DHC Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 12, 2015

S. Jiang, Ed.
Huawei Technologies Co., Ltd
S. Shen
CNNIC
D. Zhang

T. Jinmei
Infoblox Inc.
June 10, 2015

# Secure DHCPv6
# draft-ietf-dhc-sedhcpv6-08

## Abstract

The Dynamic Host Configuration Protocol for IPv6 (DHCPv6) enables DHCPv6 servers to pass configuration parameters. It offers configuration flexibility. If not being secured, DHCPv6 is vulnerable to various attacks, particularly spoofing attacks. This document analyzes the security issues of DHCPv6 and specifies a Secure DHCPv6 mechanism for communications between DHCPv6 clients and DHCPv6 servers. This document provides a DHCPv6 client/server authentication mechanism based on sender’s public/private key pairs or certificates with associated private keys. The DHCPv6 message exchanges are protected by the signature option and the timestamp option newly defined in this document.

## Status of This Memo

Copyright Notice



Table of Contents

## 1. Introduction

The Dynamic Host Configuration Protocol for IPv6 (DHCPv6, [RFC3315]) enables DHCPv6 servers to pass configuration parameters and offers configuration flexibility. If not being secured, DHCPv6 is vulnerable to various attacks, particularly spoofing attacks.

This document analyzes the security issues of DHCPv6 in details. This document provides mechanisms for improving the security of DHCPv6 between client and server:

- o the identity of a DHCPv6 message sender, which can be a DHCPv6 server or a client, can be verified by a recipient.

- o the integrity of DHCPv6 messages can be checked by the recipient of the message.

- o anti-replay protection based on timestamps.

Note: this secure mechanism in this document does not protect outer options in Relay-Forward and Relay-Reply messages, either added by a relay agent toward a server or added by a server toward a relay agent, because they are only transported within operator networks and considered less vulnerable. Communication between a server and a relay agent, and communications between relay agents, may be secured through the use of IPsec, as described in section 21.1 in [RFC3315].

The security mechanisms specified in this document is based on sender’s public/private key pairs or certificates with associated private keys. It also integrates message signatures for the integrity and timestamps for anti-replay. The sender authentication procedure using certificates defined in this document depends on deployed Public Key Infrastructure (PKI, [RFC5280]). However, the deployment of PKI is out of the scope of this document.

Secure DHCPv6 is applicable in environments where physical security on the link is not assured (such as over wireless) and attacks on DHCPv6 are a concern.

## 2. Requirements Language and Terminology

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 [RFC2119] when they appear in ALL CAPS. When these words are not in ALL CAPS (such as "should" or "Should"), they have their usual English meanings, and are not to be interpreted as [RFC2119] key words.

## 3. Security Overview of DHCPv6

DHCPv6 is a client/server protocol that provides managed configuration of devices. It enables a DHCPv6 server to automatically configure relevant network parameters on clients. In the basic DHCPv6 specification [RFC3315], security of DHCPv6 messages can be improved.

The basic DHCPv6 specifications can optionally authenticate the origin of messages and validate the integrity of messages using an authentication option with a symmetric key pair. [RFC3315] relies on pre-established secret keys. For any kind of meaningful security, each DHCPv6 client would need to be configured with its own secret key; [RFC3315] provides no mechanism for doing this.

For the keyed hash function, there are two key management mechanisms. The first one is a key management done out of band, usually through some manual process. The second approach is to use Public Key Infrastructure (PKI).

As an example of the first approach, operators can set up a key database for both servers and clients from which the client obtains a key before running DHCPv6. Manual key distribution runs counter to the goal of minimizing the configuration data needed at each host.

[RFC3315] provides an additional mechanism for preventing off-network timing attacks using the Reconfigure message: the Reconfigure Key authentication method. However, this method provides little message integrity or source integrity check, and it protects only the Reconfigure message. This key is transmitted in plaintext.

In comparison, the security mechanism defined in this document allows the public key database on the client or server to be populated opportunistically or manually, depending on the degree of confidence desired in a specific application. PKI security mechanism is simpler in the local key management respect.

## 4. Overview of Secure DHCPv6 Mechanism with Public Key

This document introduces a mechanism that uses public key signatures as a mechanism for securing the DHCPv6 protocol. In order to enable DHCPv6 clients and servers to perform mutual authentication without previous key deployment, this solution provides a DHCPv6 client/server authentication mechanism based on public/private key pairs and, optionally, PKI certificates. The purpose of this design is to make it easier to deploy DHCPv6 authentication and provides protection of DHCPv6 message within the scope of whatever trust

relationship exists for the particular key used to authenticate the message.

In this document, we introduce a public key option, a certificate option, a signature option and a timestamp option with corresponding verification mechanisms. A DHCPv6 message can include a public key option, and carrying a digital signature and a timestamp option. The signature can be verified using the supplied public key. The recipient processes the payload of the DHCPv6 message only if the validation is successful: the signature validates, and a trust relationship exists for the key. Alternatively, a DHCPv6 message can include a certificate option, and also carrying a digital signature and a timestamp option. The signature can be verified by the recipient. The recipient processes the payload of the DHCPv6 message only if the validation is successful: the certificate validates, and some trust relationship exists on the recipient for the provided certificate. The end-to-end security protection can be bidirectional, covering messages from servers to clients and from clients to servers. Additionally, the optional timestamp mechanism provides anti-replay protection.

A trust relationship for a public key can be the result either of a Trust-on-first-use (TOFU) policy, or a list of trusted keys configured on the recipient.

A trust relationship for a certificate could also be treated either as TOFU or configured in a list of trusted certificate authorities, depending on the application.

TOFU can be used by a client to authenticate a server and its messages. It can be deployed without a pre-established trust relationship between the client and the server. Unlike the Reconfigure Key Authentication Protocol defined in [RFC3315], it can also be used for other DHCPv6 messages than Reconfigure, and the same single key can be used for all clients since the server does not send a secret in plain text on the wire. Overall this will provide a reasonable balance of easy deployment and moderate level of security, as long as the risk of the attack window on the first use is acceptable.

TOFU can also be used by a server to protect an existing DHCPv6 session with a particular client by preventing a malicious client from hijacking the session. In this case the server does not even have to store the client’s public key or certificate after the session; it only has to remember the public key during that particular session and check if it can verify received messages with that key. This type of authentication can be deployed without a pre-established trust relationship.

If authentication has to be provided from the initial use, the Secure DHCPv6 mechanism needs some infrastructure such as PKI so the recipient of a public key or certificate can verify it securely. It is currently a subject of further study how such an infrastructure can be integrated to DHCPv6 in a way it makes the deployment easier.

The signature on a Secure DHCPv6 message can be expected to significantly increase the size of the message. One example is normal DHCPv6 message length plus a 1 KB for a X.509 certificate and signature and 256 Byte for a signature. IPv6 fragments [RFC2460] are highly possible. In practise, the total length would be various in a large range. Hence, deployment of Secure DHCPv6 should also consider the issues of IP fragment, PMTU, etc. Also, if there are firewalls between secure DHCPv6 clients and secure DHCPv6 servers, it is RECOMMENDED that the firewalls are configured to pass ICMP Packet Too Big messages [RFC4443].

## 4.1. New Components

The components of the solution specified in this document are as follows:

- o Servers and clients using public keys in their secure DHCPv6 messages generate a public/private key pair. A DHCPv6 option that carries the public key is defined.

- o Servers and clients that use certifiicates first generate a public/private key pair and then obtain a public key certificate from a Certificate Authority that signs the public key. Another option is defined to carry the certificate.

- o A signature generated using the private key which is used by the receiver to verify the integrity of the DHCPv6 messages and then the identity of the sender.

- o A timestamp, to detect replayed packet. The secure DHCPv6 nodes need to meet some accuracy requirements and be synced to global time, while the timestamp checking mechanism allows a configurable time value for clock drift. The real time provision is out of scope of this document.

## 4.2. Support for Algorithm Agility

Hash functions are used to provide message integrity checks. In order to provide a means of addressing problems that may emerge in the future with existing hash algorithms, as recommended in [RFC4270], this document provides a mechanism for negotiating the use of more secure hashes in the future.

In addition to hash algorithm agility, this document also provides a mechanism for signature algorithm agility.

The support for algorithm agility in this document is mainly a unilateral notification mechanism from sender to recipient. A recipient MAY support various algorithms simultaneously among different senders, and the different senders in a same administrative domain may be allowed to use various algorithms simultaneously. It is NOT RECOMMENDED that the same sender and recipient use various algorithms in a single communication session.

If the recipient does not support the algorithm used by the sender, it cannot authenticate the message. In the client-to-server case, the server SHOULD reply with an AlgorithmNotSupported status code (defined in Section 5.5). Upon receiving this status code, the client MAY resend the message protected with the mandatory algorithm (defined in Section 5.3).

### 4.3. Applicability

By default, a secure DHCPv6 enabled client or server SHOULD start with secure mode by sending secure DHCPv6 messages. If the recipient is secure DHCPv6 enabled and the key or certificate authority is trusted by the recipient, then their communication would be in secure mode. In the scenario where the secure DHCPv6 enabled client and server fail to build up secure communication between them, the secure DHCPv6 enabled client MAY choose to send unsecured DHCPv6 message towards the server according to its local policies.

In the scenario where the recipient is a legacy DHCPv6 server that does not support secure mechanism, the DHCPv6 server (for all of known DHCPv6 implementations) would just omit or disregard unknown options (secure options defined in this document) and still process the known options. The reply message would be unsecured, of course. It is up to the local policy of the client whether to accept such messages. If the client accepts the unsecured messages from the DHCPv6 server, the subsequent exchanges will be in the unsecured mode.

In the scenario where a legacy client sends an unsecured message to a secure DHCPv6 enabled server, there are two possibilities depending on the server policy. If the server’s policy requires the authentication, an UnspecFail (value 1, [RFC3315]) error status code, SHOULD be returned. In such case, the client cannot build up the connection with the server. If the server has been configured to support unsecured clients, the server MAY fall back to the unsecured DHCPv6 mode, and reply unsecured messages toward the client; depending on the local policy, the server MAY continue to send the

secured reply messages with the consumption of computing resource. The resources allocated for unsecured clients SHOULD be separated and restricted.

These are all examples of how interactions can go, but there is nothing to prevent clients from behaving adaptively in response to secure messages from servers.

## 5. Extensions for Secure DHCPv6

This section describes the extensions to DHCPv6. Four new options have been defined. The new options MUST be supported in the Secure DHCPv6 message exchange.

### 5.1. Public Key Option

The Public Key option carries the public key of the sender. The format of the Public Key option is described as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      OPTION_PUBLIC_KEY        |         option-len            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                     Public Key (variable length)              .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code OPTION_PUBLIC_KEY (TBA1).

option-len Length of public key in octets.

Public Key A variable-length field containing a SubjectPublicKeyInfo object specified in [RFC5280]. The SubjectPublicKeyInfo structure is comprised with a public key and an AlgorithmIdentifier object which is specified in section 4.1.1.2, [RFC5280]. The object identifiers for the supported algorithms and the methods for encoding the public key materials (public key and parameters) are specified in [RFC3279], [RFC4055], and [RFC4491].

### 5.2. Certificate Option

The Certificate option carries the public key certificate of the client. The format of the Certificate option is described as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      OPTION_CERTIFICATE       |         option-len            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
.                    Certificate (variable length)              .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code OPTION_CERTIFICATE (TBA2).

option-len Length of certificate in octets.

Certificate A variable-length field containing certificate. The encoding of certificate and certificate data MUST be in format as defined in Section 3.6, [RFC7296]. The support of X.509 certificate - Signature (4) is mandatory.

## 5.3. Signature Option

The Signature option allows a signature that is signed by the private key to be attached to a DHCPv6 message. The Signature option could be any place within the DHCPv6 message while it is logically created after the entire DHCPv6 header and options, except for the Authentication Option. It protects the entire DHCPv6 header and options, including itself, except for the Authentication Option. The format of the Signature option is described as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     OPTION_SIGNATURE          |        option-len             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     HA-id     |     SA-id     |                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               |
|                                                               |
.                    Signature (variable length)                .
.                                                               .
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code OPTION_SIGNATURE (TBA3).

option-len 2 + Length of Signature field in octets.

HA-id Hash Algorithm id. The hash algorithm is used for computing the signature result. This design is

adopted in order to provide hash algorithm agility. The value is from the Hash Algorithm for Secure DHCPv6 registry in IANA. The support of SHA-256 is mandatory. A registry of the initial assigned values is defined in Section 8.

SA-id Signature Algorithm id. The signature algorithm is used for computing the signature result. This design is adopted in order to provide signature algorithm agility. The value is from the Signature Algorithm for Secure DHCPv6 registry in IANA. The support of RSASSA-PKCS1-v1_5 is mandatory. A registry of the initial assigned values is defined in Section 8.

Signature A variable-length field containing a digital signature. The signature value is computed with the hash algorithm and the signature algorithm, as described in HA-id and SA-id. The signature constructed by using the sender’s private key protects the following sequence of octets:

1. The DHCPv6 message header.

2. All DHCPv6 options including the Signature option (fill the signature field with zeroes) except for the Authentication Option.

The signature field MUST be padded, with all 0, to the next octet boundary if its size is not a multiple of 8 bits. The padding length depends on the signature algorithm, which is indicated in the SA-id field.

Note: if both signature and authentication option are present, signature option does not protect the Authentication Option. It allows the Authentication Option be created after signature has been calculated and filled with the valid signature. It is because both options need to apply hash algorithm to whole message, so there must be a clear order and there could be only one last-created option. In order to avoid update [RFC3315] because of changing auth option, the authors chose not include authentication option in the signature.

## 5.4. Timestamp Option

The Timestamp option carries the current time on the sender. It adds the anti-replay protection to the DHCPv6 messages. It is optional.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     OPTION_TIMESTAMP          |        option-len             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                      Timestamp (64-bit)                       |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

option-code  OPTION_TIMESTAMP (TBA4).

option-len  8, in octets.

Timestamp  The current time of day (NTP-format timestamp [RFC5905] in UTC (Coordinated Universal Time), a 64-bit unsigned fixed-point number, in seconds relative to 0h on 1 January 1900.). It can reduce the danger of replay attacks.

## 5.5. Status Codes

The following new status codes, see Section 5.4 of [RFC3315] are defined.

- o AlgorithmNotSupported (TBD5): indicates that the DHCPv6 server does not support algorithms that sender used.
- o AuthenticationFail (TBD6): indicates that the DHCPv6 client fails authentication check.
- o TimestampFail (TBD7): indicates the message from DHCPv6 client fails the timestamp check.
- o SignatureFail (TBD8): indicates the message from DHCPv6 client fails the signature check.

# 6. Processing Rules and Behaviors

This section only covers the scenario where both DHCPv6 client and DHCPv6 server are secure enabled.

## 6.1. Processing Rules of Sender

The sender of a Secure DHCPv6 message could be a DHCPv6 server or a DHCPv6 client.

The sender must have a public/private key pair in order to create Secure DHCPv6 messages. The sender may also have a public key certificate, which is signed by a CA assumed to be trusted by the recipient, and its corresponding private key.

To support Secure DHCPv6, the Secure DHCPv6 enabled sender MUST construct the DHCPv6 message following the rules defined in [RFC3315].

A Secure DHCPv6 message sent by a DHCPv6 server or a client, which may be encapsulated by a Relay-forward or Relay-reply message (see below), MUST either contain a Public Key option, which MUST be constructed as explained in Section 5.1, or a Certificate option, which MUST be constructed as explained in Section 5.2.

A Secure DHCPv6 message MUST contain one and only one Signature option, which MUST be constructed as explained in Section 5.3. It protects the message header and all DHCPv6 options except for the Authentication Option.

A Secure DHCPv6 message SHOULD contain one and only one Timestamp option, which MUST be constructed as explained in Section 5.4. The Timestamp field SHOULD be set to the current time, according to sender’s real time clock.

Relay-forward and Relay-reply messages MUST NOT contain any additional Public Key or Certificate option or Signature Option or Timestamp Option, aside from those present in the innermost encapsulated messages from the client or server.

If the sender is a DHCPv6 client, in the failure cases, it receives a Reply message with an error status code. The error status code indicates the failure reason on the server side. According to the received status code, the client MAY take follow-up action:

- o Upon receiving an AlgorithmNotSupported error status code, the client SHOULD resend the message protected with one of the mandatory algorithms.

- o Upon receiving an AuthenticationFail error status code, the client is not able to build up the secure communication with the recipient. However, there may be more than one DHCPv6 servers, one of which may send AuthenticationFail and the other of which may succeed. The client MAY use the AuthenticationFail as a hint and switch to other public key certificate if it has another one; but otherwise treat the message containing the status code as if it had not been received. But it SHOULD NOT retry with the same certificate. However, if the client decides to retransmit using

the same certificate after receiving AuthenticationFail, it MUST NOT retransmit immediately and MUST follow normal retransmission routines defined in [RFC3315].

- o Upon receiving a TimestampFail error status code, the client MAY resend the message with an adjusted timestamp according to the returned clock from the DHCPv6 server. The client SHOULD NOT change its own clock, but only compute an offset for the communication session.

- o Upon receiving a SignatureFail error status code, the client MAY resend the message following normal retransmission routines defined in [RFC3315].

### 6.2. Processing Rules of Recipient

The recipient of a Secure DHCPv6 message could be a DHCPv6 server or a DHCPv6 client. In the failure cases, either DHCPv6 server or client SHOULD NOT process the received message, and the server SHOULD reply with a correspondent error status code, while the client behaves as if no response had been received from that server. The specific behavior depends on the configured local policy.

When receiving a DHCPv6 message, except for Relay-forward and Relay-reply messages, a Secure DHCPv6 enabled recipient SHOULD discard any DHCPv6 messages that meet any of the following conditions:

- o the Signature option is absent,

- o multiple Signature options are present,

- o both the Public Key option and the Certificate option are absent,

- o both the Public Key option and the Certificate option are present.

In such failure, if the recipient is a DHCPv6 server, the server SHOULD reply an UnspecFail (value 1, [RFC3315]) error status code.

If none of the Signature, Public Key or Certificate options is present, the sender MAY be a legacy node or in unsecured mode, then, the recipient MAY fall back to the unsecured DHCPv6 mode if its local policy allows.

The recipient SHOULD first check the support of the hash and signature algorithms that the sender used. If the check fails for a client, the message SHOULD be dropped. If the check fails for a server, the server SHOULD reply with an AlgorithmNotSupported error status code, defined in Section 5.5, back to the client. If both

hash and signature algorithms are supported, the recipient then checks the authority of this sender. The recipient SHOULD also use the same algorithms in the return messages.

If a Certificate option is provided, the recipient SHOULD validate the certificate according to the rules defined in [RFC5280]. An implementation may create a local trust certificate record for verified certificates in order to avoid repeated verification procedure in the future. A certificate that finds a match in the local trust certificate list is treated as verified.

If a Public Key option is provided, the recipient SHOULD validate it by finding a matching public key from the local trust public key list, which is pre-configured or recorded from previous communications (TOFU). A local trust public key list is a data table maintained by the recipient. It stores public keys from all senders that are considered trustworthy.

When the local policy of the recipient allows the use of TOFU, if a Public Key option is provided but it is not found in the local trust public key list, the recipient MAY accept the public key. The recipient will normally store the key in the local list for subsequent DHCPv6 sessions, but it may not necessarily have to do so depending on the purpose of the authentication (see the case of authenticating a client with TOFU described in Section 4).

The message that fails authentication check, either because the certificate validation fails or because the public key is not recognized, MUST be dropped. In such failure, the DHCPv6 server SHOULD reply an AuthenticationFail error status code, defined in Section 5.5, back to the client.

The recipient MAY choose to further process messages from a sender when there is no matched public key. When a message is authenticated using a key that has not previously been seen, the recipient may, if permitted by policy, treat the sender as trustworthy and record the key for future use (i.e, TOFU).

At this point, the recipient has either recognized the authentication of the sender, or decided to drop the message. The recipient MUST now authenticate the sender by verifying the signature and checking timestamp (see details in Section 6.4), if there is a Timestamp option. The order of two procedures is left as an implementation decision. It is RECOMMENDED to check timestamp first, because signature verification is much more computationally expensive. Depending on server’s local policy, the message without a Timestamp option MAY be acceptable or rejected. If the server rejects such a message, a TimestampFail error status code, defined in Section 5.5,

should be sent back to the client. The reply message that carries the TimestampFail error status code SHOULD carry a timestamp option, which indicates the server’s clock for the client to use.

The signature field verification MUST show that the signature has been calculated as specified in Section 5.3. Only the messages that get through both the signature verifications and timestamp check (if there is a Timestamp option) are accepted as secured DHCPv6 messages and continue to be handled for their contained DHCPv6 options as defined in [RFC3315]. Messages that do not pass the above tests MUST be discarded or treated as unsecured messages. In the case the recipient is DHCPv6 server, the DHCPv6 server SHOULD reply a SignatureFail error status code, defined in Section 5.5, for the signature verification failure; or a TimestampFail error status code, defined in Section 5.5, for the timestamp check failure, back to the client.

Furthermore, the node that supports the verification of the Secure DHCPv6 messages MAY impose additional constraints for the verification. For example, it may impose limits on minimum and maximum key lengths.

Minbits The minimum acceptable key length for public keys. An upper limit MAY also be set for the amount of computation needed when verifying packets that use these security associations. The appropriate lengths SHOULD be set according to the signature algorithm and also following prudent cryptographic practice. For example, minimum length 1024 and upper limit 2048 may be used for RSA [RSA].

A Relay-forward or Relay-reply message with any Public Key, Certificate or the Signature option is invalid. The message MUST be discarded silently.

### 6.3. Processing Rules of Relay Agent

To support Secure DHCPv6, relay agents just need to follow the same processing rules defined in [RFC3315]. There is nothing more the relay agents have to do, either verify the messages from client or server, or add any secure DHCPv6 options. Actually, by definition in this document, relay agents SHOULD NOT add any secure DHCPv6 options.

### 6.4. Timestamp Check

In order to check the Timestamp option, defined in Section 5.4, recipients SHOULD be configured with an allowed timestamp Delta value, a "fuzz factor" for comparisons, and an allowed clock drift parameter. The recommended default value for the allowed Delta is

300 seconds (5 minutes); for fuzz factor 1 second; and for clock drift, 0.01 second.

Note: the Timestamp mechanism is based on the assumption that communication peers have roughly synchronized clocks, with certain allowed clock drift. So, accurate clock is not necessary. If one has a clock too far from the current time, the timestamp mechanism would not work.

To facilitate timestamp checking, each recipient SHOULD store the following information for each sender, from which at least one accepted secure DHCPv6 message is successfully verified (for both timestamp check and signature verification):

- o The receive time of the last received and accepted DHCPv6 message. This is called RDlast.

- o The timestamp in the last received and accepted DHCPv6 message. This is called TSlast.

A verified (for both timestamp check and signature verification) secure DHCPv6 message initiates the update of the above variables in the recipient’s record.

Recipients MUST check the Timestamp field as follows:

- o When a message is received from a new peer (i.e., one that is not stored in the cache), the received timestamp, TSnew, is checked, and the message is accepted if the timestamp is recent enough to the reception time of the packet, RDnew:

  -Delta < (RDnew - TSnew) < +Delta

  After the signature verification also succeeds, the RDnew and TSnew values SHOULD be stored in the cache as RDlast and TSlast.

- o When a message is received from a known peer (i.e., one that already has an entry in the cache), the timestamp is checked against the previously received Secure DHCPv6 message:

  TSnew + fuzz > TSlast + (RDnew - RDlast) x (1 - drift) - fuzz

  If this inequality does not hold or RDnew < RDlast, the recipient SHOULD silently discard the message. If, on the other hand, the inequality holds, the recipient SHOULD process the message.

  Moreover, if the above inequality holds and TSnew > TSlast, the recipient SHOULD update RDlast and TSlast after the signature

verification also successes. Otherwise, the recipient MUST NOT update RDlast or TSlast.

An implementation MAY use some mechanism such as a timestamp cache to strengthen resistance to replay attacks. When there is a very large number of nodes on the same link, or when a cache filling attack is in progress, it is possible that the cache holding the most recent timestamp per sender will become full. In this case, the node MUST remove some entries from the cache or refuse some new requested entries. The specific policy as to which entries are preferred over others is left as an implementation decision.

An implementation MAY statefully record the latest timestamps from senders. In such implementation, the timestamps MUST be strictly monotonously increasing. This is reasonable given that DHCPv6 messages are rarely misordered.

## 7. Security Considerations

This document provides new security features to the DHCPv6 protocol.

Using public key based security mechanism and its verification mechanism in DHCPv6 message exchanging provides the authentication and data integrity protection. Timestamp mechanism provides anti-replay function.

The Secure DHCPv6 mechanism is based on the pre-condition that the recipient knows the public key of the sender or the sender’s public key certificate can be verified through a trust CA. Clients may discard the DHCPv6 messages from unknown/unverified servers, which may be fake servers; or may prefer DHCPv6 messages from known/verified servers over unsigned messages or messages from unknown/unverified servers. The pre-configuration operation also needs to be protected, which is out of scope. The deployment of PKI is also out of scope.

When a recipient first encounters a new public key, it may also store the key using a Trust On First Use policy. If the sender that used that public key is in fact legitimate, then all future communication with that sender can be protected by storing the public key. This does not provide complete security, but it limits the opportunity to mount an attack on a specific recipient to the first time it communicates with a new sender.

When using TOFU, if the recipient automatically and unlimitedly stores the public key, an attacker could force the recipient to exhaust the storage by sending DHCPv6 messages with many different keys. There are several possible ways to address this concern:

First, the new public key should only be stored after the signature and timestamp validations succeed. It does not prevent the attack itself, but will at least increase the cost of mounting the attack. Another approach is that as long as a client recipient has an uninterrupted connection to a particular network medium, it could limit the number of keys that it will remember as a result of messages received on that medium. Network events like a link state transition would clear the counter, but there might also need to be a counter based on absolute time. In addition, there should probably be a mechanism for purging keys that have only been seen once after a certain period.

Downgrade attacks cannot be avoided if nodes are configured to accept both secured and unsecured messages. A future specification may provide a mechanism on how to treat unsecured DHCPv6 messages.

[RFC6273] has analyzed possible threats to the hash algorithms used in SEND. Since the Secure DHCPv6 defined in this document uses the same hash algorithms in similar way to SEND, analysis results could be applied as well: current attacks on hash functions do not constitute any practical threat to the digital signatures used in the signature algorithm in the Secure DHCPv6.

A server, whose local policy accepts messages without a Timestamp option, may have to face the risk of replay attacks.

A window of vulnerability for replay attacks exists until the timestamp expires. Secure DHCPv6 nodes are protected against replay attacks as long as they cache the state created by the message containing the timestamp. The cached state allows the node to protect itself against replayed messages. However, once the node flushes the state for whatever reason, an attacker can re-create the state by replaying an old message while the timestamp is still valid. In addition, the effectiveness of timestamps is largely dependent upon the accuracy of synchronization between communicating nodes. However, how the two communicating nodes can be synchronized is out of scope of this work.

Attacks against time synchronization protocols such as NTP [RFC5905] may cause Secure DHCPv6 nodes to have an incorrect timestamp value. This can be used to launch replay attacks, even outside the normal window of vulnerability. To protect against these attacks, it is recommended that Secure DHCPv6 nodes keep independently maintained clocks or apply suitable security measures for the time synchronization protocols.

One more consideration is that this protocol does reveal additional client information in their certificate. It means less privacy. In

current practice, the client privacy and the client authentication are mutually exclusive.

## 8. IANA Considerations

This document defines four new DHCPv6 [RFC3315] options. The IANA is requested to assign values for these four options from the DHCPv6 Option Codes table of the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters. The four options are:

The Public Key Option (TBA1), described in Section 5.1.

The Certificate Option (TBA2), described in Section 5.2.

The Signature Option (TBA3), described in Section 5.3.

The Timestamp Option (TBA4),described in Section 5.4.

The IANA is also requested to add two new registry tables to the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters. The two tables are the Hash Algorithm for Secure DHCPv6 table and the Signature Algorithm for Secure DHCPv6 table.

Initial values for these registries are given below. Future assignments are to be made through Standards Action [RFC5226]. Assignments for each registry consist of a name, a value and a RFC number where the registry is defined.

Hash Algorithm for Secure DHCPv6. The values in this table are 8-bit unsigned integers. The following initial values are assigned for Hash Algorithm for Secure DHCPv6 in this document:

| Name | Value | RFCs |
|---|---|---|
| SHA-256 | 0x01 | this document |
| SHA-512 | 0x02 | this document |

Signature Algorithm for Secure DHCPv6. The values in this table are 8-bit unsigned integers. The following initial values are assigned for Signature Algorithm for Secure DHCPv6 in this document:

| Name | Value | RFCs |
|---|---|---|
| RSASSA-PKCS1-v1_5 | 0x01 | this document |

IANA is requested to assign the following new DHCPv6 Status Codes, defined in Section 5.5, in the DHCPv6 Parameters registry maintained in http://www.iana.org/assignments/dhcpv6-parameters:

| Code | Name | Reference |
|---|---|---|
| TBD5 | AlgorithmNotSupported | this document |
| TBD6 | AuthenticationFail | this document |
| TBD7 | TimestampFail | this document |
| TBD8 | SignatureFail | this document |

## 9. Acknowledgements

The authors would like to thank Bernie Volz, Ted Lemon, Ralph Droms, Jari Arkko, Sean Turner, Stephen Kent, Thomas Huth, David Schumacher, Francis Dupont, Tomek Mrugalski, Gang Chen, Qi Sun, Suresh Krishnan, Fred Templin, Robert Elz and other members of the IETF DHC working group for their valuable comments.

This document was produced using the xml2rfc tool [RFC2629].

## 10. Change log [RFC Editor: Please remove]

draft-ietf-dhc-sedhcpv6-08: clarified what the client and the server should do if it receives a message using unsupported algorithm; refined the error code treatment regarding to AuthenticationFail and TimestampFail; added consideration on how to reduce the DoS attack when using TOFU; other general editorial cleanups. 2015-06-10.

draft-ietf-dhc-sedhcpv6-07: removed the deployment consideration section; instead, described more straightforward use cases with TOFU in the overview section, and clarified how the public keys would be stored at the recipient when TOFU is used. The overview section also clarified the integration of PKI or other similar infrastructure is an open issue. 2015-03-23.

draft-ietf-dhc-sedhcpv6-06: remove the limitation that only clients use PKI- certificates and only servers use public keys. The new text would allow clients use public keys and servers use PKI-certificates. 2015-02-18.

draft-ietf-dhc-sedhcpv6-05: addressed comments from mail list that responsed to the second WGLC. 2014-12-08.

draft-ietf-dhc-sedhcpv6-04: addressed comments from mail list. Making timestamp an independent and optional option. Reduce the serverside authentication to base on only client’s certificate.

Reduce the clientside authentication to only Leaf of Faith base on server’s public key. 2014-09-26.

draft-ietf-dhc-sedhcpv6-03: addressed comments from WGLC. Added a new section "Deployment Consideration". Corrected the Public Key Field in the Public Key Option. Added consideration for large DHCPv6 message transmission. Added TimestampFail error code. Refined the retransmission rules on clients. 2014-06-18.

draft-ietf-dhc-sedhcpv6-02: addressed comments (applicability statement, redesign the error codes and their logic) from IETF89 DHC WG meeting and volunteer reviewers. 2014-04-14.

draft-ietf-dhc-sedhcpv6-01: addressed comments from IETF88 DHC WG meeting. Moved Dacheng Zhang from acknowledgement to be co-author. 2014-02-14.

draft-ietf-dhc-sedhcpv6-00: adopted by DHC WG. 2013-11-19.

draft-jiang-dhc-sedhcpv6-02: removed protection between relay agent and server due to complexity, following the comments from Ted Lemon, Bernie Volz. 2013-10-16.

draft-jiang-dhc-sedhcpv6-01: update according to review comments from Ted Lemon, Bernie Volz, Ralph Droms. Separated Public Key/ Certificate option into two options. Refined many detailed processes. 2013-10-08.

draft-jiang-dhc-sedhcpv6-00: original version, this draft is a replacement of draft-ietf-dhc-secure-dhcpv6, which reached IESG and dead because of consideration regarding to CGA. The authors followed the suggestion from IESG making a general public key based mechanism. 2013-06-29.

# 11. References

## 11.1. Normative References

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

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, April 2002.

[RFC3315] Droms, R., Bound, J., Volz, B., Lemon, T., Perkins, C., and M. Carney, "Dynamic Host Configuration Protocol for IPv6 (DHCPv6)", RFC 3315, July 2003.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, June 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4491] Leontiev, S. and D. Shefanovski, "Using the GOST R 34.10-94, GOST R 34.10-2001, and GOST R 34.11-94 Algorithms with the Internet X.509 Public Key Infrastructure Certificate and CRL Profile", RFC 4491, May 2006.

[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, May 2008.

[RFC5905] Mills, D., Martin, J., Burbank, J., and W. Kasch, "Network Time Protocol Version 4: Protocol and Algorithms Specification", RFC 5905, June 2010.

[RFC7296] Kaufman, C., Hoffman, P., Nir, Y., Eronen, P., and T. Kivinen, "Internet Key Exchange Protocol Version 2 (IKEv2)", STD 79, RFC 7296, October 2014.

## 11.2. Informative References

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, June 1999.

[RFC4270] Hoffman, P. and B. Schneier, "Attacks on Cryptographic Hashes in Internet Protocols", RFC 4270, November 2005.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[RFC6273] Kukec, A., Krishnan, S., and S. Jiang, "The Secure Neighbor Discovery (SEND) Hash Threat Analysis", RFC 6273, June 2011.

[RSA] RSA Laboratories, "RSA Encryption Standard, Version 2.1, PKCS 1", November 2002.

Authors' Addresses

Sheng Jiang (editor)
Huawei Technologies Co., Ltd
Q14, Huawei Campus, No.156 Beiqing Road
Hai-Dian District, Beijing, 100095
CN

Email: jiangsheng@huawei.com

Sean Shen
CNNIC
4, South 4th Street, Zhongguancun
Beijing 100190
CN

Email: shenshuo@cnnic.cn

Dacheng Zhang
Beijing
CN

Email: dacheng.zhang@gmail.com

Tatuya Jinmei
Infoblox Inc.
3111 Coronado Drive
Santa Clara, CA
US

Email: jinmei@wide.ad.jp

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

M. Miller
S. Nandakumar
Cisco Systems, Inc.
July 06, 2015

# Key Discovery Service
# draft-miller-saag-key-discovery-00

## Abstract

A typical requirement with any cryptographic key management system is to provide discovery, retrieval, distribution, and management of keys across entities needing to perform the necessary security operations. However there exists no standard mechanism to automatically discovery the keys, but rather the keys are either provisioned statically or shared beforehand via non standard mechanisms. This document defines machanisms for an entity to automatically discover the key(s) associated with other entities using the WebFinger protocol.

## Status of This Memo

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

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

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

This Internet-Draft will expire on January 7, 2016.

## Copyright Notice

Table of Contents



## 1. Introduction

With the increase in efforts towards ensuring end to end encryption for communications on the Internet, it has become necessary to improve the experience around how cryptographic primitives such as keys and certificates are discovered, distributed, and mananged. Efforts such as [I-D.barnes-acme] attempts to automate aspects of certificate retrieval and manangement, whereas efforts such as [I-D.abiggs-saag-key-management-service] provides mechanisms for dealing with keys required for secure group communications. However, today's standard efforts lack mechanisms for easy discovery of keys associated with an entity or a resource on the Internet. For example, any public key cryptograpy based system relies on being able to have acquired the public key(s) of the target entity in order to establish a secure communication with that entity. For these scenarios, the entities wanting to acquire such keys are either provisioned with the keys statically (as part of the configuration) or distributed by non standard (application specific) means.

This document describes mechanisms for entities to automatically discover the cryptographic keys associated with entities (users/

resources) using WebFinger [RFC7033] as the protocol mechanism. Such a mechanism provides an added benefit of separating key discovery from its retrieval and management.

The rest of this document is organized as follows. Section 3 shows using WebFinger protocol for entity's public key using an 'acct' URI [RFC7565], followed by Section 4 showing the same procedure for retrieving a secret key for a file resource.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 3. Locating an Entity's Public Keys

The examples below show query and response on the WebFinger resource for retrieving the public key(s), using an 'acct' URI [RFC7565]:

Query WebFinger:

```
GET /.well-known/webfinger?
    resource=acct%3Abilbo.baggins%40hobbiton.example
    HTTP/1.1
Host: hobbiton.example
```

The WebFinger response then includes links to the entity's public keys:

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: hobbiton.example
Access-Control-Allow-Methods: GET OPTIONS
Content-Type: application/jrd+json

{
  "subject": "acct:bilbo.baggins@hobbiton.example",
  ...
  "links": [
    ...
    {
      "rel":  "public-key",
      "href": "https://hobbiton.example/~bilbo.baggins/
               pubkeyset.json",
      "type": "application/jwk-set+json"
    }
  ]
}
```

The "rel" value is ’public-key’. The "href" MUST be a HTTPS URI that the entity’s public key(s) is retrieved from, formatted as a JWK or JWK-set (as defined in [RFC7517]):

```
{
  "keys": [
    {
      "kty": "EC",
      "kid": "bilbo.baggins@hobbiton.example",
      "use": "sig",
      "crv": "P-521",
      "x":   "AHKZLLOsCOzz5cY97ewNUajB957y-C-U88c3v13nmGZx6sYl_o
              JXu9A5RkTKqjqvjyekWF-7ytDyRXYgCF5cj0Kt",
      "y":   "AdymlHvOiLxXkEhayXQnNCvDX4h9htZaCJN34kfmC6pV5OhQHi
              raVySsUdaQkAgDPrwQrJmbnX9cwlGfP-HqHZR1"
    },
    {
      "kty": "RSA",
      "kid": "bilbo.baggins@hobbiton.example",
      "use": "sig",
      "n":   "n4EPtAOCc9AlkeQHPzHStgAbgs7bTZLwUBZdR8_KuKPEHLd4rH
              VTeT-O-XV2jRojdNhxJWTDvNd7nqQ0VEiZQHz_AJmSCpMaJMRB
              SFKrKb2wqVwGU_NsYOYL-QtiWN2lbzcEe6XC0dApr5ydQLrHqk
              HHig3RBordaZ6Aj-oBHqFEHYpPe7Tpe-OfVfHd1E6cS6M1FZcD
              1NNLYD5lFHpPI9bTwJlsde3uhGqC0ZCuEHg8lhzwOHrtIQbS0F
              Vbb9k3-tVTU4fg_3L_vniUFAKwuCLqKnS2BYwdq_mzSnbLY7h_
              qixoR7jig3__kRhuaxwUkRz5iaiQkqgc5gHdrNP5zw",
      "e":   "AQAB"
    },
    {
      "kty": "RSA",
      "kid": "bilbo.baggins@hobbiton.example",
      "use": "enc",
      "e":   "AQAB",
      "n":   "uTWZBa8bjLQNJ9cBrdxGV_H_pmHEDuAXpCR1NnyYQYkUGJ8F3a
              y_OM6sw82fS2ZcAXHpCVYlp30pd4D6BYwwixDt_eSkY-NLhPA3
              ouE4YwtaUVZYBZT909pISRK4WOr3nXeJ0lltrgPQ7StBR1C776
              KJnsHbBPdXO7tpAfph9GnjNUJxrpoFmhiZx3hbpEUpsxTsDuB9
              doVN9cFCpsjPpoiAvkr_Doyckbi1TnR4zwzDQyfSkhNYghFuqh
              vAQQ8yMQ29HOHYdfON2Z8yCjgAnyJCs1lnywkYaAaZGyxhozXr
              F6_Np2BHteL_XRNekhY72gt1nRZYCQArjJMACx_3iw"
    }
  ]
}
```


## 4. Locating a Resource's Key

The example below shows WebFinger query and response for retrieving the secret key associated with a resource controlled by erebor.com, identified using a 'key' URI scheme defined in Section 5.1. The URI to query could have been determined as per Appendix A.

Query WebFinger:

```
GET /.well-known/webfinger/
    ?resource=key%3Asha-256.GJa85ytSaK1pX6uwyBIEZFRLn5ZjrDd36emx
    NmAGP_s@erebor.eample
    HTTP/1.1
Host: erebor.example
```

WebFinger Response: The "rel" value is ’secret-key’. The "href" indicates where to retrieve the secret key.

```
HTTP/1.1 200 OK
Access-Control-Allow-Origin: hobbiton.example
Access-Control-Allow-Methods: GET OPTIONS
Content-Type: application/jrd+json

{
  "subject": "key:sha-256.GJa85ytSaK1pX6uwyBIEZFRLn5ZjrDd36emxNmAGP_s
              @erebor.eample",
  ...
  "links": [
    ...
    {
      "rel":  "secret-key",
      "href": "kms://rivendell.example/key/
               c8e84a7d-2ae1-435a-9738-bb00e4c8dc7a",
      "type": "application/jwk+json"
    }
  ]
}
```

If "href" is an HTTPS URI, the type SHOULD be "application/jwk+json" or "application/jwk-set+json". Other protocols might use different container formats.

## 5. IANA Considerations

### 5.1. "key:" URI Scheme

In accordance with the guidelines and registration procedures for new URI schemes [RFC4395], this section provides the information needed to register the ’key’ URI scheme.

#### 5.1.1. URI Scheme Name

key

#### 5.1.2. Status

permanent

#### 5.1.3. URI Scheme Syntax

The ’key’ URI syntax is defined here in Augmented Backus-Naur Form (ABNF) [RFC5234], borrowing the ’host’ and ’unreserved’ rules from [RFC3986]:

```
keyuri = "key" ":" keyid "@" host
keyid  = 1 * unreserved
```

#### 5.1.4. URI Scheme Semantics

The ’key’ URI scheme identifies cryptographic keys provided by organizations, identified by domain name. It is used only for identification, not for interaction. A protocol (other than the one specified in this document) that employs the ’key’ URI scheme is responsible for specifying how a ’key’ URI is dereferenced in the context of that protocol.

#### 5.1.5. Encoding Considerations

- o The keyid consists of unreserved characters as defined in [RFC3986].
- o The host consists only of Unicode code points that conform to the rules in [RFC5892].
- o Internationalized domain name (IDN) labels are encoded as A-labels [RFC5890].

#### 5.1.6. Applications/Protocols That Use This URI Scheme Name

At the time of this writing, only this protocol uses the ’key’ URI scheme, in conjunction with WebFinger. However, use is not restricted to this protocol, and the scheme might be considered for use in other protocols.

### 5.1.7. Interoperability Considerations

There are no known interoperability concerns related to the use of the 'key' URI scheme.

## 6. Security Considerations

As this document is in essence a profile of WebFinger [RFC7033], all of the security considerations from that draft apply.

Because anyone with the symmetric secret key can use it for decryption, access to symmetric secret keys SHOULD require authorization. Such authorization enforcement SHOULD be at the URI for the key, and MAY also be enforced on the WebFinger query.

## 7. References

### 7.1. Normative References

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

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[RFC4395] Hansen, T., Hardie, T., and L. Masinter, "Guidelines and Registration Procedures for New URI Schemes", RFC 4395, February 2006.

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, September 2013.

## 7.2. Informative References

[I-D.abiggs-saag-key-management-service]
Biggs, A. and S. Cooley, "Key Management Service Architecture", draft-abiggs-saag-key-management-service-02 (work in progress), July 2015.

[I-D.barnes-acme]
Barnes, R., Eckersley, P., Schoen, S., Halderman, A., and J. Kasten, "Automatic Certificate Management Environment (ACME)", draft-barnes-acme-02 (work in progress), May 2015.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC6234] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, May 2011.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, May 2015.

[RFC7565] Saint-Andre, P., "The ’acct’ URI Scheme", RFC 7565, May 2015.

## Appendix A. Determining a URI from Encrypted Content for Key Discovery

In most cases, the URI on which to perform key discovery will be known. Chat rooms, conferencing services, and even shared files oft-times have a URI for addressing the resource. Occasionally protected content will be disseminated in a manner that an explicit URI cannot be known or conveyed, but a domain name for where the content originated from might be known. The following is an algorithm that can be used to determine a URI for discoverying the key is such cases.

1. Start with the encrypted content, C.

2. Perform a SHA-2 [RFC6234] hash (e.g., SHA-256) over the encrypted content, to produce I’.

3. Perform the URL-safe Base64 encoding [RFC4648] over I’ to produce I.

4. Concatenate the following to produce the URI for key discovery, U:

   * The scheme "key:";

* The name of the hash used in Step 2 (as registerd in the IANA Hash Function Textual Names registry; e.g., "sha-256"), H;

* The character "." (U+002E FULL STOP);

* The base64url-encoded hash from Step 2, I;

* The character "@" (U+0040 COMMERCIAL AT); and

* The domain name, D

Expressed as an algorithm:

U := "key:" || H || "." || BASE64URL(SHA2(C)) || "@" || D

For example, suppose one has some encrypted content for which they do not have the key, but is known to come from "erebor.example". If the SHA-256 hash of the encrypted content were (in hex):

1896bce72b5268ad695fabb0c8120464544b9f9663ac3777e9e9b13660063feb

The URI to use for key discovery is then:

key:sha-256.GJa85ytSaK1pX6uwyBIEZFRLn5ZjrDd36emxNmAGP_s@erebor.eample

From here, the receiver of the encrypted content uses the calculated URI to perform key discovery for a resource as described in Section 4.

## Authors' Addresses

Matthew Miller
Cisco Systems, Inc.

Email: mamille2@cisco.com

Suhas Nandakumar
Cisco Systems, Inc.

Email: snandaku@cisco.com

Operations Area Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 7, 2016

S. Winter
RESTENA

July 06, 2015

# A Configuration File Format for Extensible Authentication Protocol (EAP) Deployments

draft-winter-opsawg-eap-metadata-02

## Abstract

This document specifies a YANG module and derived XML and JSON file formats for transfering configuration information of deployments of the Extensible Authentication Protocol (EAP). Such configuration files are meant to be discovered, consumed and used by EAP supplicant software to achieve secure and automatic EAP configuration on the consuming device.

## Status of This Memo



## Copyright Notice

Table of Contents



# 1. Introduction

## 1.1. Problem Statement

The IETF has produced the Extensible Authentication Protocol (EAP, [RFC3748] and numerous EAP methods (for example EAP-TTLS [RFC5281], EAP-TLS [RFC5216] and EAP-pwd [RFC5931]); the methods have many properties which need to be setup on the EAP server and matched as configuration items on the EAP peer for a secure EAP deployment.

Setting up these configuration items is comparatively easy if the end-user devices which implement the EAP peer functionality are under

central administrative control, e.g. in closed enterprise environments. Group policies or device provisioning by the IT department can push the settings to user devices.

In other environments, for example "BYOD" scenarios where users bring their own devices which are not under enterprise control, or in EAP-based WISP environments (see e.g. [HS20] and [I-D.wierenga-ietf-eduroam]) where it is not desired neither for the ISP nor for his user that the device control is in the ISPs hands, configuration of EAP is significantly harder as it has to be done by potentially very non-technical end users.

Correct configuration of all EAP deployment parameters is required to make the resulting authentications

- o functional (i.e. the end user can authenticate to an EAP server at all)

- o secure (i.e. the end user device can unambiguously authenticate the EAP server prior to releasing any sensitive client-side credentials)

- o privacy-preserving (i.e. the end user is able to conceal his username from the EAP authenticator)

It would be desirable to be able to convey the EAP configuration information of a deployment in a machine parseable way to the end-user device, so that all the gory details need not be known/ understood by the user. Instead, the EAP peer software on the device could consume the configuration information and set up all EAP authentication details automatically.

However, there is currently no standard way of communicating configuration parameters about an EAP setup to the EAP peer.

This specification defines such file formats for EAP configuration metadata. The source definition is a YANG module which allows for automatic derivation of XML and JSON formats.

The specification allows for unique identification of an EAP identity provider by scoping it into a namespace and giving it a unique name inside that namespace. Using this unique identification, other configuration files (which e.g. detail the wireless media properties of an Enterprise Wi-Fi setup) can then refer to this particular instance of EAP identity information as authentication source. The contents of the EAP configuration file may also be an embedded part of those other configuration files.

### 1.2. Other Approaches

Device manufacturers sometimes have developed their own proprietary configuration formats, examples include Apple's "mobileconfig" (MIME type application/x-apple-aspen-config), Microsoft's XML schemata for EAP methods for use with the command-line "netsh" tool, or Intel's "PRO/Set Wireless" binary configuration files. The multitude of proprietary file formats and their different levels of richness in expression of EAP details create a very heterogenous and non-interoperable landscape.

New devices which would like to benefit from machine-parseable EAP configuration currently either have to choose to follow a competitor's approach and use that competitor's file format or have to develop their own. This situation is very unsatisfactory.

### 1.3. Requirements Language

In this document, several words are used to signify the requirements of the specification. The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119. [RFC2119]

### 1.4. Terminology

## 2. YANG module for EAP Metadata

### 2.1. Location of the YANG module and derived XML Schema

The schema files are currently hosted on this location:

- o YANG module: https://www.supplicants.net/site/standardisation/eap-metadata-02.yang

- o XML Schema: https://www.supplicants.net/site/standardisation/eap-metadata-02.xml

### 2.2. Description of YANG Module Elements

#### 2.2.1. Overall structure

The root element is the container 'EAPIdentityProviderList', which contains a list of 'EAPIdentityProvider' elements; these carry the actual EAP configuration information for this identity provider. In most practical applications, the 'EAPIdentityProviderList' will contain only a single element; a longer list can be used for metadata

transfers between systems or to allow users to select from a set of providers in one file.

The global uniqueness of each 'EAPIdentityProvider' is ensured by the combination of the two leafs 'NameIDFormat' which provides a namespace identifier, and 'ID' which specifies the unique name inside the namespace. The other leafs and containers in the 'EAPIdentityProvider' list are:

- o zero or one 'ValidUntil' date-and-time timestamp with an indication of possible expiry of the information in the configuration file. EAP peers importing the configuration file can use this information for example to re-assess whether the account is still valid (e.g. if the ValidUntil timestamp has passed, and authentication attempts consistently fail, the supplicant should consider the information stale and ask the user to verify his access authorisation with the EAP identity provider)

- o exactly one 'AuthenticationMethods' container with a list of EAP methods which the EAPIdentityProvider supports. This container is described in more detail in section Section 2.2.2

- o zero or one 'ProviderInfo' container can provide additional information about the EAPIdentityProvider, e.g. a logo to allow visual identification of the provider to the user in a user interface, or Acceptable Use Policies pertaining to the use of this EAP identity. This element is described in more detail in section Section 2.2.3

### 2.2.2. The 'AuthenticationMethods' container

'AuthenticationMethods' contains a sequence of 'AuthenticationMethod' groupings. Each such grouping specifies the properties of one supported authentication method of an EAPIdentityProvider. The content of this grouping is enumerated in section Section 2.2.2.1 The set of configuration parameters specified in the grouping depends on the particular EAP method to be configured.

For instance, EAP-PWD [RFC5931] does not require any server certificate parameters; EAP-FAST and TEAP are the only ones making use of Protected Access Credential (PAC) provisioning. On the other hand, properties such as outer ("anonymous") identity or the need for a trusted root Certification Authority are common to several EAP methods. The server- and client-side credential types of EAP methods are defined as a flat list of elements to choose from (see 'ServerSideCredential' and 'ClientSideCredential' below); see section Section 6.2 for a rationale.

Where the sequence of 'AuthenticationMethod' groupings contains more than one element, the order of appearance in the file indicates the server operator's preference for the supported EAP types; occurences earlier in the file indicate a more preferred authentication method.

When a consuming device receives multiple 'AuthenticationMethod' groupings inside 'AuthenticationMethods', it should attempt to install more preferred methods first. During interactive provisioning of EAP properties, if the configuration information for a preferred method is insufficient (e.g. the 'AuthenticationMethod' is EAP-TLS, but the configuration file does not contain the client certificate/private key and the device's credential store is not pre-loaded with the client's certificate), the device should query whether this more preferred method should be used (requiring the user to supplement the missing data) or whether a less-preferred method should be configured instead. In non-interactive provisioning scenarios, all methods should be tried non-interactively in order until one method can be installed; if no method can be installed in a fully automated way, provisioning is aborted.

#### 2.2.2.1. Authentication Method Properties

The 'AuthenticationMethod' grouping contains

- o exactly one 'EAPMethod' leaf, which is an enumerated integer of the EAP method identifier as assigned by IANA (typedef eap-method)

- o zero or one container 'ServerSideCredential' which defines means to authenticate the EAP server to the EAP peer (for a list of the elements comprising this container, see section Section 2.2.2.2)

- o zero or one container 'ClientSideCredential' which defines means to authenticate the EAP peer to the EAP server (for a list of the elements comprising this container, see section Section 2.2.2.3)

- o zero or more 'InnerAuthenticationMethod' lists. Occurence of this list indicates that a tunneled EAP method is in use, and that further server-side and/or client-side credentials are defined inside the tunnel. The presence of more than one 'InnerAuthenticationMethod' indicates that EAP Method Chaining is in use, i.e. that several inner EAP methods are to be executed in sequence inside the tunnel. The order of occurence of the inner EAP methods defines the chaining order of the methods.

The 'InnerAuthenticationMethod' list itself contains the same 'EAPMethod', 'ServerSideCredentials' and 'ClientSideCredentials' elements as described in the preceding list, but differs in two points:

- o It can optionally contain the leaf 'NonEAPAuthMethod' (an enumerated integer of authentication methods not based on EAP) instead of 'EAPMethod' because some tunneled EAP types do not necessarily contain EAP inside the tunnel (e.g. TTLS-PAP, TEAP). The YANG definition ensures that EAPMethod and NonEAPAuthMethod are mutually exclusive in instantiations of the YANG module.

- o It can NOT contain a further 'InnerAuthenticationMethod' because establishing a secure tunnel inside an already established secure tunnel is considered a pathological case which needs not be considered. See section Section 6.3 for a rationale.

#### 2.2.2.2. The 'ServerSideCredential' container

The server-side authentication of a mutually authenticating EAP method is typically based on X.509 certificates, which requires the EAP peer to be pre-provisioned with one or more trusted root Certification Authority (CA) prior to authenticating. A server is uniquely identified by presenting a certificate which is signed by these trusted CAs, and by the EAP peer verifying that the name of the server matches the expected one. Consequently, a (set of) CAs and a (set of) server names make up the ServerSideCredentials block.

Note that different EAP methods use different terminology when referring to trusted CA roots, server certificates, and server name identification. They also differ or have inherent ambiguity in their interpretation on where to extract the server name from (e.g. is the server name the CN part of the DistinguishedName, or is the server name one of the subjectAltName:DNS entries; what to do if there is a mismatch?). This specification introduces one single element for CA trust roots and naming; these notions map into the naming of the particular EAP methods very naturally. This specification can not remove the CN vs. sAN:DNS ambiguity in many EAP methods.

- o zero or more 'CA' lists: a Certification Authority which is trusted to sign the expected server certificate. The set of 'CA' occurences SHOULD contain self-signed root certificates to establish trust, and MAY contain additional intermediate CA certificates which ultimately root in these self-signed root CAs. A configuration file can, but SHOULD NOT include only an intermediate CA certificate (i.e. without also including the corresponding self-signed root) because trusting only an intermediate CA without being able to verify to a self-signed root is an unsupported notion in many EAP peers.

- o zero or more 'ServerID' leafs: these leafs contain the expected server names in incoming X.509 EAP server certificates. For EAP methods not using X.509 certificates for their mutual

authentication, these elements contain other string-based handles which identify the server (Example: EAP-pwd).

2.2.2.3. The 'ClientSideCredential' container

There is a variety of means to identify the EAP peer to the EAP server. EAP methods use a subset of these criteria. As with server-side credentials, the terminology for the credential type may differ slightly between EAP types. The naming convention in this specification maps nicely into the method-specific terminology. Not all the criteria make sense in all contexts; for EAP methods which do not support a criterion, configuration files SHOULD NOT contain the corresponding elements, and consumers of the file MUST ignore these elements.

Specifying any one of these elements is optional and they can occur at most once. Consumers of configuration files MUST be able to fall back to user-interactive configuration for these parts if they are not specified (e.g. ask for the username and password for an EAP method during import of the EAP configuration data). Configuration files which contain sensitive elements such as 'Password' MUST be handled with due care after the import on the device (e.g. ensure minimal file permissions, or delete the source file after installing). See also the leaf 'allow-save' below.

The leaf 'allow-save' specifies whether consumers should allow the user to save the credential persistently; if it is set to false, sensitive parts of the client-side credentials MUST NOT be persistently saved on the device. See also section Section 4 for transport security considerations.

Leaf 'AnonymousIdentity' is typically used on the outside of a tunneled EAP method and allows to specify which user identity should be used outside the tunnel. This string is not used for actual user authentication, but may contain routing hints to send the request to the right EAP server.

'UserName' contains the actual username to be used for user authentication. For tunneled EAP methods, this element SHOULD only occur in the 'InnerAuthenticationMethod's 'ClientSideCredentials' - if differing outer identities are not desired in the deployment, the 'OuterIdentity' element should be populated for the 'AuthenticationMethod' element but be populated with the actual username then.

The 'ClientCertificate' container holds a X.509 certificate and private key; if the key is protected, the 'Passphrase' leaf MAY be used to indicate the passphrase, see below

’Passphrase’ contains the passphrase needed to unlock a cryptographic credential internally on the device (i.e. it is not used itself for the actual authentication during the EAP conversation)

’Password’ contains the user’s password, or an otherwise secret string which the user needs to authenticate to the EAP server

’PAC’ contains the Protected Access Credential, typically used in EAP-FAST and TEAP.

’ProvisionPAC’ is a boolean which indicates whether a PAC should be provisioned on the first connection. Note that this specification allows to use ’ProvisionPAC’ without a CA nor ServerID in ’ServerSideCredential’. While this allows the operation mode of "Anonymous PAC Provisioning" as used in many field deployments of EAP-FAST (and is thus supported here), due to the known security vulnerabilities of anonymous PAC provisioning, this combination SHOULD NOT be used.

### 2.2.3. The ’ProviderInfo’ container

This specification needs to consider that user interaction during the installation time may be required; the user at the very least must be empowered to decide whether the configuration file was issued by a provider he has an account with; the provider may have hints for the user (e.g. which password to use for the login), or may want to display links to helpdesk pages in case the user has problems with the setup or use of his identity.

The ’ProviderInfo’ container allows to specify a range of potentially useful information for display to the user (some of which is relevant only during installation time, other pieces of information could be retained by the EAP peer implementation and displayed e.g. in case of failed authentication):

- ’DisplayName’ specifies a user-friendly name for the EAP Identity Provider. Consumers of this specification should be aware that this is simple text, and self-asserted by the producer of the configuration file. If more authoritative information about the issuer is available (e.g. if the file is signed with S/MIME and carries an Organisation name (O attribute) in the signing certificate) then the more authoritative information should be displayed with more prominence than the self-asserted one.

- ’Description’ specifies a generic descriptive text which should be displayed to the user prior to the installation of the configuration data.

- o ’ProviderLocation’ specifies the approximate geographic location(s) of the EAP Identity Provider and/or his Points of Presence. This can be useful if the configuration file contains multiple ’EAPIdentityProvider’ elements; the user device can then make an informed guess which of the Identity Providers could be a good match to suggest to the user.

- o ’ProviderLogo’ specifies the logo of the EAP Identity Provider. The same self-assertion considerations as for ’DisplayName’ above apply.

- o ’TermsOfUse’ contains terms of use to be displayed to and acknowledged by the user prior to the installation of the configuration on the user’s system

- o ’Helpdesk’ is a container with three possible sub-elements: ’EmailAddress’, ’WebAddress’ and ’Phone’, all of which can be displayed to the user and possibly retained for future debugging hints.

## 2.3. Internationalisation / Multi-language support

Some elements in this specification contain text to be displayed in User Interfaces; depending on the user’s language preferences, it would be desirable to present the information in a local language. Other elements contain contact information, and those contact points may only be able to handle requests in a number of languages; it may be desirable to present only contact points to the user which are compatible with his language capabilities.

All elements which either contain localisable text, or which point to external resources in localised languages, use the grouping ’localized-non-interactive’ or ’localized-interactive’. These groupings can occur more than once in the specification, which enables an iteration of all applicable languages. If the grouping is omitted or its ’lang’ leaf is set to "C", the instance of the element is considered a default choice which is to be displayed if no other language is a better match.

If the entire file content consistently uses only one language set, e.g. all the elements are to be treated as "default" choices, the language can also be set for the entire ’EAPIdentityProvider’ element in its own ’lang-tag’ leaf.

## 3. Derivation of formats from YANG source

The utility ’pyang’ is used to derive XML Schema (XSD) from the YANG source. The Schema for this Internet-Draft was generated with pyang 1.4.1.

## 4. Issuer Authentication, Integrity Protection and Encryption of EAP Metadata configuration files

S/MIME or underlying transport security. Nuff said :-)

## 5. XML Farget Format: File Discovery

### 5.1. By MIME-Type: application/eap-config-xml

For transports where the categorisation of file types via MIME types is possible (e.g. HTTP, E-Mail), this document assigns the MIME type

application/eap-config-xml

Edge devices can associate this MIME type to incoming files on such transports, and register the application which can consume the EAP Metadata in XML format as the default handler for this file type. By doing so, for example a single click or tap on a link to the file in the device’s browser will invoke the configuration process.

This method of discovery is analogous to the Apple "mobileconfig" discovery on recent versions of Mac OS and iOS.

### 5.2. By filename extension: .eap-config-xml

In situations where file types can not be determined by MIME type meta-information (e.g. when the file gets stored on a local filesystem), this document RECOMMENDs that EAP Metadata in XML format files be stored with the extension

.eap-config-xml

to identify the file as containing EAP Metadata configuration information in XML format. Edge devices can register the application which can consume the EAP Metadata with this file extension. By doing so, for example a single click or tap on the filename in the device’s User Interface will invoke the configuration process.

### 5.3. By network location: SCAD

## 6. Design Decisions

### 6.1. Why YANG and not directly XML, JSON or $FOO?

XML is a popular choice for EAP configurations: Microsoft’s "netsh" files, Apple’s "mobileconfig" files, the Wi-Fi Alliance’s "PerProviderSubscription Managed Object", and other vendor/SDO definitions are all using XML.

JSON file formats for EAP configuration exist as well; most notable are Google’s most recent efforts for their Chromebook Operating system.

YANG has a very rich feature set, and can codify restrictions on which element is allowed when in a much more fine-grained way than XML Schema could. Since YANG modules can be converted to XML Schema and be instantiated as XML or JSON, they can serve as an abstract notion of EAP configuration which can be deployed on consumer devices in either of those two more popular formats as needed by the device in question.

### 6.2. Shallow vs. Deep definition of EAP method properties

### 6.3. EAP tunneling inside EAP tunnels

### 6.4. Placement of ’OuterIdentity’ inside ’AuthenticationMethod’

## 7. Implementation Status

RFC Editor Note: Please remove this section and the reference to [RFC6982] prior to publication.

This section records the status of known implementations of the protocol defined by this specification at the time of posting of this Internet-Draft, and is based on a proposal described in [RFC6982]. The description of implementations in this section is intended to assist the IETF in its decision processes in progressing drafts to RFCs. Please note that the listing of any individual implementation here does not imply endorsement by the IETF. Furthermore, no effort has been spent to verify the information presented here that was supplied by IETF contributors. This is not intended as, and must not be construed to be, a catalog of available implementations or their features. Readers are advised to note that other implementations may exist.

According to [RFC6982], "this will allow reviewers and working groups to assign due consideration to documents that have the benefit of running code, which may serve as evidence of valuable experimentation and feedback that have made the implemented protocols more mature. It is up to the individual working groups to use this information as they see fit".

All of the implementations listed below interoperate from producer- to consumer-side of the EAP metadata specification.

Producers of the configuration files

o eduroam Configuration Assistant Tool

Organisation: Nicolaus Copernicus University, Torun, Poland

Implementation Name: eduroam Configuration Assistant Tool

This existing tool already produces EAP configuration files in various proprietary formats for hundreds of EAP Identity Providers. A module which produces configuration files in the XML variant as specified in an eralier revision of this draft (-00) is in production deployment.

Link to production version: https://cat.eduroam.org

Maturity: production

Coverage: entire specification; XML structure aligns with version -00 of this draft

Licensing: freely distributable with acknowledgement (BSD style)

Implementation experience: given that the specification is XML, it is easy to produce a configuration file with common XML libraries. The CAT Framework is written in PHP, which provides ample procedures to produce well-formed XML.

Contact Information: Tomasz Wolniewicz (see Section 10); the CAT software homepage at http://forge.geant.net/CAT/

Consumers of the configuration files

o Android

Organisation: Swansea University, Swansea, Wales, U.K.

Implementation Name: eduroam CAT app

An Android app, compatible with API level 18 of Android (i.e. version 4.3 and above); the app consumes the -00 revision of this specification. The information in the config files is used to push settings to the SSID 'eduroam' (hard-coded) via the WifiEnterpriseConfig API. The app is in production deployment, with a 4-four digit amount of downloads one month after launch.

Link to production version: https://play.google.com/store/apps/details?id=uk.ac.swansea.eduroamcat

Maturity: production

Coverage: entire specification; XML structure aligns with version -00 of this draft

Licensing: Apache 2.0

Implementation experience: parsing XML is rather straightfoward. The ability to verify signatures on XML files (S/MIME vs. XMLDSIG as discussed in Section 4) remains unclear at this point.

Contact Information: eduroam CAT Play Store app contact address ( playstore@eduroam.org )

- Windows

Organisation: Amebis, d.o.o.i, Kamnik, Slovenia

Implementation Name: ArnesLink

A Windows supplicant/Enterprise WiFi installer/debugging assistant. The application consumes the -02 revision of this specification. The information from the XML variant of this specification is embedded in a larger XML file. The additional parts of the overall configuration file include information regarding the SSID to configure and other useful, but not EAP-specific information. The complete set of information is used to push settings into the Windows Wi-Fi configuration via the 'netsh' tool. The app is in production deployment.

Link to production version: http://ftp.arnes.si/software/eduroam/ArnesLink/

Maturity: production

Coverage: entire specification; XML structure aligns with version -02 of this draft

Licensing: GPL

Implementation experience: parsing XML is rather straightfoward. For Wi-Fi configuration use, the lack of 802.11 specific details in the config file is an issue.

Contact Information: info@amebis.si

- o Linux: the authors of this specification are currently developing an application for UNIX-like operating systems which configure enterprise networks via the NetworkManager daemon; the application can consume the file format as defined in this draft specification (XML format) and configure the settings via Networkmanager’s D-BUS interface.

## 8. Security Considerations

## 9. IANA Considerations

IANA is requested to allocate the MIME type "application/eap-config-xml" in the MIME Media Types / application registry (see section Section 5.1). The allocation should contain the following values:

- o Name: eap-config-xml
- o Template: see Appendix A (RFC editor note: remove this appendix prior to publication; replace this line with the URL to the application as posted online)
- o Reference: RFCabcd (RFC editor note: replace with the RFC number of this document)

IANA is requested to allocate the location "TBD" in the "well-known URIs" registry. The allocation should contain the following values:

- o URI Suffix: TBD
- o Change Controller: IETF
- o Reference: RFCabcd (RFC editor note: replace with the RFC number of this document)
- o Related Information: none

IANA is requested to register the XML namespace "urn:ietf:params:xml:ns:eap-config-xml" in the "IETF XML Registry / ns". The allocation should contain the following values:

- o ID: eap-config-xml
- o URI: urn:ietf:params:xml:ns:eap-config-xml
- o Filename: https://www.iana.org/assignments/xml-registry/ns/eap-config-xml.txt (to be created by IANA)
- o Reference: RFCabcd (RFC editor note: replace with the RFC number of this document)

IANA is requested to register the XML schema "urn:ietf:params:xml:schema:eap-config-xml" in the "IETF XML Registry / schema". The allocation should contain the following values:

- o ID: eap-config-xml
- o URI: urn:ietf:params:xml:schema:eap-config-xml
- o Filename: https://www.iana.org/assignments/xml-registry/schema/eap-config-xml.xsd (to be created by IANA; current XSD file is linked to in section Section 2.1)
- o Reference: RFCabcd (RFC editor note: replace with the RFC number of this document)

## 10. Contributors

Tomasz Wolniewicz of Nicolaus Copernicus University in Torun, Poland, provided significant input into this specification.

## 11. References

### 11.1. Normative References

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

### 11.2. Informative References

[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.

[RFC5281] Funk, P. and S. Blake-Wilson, "Extensible Authentication Protocol Tunneled Transport Layer Security Authenticated Protocol Version 0 (EAP-TTLSv0)", RFC 5281, August 2008.

[RFC5931] Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, August 2010.

[RFC6982] Sheffer, Y. and A. Farrel, "Improving Awareness of Running Code: The Implementation Status Section", RFC 6982, July 2013.

[I-D.wierenga-ietf-eduroam]
Wierenga, K., Winter, S., and T. Wolniewicz, "The eduroam architecture for network roaming", draft-wierenga-ietf-eduroam-05 (work in progress), March 2015.

[HS20] Wi-Fi Alliance, "Hotspot 2.0 Technical Specification", 2012, <https://www.wi-fi.org/hotspot-20-technical-specification-v100>.

## Appendix A. Appendix A: MIME Type Registration Template

The following values will be used for the online MIME type registration at https://www.iana.org/form/media-types

Your Name: Stefan Winter

Your Email Address: stefan.winter@restena.lu

Media Type Name: Application

Subtype name: (Standards tree) eap-config-xml

Required parameters: (none)

Optional parameters: (none)

Encoding Considerations: 8-Bit text

Security Considerations: This file type carries configuration information for consumer devices. It has the potential to substantially alter the consumer's device; particularly to install a new trusted Certification Authority. Applications consuming files of this type need to be cautious to explain to the end user what is being altered, so that they understand the consequences. For further explanations, see Section 8 of draft-winter-opsawg-eap-metadata. (Note to RFC Editor: replace this reference with the RFC number of this document once known)

Interoperability Considerations: The file content is XML version 1.0 or later. The encoding SHOULD be UTF-8, but implementations consuming the file SHOULD be prepared to encounter different encodings.

Published Specification: draft-winter-opsawg-eap-metadata (Note to RFC Editor: replace this reference with the RFC number of this document once known)

Applications which use this media type: files of this type are intended for consumption by sortware on edge devices; they consume the information therein to configure authentication parameters (EAP protocol and EAP method payload configurations) which are then applied to network or application authentication scenarios.

Fragment Identifier Considerations: files of this type are expected to be transmitted in their entirety. If a reference to a specific part of the content is to be made, XML XPath expressions

are to be used. I.e. fragment identifier formats are not expected to be used.

Restrictions on Usage: none

Provisional registration: initial submission of this form will be executed after adoption in the IETF; it will be a provisional registration. Final registration will be done after IESG review.

Additional information:

Deprecated alias types for this name: none

Magic numbers: none

File extensions: eap-config-xml

Macintosh File Type Codes: TBD

Object Identifiers or OIDs: none

Intended Usage: Common (no further provisions)

Other Information/General Comment: none

Person to contact for further information:

Name: Stefan Winter

E-Mail: stefan.winter@restena.lu

Author/Change controller: IETF

DATA

## Author’s Address

Stefan Winter
Fondation RESTENA
6, rue Richard Coudenhove-Kalergi
Luxembourg 1359
LUXEMBOURG

Phone: +352 424409 1
Fax: +352 422473
EMail: stefan.winter@restena.lu
URI: http://www.restena.lu.