to use DNS to bootstrap the certificate installation. This bootstrapping process requires no changes to supplicants or to systems which certificates. Instead, it requires only that:

- * certificates be placed at a well-known URI,
- * this URI is found DNS CERT records [RFC4398],
- * an independent tool downloads the certificates and uses them to configure EAP as described above,
- * the end user or device knows the NAIRealm to which it is supposed to connect.

This process leverages both DNS, and the existing "web" root CA infrastructure in order to securely configure EAP, with minimal manual intervention.

6.2.1. CERT records

The process begins with the administrator obtaining one or more server certificates as described above, and then placing them at a well-known URI. The administrator then adds records to DNS which point to the certificates. The client can then download these server certificates, and configure its EAP system to use these certificates when authenticating to the relevant NAI realm.

For the purpose of this section, we assume that the server certificates are signed by a CA which is already known to the client system. In the next section, we extend this process to downloading new CA certificates.

The information stored in DNS is a CERT record as described in [RFC4398] Section 2. If the DNS record is served over a secure transport such as DNSSEC, DNS over HTTPS, or DNS over TLS, then the record can directly contain the certificate. If the DNS record is served over an insecure transport, then the "type" field MUST be one which contains a URI. These requirements typically mean that value of the "type" field will be (4), for IPKIX, which points to the URL of an X.509 data object.

In order for this process to be secure, the URL MUST be within same domain (NAI realm) as the CERT resource record. The URL MUST be secured with TLS transport. The certificate presented at that URL

MUST be issued by a root CA which is generally already known to the device. The certificate presented at the URL MUST pass all normal HTTPS validation, including that for id-ce-subjectAltName.

That is, when the client accesses a URL pointed to by a CERT record, certificate validation for that access MUST be performed as per [RFC5280]. If any of these validation steps fail, then the client MUST NOT download or use any further data presented by that server.

Further, the contents of the data at that URL MUST be a X.509 certificate.

The downloaded certificate MUST be one with is suitable for TLS-based EAP methods, as described in [EAPTLS]. The client system MUST verify that the server certificate matches the NAI realm which is being used, either via the steps defined here, or via the host name matching defined in [EAPTLS]. If the certificate does not match the NAI realm, then it is discarded and not used for EAP.

An issue is that [EAPTLS] provides for host name matching, but not for NAI realm matching. The two are similar, but not identical. If we recall [RFC7542] Section 2.5, the NAI realm is defined as:

* Realms MUST be of the form that can be registered as a Fully Qualified Domain Name (FQDN) within the DNS.

Certificates of the form supported by [EAPTLS] therefore may still match the given NAI realm.

The downloaded certificate SHOULD also contain id-pe-wlanSSID, in order to inform the device as to which SSID is suggested for network access.

Note that there is no requirement for this server certificate to contain the id-kp-eapServer OID defined here. It is RECOMMENDED to include that OID, but it is not required.

These requirements ensure that devices can leverage the existing web framework to securely download certificates which are to be used for EAP.

The use of insecure transport for DNS is acceptable, as it is only being used to transport a URL, which is itself protected by TLS. The URL validation requirements above ensure that an attacker can only point the device to pre-existing URIs within the given domain, which contain information not under the control of the attacker.

6.2.2. CERT record labels for Server Certificates

The next problem is to define a well-known name for this record. We leverage the [RFC8552] "Underscored" naming of attribute leaves in order to provide for well-known names. We define a series of names, which are all rooted from the NAI realm given by the user.

We note here that the insecurity of plain UDP DNS may, in fact, be of use here. For example, the administrator of a captive portal can modify the captive portal DNS server in order to serve records for the "top level" domain, which not normally be permitted. Since this use of DNS names is only visible from within the captive portal, there is no security impact outside of this limited network.

The format of the CERT record is as follows:

```
_server._cert._eap.<naiRealm>
```

It can be beneficial to use a DNS CERT record instead of Enrolment over Secure Transport (EST) [RFC7030], as our goal here is to simply obtain a pre-existing certificate, and not to generate new certificates. In some cases, the URL provided by DNS can just be the URL of a certificate hosted by an EST server.

In some cases, the downloaded certificate may be from a CA which is not known to the device. For example, when the CA is a "private CA" which is not in the root CA list for web PKI. The next section addresses this issue.

6.2.3. CERT record labels for CA Certificates

This CA certificate may be obtained via EST, as described in [RFC7030] Sections 2.1 and 4.1.2. The device can also look up the CA certificate via a similar process to obtaining the server certificate, by querying for a CERT record at the following name:

```
_ca._cert._eap.<naiRealm>
```

Again, the URL presented here MUST match all of the requirements given earlier for the certificate obtained from the "_server._cert._eap.<naiRealm>" record. The only restriction on the contents and/or format of the downloaded CA certificate is that it MUST permit the previously downloaded server certificate to be verified. If the server certificate cannot be verified using this CA certificate, then both certificates MUST be discarded.

Since this new CA has been downloaded from a trusted source, the CA can also be given limited trust. That is, the downloaded CA SHOULD

be trusted to issue certificates for use with EAP, but only for the NA realm in question. The downloaded CA MUST NOT be trusted for any other use-cases or purposes. This limitation ensures that private CAs cannot be used to spoof public web sites from unrelated organizations.

This requirement in effect mandates implementations to create multiple certificate stores. This limitation is the minimal change required to supplicant implementations in order to support the core of this specification. Every other change suggested here can either be pushed to an auxiliary tool, or can be delayed until a later step.

That is, the user-visible workflow here can be implemented with minimal changes to the supplicant software which implements EAP.

7. Related issues

We discuss related issues in this section. The items discussed here are individually useful to discuss, but do not follow a clear developmental flow. As such, they are placed into a separate section.

7.1. Provisioning Issues

There are a number of issues related to provisioning. We show that there is no need to use a single network for all of the above discovery and configuration. We show that configuration updates are simple, and are no more difficult than repeating the initial provisioning. Finally, we describe why the methods defined herein are significantly more secure than ToFU.

7.1.1. Bootstrapping via a Separate Network

There is no requirement that a particular network provide all of the bootstrapping outlined above via a "guest network". It is also possible to leverage the public Internet in order to bootstrap authentication to a private network which requires EAP authentication.

For example, a mobile phone may be trying to connect to a WiFi SSID, while it also has additional network access via 3G or LTE. There is no requirement here for the WiFi network to provide a guest network with full provisioning capabilities. Instead, the phone can simply try to do unauthenticated EAP-TLS. During the EAP-TLS negotiation, the device will obtain a copy of the server certificate. This certificate should contain id-on-naiRealm.

The device can then use the LTE connection, and the process outlined

above (DNS, EST, etc.) in order to verify that the server certificate is the one which meets all of the criteria necessary for full authentication. Once the server certificate is validated and the device has updated its configuration, it can drop the EAP-TLS connection, and re-authenticate using any TLS-based EAP method.

With this process, the experience for the end user would be little more than:

- * select an SSID,
- * be informed that this is a network associated with a particular NAI realm (i.e. domain),
- * be informed that the network is secure and is trusted,
- * be requested to enter an identity and password within that domain,
- * enter that identity and password, and obtain network access.

This process is little different from using a web browser to navigate to a web site, and ensuring that the green "lock" icon is set for that site.

7.1.2. Configuration Change is just Refresh

Any automatic provisioning scheme has the problem of performing change control. In our case, updating the configuration with a new set of data is largely just repeating the bootstrapping process. The questions then become how often to check for updates, how long to cache configuration, etc.

It is RECOMMENDED that HTTPS servers which provide the certificates described in the previous section set the Cache-Control [RFC7234] directive in the response. It is RECOMMENDED that the "max-age" directive ([RFC7234] Section 5.2.2.8) be used. The value returned SHOULD NOT be less than one day (86400 seconds), and MUST NOT go past the expiry date of the certificate which is being returned.

The supplicant which is retrieving the certificate SHOULD annotate the certificate with the value of the "max-age" directive. The supplicant SHOULD perform the bootstrapping checks again prior to the "max-age" time limit being reached.

Where "max-age" is not returned, the supplicant SHOULD refresh the bootstrapping checks again no more than once per day. It SHOULD track when the certificate was downloaded, and then perform these checks no later than when the certificate is halfway to expiry, taken

from when the supplicant first downloaded the certificate.

When either the root CA or server CA has expired, the supplicant **MUST** NOT use them to obtain network access. It **SHOULD** refresh the certificates at that time. If the certificates are not refreshed, then the relevant configuration **SHOULD** be deleted.

If the refreshed certificate is identical to the previously downloaded certificate, then the supplicant makes no configuration changes other than to update its refresh timers. The supplicant **SHOULD** still perform other certificate validation checks, such as checking for certificate revocation.

If the refreshed certificate has changed, then the supplicant performs all of the validation checks described above. If the tests pass, the new certificate can be used in place of the previous one. Note that there is no need to "tear down" the current network connection if the current certificate is still valid. The new configuration should be used only when the device next requests network access.

As the old credentials are usually still valid, device **SHOULD** keep the old credentials around until such time as it has verified that the new credentials work. If the new credentials do not obtain satisfactory network access, then they should be discarded, and the device should try again not sooner than one day later.

TBD: There should also be fine-grained methods to control when a new configuration is downloaded, and separately when it is applied. For client certificates, we can use the "notBefore" field, which indicates that the certificate is not valid before a particular time.

7.1.3. Secure versus Insecure Provisioning

We now revisit the discussion of ToFU first mentioned above. We note that the process defined here isn't even "trust on first use". Instead, it is leveraging the web PKI in order to get secure, authenticated downloads of non-web certificates. ToFU provision such as used in TEAP is essentially a standardized way to download security configuration from an insecure source.

Our proposal here begins with the naiRealm, and then uses trusted roots and secure protocols to download security configuration from a known and trusted source. While this process is more complex than TEAP, in that it requires DNS and HTTPS, it is also more secure.

7.2. Issues related to Security

We explain why id-on-naiRealm was chosen. We describe some issues related to resumption, and the use of certificates in a multi-server environment. We explain how this solution can be extended to configure individual EAP types. We explain how this solution is applicable when either private or public CAs are used. We conclude by explaining how the user experience offered by this solution creates a simple and clear user experience.

7.2.1. Why id-on-naiRealm

Server certificates used with EAP have historically contained DNS names. This practice is largely because the certificates are "TLS web server" certificates. However, [RFC7585] Section 2.2 explains why DNS names are not appropriate:

Current subjectAltName fields do not semantically allow an NAI realm to be expressed; the field subjectAltName:dNSName is syntactically a good match but would inappropriately conflate DNS names and NAI realm names. Thus, this specification defines a new subjectAltName field to hold either a single NAI realm name or a wildcard name matching a set of NAI realms.

We extend the above requirement to say that the wildcard name MUST be limited to a subset of one realm. That is, a wildcard of "*.example.com" is permitted, but a wildcard of "*", or "*.com", is forbidden.

Although this recommendation was done in the context of RADIUS, this field is exactly what is needed for EAP. The definition is the same (NAI), and the use-cases are the same.

7.2.2. Resumption

[RFC8446] Section 4.6.1 discusses resumption:

Clients MUST only resume if the new SNI value is valid for the server certificate presented in the original session and SHOULD only resume if the SNI value matches the one used in the original session. The latter is a performance optimization: normally, there is no reason to expect that different servers covered by a single certificate would be able to accept each other's tickets; hence, attempting resumption in that case would waste a single-use ticket. If such an indication is provided (externally or by any other means), clients MAY resume with a different SNI value.

-0.3i

Similar requirements apply for EAP, except that clients check id-on-naiRealm instead of using SNI.

Where multiple servers are in an high availability or load-balance group, they SHOULD use the same certificate. Where the same certificate is used, then either the resumption master secret MUST be shared among all systems, or the tickets MUST be accessible to all systems. Preferably by putting them into an external data store.

7.2.3. Choosing EAP Types

We note that this specification does not define which EAP type is used by the supplicant, except implicitly. That is, if the supplicant is given a client certificate, then it is presumed that EAP-TLS is being used. Otherwise, the supplicant should choose some other TLS-based EAP type.

It would be possible to define new OIDs which define a list of EAP types that the EAP server will accept. These OIDs can then be placed in a server certificate, where they can inform the supplicant as to which EAP types should be used.

7.2.4. User Experience

It is RECOMMENDED that the system notify any end user of the configuration changes being performed. It is RECOMMENDED that these notifications give sufficient information to the end user, so that an informed decision can be made. It is RECOMMENDED that these notifications allow the user to stop or cancel the process at any time.

The goal of the user experience described is that it should be little different from using a web browser to navigate to a web site, and ensuring that the green "lock" icon is set for that site.

It is therefore RECOMMENDED that supplicant vendors update their user interfaces to clearly distinguish between "trusted" and "untrusted" network access. A "trusted" network is one which satisfies all of the criteria outlined herein. An "untrusted" network is one which satisfies only some, or none of the criteria outlined here.

It is likely a good idea to update the graphical user interface (GUI) for the supplicant with a green lock / red unlocked icon, similar to that used in web browsers. Further, the GUI should also include the naiRealm which has been verified, as web browsers show the domain name which has been verified. This information is enough to give the user enough information to meaningfully consent to obtaining network

access, and to enter credentials.

That is, if the user sees that the operating-system GUI window says "this site is trusted", and also "you are accessing the example.com domain", then the user can safely enter credentials for that domain.

This workflow is familiar to end users, and has been proven to be at least moderately successful in the web.

7.3. Issues related to Certificates

There are a number of other issues related to certificates, in addition to those which have been raised above.

TBD: more explanation of the trailing sections.

7.3.1. Public CA versus Private CA

Nothing in this specification requires the use of either a public or private CA. Both are possible, and both have issues.

The main issue with using a private CA is that it is not already on the device, and has to be provisioned. While there are many possible methods of provisioning this information, we define here only a few straightforward methods. We hope that the method proposed here (DNS + HTTPS) is clear, and simple for administrators to implement.

There is still the requirement that the client device have new software to obtain this information and call the supplicant API. However, this process is no different than installing custom MDM software.

Private CAs have the benefit of being able to sign certificates with any EKU they desire. These certificates can then be marked with an EKU as being intended for a particular use, and supplicant software can verify these EKU fields.

The issues with public CAs are described above. Public CAs are likely to refuse to sign certificates which contain the EKUs proposed here, and appear to be uninterested in offering a different product which would sign such certificates. Further, using these certificates for EAP appears to be against their intended purpose, and is therefore misuse.

However, the benefit of using public CAs is that they are already configured on most devices, and it is relatively simple to obtain certificates from them.

In the end, local administrators can choose whatever CA is best for them. Our goal here is to simplify the process of using a CA and server certificate for EAP. It is best to give administrators and implementors a few simple options which meet their needs, rather than mandating one particular solution which is likely to not meet the needs of a large set of users.

7.3.2. Limitations of public CAs

Recent changes to the CAB guidelines limit certificate validity periods to 397 days. While this change may be good for the larger WWW framework, it is not clear how it benefits other protocols such as EAP. Additional changes include limiting the kind of domain validation methods permitted, and forbidding file-based validation for wildcard certificates.

Part of EAP "best practices" is to ensure that EAP (or AAA) servers have minimal exposure to the public Internet. In order to use certificates from a public CA therefore, administrators must choose between either exposing their EAP server via WWW (in order to perform validation), or to expose a different WWW server, and then also simultaneously install the same certificate on the EAP system. Neither option is a good one.

In addition, limiting the certificate validity period means that clients see the server certificate change much more often than has been previously the case. This higher volume of changes was historically perhaps not an issue, as we have seen above that some systems perform limited checks on server certificates.

The benefit of the process outlined here is that the server certificate either becomes trivially verifiable (even if it changes), or installing a new server certificate becomes a trivial and secure process.

On the other hand, if there is no automated process to update the EAP client configuration, then users will simply be trained to mindlessly click "accept" when they are presented with a new certificate. As this process will happen much more often than has historically been the case, this maladaptive behavior by users will be even more strongly enforced.

Another issue with public CAs is that intermediate CA certificates are significantly more expensive than a server certificate. The reason here appears to be economic: if intermediate CA certs were cheap, then an organization would simply purchase one, and then use that CA cert to issue many server certificates.

This limitations means that EAP-TLS is significantly more difficult to deploy in practice and PEAP or TTLS. The administrator has to choose between either purchasing an extremely expensive intermediate CA certificate, or using a private CA.

The use of intermediate CAs has other issues, as we will see in the next section.

7.3.3. CA Chains

Some administrators wish to use multiple CAs for security. For example, a large organization could have one CA which is controlled by a security group. That CA could issue intermediate CA certificates to other groups with the organization. These CAs could be issued on multiple grounds, such as geographic location, or function units.

In practice, this process could result in there being one CA which is used for EAP / AAA, another CA which is used for internal web sites, and another CA for organizational Virtual Private Network (VPN) usage.

We have worked through all of the examples and discussion above by largely assuming that there was one "root" CA. Now that we see this assumption is insufficient, we must then discuss, and solve, the issue of intermediate CAs.

If an application were to simply trust the "root" CA, then by inference it would also trust all intermediate CAs. This trust means that an EAP administrator who can issue client certificates could potentially configure that client certificate for use in another protocol, such as with a VPN. Such misuse could lead to unauthorized users obtaining access to resources.

The solution here is two-fold. One, applications which accept client certificates SHOULD be configured to trust a particular issuing CA, which may not be the "root" CA. That is, simply having a certificate store of root CAs is insufficient. Instead, the application needs to track a particular intermediate CA.

Another solution is to simply move to purpose-specific EKU fields. An EAP client which follows this specification can require that the EAP server contain an id-kp-eapServer field. The EAP client can then rely on policy within the issuing framework to ensure that all relevant certificates also have an id-kp-eapServer field. Similarly, and EAP server which follows this specification can ensure that EAP clients are presenting certificates which contain id-kp-eapClient.

That is, the use of id-kp-serverAuth for all possible applications means that it is impossible to limit the use of certificates to one particular application. An administrator or end user is free to (mis-)use any certificate for almost any purpose.

We would suggest that having purpose-specific key usage fields is preferable. Such fields would make it simpler for both clients and servers to have more fine-grained control over certificate usage.

7.3.4. Delegated Authentication

In some cases an organization may delegate EAP / AAA functionality to another organization. This can happen for example, when an organization does not wish to run authentication servers itself, but instead delegates that functionality, say to an identity provider (IdP). The delegated functionality may be operated by an organization which handles "authentication as a service" for multiple customers.

A current solution is for the IdP system to present a server certificate which contains a list all of the domain names which it services. The problem is that this list can change often, which means that the old certificate must be revoked, and a new one issued. If these changes happen regularly, then this "churning" of certificates can cause problems for clients which cache the server certificate. There is also the management overhead of updating the certificate. Over all, this process is not scalable.

The processes outlined here allow for simple discovery and configuration of TLS-based EAP methods, but they do not entirely solve this problem.

The problem can be solved, however, by noting that the public EAP Response Identity used should be in the form "@realm", as per [EAPTLS] Section 2.1.7. An EAP server will receive this identity in the first EAP packet, at which point the server can select and present a certificate which is appropriate for that realm.

The result is that an IdP needs to be configured only with one server certificate for each NAI realm that it manages. When an NAI realm is added, deleted, or updated, those changes affect only the configuration for the modified realm. Any other organization or NAI realm is not affected.

This solution is simple and scalable.

7.3.5. Identification of Networks

While the examples above used an SSID to identify a network, there are other ways of network identification.

One is the Roaming Consortium Organisation Identifiers (RCOI), which are organizational identifiers which are assigned by the IEEE (REF TBD). They can be 24 or 36 bits. These organizations are global, and can identify a vendor, operator, consortium, or other organization.

This section defines the RCOIdentifier name as a form of otherName from the GeneralName structure in subjectAltName defined in [RFC5280].

```
id-on-RCOIdentifier OBJECT IDENTIFIER ::= { id-on TBD }
```

```
ub-RCOIdentifier-length INTEGER ::= 255
```

```
RCOIdentifier ::= OCTET String (SIZE (1..ub-RCOIdentifier-length))
```

This field can be used in either a client certificate or a server certificate. With either usage, it indicates to the client which RCOI should be used for accessing network services.

7.4. Anti-solutions

In this section, we explain why a number of existing technologies do not solve the problems which are addressed by this specification.

7.4.1. MDM Products Are not the Solution

MDM products are not the solution. Solutions like Eduroam CAT [CAT] are simple and easy to use, but they are only one of many possible products. In the extreme case, each end user has to download one MDM product for each network being accessed, and then repeat that process across each of many devices being used to access those networks.

These solutions are not just expensive, and non-standard, they are not scalable. It is difficult to scale the solutions to millions of disparate devices, as software has to be written and verified for each vendor, and often for each firmware version supplied by a vendor.

In addition, MDM products do not scale for an individual device. Each MDM product usually assumes that it is in complete control of the device, which makes it difficult or impossible to install multiple products. For example, a contractor who works for multiple companies may need multiple conflicting MDM products. Or, an

employee may be required to install an MDM product on a personal device, which makes it difficult to say who actually owns that device.

These MDM products typically also are capable of remotely wiping the device, such as when a contractor or employee leaves an organization. If the device was bought for personal use, there are ethical and may also be legal implications. Other issues are the loss of critical data such as documents or personal photographs.

Or perhaps even worse, when an MDM product is in complete control of a device, then there is plausible deniability for a user, for any action taken on that device. It is likely defensible to claim that the user is not responsible, because "the remote admin had full control", or perhaps "the remote admin is running software which controls my device".

There are serious security issues with a user not being in control of their own device.

In contrast, a standard discovery and configuration method, run by devices at the edge, which leverages DNS and HTTP is proven to work at Internet scales. They are implemented once by each vendor, and then maintained afterwards. As the configuration for each organization (NAI realm) is separate, there are minimal issues with installing multiple configurations on the same machine.

However, in the interest of enabling multiple solutions, we also define an [RFC8552] URI record. This record points to a location where a client device can download an MDM solution which is specific to a particular organization.

The format of the URI record is as follows:

```
_install._mdm.<naiRealm>
```

This record SHOULD NOT be visible on the public Internet, i.e. the public DNS servers for that NAI Realm. Doing so would permit malicious actors to download and examine the MDM software. Instead, this record SHOULD be available only inside of the organizations private network. Either to devices which have used 802.1X in order to authenticate themselves to the organization, or to devices which are using private IP address ranges.

The server which hosts the URI SHOULD use device fingerprinting in order to provide a system-dependent MDM solution.

As with the requirements on certificates above, the URI MUST be

within same domain (NAI realm) as the CERT resource record. The URI MUST be secured with TLS transport. The certificate presented at that URI MUST be issued by a root CA which is generally already known to the device. The certificate presented at the URI MUST pass all normal HTTPS validation, including that for id-ce-subjectAltName.

If any of these validation steps fail, then the client MUST NOT download or use any further data presented by that server.

If the validation steps succeed, then the client device can download and run the MDM software which has been provided at that URI.

Where possible, the client MUST inform any human user that these steps are being taken, and MUST give the user the ability to prevent this download from happening. There are many situations where the client device is owned by the end user, and not the organization which is being accessed. As such, it is inappropriate to mandate that software be automatically installed.

However, there is also no requirement that an organization grant access to devices which do not follow the organizations policy. The organization is free to deny the device network access until such time as the MDM software has been installed.

7.4.2. EST and similar protocols do not solve all of the problem

Certificate provisioning solutions like EST [RFC7030] or Simple Certificate Enrolment Protocol (SCEP) [RFC8894] are useful, but they do not solve the underlying problem we solve here.

EST and SCEP are useful for provisioning CA certificates to end devices, and for end devices to request and provision client certificates. However, these processes generally require additional configuration on the client device, and also an unrestricted network connection.

Part of the problem we are trying to solve here is supplicant configuration, and EST / SCEP do not help.

Further, EST can require complex bootstrapping. Section 2 of [RFC7030] says:

Both the EST clients and server are configured with information that provides the basis for mutual authentication and for authorization. The specific initialization data depends on the methods available in the client and server, but it can include shared secrets, network service names and locations (e.g., a Uniform Resource Identifier (URI) ...), trust anchor information

(e.g., a CA certificate or a hash of a TA's certificate), and enrollment keys and certificates.

In contrast, the method proposed here requires that the client device have a known root CA from the web PKI, and the ability to do DNS and HTTPS. This capability is available on essentially all systems which can access the public internet.

The reason for this simplification is that the problem we are trying to solve for EAP is substantially smaller than the problem that EST is trying to solve.

We conclude by noting that our solution is entirely compatible with EST, in that the DNS query for "_ca._cert._eap.example.com" could return a CERT record which points to the URL of the EST server, for example "https://example.com/.well-known/est/cacerts", as described in [RFC7030] Section 4.1.2.

7.4.3. Captive Portals and Hotspots

Captive portals and hotspots have been traditionally used as a method of controlling network access, as with EAP. The use-case for captive portals is that the client devices can do DNS and HTTP, but that they do not have credentials already provisioned. The captive portal is usually a way to introduce humans into the process, by displaying information about the network, and asking the user for credentials such as credit cards, etc.

There are, of course, a number of issues with captive portals. It may take the client device some time to determine that it is in a captive portal. The information displayed on a captive portal page may be confusing to the end user, or may even be in a language which the user does not understand.

Automating the onboarding process means that almost all of these issues are resolved.

7.4.4. Fully Anonymous Network Access and Provisioning

In some situations, the device may have minimal authentication credentials. There is still, however, a need to provision the device.

Existing solutions such as [RFC7170] Section 3.8 rely on exchanging data inside of a tunneled EAP method. This process is useful, but does not permit either device to use the full suite of protocols available on the wider Internet. We therefore define a different provisioning method here.

When devices need to provision themselves, they SHOULD attempt to authenticate using EAP-TLS, with no client certificate, and an NAI realm of "@eap.arpa". This realm indicates to the server that the device wishes to perform provisioning.

In some situations, the client device can authenticate the server certificate presented by the EAP server. More generally, however, it cannot. The client device therefore SHOULD NOT attempt to verify the server certificate when using this provisioning method. The device MUST treat the network as untrusted, and the server MUST ensure that the device is placed into a "captive portal" network as per [EAPTLS] Section 2.1.5.

Further discussion of such provisioning is outside of the scope of this document.

7.5. id-kp-eapOverLAN May not be sufficient

While [RFC4334] Section 2 defines id-kp-eapOverLAN, it gives no explicit use-case for that EKU. That document suggests that the EKU is intended for use in client certificates. However, it also can be read to suggest that the EKU could also be used in server certificates.

As such, we define a new EKU, id-kp-eapClient. If it is determined that this new EKU is not needed, this document can be updated before final publication to use id-kp-eapOverLAN instead of id-kp-eapClient.

7.6. Guest Networks

For EAP-TLS, [EAPTLS] Section 2.1.5 provides for the protocol to be used without peer authentication. The methods outlined here can be extended to perform provisioning within guest networks. That is, the device suspects the identity of the network, but also knows that it does not yet have credentials for use within that network.

Where an EAP peer wishes to connection to the network, but does not know the identity of the network, it SHOULD use EAP-TLS without peer authentication. That is, it should obtain the server certificate without providing a client certificate.

This server certificate can be examined for identification fields, such as id-on-naiRealm. The supplicant SHOULD query the network for the expected server certificate, using the DNS discovery process outlined above.

These certificates and related network configuration which are discovered this way SHOULD NOT be cached for more than one day.

While on the provisioning network, the device can use almost any method to authenticate and authorize the end user. For example, having a "self service" registration page, obtaining temporary credentials, etc.

It is RECOMMENDED that the guest network permit the device to obtain email from anywhere on the Internet, via standard email reception protocols. It is RECOMMENDED that all other ports be blocked. Port 25 (SMTP) MUST be blocked. All DNS ports MUST either be blocked, or be forwarded to a DNS server controlled by the administrator of the guest network.

Network access SHOULD be restricted in both time and usage. There is no reason to allow unauthenticated guest access for more than about 30 minutes. There is no reason to allow unauthenticated guests to transfer gigabytes of data.

These requirements allow the provisioning process to be simple. A device uses EAP-TLS without peer authentication to connect to a network. The user enters an email address in a "self service" registration page. The visited network determines whether or not the person using that email address is authorized. If so, it sends a message to that address with a unique URL.

The user obtains the email, clicks on the URL, and downloads credentials for the visited network. These credentials could be an EAP-TLS client certificate, which has the following properties:

- * issued by the visited network
- * containing information identifying the end user
- * ideally also contains information identifying the device
- * has a limited lifetime, ideally one day
- * lifetime MUST be less than 30 days.

The device then provisions the credentials as above.

This process could also be extended by leveraging DNS even more. The client device could look up a CERT record based on the user's identifying information, e.g. for a user with identifier "user@example.com", visiting a particular naiRealm it could look up a CERT RR of:

```
user.example.com._guest._cert._eap.<naiRealm>
```

The local network could return a custom CERT RR, pointing to a URL for a page which contains a custom client certificate for use with EAP-TLS.

As most DNS servers have limited policy capabilities, this

functionality is likely difficult to implement in practice.

7.7. Using TLS with protocols other than EAP

While the discussion so far has been about EAP, there is no reason to limit this process to EAP. However, we do note that the methods defined here are intended for bootstrapping access to secure networks. They are not intended for use with generic web browsing.

For example, it is possible to use similar methods (though with different DNS names and EKU fields) in order to configure clients for other protocols such as RADIUS/TLS [RFC6614] or RADIUS/DTLS [RFC7360].

These methods are not limited to RADIUS and EAP. For example, discovery of an IMAP server could be done via looking up a SRV record for "_imap._tcp.<naiRealm>", while discovery of an email submission server could be done via looking up "_submit._tcp.<naiRealm>". If a private CA is used for those services, it could be discovered via looking up a CERT record for "_ca._imap._tcp.<naiRealm>".

Similarly, the issue of intermediate CAs discussed earlier is also applicable to other protocols, and therefore requires similar solutions. We also note that there are issues with many other non-WWW protocols which appear to (mis)-use the id-kp-serverAuth field. We offer no solution here to that problem.

It may be possible to use these processes in many other situations, but we do not discuss those use-cases in detail here. We only discuss them as a side note, to demonstrate that automatic provisioning of a client system can be done simply, securely, and with minimal intervention by an end user.

8. Moving to the new methods

The methods given herein are intended to give parties in an EAP conversation more, and better information about what should be happening, and about what is happening. If all of the recommended information is available, then all parties in an EAP conversation have strong, positive indications that the system is secure. If any information is missing or conflicting information is seen, then the system may or may not be secure.

That is, following the recommendations is a positive signal of security. Lack of positive signals does not necessarily indicate insecurity.

It is RECOMMENDED that EAP peers and authenticators which implement these processes add configurable flags which allow the recommendations to be made mandatory. These configurable flags SHOULD permit the recommendations to be enforced in a wide range of conditions, such as per SSID, per realm, per CA, etc. Doing so will allow administrators to make and enforce site-local policies.

For example, a company might mandate that all devices which connect to WiFi use EAP with client certificates, that those client certificates contain the fields defined above, and that those devices only send authentication credentials to EAP authenticators which also satisfy the recommendations above. When an EAP peer follows these mandates, it will not be vulnerable to any of the attacks outlined earlier.

These guidelines allows existing systems to operate unchanged. They also allow updated systems to gain the benefit of increased, and mandated, security.

8.1. Using the new OIDs

In general, we recommend using private CAs for EAP. Such uses avoid the issue of certificate misuse under the [CAB] guidelines.

We also have to address how systems which are unaware of this specification will interact with certificates containing the new OIDs.

Happily, the requirements of [RFC5216] and [EAPTLS] are requirements on what should exist, and not on what should not exist. Tests with implementations, and (where possible) checks of publicly available source code lead us to conclude that these requirements are accurately followed.

The solution then is for server certificates to contain both id-kp-serverAuth, in order to satisfy [RFC5280], and also to contain id-kp-eapServer, in order to satisfy this document. The result is that both old, and new behavior is supported, and that the transition path from one to the other is seamless.

8.2. Recommendations for EAP peers and authenticators

It is RECOMMENDED that EAP peers use a dedicated certificate store for EAP. Where a dedicated certificate store cannot be used, each certificate MUST have additional metadata stored with it, which indicates its permitted uses. This metadata serves as a way of creating "per-use-case" certificate stores.

It is RECOMMENDED that no CAs are enabled by default for EAP. User credentials are provisioned from a known authentication source. If there are no local user credentials configured, then by definition there are no known sources. When credentials are configured, known sources can be enabled at the same time.

It is RECOMMENDED that "web" CAs are not used for EAP. The two use-cases are different, and misuse of certificates opens both EAP and WWW systems to attacks.

It is RECOMMENDED that EAP peers do not perform TLS resumption across different media.

It is RECOMMENDED that when EAP peers use any TLS-based EAP method with a client certificate, that the client certificate contains id-kp-eapClient in order to indicate that the certificate is intended to be used by an EAP peer.

It is RECOMMENDED that EAP peers expose configuration settings which allow the user to permit this new behavior, or require it, on a per-NAI realm basis.

It is RECOMMENDED that EAP servers which permit the use of client certificates mark one or more CAs as being permitted to issue client certificates. These CAs SHOULD be the one which is the "lowest" in the certificate chain. That is, the one which is closest to the client certificate. EAP servers SHOULD NOT mark a global "root" CA as being permitted to issue client certificates, as that root CA may sign many intermediate CAs, each of which could then issue client certificates.

It is RECOMMENDED to use id-pe-wlanSSID [RFC4334] in client and server certificates. When used in a client certificate, it informs the client that this certificate should be used when the given SSID

is seen. When used in a server certificate, it informs the client that the server is intended to be reachable from this particular SSID. Note that a mismatch is not necessarily an error.

It is RECOMMENDED that when EAP authenticators use any TLS-based EAP method with a server certificate, that the server certificate contains `id-kp-eapServer` in order to indicate that the certificate is intended to be used by an EAP authenticator.

It is RECOMMENDED that when EAP authenticators use any TLS-based EAP method with a server certificate, that the server certificate contains one or more `naiRealm`, to indicate that the EAP authenticator is authorized to accept authentication requests for users in those realms.

The requirements of [RFC7585] Section 2.2 on the definition, number, and format of `naiRealm` are included here by reference.

It is RECOMMENDED that when EAP peers use any TLS-based EAP method, that the EAP peer verify that the server certificate presented contains `id-kp-eapServer`, and an `naiRealm` which matches the NAI (if used) in the EAP Identity Response. Any mismatch indicates to the client that the server is not trusted to authenticate users for that realm. Therefore user credentials for that NAI realm should not be sent to the server.

It is RECOMMENDED that when EAP authenticators use any TLS-based EAP method and a client certificate is presented, that the EAP authenticator verify that the client certificate contains `id-kp-eapClient`, and that the NAI (if used) given in the EAP Identity Response field matches one of the `naiRealm` fields in the server certificate. Any mismatch indicates to the server that the client is either misconfigured, or is acting maliciously. The server should therefore treat this mismatch as an authentication failure.

As noted above, the use of the label "*" in `id-on-naiRealm` is forbidden for this specification.

Where the server certificates do not contain `naiRealm`, but do contain one or more `subjectAltName` field of type `dNSName`, clients SHOULD verify that the NAI realm used by the client is an exact suffix of the `dNSName` field.

8.3. Principles and Guidelines

After analysis of the historical practices and standards for EAP, we came to a set of guidelines which are outlined in this section. Application of these guidelines drove the rest of the specification which we define herein.

It is RECOMMENDED that the guidelines given below are followed when developing new network configuration standards and methods:

- * Automated provisioning is strongly preferred to manual provisioning. We define "automated provisioning" as provisioning which is performed via software, with little or no user intervention. Automation minimizes the possibility for end users to create broken or insecure configurations.
- * Manual provisioning should be limited to "Trust on first use" (ToFU), and cached or "pinned" after that. That is, manual provisioning should be limited to allowing a user to approve validation decisions which have been made by the system.
- * Relying on end users to manually configure complex systems is strongly discouraged. Complex systems are difficult to configure, and improperly configured systems create many issues related to security, usability, and network access.
- * Configuration should be "pinned" in order to permit systems to detect and prevent unauthorized changes, and to detect malicious networks which claim to be updated versions of the true network.
- * The identity and role of both parties should be exchanged, and verified. In practice, this suggestion often means that TLS-based EAP methods are preferred to ones which only do name / password credential verification.
- * The previous requirement usually means that the both parties know which RFC 7542 NAI realm is being used. This realm serves a similar purpose to the the DNS host name used in other TLS-based protocols such as HTTPS. As such, similar methods can be used to validate certificate authenticity. This NAI realm is contained in an id-on-naiRealm field, as defined in [RFC7585] Section 2.2
- * For TLS-based EAP methods, trust should be based on a certification authority (CA), which signs certificates for a particular realm. If the CA is trusted, then everything derived from that CA can be trusted. If the CA is not trusted, then it is impossible to trust anything derived from an untrusted CA.

- * CAs should also be associated with permitted uses. For example, a root CA which is trusted for web surfing is not necessarily trusted for use with EAP authentication. In practice this means either having separate certificate stores for different purposes, or annotating root certificates with their permitted uses.

We believe that these recommendations are correct, simple, practical, and will improve security and usability for all participants in EAP. We show that there is a clear upgrade path from current behavior to better behavior. Each step of that upgrade path is simple, and involves minimal change for end users or administrators.

9. Security Considerations

There are a large number of security issues with current practices. This document attempts to give both fixes, and a transition path to a better system. As such, the entire document is discussing security issues.

One of the main points of this document is that systems which are difficult to configure are likely to be insecure.

This document also highlights problems with misuse of certificates containing id-kp-serverAuth, and id-kp-clientAuth. If such misused certificates were to be widely reported, then large parts of the Internet could be taken offline.

We note that distribution of these certificates MUST NOT be done via email. There is just too much possibility for forgery and user mistakes for that process to be secure. We instead rely on secure transport layers and cryptographically signed data in order to bootstrap authenticated network access.

9.1. On Identities and Service Discovery

All the user needs is an identity (e.g. "user@example.com"), and a password. Essentially everything else required for network access can be derived automatically, and provisioned with no additional user input. This process is significantly more secure than manual provisioning.

9.2. Password Hashing and Storage

In some situations, using client certificates with EAP is preferable to using password-based methods. Password-based EAP methods often hash the password along with a salt or a challenge, and then send the hashed version of the password. However, this hashing can conflict with the desire of administrators to store hashed passwords in their

user databases. The two different hashing methods are almost always incompatible, which means that the administrator has to choose either to send passwords via a method such as PAP inside of TTLS, or to store clear-text passwords in their local user database.

Neither choice is optimal. Where there is a trade-off, it is RECOMMENDED that systems use a method such as TTLS with PAP, and then store hashed or encrypted passwords in the local user database. The "clear-text" password which is sent in TTLS is, in fact, secured via TLS when it is sent "over the wire". So it is incorrect to claim that EAP is sending passwords "in the clear".

The caveat here is that supplicants must verify the identity of the server certificate before sending the password to the EAP server. As we have seen in Section 2 above, many supplicants skipped this step. As a result, it has been common practice to recommend that supplicants use EAP methods which do not send clear-text passwords. However, all this recommendation does is to move the security risk from the supplicant to the database, which is now required to store clear-text passwords for all users.

While some would argue that exposing the users clear-text password to an EAP Server is a security risk, it is in practice irrelevant. The EAP server is almost always co-located with an AAA server (e.g. RADIUS or Diameter). Those servers control network access for entire organizations, including setting complex policies. Any attacker who gains control of an AAA server can take many more, and worse actions than to simply observe peoples passwords.

In contrast, history shows that exposure of user databases (with names and passwords) is not uncommon. In fact, as the EAP or AAA server usually has complete access to the user database (including passwords), compromise of the AAA server almost by definition leads to compromise of the local user password database.

We therefore make the trade-off which has the lowest possible security impact, for all failure cases. Passwords SHOULD be stored hashed or encrypted in a user database. TLS-based EAP methods which rely on passwords SHOULD use authentication methods which are compatible with such password storage methods, which generally means that the passwords are sent by the user in clear-text, but are protected by TLS.

10. IANA Considerations

This section this specification requests from Internet Assigned Numbers Authority (IANA) registration of the following items.

10.1. Key Purpose OIDs

We request registration of values related to the certificate key purpose OIDs in accordance with [RFC8126].

- * id-kp-eapServer

- * id-kp-eapClient

10.2. Underscored and Globally Scoped DNS Node Names

Per RFC 8552, please add the following entry to the "Underscored and Globally Scoped DNS Node Names" registry:

RR Type	_NODE NAME	Reference
CERT	_ca._cert._eap	<this document>
CERT	_client._cert._eap	<this document>
CERT	_server._cert._eap	<this document>
SRV	_est._eap	<this document>
URI	_install._mdm	<this document>

We note that [RFC8552] does not provide for "sub" registries, as we have defined above. However, we believe that these definitions fall within both the intent of [RFC8552], and common practice.

11. References

11.1. Normative References

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC4334]

Housley, R., and Moore, T., "Certificate Extensions and Attributes Supporting Authentication in Point-to-Point Protocol (PPP) and Wireless Local Area Networks (WLAN)", RFC 4334, February 2006

- [RFC4398]
Josefsson, S., "Storing Certificates in the Domain Name System (DNS)", RFC 4398, March 2006.
- [RFC5216]
Simon, D., Aboba, B., and R. Hurst, "The EAP-TLS Authentication Protocol", RFC 5216, March 2008
- [RFC7170]
Zhou, H., et al., "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, May 2014.
- [RFC7234]
Fielding, Ed., et al, "Hypertext Transfer Protocol (HTTP/1.1): Caching", RFC 7234, June 2014
- [RFC7542]
DeKok, A., "The Network Access Identifier", RFC 7542, May 2015.
- [RFC8126]
Cotton, M., et al, "Guidelines for Writing an IANA Considerations Section in RFCs", RC 8126, June 2017.
- [RFC8174]
Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.
- [RFC8552]
Crocker, D., "Scoped Interpretation of DNS Resource Records through "Underscored" Naming of Attribute Leaves", RFC 8552, March 2019.
- [EAPTLS]
Mattsson, J., and Sethi, M., "Using EAP-TLS with TLS 1.3", draft-ietf-emu-eap-tls13-15, May 2021.

11.2. Informative References

- [CAT]
<https://cat.eduroam.org/>
- [EDUROAM]
<https://eduroam.org/>
- [MSPEAP]
<https://msdn.microsoft.com/en-us/library/cc238354.aspx>

[PEAP]

Palekar, A. et al, "Protected EAP Protocol (PEAP)", draft-josefsson-pppext-eap-tls-eap-06.txt, March 2003.

[CAB]

CA/Browser Forum, "Baseline Requirements for the Issuance and Management of Publicly-Trusted Certificates" Version 1.7.4, 5 April 2021 <https://cabforum.org/wp-content/uploads/CA-Browser-Forum-BR-1.7.4.pdf>

[RFC2716]

Aboba, B., and Simon, D., "PPP EAP TLS Authentication Protocol", RFC 2716, October 1999.

[RFC2865]

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

[RFC2818]

Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[RFC4851]

Cam-Winget, N., et al, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

[RFC5280]

Cooper, D., et al., "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[RFC5281]

Funk, P., and Blake-Wilson, S., "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.

[RFC5785]

Nottingham, M. and Hammer-Lahav, E., "Defining Well-Known Uniform Resource Identifiers (URIs)", April 2010.

[RFC5931]

Harkins, D., and Zorn, G., "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, August 2010.

[RFC6614]

Winter, S., et al., "Transport Layer Security (TLS) Encryption for RADIUS", RFC 6614, May 2012.

[RFC6677]

Hartman, S. (ed), et al., "Channel-Binding Support for Extensible Authentication Protocol (EAP) Methods", RFC 6677, July 2012.

[RFC7030],

Pritikin, M. (Ed), Et al, "Enrollment over Secure Transport", October 2013

[RFC7360]

DeKok, A., "Datagram Transport Layer Security (DTLS) as a Transport Layer for RADIUS", RFC 7360, September 2014.

[RFC7585]

Winter, S, and McCauley, M., "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, October 2015.

[RFC7593]

Weiringa, K. et al, "The eduroam Architecture for Network Roaming", RFC 7593, September 2015.

[RFC8446]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", August 2018.

[RFC8894]

Gutmann, P., "Simple Certificate Enrolment Protocol", RFC 8894, September 2020.

Acknowledgments

This document would not be possible without the input of many people.

Jorge Vergara provided reviews of previous documents which led to a clearer articulation of the problem described here, and which therefore motivated this document. He has also provided reviews for early versions of this document.

Tom Rixom provided a significant amount of information on issues seen by supplicants, which motivated much of the text in Section 3.1.

Arran Cudbard-Bell suggested using DNS for the "out of band" provisioning of certificates in Section 3. He also provided detailed reviews of multiple versions of this document.

Stefan Winter provided feedback on a number of issues related to certificates, roaming, and provisioning.

Karri Huhtanen raised issues related to purpose-specific CAs, and provided suggestions to help address those issues.

Authors' Addresses

Alan DeKok
Network RADIUS SAS
26 rue Colonel Dumont
38000 Grenoble
FRANCE

Email: aland@networkradius.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2022

O. Friel
Cisco
D. Harkins
Hewlett-Packard Enterprise
February 04, 2022

Bootstrapped TLS Authentication
draft-friel-tls-eap-dpp-04

Abstract

This document defines a TLS extension that enables a server to prove to a client that it has knowledge of the public key of a key pair where the client has knowledge of the private key of the key pair. Unlike standard TLS key exchanges, the public key is never exchanged in TLS protocol messages. Proof of knowledge of the public key is used by the client to bootstrap trust in the server. The use case outlined in this document is to establish trust in an 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 <https://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 August 8, 2022.

Copyright Notice

Copyright (c) 2022 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 (<https://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	2
1.1. Bootstrap Key Pair	2
1.2. Alignment with Wi-Fi Alliance Device Provisioning Profile	3
2. Bootstrapping in TLS 1.3	4
2.1. Bootstrap Extended PSK	4
2.2. Changes to TLS 1.3 Handshake	5
3. Using TLS Bootstrapping in EAP	6
4. Summary of Work	7
5. IANA Considerations	7
6. Security Considerations	7
7. References	8
7.1. Normative References	8
7.2. Informative References	8
Authors' Addresses	9

1. Introduction

On-boarding of devices with no, or limited, user interface can be difficult. Typically, a credential is needed to access the network and network connectivity is needed to obtain a credential. This poses a catch-22.

If trust in the integrity of a device's public key can be obtained in an out-of-band fashion, a device can be authenticated and provisioned with a usable credential for network access. While this authentication can be strong, the device's authentication of the network is somewhat weaker. [duckling] presents a functional security model to address this asymmetry.

There are on-boarding protocols, such as [DPP], to address this use case but they have drawbacks. [DPP] for instance does not support wired network access. This document describes an on-boarding protocol, which we refer to as TLS Proof of Knowledge or TLS-POK.

1.1. Bootstrap Key Pair

The mechanism for on-boarding of devices defined in this document relies on bootstrap key pairs. A client device has an associated elliptic curve (EC) key pair. The key pair may be static and baked into device firmware at manufacturing time, or may be dynamic and generated at on-boarding time by the device. If this public key, specifically the ASN.1 SEQUENCE SubjectPublicKeyInfo from [RFC5280],

can be shared in a trustworthy manner with a TLS server, a form of "origin entity authentication" (the step from which all subsequent authentication proceeds) can be obtained.

The exact mechanism by which the server gains knowledge of the public key is out of scope of this specification, but possible mechanisms include scanning a QR code to obtain a base64 encoding of the ASN.1-formatted public key or uploading of a Bill of Materials (BOM) which includes the public key. If the QR code is physically attached to the client device, or the BOM is associated with the device, the assumption is that the public key obtained in this bootstrapping method belongs to the client. In this model, physical possession of the device implies legitimate ownership.

The server may have knowledge of multiple bootstrap public keys corresponding to multiple devices, and TLS extensions are defined in this document that enable the server to identify a specific bootstrap public key corresponding to a specific device.

Using the process defined herein, the client proves to the server that it has possession of the private analog to its public bootstrapping key. Provided that the mechanism in which the server obtained the bootstrapping key is trustworthy, a commensurate amount of authenticity of the resulting connection can be obtained. The server also proves that it knows the client's public key which, if the client does not gratuitously expose its public key, can be used to obtain a modicum of correctness, that the client is connecting to the correct network (see [duckling]).

1.2. Alignment with Wi-Fi Alliance Device Provisioning Profile

The definition of the bootstrap public key aligns with that given in [DPP]. This, for example, enables the QR code format as defined in [DPP] to be reused for TLS-POK. Therefore, a device that supports both wired LAN and Wi-Fi LAN connections can have a single QR code printed on its label, and the bootstrap key can be used for DPP if the device bootstraps against a Wi-Fi network, or TLS-POK if the device bootstraps against a wired network. Similarly, a common bootstrap public key format could be imported in a BOM into a server that handles devices connecting over both wired and Wi-Fi networks.

Any bootstrapping method defined for, or used by, [DPP] is compatible with TLS-POK.

2. Bootstrapping in TLS 1.3

The bootstrapping modifications introduce an extension to identify a "bootstrapping" key which is converted into an external PSK and used directly in the TLS 1.3 handshake. This key MUST be from a cryptosystem suitable for doing ECDSA.

2.1. Bootstrap Extended PSK

This document defines the "bskey" extended PSK type by expanding on the work in [extensible-psks].

```
enum {  
    bskey(TBD), (255)  
} ExtendedPskIdentityType;
```

A bskey PSK is a variant of an external PSK which, in this case, is derived from a public key.

The PSKIdentity of a bskey extended PSK is encoded with a string derived from the DER-encoded ASN.1 subjectPublicKeyInfo representation of the bootstrapping public key.

```
struct {  
    opaque identity<1..232-1>  
} BootstrapPSKIdentity;
```

Both the bskey PSK and the BootstrapPSKIdentity are computed using [RFC5869] with the hash algorithm from the ciphersuite:

```
bskeypsk = HKDF-Expand(HKDF-Extract(<>, bskey),  
                        "tls13-extended-psk-bskey", L)  
identity = HKDF-Expand(HKDF-Extract(<>, bskey),  
                        "tls13-psk-identity-bskey", L)
```

where:

- <> is a NULL salt
- bskey is the DER-encoded ASN.1 subjectPublicKeyInfo representation of the bootstrapping key
- L is the length of the digest of the underlying hash algorithm

A performance versus storage tradeoff a server can choose is to precompute the identity of every bootstrapped key with every hash algorithm that it uses in TLS and use that to quickly lookup the bootstrap key and generate the PSK. Servers that choose not to employ this optimization will have to do a runtime check with every bootstrap key it holds against the identity the client provides.

2.2. Changes to TLS 1.3 Handshake

The client includes the "tls_cert_with_extern_psk" extension in the ClientHello, per [RFC8773], and identifies the bootstrapping key using the BootstrapPSKIdentity extension. The server looks up the client's bootstrapping key in its database by checking the hash of each entry with the value received in the ClientHello. If no match is found, the server SHALL terminate the TLS handshake with an alert.

If the server found the matching bootstrap key, it generates the bskeypsk and includes the "tls_cert_with_extern_psk" extension in the ServerHello message. When these extensions have been successfully negotiated, the TLS 1.3 key schedule SHALL include both the bskeypsk in the Early Secret derivation and an (EC)DHE shared secret value in the Handshake Secret derivation.

After successful negotiation of these extensions, the full TLS 1.3 handshake is performed with the additional caveat that the client authenticates with a raw public key (its bootstrapping key) per [RFC7250]. The bootstrapping key is always an elliptic curve public key, therefore the ClientCertTypeExtension SHALL always indicate RawPublicKey and the type of the client's Certificate SHALL be ECDSA and contain the client's bootstrapping key as a DER-encoded ASN.1 subjectPublicKeyInfo SEQUENCE.

When the server processes the client's Certificate it MUST ensure that it is identical to the bootstrapping public key that it used to generate an external PSK and PSKIdentifier for this handshake.

When clients use the [duckling] form of authentication, they MAY forgo the checking of the server's certificate in the CertificateVerify and rely on the integrity of the bootstrapping method employed to distribute its key in order to validate trust in the authenticated TLS connection.

The handshake is shown in Figure 1.

```

Client
-----
ClientHello
+ bskey_id
+ cert_with_extern_psk
+ client_cert_type=RawPublicKey
+ key_share
----->

Server
-----
ServerHello
+ bskey_id
+ cert_with_extern_psk
+ client_cert_type=RawPublicKey
+ key_share
{EncryptedExtensions}
{CertificateRequest}
{Certificate}
{CertificateVerify}
<----- {Finished}
{Certificate}
{CertificateVerify}
{Finished}
----->
[Application Data] <-----> [Application Data]

```

Figure 1: TLS 1.3 TLS-POK Handshake

3. Using TLS Bootstrapping in EAP

Enterprise deployments typically require an 802.1X/EAP-based authentication to obtain network access. Protocols like [RFC7030] can be used to enroll devices into a Certification Authority to allow them to authenticate using 802.1X/EAP. But this creates a Catch-22 where a certificate is needed for network access and network access is needed to obtain certificate.

Devices whose bootstrapping key can be obtained in an out-of-band fashion can perform an EAP-TLS-based exchange, for instance [RFC7170], and authenticate the TLS exchange using the bootstrapping extensions defined in Section 2. This network connectivity can then be used to perform an enrollment protocol (such as provided by [RFC7170]) to obtain a credential for subsequent network connectivity and certificate lifecycle maintenance.

Upon "link up", an Authenticator on an 802.1X-protected port will issue an EAP Identify request to the newly connected peer. For unprovisioned devices that desire to take advantage of TLS-POK, there is no initial realm in which to construct an NAI (see [RFC4282]) so the initial EAP Identity response SHOULD contain simply the name "TLS-POK" in order to indicate to the Authenticator that an EAP method that supports TLS-POK SHOULD be started.

Authenticating Peer	Authenticator
-----	-----
	<- EAP-Request/ Identity
EAP-Response/ Identity (TLS-POK) ->	
	<- EAP-Request/ EAP-Type=TEAP (TLS Start)
	.
	.
	.

4. Summary of Work

The protocol outlined here can be broadly broken up into 4 distinct areas:

- o TLS extensions to transport the bootstrap public key identifier
- o Use of the TLS 1.3 extension for certificate-based authentication with an external PSK
- o The client's use of a raw public key in its certificate
- o TEAP extensions to leverage the new TLS-POK handshake for trust establishment

This document captures all 4 areas.

5. IANA Considerations

IANA will allocated an ExtensionPSKIdentityType for the bskey type from the TLS 1.3 repository created by [extensible-psks] and replace TBD in this document with that number.

6. Security Considerations

Bootstrap and trust establishment by the TLS server is based on proof of knowledge of the client's bootstrap public key, a non-public datum. The TLS server obtains proof that the client knows its bootstrap public key and, in addition, also possesses its corresponding private analog.

Trust on the part of the client is based on validation of the server certificate and the TLS 1.3 handshake. In addition, the client

assumes that knowledge of its public bootstrapping key is not widely disseminated and therefore any device that proves knowledge of its bootstrapping key is the appropriate device from which to receive provisioning, for instance via [RFC7170]. [duckling] describes a security model for this type of "imprinting".

An attack on the bootstrapping method which substitutes the public key of a corrupted device for the public key of an honest device can result in the TLS sever on-boarding and trusting the corrupted device.

If an adversary has knowledge of the bootstrap public key, the adversary may be able to make the client bootstrap against the adversary's network. For example, if an adversary intercepts and scans QR labels on clients, and the adversary can force the client to connect to its server, then the adversary can complete the TLS-POK handshake with the client and the client will connect to the adversary's server. Since physical possession implies ownership, there is nothing to prevent a stolen device from being on-boarded.

7. References

7.1. Normative References

- [RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <<https://www.rfc-editor.org/info/rfc5869>>.
- [RFC7250] Wouters, P., Ed., Tschofenig, H., Ed., Gilmore, J., Weiler, S., and T. Kivinen, "Using Raw Public Keys in Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS)", RFC 7250, DOI 10.17487/RFC7250, June 2014, <<https://www.rfc-editor.org/info/rfc7250>>.
- [RFC8773] Housley, R., "TLS 1.3 Extension for Certificate-Based Authentication with an External Pre-Shared Key", RFC 8773, DOI 10.17487/RFC8773, March 2020, <<https://www.rfc-editor.org/info/rfc8773>>.

7.2. Informative References

- [DPP] Wi-Fi Alliance, "Device Provisioning Profile", 2020.
- [duckling] Stajano, F. and E. Rescorla, "The Resurrecting Duckling: Security Issues for Ad-Hoc Wireless Networks", 1999.

[extensible-psks]

Wood, C. and R. Anderson, "Extensible Pre-Shared Key Types for TLS", n.d., <<https://chris-wood.github.io/draft-tls-extensible-psks/draft-group-tls-extensible-psks.html>>.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, DOI 10.17487/RFC4282, December 2005, <<https://www.rfc-editor.org/info/rfc4282>>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <<https://www.rfc-editor.org/info/rfc5280>>.

[RFC7030] Pritikin, M., Ed., Yee, P., Ed., and D. Harkins, Ed., "Enrollment over Secure Transport", RFC 7030, DOI 10.17487/RFC7030, October 2013, <<https://www.rfc-editor.org/info/rfc7030>>.

[RFC7170] Zhou, H., Cam-Winget, N., Salowey, J., and S. Hanna, "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, DOI 10.17487/RFC7170, May 2014, <<https://www.rfc-editor.org/info/rfc7170>>.

Authors' Addresses

Owen Friel
Cisco

Email: ofriel@cisco.com

Dan Harkins
Hewlett-Packard Enterprise

Email: daniel.harkins@hpe.com

Network Working Group
INTERNET-DRAFT
Updates: 5247, 5281, 7170
Category: Standards Track
Expires: September 05, 2022

DeKok, Alan
FreeRADIUS
5 March 2022

TLS-based EAP types and TLS 1.3
draft-ietf-emu-tls-eap-types-05.txt

Abstract

EAP-TLS (RFC 5216) has been updated for TLS 1.3 in RFC 9190. Many other EAP types also depend on TLS, such as FAST (RFC 4851), TTLS (RFC 5281), TEAP (RFC 7170), and possibly many vendor specific EAP methods. This document updates those methods in order to use the new key derivation methods available in TLS 1.3. Additional changes necessitated by TLS 1.3 are also discussed.

Status of this Memo

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

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at <http://www.ietf.org/ietf/lid-abstracts.txt>.

The list of Internet-Draft Shadow Directories can be accessed at <http://www.ietf.org/shadow.html>.

This Internet-Draft will expire on January 29, 2021.

Copyright Notice

Copyright (c) 2022 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	4
1.1. Requirements Language	4
2. Using TLS-based EAP methods with TLS 1.3	5
2.1. Key Derivation	5
2.2. TEAP	6
2.3. FAST	7
2.4. TTLS	8
2.4.1. Client Certificates	8
2.5. PEAP	9
2.5.1. Client Certificates	9
3. Application Data	9
3.1. Identities	11
4. Resumption	13
5. Implementation Status	14
6. Security Considerations	14
6.1. Protected Success and Failure indicators	15
7. IANA Considerations	16
8. References	17
8.1. Normative References	17
8.2. Informative References	18

1. Introduction

EAP-TLS has been updated for TLS 1.3 in [RFC9190]. Many other EAP types also depend on TLS, such as FAST [RFC4851], TTLS [RFC5281], TEAP [RFC7170], and possibly many vendor specific EAP methods such as PEAP [PEAP]. All of these methods use key derivation functions which are no longer applicable to TLS 1.3. As such, all of those methods are incompatible with TLS 1.3.

This document updates those methods in order to be used with TLS 1.3. These changes involve defining new key derivation functions. We also discuss implementation issues in order to highlight differences between TLS 1.3 and earlier versions of TLS.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

2. Using TLS-based EAP methods with TLS 1.3

In general, all of the requirements of [RFC9190] apply to other EAP methods that wish to use TLS 1.3. Unless otherwise required herein, implementations of EAP methods that wish to use TLS 1.3 MUST follow the guidelines in [RFC9190].

There remain some differences between EAP-TLS and other TLS-based EAP methods which necessitates this document. The main difference is that [RFC9190] uses the EAP-TLS Type (value 0x0D) in a number of calculations, whereas other method types will use their own Type value instead of the EAP-TLS Type value. This topic is discussed further below in Section 2.

An additional difference is that [RFC9190] Section 2.5 requires that once the EAP-TLS handshake has completed, the EAP server sends a protected success result indication. This indication is composed of one octet (0x00) of application data. Other TLS-based EAP methods also use this indicator, but only during resumption. When other TLS-based EAP methods use full authentication, the indicator is not needed, and is not used. This topic is explained in more detail below, in Section 3 and Section 4.

Finally, the document includes clarifications on how various TLS-based parameters are calculated when using TLS 1.3. These parameters are different for each EAP method, so they are discussed separately.

2.1. Key Derivation

The key derivation for TLS-based EAP methods depends on the value of the EAP Type as defined by [IANA] in the Extensible Authentication Protocol (EAP) Registry. The most important definition is of the Type field, as first defined in [RFC3748] Section 2:

Type = value of the EAP Method type

For the purposes of this specification, when we refer to logical Type, we mean that the logical Type is defined to be 1 octet for values smaller than 254 (the value for the Expanded Type), and when Expanded EAP Types are used, the logical Type is defined to be the concatenation of the fields required to define the Expanded Type, including the Type with value 0xfe, Vendor-Id (in network byte order) and Vendor-Type fields (in network byte order) defined in [RFC3748] Section 5.7, as given below:

Type = 0xFE || Vendor-Id || Vendor-Type

This definition does not alter the meaning of Type in [RFC3748], or

change the structure of EAP packets. Instead, this definition allows us to simplify references to EAP Types, by just using a logical "Type" instead of referring to "the Type field or the Type field with value 0xfe, plus the Vendor-ID and Vendor-Type". For example, the value of Type for PEAP is simply 0x19.

Unless otherwise discussed below, the key derivation functions for all TLS-based EAP Types are defined in [RFC9190] Section 2.3, and reproduced here for clarity:

```
Key_Material = TLS-Exporter("EXPORTER_EAP_TLS_Key_Material",
                             Type, 128)
Method-Id    = TLS-Exporter("EXPORTER_EAP_TLS_Method-Id",
                             Type, 64)
Session-Id   = Type || Method-Id
MSK          = Key_Material(0, 63)
EMSK         = Key_Material(64, 127)
```

We note that these definitions re-use the EAP-TLS exporter labels, and change the derivation only by adding a dependency on the logical Type. The reason for this change is simplicity. There does not appear to be compelling reasons to make the labels method-specific, when they can just include the logical Type in the key derivation.

These definitions apply in their entirety to TTLS [RFC5281] and PEAP as defined in [PEAP] and [MSPEAP]. Some definitions apply to FAST and TEAP, with exceptions as noted below.

It is RECOMMENDED that vendor-defined TLS-based EAP methods use the above definitions for TLS 1.3. There is no compelling reason to use different definitions.

2.2. TEAP

[RFC7170] Section 5.2 gives a definition for the Inner Method Session Key (IMSK), which depends on the TLS-PRF. When the inner methods generates an EMSK, we update that definition for TLS 1.3 as:

```
IMSK = TLS-Exporter("TEAPbindkey@ietf.org", EMSK, 32)
```

If an inner method does not support export of an Extended Master Session Key (EMSK), then IMSK is the MSK of the inner method as per [RFC7170] Section 5.2.

For MSK and EMSK, TEAP [RFC7170] uses an inner tunnel EMSK to calculate the outer EMSK. As such, those key derivations cannot use the above derivation.

The other key derivations for TEAP are given here. All derivations not given here are the same as given above in the previous section. These derivations are also used for FAST, but using the FAST Type.

```
session_key_seed = TLS-Exporter("EXPORTER: session key seed",
                                Type, 40)

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

Where | denotes concatenation. The outer MSK and EMSK are then derived from the above definitions, as:

```
MSK = TLS-Exporter("EXPORTER: Session Key Generating Function",
                   S-IMCK[j], 64)

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

The TEAP Compound MAC defined in [RFC7170] Section 5.3 is updated to use the definition of CMK[j] given above, which then leads to the following definition

```
CMK = CMK[j]

Compound-MAC = MAC( CMK, BUFFER )
```

where j is the number of the last successfully executed inner EAP method. For TLS 1.3, the message authentication code (MAC) is computed with the HMAC algorithm negotiated for HKDF in the key schedule, as per section 7.1 of RFC 8446. The definition of BUFFER is unchanged from [RFC7170] Section 5.3

2.3. FAST

For FAST, the session_key_seed is also part of the key_block, as defined in [RFC4851] Section 5.1.

The definition of S-IMCK[n], MSK, and EMSK are the same as given above for TEAP. We reiterate that the EAP-FAST Type must be used when deriving the session_key_seed, and not the TEAP Type.

Unlike [RFC4851] Section 5.2, the definition of IMCK[j] places the reference to S-IMCK after the textual label, and the concatenates the

IMSK instead of MSK.

EAP-FAST previously used a PAC, which is a session ticket that contains a pre-shared key (PSK) along with other data. As TLS 1.3 allows session resumption using a PSK, the use of a PAC is deprecated for EAP-FAST in TLS 1.3. PAC provisioning [RFC5422] is also no longer part of EAP-FAST when TLS 1.3 is used.

The T-PRF given in [RFC4851] Section 5.5 is not used for TLS 1.3. Instead, it is replaced with the TLS 1.3 TLS-Exporter function.

2.4. TTLS

[RFC5281] Section 11.1 defines an implicit challenge when the inner methods of CHAP [RFC1994], MS-CHAP [RFC2433], or MS-CHAPv2 [RFC2759] are used. The derivation for TLS 1.3 is instead given as

```
EAP-TTLS_challenge = TLS-Exporter("ttls challenge",, n)
```

There is no "context_value" ([RFC8446] Section 7.5) passed to the TLS-Exporter function. The value "n" given here is the length of the data required, which [RFC5281] requires it to be 17 octets for CHAP (Section 11.2.2) and MS-CHAP-V2 (Section 11.2.4), and to be 9 octets for MS-CHAP (Section 11.2.3).

Note that unlike TLS 1.2 and earlier, the calculation of TLS-Exporter depends on the length passed to it. Implementations therefore MUST pass the correct length instead of passing a large length and truncating the output. Any output calculated using a larger length value, and which is then truncated, will be different from the output which was calculated using the correct length.

2.4.1. Client Certificates

[RFC5281] Section 7.6 permits "Authentication of the client via client certificate during phase 1, with no additional authentication or information exchange required.". This practice is forbidden when TTLS is used with TLS 1.3. If there is a requirement to use client certificates with no inner tunnel methods, then EAP-TLS should be used instead of TTLS.

The use of client certificates is still permitted when using TTLS with TLS 1.3. However, if the client certificate is accepted, then the EAP peer MUST proceed with additional authentication of Phase 2, as per [RFC5281] Section 7.2 and following. If there is no Phase 2 data, then the EAP server MUST reject the session.

2.5. PEAP

When PEAP uses crypto binding, it uses a different key calculation defined in [PEAP-MPPE] which consumes inner method keying material. The pseudo-random function (PRF+) used here is not taken from the TLS exporter, but is instead calculated via a different method which is given in [PEAP-PRF]. That derivation remains unchanged in this specification.

However, the key calculation uses a PEAP Tunnel Key [PEAP-TK] which is defined as:

```
... the TK is the first 60 octets of the Key_Material, as
specified in [RFC5216]: TLS-PRF-128 (master secret, "client EAP
encryption", client.random || server.random).
```

We note that this text does not define Key_Material. Instead, it defines TK as the first octets of Key_Material, and gives a definition of Key_Material which is appropriate for TLS versions before TLS 1.3.

For TLS 1.3, the TK should be derived from the Key_Material defined above in Section 2.1, instead of using the TLS-PRF-128 derivation given above.

2.5.1. Client Certificates

As with TTLS, [PEAP] permits the use of client certificates in addition to inner tunnel methods.

The use of client certificates is still permitted when using PEAP with TLS 1.3. However, if the client certificate is accepted, then the EAP peer MUST proceed with additional authentication of the inner tunnel. If there is no inner tunnel authentication data, then the EAP server MUST reject the session.

3. Application Data

Unlike previous TLS versions, TLS 1.3 can continue negotiation after the initial TLS handshake has been completed, which TLS 1.3 calls the "CONNECTED" state. Some implementations use a "TLS finished" determination as an indication that TLS negotiation has completed, and that an "inner tunnel" session can now be negotiated. This assumption is not always correct with TLS 1.3.

Earlier TLS versions did not always send application data along with the "TLS finished" method. It was then possible for implementations to assume that a transition to "TLS finished" also meant that there

was no application data available, and that another round trip was required. This assumption is not true with TLS 1.3, and applications relying on that behavior will not operate correctly with TLS 1.3.

As a result, implementations MUST check for application data once the TLS session has been established. This check MUST be performed before proceeding with another round trip of TLS negotiation. If application data is available, it MUST be processed according to the relevant resumption and/or EAP type.

TLS 1.3 also permits NewSessionTicket messages to be sent before the TLS "Finished", and after application data is sent. This change can cause many implementations to fail in a number of different ways, due to a reliance on implicit behavior seen in earlier TLS versions.

In order to correct this failure, we require that if the underlying TLS connection is still performing negotiation, then implementations MUST NOT send, or expect to receive application data in the TLS session. Implementations MUST delay processing of application data until such time as the TLS negotiation has finished. If the TLS negotiation is successful, then the application data can be examined. If the TLS negotiation is unsuccessful, then the application data is untrusted, and therefore MUST be discarded without being examined.

The default for many TLS library implementations is to send a NewSessionTicket message immediately after, or along with, the TLS Finished message. This ticket could be used for resumption, even if the "inner tunnel" authentication has not been completed. If the ticket could be used, then it could allow a malicious EAP peer to completely bypass the "inner tunnel" authentication.

Therefore, the EAP server MUST NOT permit any session ticket to successfully resume authentication, unless the inner tunnel authentication has completed successfully. The alternative would allow an attacker to bypass authentication by obtaining a session ticket, and then immediately closing the current session, and "resuming" using the session ticket.

To protect against that attack, implementations SHOULD NOT send NewSessionTicket messages until the "inner tunnel" authentication has completed. There is no reason to send session tickets which will later be invalidated or ignored. However, we recognize that this suggestion may not always be possible to implement with some available TLS libraries. As such, EAP servers MUST take care to either invalidate or discard session tickets which are associated with sessions that terminate in EAP Failure.

The NewSessionTicket message SHOULD also be sent along with other

application data, if possible. Sending that message alone prolongs the packet exchange to no benefit.

[RFC9190] Section 2.5 requires a protected result indicator which indicates that TLS negotiation has finished. Methods which use "inner tunnel" methods MUST instead begin their "inner tunnel" negotiation by sending Type-specific application data.

3.1. Identities

[RFC9190] Sections 2.1.3 and 2.1.7 recommend the use of anonymous Network Access Identifiers (NAIs) [RFC7542] in the EAP Identity Response packet. However, as EAP-TLS does not send application data inside of the TLS tunnel, that specification does not address the subject of "inner" identities in tunneled EAP methods. This subject must, however, be addressed for the tunneled methods.

Using an anonymous NAI as per [RFC7542] Section 2.4 has two benefits. First, an anonymous identity makes it more difficult to track users. Second, an NAI allows the EAP session to be routed in an AAA framework as described in [RFC7542] Section 3.

For the purposes of tunneled EAP methods, we can therefore view the outer TLS layer as being mainly a secure transport layer. That transport layer is responsible for getting the actual (inner) authentication credentials securely from the EAP peer to the EAP server. As the outer identity is often used as an anonymous routing identifier for AAA ([RFC7542] Section 3), there is little reason for it to be the same as the inner identity. We therefore have a few recommendations on the inner identity, and its relationship to the outer identity.

For the purpose of this section, we define the inner identity as the identification information carried inside of the TLS tunnel. For PEAP, that identity may be an EAP Response Identity. For TTLS, it may be the User-Name attribute. Vendor-specific EAP methods which use TLS will generally also have an inner identity.

Implementations MUST NOT use anonymous identities for the inner identity. If anonymous network access is desired, eap peers MUST use EAP-TLS without peer authentication, as per [RFC9190] section 2.1.5. EAP servers MUST cause authentication to fail if an EAP peer uses an anonymous "inner" identity for any TLS-based EAP method.

Implementations SHOULD NOT use inner identities which contain an NAI realm. The outer identity contains an NAI realm, which ensures that the inner authentication method is routed to the correct destination. As such, any NAI realm in the inner identity is almost always

redundant.

However, if the inner identity does contain an NAI realm, the inner realm SHOULD be either an exact copy of the outer realm, or be a subdomain of the outer realm. The inner realm SHOULD NOT be from a different realm than the outer realm. There are very few reasons for those realms to be different.

In general, routing identifiers should be associated with the authentication data that they are routing. For example, if a user has an inner identity of "user@example.com", then it generally makes no sense to have an outer identity of "@example.org". The authentication request would then be routed to the "example.org" domain, which may have no idea what to do with the credentials for "user@example.com". At best, the authentication request would be discarded. At worst, the "example.org" domain could harvest user credentials for later use in attacks on "example.com".

In addition, associating disparate inner/outer identities in the same EAP authentication session means that otherwise unrelated realms are tied together, which can make networks more fragile.

For example, an organization which uses a "hosted" AAA provider may choose to use the realm of the AAA provider as the outer identity. The inner identity can then be fully qualified: user name plus realm of the organization. This practice can result in successful authentications, but it has difficulties.

Other organizations may host their own AAA servers, but use a "cloud" identity provider to hold user accounts. In that situation, the organizations may use their own realm as the outer (routing) identity, then use an identity from the "cloud" provider as the inner identity. This practice is NOT RECOMMENDED. User accounts for an organization should be qualified as belonging to that organization, and not to an unrelated third party.

Both of these practices mean that changing "cloud" providers is difficult. When such a change happens, each individual supplicant must be updated with a different outer identity which points to the new "cloud" provider. This process can be expensive, and some supplicants may not be online when this changeover happens. The result could be devices or users who are unable to obtain network access, even if all relevant network systems are online and functional.

Further, standards such as [RFC7585] allow for dynamic discovery of home servers for authentication. That specification has been widely deployed, and means that there is minimal cost to routing

authentication to a particular domain. The authentication can also be routed to a particular identity provider, and changed at will, with no loss of functionality. That specification is also scalable, in that it does not require changes to many systems when a domain updates its configuration. Instead, only one thing has to change: the configuration of that domain. Everything else is discovered dynamically.

That is, changing the configuration for one domain is significantly simpler and more scalable than changing the configuration for potentially millions of end-user devices.

We recognize that there may be existing use-cases where the inner and outer identities use different realms. As such, we cannot forbid that practice. We hope that the discussion above shows not only why such practices are problematic, but also that it shows how alternative methods are more flexible, more scalable, and are easier to manage.

4. Resumption

[RFC9190] Section 2.1.3 defines the process for resumption. This process is the same for all TLS-based EAP types. The only practical difference is that the value of the Type field is different.

Note that if resumption is performed, then the EAP server MUST send the protected success result indicator (one octet of 0x00) inside the TLS tunnel as per [RFC9190]. If either peer or server instead initiates an inner tunnel method, then that method MUST be followed, and resumption MUST NOT be used. The EAP peer MUST in turn check for the existence the protected success result indicator (one octet of 0x00), and cause authentication to fail if that octet is not received.

All TLS-based EAP methods support resumption, as it is a property of the underlying TLS protocol. All EAP servers and peers MUST support resumption for all TLS-based EAP methods. We note that EAP servers and peers can still choose to not resume any particular session. For example, EAP servers may forbid resumption for administrative, or other policy reasons.

It is RECOMMENDED that EAP servers and peers enable resumption, and use it where possible. The use of resumption decreases the number of round trips used for authentication. This decrease leads to lower latency for authentications, and less load on the EAP server. Resumption can also lower load on external systems, such as databases which contain user credentials.

As the packet flows for resumption are essentially identical across all TLS-based EAP types, it is technically possible to authenticate using EAP-TLS (Type 13), and then perform resumption using another EAP type, just as EAP-TTLS (Type 21). However, there is no practical benefit to doing so. It is also not clear what this behavior would mean, or what (if any) security issues there may be with it. As a result, this behavior is forbidden.

EAP servers therefore MUST NOT resume sessions across different EAP Types, and EAP servers MUST reject resumptions in which the EAP Type value is different from the original authentication.

5. Implementation Status

TTLS and PEAP are implemented and tested to be inter-operable with wpa_supplicant 2.10 and Windows 11 as clients, and FreeRADIUS 3.0.26 and Radiator as RADIUS servers.

The wpa_supplicant implementation requires that a configuration flag be set "tls_disable_tlsv1_3=0", and describes the flag as "enable TLSv1.3 (experimental - disabled by default)". However, interoperability testing shows that PEAP and TTLS both work with Radiator and FreeRADIUS.

Implementors have demonstrated significant interest in getting PEAP and TTLS working for TLS 1.3, but less interest in EAP-FAST and TEAP. As such, there is no implementation experience with EAP-FAST or TEAP. However, we believe that the definitions described above are correct, and are workable.

6. Security Considerations

[RFC9190] Section 5 is included here by reference.

Updating the above EAP methods to use TLS 1.3 is of high importance for the Internet Community. Using the most recent security protocols can significantly improve security and privacy of a network.

In some cases, client certificates are not used for TLS-based EAP methods. In those cases, the user is authenticated only after successful completion of the inner tunnel authentication. However, the TLS protocol may send one or more NewSessionTicket after receiving the TLS Finished message from the client, and therefore before the user is authenticated.

This separation of data allows for a "time of use, time of check" security issue. Malicious clients can begin a session and receive a NewSessionTicket. The malicious client can then abort the

authentication session, and the obtained NewSessionTicket to "resume" the previous session.

As a result, EAP servers MUST NOT permit sessions to be resumed until after authentication has successfully completed. This requirement may be met in a number of ways. For example, by not caching the session ticket until after authentication has completed, or by marking up the cached session ticket with a flag stating whether or not authentication has completed.

For PEAP, some derivations use HMAC-SHA1 [PEAP-MPPE]. In the interests of interoperability and minimal changes, we do not change that derivation, as there are no known security issues with HMAC-SHA1. Further, the data derived from the HMAC-SHA1 calculations is exchanged inside of the TLS tunnel, and is visible only to users who have already successfully authenticated. As such, the security risks are minimal.

6.1. Protected Success and Failure indicators

[RFC9190] provides for protected success and failure indicators as discussed in Section 4.1.1 of [RFC4137]. These indicators are provided for both full authentication, and for resumption.

Other TLS-based EAP methods provide these indicators only for resumption.

For full authentication, the other TLS-based EAP methods do not provide for protected success and failure indicators as part of the outer TLS exchange. That is, the protected result indicator is not used, and there is no TLS-layer alert sent when the inner authentication fails. Instead, there is simply either an EAP-Success or EAP-Failure sent. This behavior is the same as for previous TLS versions, and therefore introduces no new security issues.

We note that most TLS-based EAP methods provide for success and failure indicators as part of the authentication exchange performed inside of the TLS tunnel. These indicators are therefore protected, as they cannot be modified or forged.

However, some inner methods do not provide for success or failure indicators. For example, the use of TTLS with inner PAP or CHAP. Those methods send authentication credentials to the server via the inner tunnel, with no method to signal success or failure inside of the tunnel.

There are functionally equivalent authentication methods which can be used to provide protected indicators. PAP can often be replaced with

EAP-GTC, and CHAP with EAP-MD5. Both replacement methods provide for similar functionality, and have protected success and failure indicator. The main cost to this change is additional round trips.

It is RECOMMENDED that implementations deprecate inner tunnel methods which do not provided protected success and failure indicators. Implementations SHOULD use EAP-GTC instead of PAP, and EAP-MD5 instead of CHAP. New TLS-based EAP methods MUST provide protected success and failure indicators inside of the TLS tunnel.

When the inner authentication protocol indicates that authentication has failed, then implementations MUST fail authentication for the entire session. There MAY be additional protocol exchanges in order to exchange more detailed failure indicators, but the final result MUST be a failed authentication. As noted earlier, any session tickets for this failed authentication MUST be either invalidated or discarded.

Similarly, when the inner authentication protocol indicates that authentication has succeed, then implementations SHOULD cause authentication to succeed for the entire session. There MAY be additional protocol exchanges in order which could cause other failures, so success is not required here.

In both of these cases, the EAP server MUST send an EAP-Failure or EAP-Success message, as indicated by Section 2, item 4 of [RFC3748]. Even though both parties have already determined the final authentication status, the full EAP state machine must still be followed.

7. IANA Considerations

This section provides guidance to the Internet Assigned Numbers Authority (IANA) regarding registration of values related to the TLS-based EAP methods for TLS 1.3 protocol in accordance with [RFC8126].

This memo requires IANA to add the following labels to the TLS Exporter Label Registry defined by [RFC5705]. These labels are used in the derivation of Key_Material and Method-Id as defined above in Section 2.

The labels below need to be added to the "TLS Exporter Labels" registry. These labels are used only for TEAP.

- * EXPORTER: session key seed
- * EXPORTER: Inner Methods Compound Keys
- * EXPORTER: Session Key Generating Function
- * EXPORTER: Extended Session Key Generating Function

* TEAPbindkey@ietf.org

8. References

8.1. Normative References

[RFC2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March, 1997, <<http://www.rfc-editor.org/info/rfc2119>>.

[RFC3748]

Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC5216]

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

[RFC5247]

Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008,

[RFC5705]

Rescorla, E., "Keying Material Exporters for Transport Layer Security (TLS)", RFC 5705, March 2010

[RFC7170]

Zhou, H., et al., "Tunnel Extensible Authentication Protocol (TEAP) Version 1", RFC 7170, May 2014.

[RFC8126]

Cotton, M., et al., "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 8126, June 2017.

[RFC8174]

Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", RFC 8174, May 2017, <<http://www.rfc-editor.org/info/rfc8174>>.

[RFC8446]

Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, August 2018.

[RFC9190]

Mattsson, J., and Sethi, M., "Using EAP-TLS with TLS 1.3", draft-

ietf-emu-eap-tls13-18, July 2021.

[IANA]

<https://www.iana.org/assignments/eap-numbers/eap-numbers.xhtml#eap-numbers-4>

8.2. Informative References

[MSPEAP]

<https://msdn.microsoft.com/en-us/library/cc238354.aspx>

[PEAP]

Palekar, A. et al, "Protected EAP Protocol (PEAP)", draft-josefsson-pppext-eap-tls-eap-06.txt, May 2003.

[PEAP-MPPE]

https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/e75b0385-915a-4fc3-a549-fd3d06b995b0

[PEAP-PRF]

https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/0de54161-0bd3-424a-9b1a-854b4040a6df

[PEAP-TK]

https://docs.microsoft.com/en-us/openspecs/windows_protocols/MS-PEAP/41288c09-3d7d-482f-a57f-e83691d4d246

[RFC1994]

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

[RFC2433]

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

[RFC2759]

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

[RFC4137]

Vollbrecht, J., et al, "State Machines for Extensible Authentication Protocol (EAP) Peer and Authenticator ", RFC 4137, August 2005.

[RFC4851]

Cam-Winget, N., et al, "The Flexible Authentication via Secure Tunneling Extensible Authentication Protocol Method (EAP-FAST)", RFC 4851, May 2007.

[RFC5281]

Funk, P., and Blake-Wilson, S., "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.

[RFC5422]

Cam-Winget, N., et al, "Dynamic Provisioning Using Flexible Authentication via Secure Tunneling Extensible Authentication Protocol (EAP-FAST)", RFC 5422, March 2009.

[RFC7542]

DeKoK, A, "The Network Access Identifier", RFC 7542, May 2015.

[RFC7585]

Winter, S, and McCauley, M., "Dynamic Peer Discovery for RADIUS/TLS and RADIUS/DTLS Based on the Network Access Identifier (NAI)", RFC 7585, October 2015.

Acknowledgments

Thanks to Jorge Vergara for a detailed review of the requirements for various EAP types.

Thanks to Jorge Vergara, Bruno Periera Vidal, Alexander Clouter, Karri Huhtanen, and Heikki Vatiainen for reviews of this document, and for assistance with interoperability testing.

Authors' Addresses

Alan DeKok
The FreeRADIUS Server Project

Email: aland@freeradius.org

