Network Working Group
Request for Comments: 4793
Category: Informational

M. Nystroem
RSA Security

February 2007

The EAP Protected One-Time Password Protocol (EAP-POTP)

Status of This Memo

Copyright Notice

Abstract

This document describes a general Extensible Authentication Protocol (EAP) method suitable for use with One-Time Password (OTP) tokens, and offers particular advantages for tokens with direct electronic interfaces to their associated clients. The method can be used to provide unilateral or mutual authentication, and key material, in protocols utilizing EAP, such as PPP, IEEE 802.1X, and Internet Key Exchange Protocol Version 2 (IKEv2).

Table of Contents

1. Introduction

1.1. Scope

This document describes an Extensible Authentication Protocol (EAP) [1] method suitable for use with One-Time Password (OTP) tokens, and offers particular advantages for tokens that are electronically connected to a user's computer, e.g., through a USB interface. The method can be used to provide unilateral or mutual authentication, and key material, in protocols utilizing EAP, such as PPP [10], IEEE 802.1X [11], and IKEv2 [12].

1.2. Background

A One-Time Password (OTP) token may be a handheld hardware device, a hardware device connected to a personal computer through an electronic interface such as USB, or a software module resident on a personal computer, which generates one-time passwords that may be used to authenticate a user towards some service. This document describes an EAP method intended to meet the needs of organizations wishing to use OTP tokens in an interoperable manner to authenticate users over EAP. The method is designed to be independent of particular OTP algorithms and to meet the requirements on modern EAP methods (see [13]).

The basic variant of this method provides client authentication only. This mode is only to be used within a secured tunnel. A more advanced variant provides mutual authentication, integrity protection of the exchange, protection against eavesdroppers, and establishment of authenticated keying material. Both variants allow for fast session resumption.

While this document also includes a profile of the general method for the RSA SecurID(TM) mechanism, it is described in terms of general constructions. It is therefore intended that the document will also serve as a framework for use with other OTP algorithms.

Note: The term "OTP" as used herein shall not be confused with the EAP OTP method defined in [1].

1.3. Rationale behind the Design

EAP-POTP has been designed with the intent that its messages and data elements be easily parsed by EAP implementations. This makes it easier to programmatically use the EAP method in the peer and the authenticator, reducing the need for user interactions and allowing for local generation of user prompts, when needed. In contrast, the Generic Token Card (GTC) method from [1], which uses text strings

generated by the EAP server, is intended to be interpreted and acted upon by humans. Furthermore, EAP-POTP allows for mutual authentication and establishment of keying material, which GTC does not. To retain the generic nature of GTC, the EAP-POTP method has been designed to support a wide range of OTP algorithms, with profiling expected for specific such algorithms. This document provides a profile of EAP-POTP for RSA SecurID tokens.

1.4. Relationship with EAP Methods in RFC 3748

The EAP OTP method defined in [1], which builds on [14], is an example of a particular OTP algorithm and is not related to the EAP method defined in this document, other than that a profile of EAP-POTP may be created for the OTP algorithm from [14].

The Generic Token Card EAP method defined in [1] is intended to work with a variety of OTP algorithms. The same is true for EAP-POTP, the EAP method defined herein. Advantages of profiling a particular OTP algorithm for use with EAP-POTP, compared to using EAP GTC, are described in Section 1.3.

2. Conventions Used in This Document

The key words "MUST", "MUST NOT", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", and "MAY", in this document are to be interpreted as described in RFC 2119 [2].

3. Authentication Model

The EAP-POTP method provides user authentication as defined below. Additionally, it may provide mutual authentication (authenticating the EAP server to the EAP client) and establish keying material.

There are basically three entities in the authentication method described here:

- A client, or "peer", using EAP terminology, acting on behalf of a user possessing an OTP token;
- A server, or "authenticator", using EAP terminology, to which the user needs to authenticate; and
- A backend authentication server, providing an authentication service to the authenticator.

The term "EAP server" is used here with the same meaning as in [1]. Any protocol used between the authenticator and the backend authentication server is outside the scope of this document, although

RADIUS [15] is a typical choice. It is assumed that the EAP client and the peer are located on the same host, and hence only the term "peer" is used in the following for these entities.

The EAP-POTP method assumes the use of a shared secret key, or "seed", which is known both by the user and the backend authentication server. The secret seed is stored on an OTP token that the user possesses, as well as on the authentication server.

In its most basic variant, the EAP-POTP method provides only one Service (namely, user authentication) where the user provides information to the authentication server so that the server can authenticate the user. A more advanced variant provides mutual authentication, protection against eavesdropping, and establishment of authenticated keying material.

4. Description of the EAP-POTP Method

4.1. Overview

Note: Since the EAP-POTP method is general in nature, the term "POTP-X" is used below as a placeholder for an EAP method type identifier, identifying the use of a particular OTP algorithm with EAP-POTP. As an example, in the case of using RSA SecurID tokens within EAP-POTP, the EAP method type shall be 32 (see Appendix A).

A typical EAP-POTP authentication is performed as follows (Appendix B provides more detailed examples):

a. The optional EAP Identity Request/Response is exchanged, as per RFC 3748 [1]. An identity provided here may alleviate the need for a "User Identifier" or a "Token Key Identifier" triplet (TLV), defined below, later in the exchange.

b. The EAP server sends an EAP-Request of type POTP-X with a Version TLV. The Version TLV indicates the highest and lowest version of this method supported by the server. The EAP server typically also includes an OTP TLV in the EAP-Request. The OTP TLV instructs the peer to respond with the current OTP (possibly in protected form), and may contain a challenge and some other information, like server policies. The EAP server should also include a Server-Info TLV in the request, and must do so if it supports session resumption. The Server-Info TLV identifies the authentication server, contains an identifier for this (new) session, and may be used by the peer to find an already existing session with the EAP server.

c. The peer responds with an EAP-Response of type Nak (3) if it does not support POTP-X or if it does not support a version of this method that is also supported by the server, as indicated in the server's Version TLV.

If the peer supports a version of this method that is also supported by the EAP server, the peer generates an EAP-Response of type POTP-X as follows:

* First, it generates a Version TLV, which indicates the peer's highest supported version within the range of versions offered by the server. This Version TLV will be part of the EAP-Response to the EAP server.

* Next, if the peer's highest supported version equals that of the EAP server, and the EAP server sent a Server-Info TLV, the peer checks if it has a saved session with the EAP server. If an existing session with the server is found, and session resumption is possible (the Server-Info TLV may explicitly disallow it), the peer calculates new session keys (if the session is a protected-mode session) and responds with a Resume TLV and the Version TLV.

* Otherwise, if the peer's highest supported version equals that of the EAP server, and the received EAP-Request message contains an OTP TLV, the peer requests (possibly through user interaction) the OTP token to calculate a one-time password based on the information in the received EAP-Request message (which could, for example, carry a challenge), the current token state (e.g., token time), a shared secret (the "seed"), and a user-provided PIN (note that, depending on the OTP token type, some of the information in the EAP-Request may not be used in the OTP calculation, and the PIN may be optional too). If the received OTP TLV has the P bit set (see below), the peer then combines the token-provided OTP with other information, and provides the combined data to a key derivation function. The key derivation function generates several keys, of which one is used to calculate a Message Authentication Code (MAC) on the received message, together with some other information. The resulting MAC, together with some additional information, is then placed in an OTP TLV (with the P bit set) that is sent in a response to the EAP server, together with the Version TLV. If the P bit is not set in the received OTP TLV, the peer instead inserts the calculated OTP value directly in an OTP TLV, which then is sent to the EAP server together with the Version TLV.

* Finally, if the peer's highest supported version differs from the server's, or if the server did not provide any TLVs besides the Version TLV in its initial request, the peer just sends back the generated Version TLV as an EAP-Response to the EAP server.

d. If the EAP server receives an EAP-Response of type Nak (3), the session negotiation failed and the EAP server may try with another EAP method. Otherwise, the EAP server checks the peer's supported version. If the peer did not support the highest version supported by the server, the server will send a new EAP-Request with TLVs adjusted for that version. Otherwise, assuming the EAP server did send additional TLVs in its initial EAP-Request, the EAP server will attempt to authenticate the peer based on the response provided in c). Depending on the result of this authentication, the EAP server may do one of the following:

   * send a new EAP-Request of type POTP-X to the peer indicating that session resumption was not possible, and ask for a new OTP (this would be the case when the peer responded with a Resume TLV, and the session indicated in the Resume TLV was not valid),

   * send a new EAP-Request of type POTP-X to the peer (e.g., to ask for the next OTP),

   * accept the authentication (and send an EAP-Request message containing a Confirm TLV to the peer if the received response has the P bit set or was a successful attempt at a protected-mode session resumption; otherwise, send an EAP-Success message to the peer), or

   * fail the authentication (and send an EAP-Failure message -- possibly preceded by an EAP-Request message of type Notification (2) -- to the peer).

e. If the peer receives an EAP-Success or an EAP-Failure message the protocol run is finished. If the peer receives an EAP-Request of type Notification, it responds as specified by RFC 3748 [1]. If the peer receives an EAP-Request of type POTP-X with a Confirm TLV, it attempts to authenticate the EAP server using the provided data. If the authentication is successful, the peer responds with an EAP-Response of type POTP-X with a Confirm TLV. If it is unsuccessful, the peer responds with an empty EAP-Response of type POTP-X. If the peer receives an EAP-Request of type POTP-X containing some other TLVs, it continues as specified in c) above (though no version negotiation will take place in this case) or as described for those TLVs.

f. When an EAP server, which has sent an EAP-Request of type POTP-X with a Confirm TLV, receives an EAP-Response of type POTP-X with a Confirm TLV present, it can proceed in one of two ways: If it has detected that there is a need to send additional EAP-Requests of type POTP-X, it shall enter a "protected state", where, from then on, all POTP-X TLVs must be encrypted and integrity-protected before being sent (at this point, the parties shall have calculated a master session key as described in Section 4.5). One reason to continue the POTP-X conversation after exchange of the Confirm TLV could be that the user needs to update her OTP PIN; hence, the EAP server needs to send a New PIN TLV. At that point, the handshake is back at step c) above (except for the version negotiation and the protection of all TLVs). If there is no need to send additional EAP-Request packets, the EAP server shall instead send an EAP-Success method to the peer to indicate successful protocol completion. The EAP server may not continue the conversation unless it indicates its intent to do so in the Confirm TLV.

   An EAP server, which has sent an EAP-Request of type POTP-X with a Confirm TLV and receives an EAP-Response of type POTP-X, which is empty (i.e., does not contain any TLVs), shall respond with an EAP-Failure and terminate the handshake.

As implied by the description, steps c) through f) may be carried out a number of times before completion of the exchange. One example of this is when the authentication server initially requests an OTP, accepts the response from the peer, performs an (intermediary) Confirm TLV exchange, requests the peer to select a new PIN, and finally asks the peer to authenticate with an OTP based on the new PIN (which again will be followed with a final Confirm TLV exchange).

4.2. Version Negotiation

The EAP-POTP method provides a version negotiation mechanism that enables implementations to be backward compatible with previous versions of the protocol. This specification documents the EAP-POTP protocol version 1. Version negotiation proceeds as follows:

a. In the first EAP-Request of type POTP-X, the EAP server MUST send a Version TLV in which it sets the "Highest" field to its highest supported version number, and the "Lowest" field to its lowest supported version number. The EAP server MAY include other TLV triplets, as described below, that are compatible with the "Highest" supported version number to optimize the number of round-trips in the case of a peer supporting the server’s "Highest" version number.

b. If the peer supports a version of the protocol that falls within the range of versions indicated by the EAP server, it MUST respond with an EAP-Response of type POTP-X that contains a Version TLV with the "Highest" field set to the highest version supported by the peer. The peer MUST also respond to any TLV triplets included in the EAP-Request, if it supported the "Highest" supported version indicated in the server's Version TLV.

   The EAP peer MUST respond with an EAP-Response of type Nak (3) if it does not support a version that falls within the range of versions indicated by the EAP server. This will allow the EAP server to use another EAP method for peer authentication.

c. When the EAP server receives an EAP-Response containing a Version TLV from the peer, but the "Highest" supported version field in the TLV differs from the "Highest" supported version field sent by the EAP server, or when the version is the same as the one originally proposed by the EAP server, but the EAP server did not include any TLV triplets in the initial request, the EAP server sends a new EAP-Request of type POTP-X with the negotiated version and TLV triplets as desired and described herein.

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

The EAP-POTP version field may be modified in transit by an attacker. It is therefore important that EAP entities only accept EAP-POTP versions according to an explicit policy.

4.3. Cryptographic Algorithm Negotiation

Cryptographic algorithms are negotiated through the use of the Crypto Algorithm TLV. EAP-POTP provides a default digest algorithm (SHA-256) [3], a default encryption algorithm (AES-CBC) [4] , and a default MAC algorithm (HMAC) [5], and these algorithms MUST be supported by all EAP-POTP implementations. An EAP server that does not want to make use of any other algorithms than the default ones need not send a Crypto Algorithm TLV. An EAP server that does want to negotiate use of some other algorithms MUST send the Crypto Algorithm TLV in the initial EAP-Request of type POTP-X that also contains an OTP TLV with the P bit set. The TLV MUST NOT be present in any other EAP-Request in the session. (The two exceptions to this are 1) if the client attempted a session resumption that failed and therefore did not evaluate a sent Crypto Algorithm TLV, or 2) if the

Crypto Algorithm TLV was part of the initial message from the EAP server, and the client negotiated another EAP-POTP version than the highest one supported by the EAP server. When either of these cases apply, the server MUST include the Crypto Algorithm TLV in the first EAP-Request that also contains an OTP TLV with the P bit set subsequent to the failed session resumption / protocol version negotiation.) In the Crypto Algorithm TLV, the EAP server suggests some combination of digest, encryption, and MAC algorithms. (If the server only wants to negotiate a particular class of algorithms, then suggestions for the other classes need not be present, since the default applies.)

The peer MUST include a Crypto Algorithm TLV in an EAP-Response if and only if an EAP-Request of type POTP-X has been received containing a Crypto Algorithm TLV, it was legal for that EAP-Request to contain a Crypto Algorithm TLV, the peer does not try to resume an existing session, and the peer and the EAP server agree on at least one algorithm not being the default one. If the peer does not supply a value for a particular class of algorithms in a responding Crypto Algorithm TLV, then the default algorithm applies for that class. When resuming an existing session (see the next section), there is no need for the peer to negotiate since the session already is associated with a set of algorithms. Servers MUST fail a session (i.e., send an EAP-Failure) if they receive an EAP-Response TLV containing both a Resume TLV and a Crypto Algorithm TLV.

Clearly, EAP servers and peers MUST NOT suggest any other algorithms than the ones their policy allows them to use. Policies may also restrict what combinations of cryptographic algorithms are acceptable.

4.4. Session Resumption

This method makes use of session identifiers and server identifiers to allow for improved efficiency in the case where a peer repeatedly attempts to authenticate to an EAP server within a short period of time. This capability is particularly useful for support of wireless roaming.

In order to help the peer find a session associated with the EAP server, an EAP server that supports session resumption MUST send a Server-Info TLV containing a server identifier in its initial EAP-Request of type POTP-X that also contains an OTP TLV. The identifier may then be used by the peer for lookup purposes.

It is left to the peer whether or not to attempt to continue a previous session, thus shortening the negotiation. Typically, the peer’s decision will be made based on the time elapsed since the

previous authentication attempt to that EAP server. If the peer decides to attempt to resume a session with the EAP server, it sends a Resume TLV identifying the chosen session and other contents, as described below, to the EAP server.

Based on the session identifier chosen by the peer, and the time elapsed since the previous authentication, the EAP server will decide whether to allow the session resumption, or continue with a new session.

- If the EAP server is willing to resume a previously established session, it MUST authenticate the peer based on the contents of the Resume TLV. If the authentication succeeds, the handshake will continue in one of two ways:

  * If the session is a protected-mode session, then the server MUST respond with a request containing a Confirm TLV. If the Confirm TLV authenticates the EAP server, then the peer responds with an empty Confirm TLV, to which the EAP server responds with an EAP-Success message. If the Confirm TLV does not authenticate the EAP server, the peer responds with an empty EAP-Response of type POTP-X.

  * If the session is not a protected-mode session, i.e., it is a session created from a basic-mode peer authentication, then the server MUST respond with an EAP-Success message.

  If the authentication of the peer fails, the EAP server SHOULD send another EAP-Request containing an OTP TLV and a Server-Info TLV with the N bit set to indicate that no session resumption is possible. The EAP server MAY also send an EAP-Failure message, possibly preceded by an EAP-Request of type Notification (2), in which case, the EAP run will terminate.

- If the EAP server is not willing or able to resume a previously established session, it will respond with another EAP-Request containing an OTP TLV and a Server-Info TLV with the N bit set (indicating no session resumption).

Sessions SHOULD NOT be maintained longer than the security of the exchange which created the session permits. For example, if it is estimated that an attacker could be successful in brute-force searching for the OTP in 24 hours, then EAP-POTP session lifetimes should be clearly less than this value.

4.5. Key Derivation and Session Identifiers

The EAP-POTP method described herein makes use of a key derivation function denoted "PBKDF2". PBKDF2 is described in [6], Section 5.2. The PBKDF2 PRF SHALL be set to the negotiated MAC algorithm. The default MAC algorithm, which MUST be supported, is HMAC-SHA256. HMAC is defined in [5], and SHA-256 is defined in [3]. HMAC-SHA256 is the HMAC construct from [5] with SHA-256 as the hash function H. The output length of HMAC-SHA256, when used as a PRF for PBKDF2, shall be 32 octets (i.e., the full output length).

The output from PBKDF2 as described here will consist of five keys (see Section 4.11.3 for details on how to calculate these keys):

- K_MAC, a MAC key used for mutual authentication and integrity protection,
- K_ENC, an encryption key used to protect certain data during the authentication,
- SRK, a session resumption key only used for session resumption purposes,
- MSK, a Master Session Key, as defined in [1], and
- EMSK, an Extended Master Session Key, also as defined in [1].

  For the default algorithms, K_MAC, K_ENC, and SRK SHALL be 16 octets. For other cases, the key lengths will be as determined by the negotiated algorithms. The MSK and the EMSK SHALL each be 64 octets, in conformance with [1]. Therefore, in the case of default algorithms, the "dkLen" parameter from Section 5.2 of [6] SHALL be set to 176 (the combined length of K_MAC, K_ENC, SRK, MSK, and EMSK).

[1] and [16] define usage of the MSK and the EMSK . For a particular use case, see also Appendix C.

4.6. Error Handling and Result Indications

EAP does not allow for the sending of an EAP-Response of type Nak (3) within a method after the initial EAP-Request and EAP-Response pair of that particular method has been exchanged (see [1], Section 2.1). Instead, when a peer is unable to continue an EAP-POTP session, the peer MAY respond to an outstanding EAP-Request by sending an empty EAP-Response of type POTP-X rather than immediately terminating the conversation. This allows the EAP server to log the cause of the error.

To ensure that the EAP server receives the empty EAP-Response, the peer SHOULD wait for the EAP server to reply before terminating the conversation. The EAP server MUST reply with an EAP-Failure.

When EAP-POTP is run in protected mode, the exchange of the Confirm TLV (Section 4.11.6) serves as a success result indication; when the peer receives a Confirm TLV, it knows that the EAP server has successfully authenticated it. Similarly, when the EAP server receives the Confirm TLV response from the peer, it knows that the peer has authenticated it. In protected mode, the peer will not accept an EAP-Success packet unless it has received and validated a Confirm TLV. The Confirm TLV sent from the EAP server to the peer is a "protected result indication" as defined in [1], as it is integrity protected and cannot be replayed. The Confirm TLV sent from the peer to the EAP server is, however, not a protected result indication. An empty EAP-POTP response sent from the peer to the EAP server serves as a failure result indication.

4.7. Use of the EAP Notification Method

Except where explicitly allowed in the following, the EAP Notification method MUST NOT be used within an EAP-POTP session. The EAP Notification method MAY be used within an EAP-POTP session in the following situations:

- The EAP server MAY send an EAP-Request of type Notification (2) when it has received an EAP-Response containing an OTP TLV and is unable to authenticate the user. In this case, once the EAP-Response of type Notification is received, the EAP server MAY retry the authentication and send a new EAP-Request containing an OTP TLV, or it MAY fail the session and send an EAP-Failure message.

- The EAP server MAY send an EAP-Request of type Notification (2) when it has received an unacceptable New PIN TLV. In this case, once the EAP-Response of type Notification is received, the EAP server MAY retry the PIN update and send a new EAP-Request with a New PIN TLV, or it MAY fail the session and send an EAP-Failure message.

4.8. Protection against Brute-Force Attacks

Since OTPs may be relatively short, it is important to slow down an attacker sufficiently so that it is economically unattractive to brute-force search for an OTP, given an observed EAP-POTP handshake in protected mode. One way to do this is to do a high number of iterated hashes in the PBKDF2 function. Another is for the client to include a value ("pepper") unknown to the attacker in the hash

computation. Whereas a traditional "salt" value normally is sent in the clear, this "pepper" value will not be sent in the clear, but may instead be transferred to the EAP server in encrypted form. In practice, the procedure is as follows:

a. The EAP server indicates in its OTP TLV whether it supports pepper searching. Additionally, it may indicate to the peer that a new pepper shall be chosen.

b. If the peer supports the use of pepper, the peer checks whether it already has established a shared pepper with this server:

   If it does have a pepper stored for this server, and the server did not indicate that a new pepper shall be generated, then it uses the existing pepper value, as specified in Section 4.11.3 below, to calculate an OTP TLV response. In this case, the iteration count shall be kept to a minimum, as the security of the scheme is provided through the pepper, and efficiency otherwise is lost.

   If the peer does not have a pepper stored for this server, but the server indicated support for pepper searching, or the server indicated that a new pepper shall be generated, then the peer generates a random and uniformly distributed pepper of sufficient length (the maximum length supported by the server is provided in the server’s OTP TLV), and includes the new pepper in the PBKDF2 computation.

   If the peer does not have a pepper stored for this server, and the server did not indicate support for pepper searching, then a pepper will not be used in the response computation.

   Clearly, if the peer itself does not support the use of pepper, then a pepper will not be used in the response computation.

c. The EAP server may, in its subsequent Confirm TLV, provide a pepper to the peer for later use. In this case, the pepper will be substantially longer than a peer-chosen pepper, and encrypted with a key derived from the PBKDF2 computation.

The above procedure allows for pepper updates to be initiated by either side, e.g., based on policy. Since the pepper can be seen as a MAC key, its lifetime should be limited.

An EAP server that is not capable of storing pepper values for each user it is authenticating may still support the use of pepper; the cost for this will be the extra computation time to do pepper searches. This cost is still substantially lower than the cost for

an attacker, however, since the server already knows the underlying OTP.

4.9. MAC Calculations in EAP-POTP

4.9.1. Introduction

In protected mode, EAP-POTP uses MACs for authentication purposes, as well as to ensure the integrity of protocol sessions. This section defines how the MACs are calculated and the rationale for the design.

4.9.2. MAC Calculation

In protected mode, and when resuming a previous session, rather than sending authenticating credentials (such as one-time passwords or shared keys) directly, evidence of knowledge of the credentials is sent. This evidence is a MAC on the hash of (certain parts of) EAP-POTP messages exchanged so far in a session using a key K_MAC:

mac = MAC(K_MAC, msg_hash(msg_1, msg_2, ..., msg_n))

where

"MAC" is the negotiated MAC algorithm, "K_MAC" is a key derived as specified in Section 4.5, and "msg_hash(msg_1, msg_2, ..., msg_n)" is the message hash defined below of messages msg_1, msg_2, ..., msg_n.

4.9.3. Message Hash Algorithm

To compute a message hash for the MAC, given a sequence of EAP messages msg_1, msg_2, ..., msg_n, the following operations shall be carried out:

a. Re-transmitted messages are removed from the sequence of messages.

   Note: The resulting sequence of messages must be an alternating sequence of EAP Request and EAP Response messages.

b. The contents (i.e., starting with the EAP "Type" field and excluding the EAP "Code", "Identifier", and "Length" fields) of each message, msg_1, msg_2, ..., msg_n, is concatenated together.

c. User identifier TLVs MUST NOT be included in the hash (this is to allow for a backend service that does not know about individual user names), i.e., any such TLV is removed from the message in which it appeared.

d. The resulting string is hashed using the negotiated hash algorithm.

4.9.4. Design Rationale

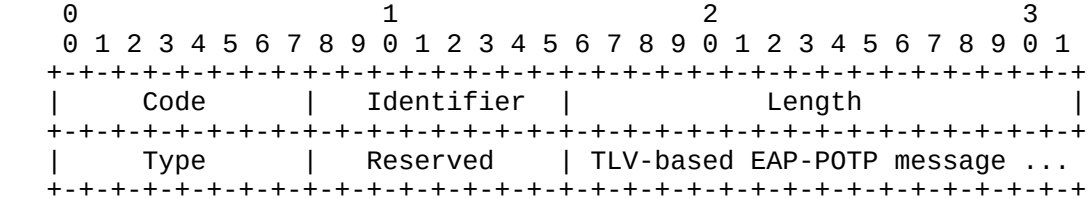
The reason for excluding the "Identifier" field is that the actual, transmitted "Identifier" field is not always known to the EAP method layer. The reason for excluding the "Length" field is to allow the possibility for an intermediary to remove or replace a Username TLV (e.g., for anonymity or service reasons) before passing a received response on to an authentication server. While this on the surface may appear as bad security practice, it may in practice only result in denial of service, something which always may be achieved by an attacker able to modify messages in transit. By excluding the "Code" field, the hash is simply calculated on applicable sent and received message contents. Excluding the "Code" field is regarded as harmless since the hash is to be made on the sequence of POTP-X messages, all having alternating (known) Code values, namely 1 (Request) and 2 (Response).

4.9.5. Implementation Considerations

To save on storage space, each EAP entity may partially hash messages as they are sent and received (e.g., HashInit(); HashUpdate(message 1); ...; HashUpdate(message n-1); HashFinal(message n)). This reduces the amount of state needed for this purpose to the internal state required for the negotiated hash algorithm.

4.10. EAP-POTP Packet Format

A summary of the EAP-POTP packet format is shown below. The fields are transmitted from left to right.



Code

1 - Request

2 - Response

Identifier

The Identifier field is 1 octet and aids in matching responses with requests. For a more detailed description of this field and how to use it, see [1].

Length

The Length field is 2 octets and indicates the length of the EAP packet including the Code, Identifier, Length, Type, Version, Flags, and TLV-based EAP-POTP message fields.

Type

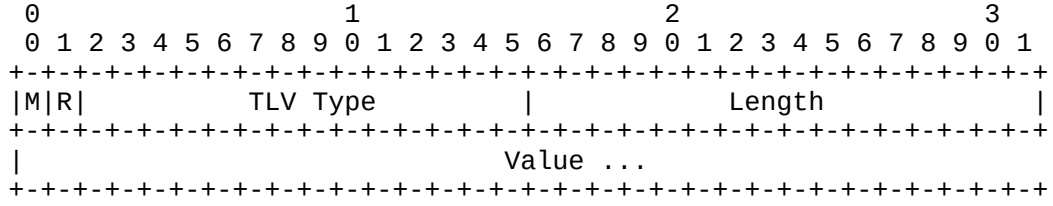
Identifies use of a particular OTP algorithm with EAP-POTP.

Reserved

This octet is reserved for future use. It SHALL be set to zero for this version. Recipients SHALL ignore this octet for this version of EAP-POTP.

TLV-based EAP-POTP message

This field will contain 0, 1, or more Type-Length-Value triplets defined as follows (this is similar to the EAP-TLV TLVs defined in PEAPv2 [17], and the explanation of the generic fields is borrowed from that document).



M

0 - Non-mandatory TLV

1 - Mandatory TLV

The TLVs within EAP POTP-X are used to carry parameters between the EAP peer and the EAP server. An EAP peer may not necessarily implement all the TLVs supported by an EAP server, and to allow for interoperability, a special TLV allows an EAP server to discover if a TLV is supported by the EAP peer.

The mandatory bit in a TLV indicates that if the peer or server does not support the TLV, it MUST send a NAK TLV in response; all other TLVs in the message MUST be ignored. If an EAP peer or server finds an unsupported TLV that is marked as non-mandatory (i.e., optional), it MUST NOT send a NAK TLV on this ground only.

The mandatory bit does not imply that the peer or server is required to understand the contents of the TLV. The appropriate response to a supported TLV with content that is not understood is defined by the specification of the particular TLV.

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of the EAP-POTP.

TLV Type

The following TLV types are defined for use with EAP-POTP:

```
 0 - Reserved for future use
 1 - Version
 2 - Server-Info
 3 - OTP
 4 - NAK
 5 - New PIN
 6 - Confirm
 7 - Vendor-Specific
 8 - Resume
 9 - User Identifier
10 - Token Key Identifier
11 - Time Stamp
12 - Counter
13 - Keep-Alive
14 - Protected
15 - Crypto Algorithm
16 - Challenge
```

These TLVs are defined in the following. With the exception of the NAK TLV, a particular TLV type MUST NOT appear more than once in a message of type POTP-X.

Length

The length of the Value field in octets.

Value

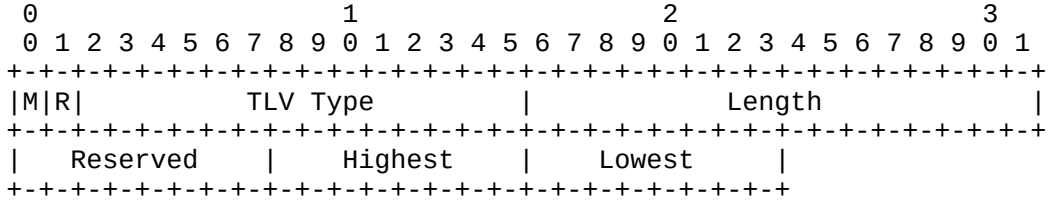
The value of the TLV.

4.11. EAP-POTP TLV Objects

4.11.1. Version TLV

The Version TLV carries information about the supported EAP-POTP method version.

This TLV MUST be present in the initial EAP-Request of type POTP-X from the EAP server and in the initial response of type POTP-X from the peer. It MUST NOT be present in any subsequent EAP-Request or EAP-Response in the session. The Version TLV MUST be supported by all peers, and all EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV. The version negotiation procedure is described in detail in Section 4.2.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

1

Length

3 in EAP-Requests, 2 in EAP-Responses

Reserved

Reserved for future use. This octet MUST be set to zero for this version. Recipients SHALL ignore this octet for this version of EAP-POTP.

Highest

This field contains an unsigned integer representing the highest protocol version supported by the sender. If a value provided by a peer to an EAP server falls between the server's "Highest" and "Lowest" supported version (inclusive), then that value will be the negotiated version for the authentication session.

Lowest

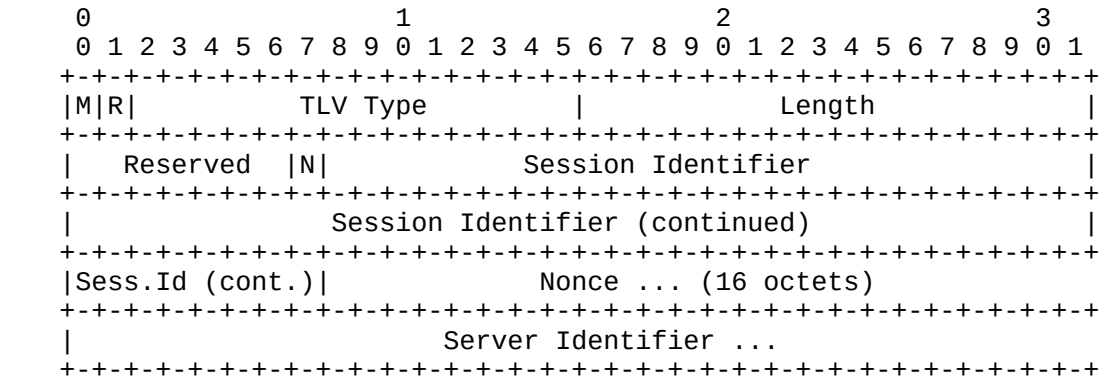
This field contains an unsigned integer representing the lowest version acceptable by the EAP server. The field MUST be present in an EAP-Request. The field MUST NOT be present in an EAP-Response. A peer SHALL respond to an EAP-Request of type POTP-X with an EAP-Response of type Nak (3) if the peer's highest supported version is lower than the value of this field.

This document defines version 1 of the protocol. Therefore, EAP server implementations conforming to this document SHALL set the "Highest" field to 1. Peer implementations conforming to this document SHALL set the "Highest" field to 1.

4.11.2. Server-Info TLV

The Server-Info TLV carries information about the EAP server and the session (when applicable). It provides one piece in the framework for fast session resumption.

This TLV SHOULD always be present in an EAP-Request of type POTP-X that also carries an OTP TLV, as long as the peer has not been authenticated, and MUST be present in such a request if the server supports session resumption. It MUST NOT be present in any other EAP-Request of type POTP-X or in any EAP-Response packets. This TLV type MUST be supported by all peers conforming to this specification and MUST NOT be responded to with a NAK TLV (this is not to say that all peers need to support session resumption, only that they cannot respond to this TLV with a NAK TLV).



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

2

Length

25 + length of Server Identifier field

Reserved

Reserved for future use. All 7 bits MUST be set to zero for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

N

The N bit signals that the peer MUST NOT attempt to resume any session it has stored associated with this server.

Session Identifier

An 8-octet identifier for the session about to be negotiated. Note that, in the case of session resumption, this session identifier will not be used (the session identifier for the resumed session will continue to be used).

Nonce

A 16-octet nonce chosen by the server. During session resumption, this nonce is used when calculating new K_ENC, K_MAC, SRK, MSK, and EMSK keys as specified below.

Server Identifier

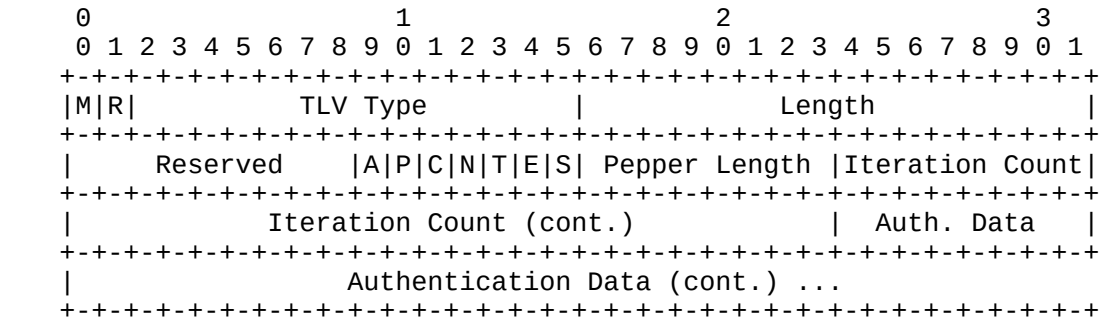
An identifier for the authentication server. The peer MAY use this identifier to search for a stored session associated with this server, or to associate the session to be negotiated with the server. The value of the identifier SHOULD be chosen so as to reduce the risk of collisions with other EAP server identifiers as much as possible. One possibility is to use the DNS name of the EAP server. The identifier MAY also be used by the peer to select a suitable key on the OTP token (when there are multiple keys available).

The identifier MUST NOT be longer than 128 octets. The identifier SHALL be a UTF-8 [7] encoded string of printable characters (without any terminating NULL character).

4.11.3. OTP TLV

In an EAP-Request, the OTP TLV is used to request an OTP (or a value derived from an OTP) from the peer. In an EAP-Response, the OTP TLV carries an OTP or a value derived from an OTP.

This TLV type MUST be supported by all peers and all EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV. The OTP TLV MUST NOT be present in an EAP-Request of type POTP-X that contains a New PIN TLV. Further, the OTP TLV MUST NOT be present in an EAP-Response of type POTP-X unless the preceding EAP-Request of type POTP-X contained an OTP TLV and it was valid for it to do so. Finally, an OTP TLV MUST NOT be present in an EAP-Response of type POTP-X that also contains a Resume TLV. The OTP TLV is defined as follows:



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

3

Length

7 + length of Authentication Data field

Reserved

Reserved for future use. All 9 bits SHALL be set to zero (0) for this version. Recipients SHALL ignore these bits for this version of EAP-POTP.

A

The A bit MUST be set in an EAP-Request if and only if the request immediately follows an EAP-Response of type POTP-X containing a New PIN TLV (see Section 4.11.5), and the new PIN in the response was accepted by the EAP server. In this case, the A bit signals that the EAP-server has accepted the PIN, and that the peer SHALL use the newly established PIN when calculating the response (when applicable). The A bit MUST NOT be set if the S bit is set. If a request has both the S bit and the A bit set, the peer SHALL regard the request as invalid, and return an empty POTP-X EAP-Response message.

In an EAP-Response, the A bit, when set, indicates that the OTP was calculated with the use of the newly selected user PIN. The A bit MUST be set in a response if and only if the EAP-Request which triggered the response contained an OTP TLV with the A bit set.

P

In an EAP-Request, the P bit indicates that the OTP in the response MUST be protected. Use of this bit also indicates that mutual authentication will take place, as well as generation of keying material. It is RECOMMENDED to always set the P bit. If a peer receives an EAP-Request with an OTP TLV that does not have the P bit set, and the peer's policy dictates protected mode, the peer MUST respond with an empty POTP-X EAP-Response message. All peers MUST support protected mode.

In an EAP-Response, this bit indicates that the provided OTP has been protected (see below). The P bit MUST be set in a response (and hence the OTP MUST be protected) if and only if the EAP-Request that triggered the response contained an OTP TLV with the P bit set.

In an 802.1x EAP over LAN (EAPOL) environment (this includes wireless LAN environments), the P bit MUST be set, or, alternatively, the EAP-POTP method MUST be carried out inside an authenticated tunnel that provides a cryptographic binding with inner EAP methods such as the one provided by PEAPv2 [17].

C

The C bit carries meaning only when the OTP algorithm in question makes use of server challenges. For other OTP algorithms, the C bit SHALL always be set to zero.

In an EAP-Request, the C bit ("Combine") indicates that the OTP SHALL be calculated using both the provided challenge and internal state (e.g., current token time). The OTP SHALL be calculated based only on the provided challenge (and the shared secret) if the C bit is not set, and a challenge is present. The returned OTP SHALL always be calculated based on the peer's current state (and the shared secret) if no challenge is present. If the C bit is set but no challenge is provided, the peer SHALL regard the request as invalid, and return an empty POTP-X EAP-Response message.

In an EAP response, this bit indicates that the provided OTP has been calculated using a provided challenge and the token state. The C bit MUST be set in a response if and only if the EAP-Request that triggered the response contained an OTP TLV with the C bit set and a challenge.

N

In an EAP-Request, the N bit, when set, indicates that the OTP to calculate SHALL be based on the next token "state", and not the current one. As an example, for a time-based token, this means the next time slot. For an event-based token, this could mean the next counter value, if counter values are used. This bit will normally not be set in initial EAP-Request messages, but may be set in subsequent ones. Further, the N bit carries no meaning in an EAP-Request if a challenge is present and the C bit is not set, and SHALL be set to 0, in this case. If a request that has the N bit set also contains a challenge, but does not have the C bit set, the peer SHALL regard the request as invalid, and return an empty POTP-X EAP-Response message. Note that setting the N bit in an EAP-Request will normally advance the internal state of the token.

In an EAP-Response, the N bit, when set, indicates that the OTP was calculated based on the next token "state" (as explained above), and not the current one. The N bit MUST be set in a response if and only if the EAP-Request that triggered the response contained an OTP TLV with the N bit set.

T

The T bit only carries meaning for OTP methods normally incorporating a user PIN in the OTP computation.

In an EAP-Request, the T bit, when set, indicates that the OTP to calculate MUST NOT include a user PIN.

In an EAP-Response, the T bit, when set, indicates that the OTP was calculated without the use of a user PIN. The T bit MUST be set in a response if and only if the EAP-Request that triggered the response contained an OTP TLV with the T bit set. Note that client policy may prohibit PIN-less calculations; in these cases, the client MAY respond with an empty POTP-X EAP response message.

E

In an EAP-Request, the E bit, when set, indicates that the peer MUST NOT use any stored pepper value associated with this server in the PBKDF2 computation. Rather, it MUST generate a new pepper (if supported by the peer) and/or use the iteration count parameter to protect the OTP (if the server's Max Pepper Length is 0, then the peer MUST rely on the iteration count only to protect the OTP). This bit will usually not be set in initial EAP-Request messages, but may be set in subsequent ones, e.g., if the server, upon receipt of an OTP TLV with a pepper identifier, detects that it does not have a pepper with that identifier in storage. This bit carries no meaning, and MUST be set to zero, when the P bit is not set. If a request has the E bit set but not the P bit, a peer SHALL regard the request as invalid, and return an empty POTP-X EAP-Response message.

In an EAP-Response, the E bit indicates that the response has been calculated without use of any stored pepper value.

S

In an EAP-Request, the S bit ("Same"), when set, indicates that the peer SHOULD calculate its response based on the same OTP value as was used for the preceding response. This bit MAY be set when the EAP server has received an OTP TLV from the peer protected with a pepper, of which the server is no longer in possession. Since the server has not attempted validation of the provided data, there is no need for the EAP peer to retrieve a new OTP value. This bit carries no meaning, and MUST be set to zero, when the E bit is not set. A peer SHALL regard a request where the S bit is set, but not the E bit, as invalid, and return an empty POTP-X EAP-Response message. Further, the S bit MUST NOT be set when the A bit also is set; see above.

In an EAP-Response, the S bit is never set.

Pepper Length

This octet SHALL be present if and only if the P bit is set. When present, it contains an unsigned integer, having a value between 0 and 255 (inclusive). In an EAP-Request, the integer represents the maximum length (in bits) of a client-generated pepper the server is prepared to search for. Peers MUST NOT generate peppers longer than this value. If the value is set to zero, it means the peer MUST NOT generate a pepper for the PBKDF2 calculation. In an EAP-Response, it indicates the length of the used pepper.

Iteration Count

These 4 octets SHALL be present if and only if the P bit is set. When present, they contain an unsigned, 4-octet integer in network byte order. In an EAP-Request, the integer represents the maximum iteration count the peer may use in the PBKDF2 computation. Peers MUST NOT use iteration counts higher than this value. In an EAP-Response, it indicates the actual iteration count used.

Note regarding the Pepper Length and Iteration Count parameters: A peer MUST compare these policy parameters provided by the EAP server with local policy and MUST NOT continue the handshake if use of the EAP server's suggested parameters would result in a lower security than the client's acceptable policy. If the security given by the EAP server's provided policy parameters surpasses the security level given by the peer's local policy, the client SHOULD use the server's parameters (subject to reason - active attackers could otherwise mount simple denial-of-service attacks against peers or servers, e.g., by providing unreasonably high values for the iteration count). Note that the server-provided parameters only apply to the case where the peer cannot use or does not have a previously provided server-provided pepper. If a peer cannot continue the handshake due to the server's policy being unacceptable, it MUST return an empty POTP-X EAP-Response message.

Authentication Data

EAP-Request: In an EAP-Request, the Authentication Data field, when present, contains an optional "challenge". The challenge is an octet string that SHOULD be uniquely generated for each request in which it is present (i.e., it is a "nonce"), and SHOULD be 8 octets or longer. To avoid fragmentation (i.e., EAP messages longer than the minimum EAP MTU size; see [1]), the challenge MUST NOT be longer than 64 octets. When the challenge is not present, the OTP will be calculated on the current token state only. The peer MAY ignore a provided challenge if and only if the OTP token the peer is interacting with is not capable of including a challenge in the OTP calculation. In this case, EAP server policies will determine whether or not to accept a provided OTP value.

EAP-Response: The following applies to the Authentication Data field in an EAP-Response:

* When the P bit is not set, the peer SHALL directly place the OTP value calculated by the token in the Authentication Data field. In this case, the EAP server MUST NOT send a Confirm

TLV upon successful authentication of the peer (instead, it sends an EAP-Success message).

* When the P bit is set, the peer SHALL populate this field as follows. After the token has calculated the OTP value, the peer SHALL compute:

  K_MAC | K_ENC | MSK | EMSK | SRK = PBKDF2(otp, salt | pepper | auth_id, iteration_count, key_length)

  where

  "|" denotes concatenation,

  "otp" is the already computed OTP value,

  "salt" is a 16-octet nonce,

  "pepper" is an optional nonce (at most, 255 bits long, and, if necessary, padded to be a multiple of 8 bits long; see below) included to complicate the task of finding a matching "otp" value for an attacker,

  "auth_id" is an identifier (at most, 255 octets in length) for the authenticator (i.e., the network access server) as reported by lower layers and as specified below,

  "iteration_count" is an iteration count chosen such that the computation time on the peer is acceptable (based on the server's indicated policy and the peer's local policy), while an attacker, having observed the response and initiating a search for a matching OTP, will be sufficiently slowed down. The "iteration_count" value MUST be chosen to provide a suitable level of protection (e.g., at least 100,000) unless a server-provided pepper is being used, in which case, it SHOULD be 1.

  "key_length" is the combined length of the desired key material, in octets. When the default algorithms are used, key_length is 176.

  The "pepper" values are only included in PBKDF2 calculations and are never sent to EAP servers (though the peers do send their length, in bits). The purpose of the pepper values are, as mentioned above, to slow down an attacker's search for a matching OTP, while not slowing down the peer (which iterated hashes do). If the pepper has been generated by the peer, and the chosen pepper length in bits is not a

multiple of 8, then the pepper value SHALL be padded to the left, with '0' bits to the nearest multiple of 8 before being used in the PBKDF2 calculation. This is to ensure the input to the calculation consists only of whole octets. As an example, if the chosen pepper length is 4, the pepper value will be padded to the left, with 4 '0' bits to form an octet before being used in the PBKDF2 calculation.

When pepper is used, it is RECOMMENDED that the length of the pepper and the iteration count are chosen in such a way that it is computationally infeasible/unattractive for an attacker to brute-force search for the given OTP within the lifetime of that OTP.

As mentioned previously, a peer MUST NOT include a newly generated pepper value in the PBKDF2 computation if the server did not indicate its support for pepper searching in this session. If the server did not indicate support for pepper searching, then the PBKDF2 computation MUST be carried out with a sufficiently higher number of iterations so as to compensate for the lack of pepper (see further Appendix D).

A server may, in an earlier session, have transferred a pepper value to the peer in a Confirm TLV (see below). When this is the case, and the peer still has that pepper value stored for this server, the peer MUST NOT generate a new pepper but MUST, instead, use this transferred pepper value in the PBKDF2 calculations. The only exception to this is when a local policy (e.g., timer) dictates that the peer must switch to a new pepper (and the server indicated support for pepper searching).

The following applies to the auth_id component:

- For dial-up, "auth_id" SHALL be either the empty string or the phone number called by the peer. The phone number SHALL be specified in the form of a URL conformant with RFC 3966 [8], e.g., "tel:+16175550101". Processing of received phone numbers SHALL be conformant with RFC 3966 (this assumes that "tel" URIs will be shorter than 256 octets, which would normally be the case).

- For use with IEEE 802.1X, "auth_id" SHALL be either the empty string or the MAC address of the authenticator in canonical binary format (6 octets).

- For IP-based EAP, "auth_id" SHALL be either the empty string or the IPv4 or IPv6 address of the authenticator as seen by the peer and in binary format (4 or 16 octets, respectively). As an example, the IPv4 address "192.0.2.5" would be represented as (in hex) C0 00 02 05, whereas the IPv6 address "2001:DB8::101" would be represented as (in hex) 20 01 0D B8 00 00 00 00 00 00 00 00 00 00 01 01.

Note: Use of the authenticator's identifying information within the computation aids in protection against man-in-the-middle attacks, where a rogue authenticator seeks to intercept and forward the Authentication Data in order to impersonate the peer at a legitimate authenticator (but see also the discussion around spoofed authenticator addresses in Section 6). For these reasons, a peer SHOULD NOT set the auth_id component to the empty string unless it is unable to learn the identifying information of the authenticator. In these cases, the EAP server's policy will determine whether or not the session may continue.

As an example, when otp = "12345678", salt = 0x5443453454344543545678789099880, pepper is not used, auth_id = "192.0.2.5", iteration_count = 2000 (decimal), and key_length = 176 (decimal), the input to the PBKDF2 calculation will be (first two parameters in hex, line wrap for readability):

(3132333435363738, 5443453454344543545678789099880 | c0000205, 2000, 176)

As described, when the default algorithms are used, K_MAC is the first 16 octets of the output from PBKDF2, K_ENC the next 16 octets, MSK the following 64 octets, EMSK the next 64 octets, and SRK the final 16 octets. Using K_MAC, the peer calculates:

mac = MAC(K_MAC, msg_hash(msg_1, msg_2, ..., msg_n))

as specified in Section 4.9 and where msg_1, msg_2, ..., msg_n is a sequence of all EAP messages of type POTP-X exchanged so far in this session, as sent and received by the peer (for the peer's initial MAC, it will typically be just one message: the EAP server's initial EAP-Request of type POTP-X).

The peer then places the first 16 octets of "mac" in the Authentication Data field, followed by the "salt" value, followed by one octet representing the length of the "auth_id" value in octets, followed by the actual "auth_id" value in binary form, and optionally followed by a pepper identifier (only when the peer made use of a pepper value previously provided by the EAP server). Pepper identifiers, when present, are always 4 octets. All variables SHALL be present in the form they were input to the PBKDF2 algorithm. This will result in the Authentication Data field being 33 + (length of auth_id in octets) + (4, for pepper identifier, when present) octets in length.

Continuing the previous example, the Authentication Data field will be populated with (in hex, line wrap for readability):

< 16 octets of mac > | 54434534543445435465768789099880 | 04 | c0000205

Note: Since in this case (i.e., when the P bit is set) successful authentication of the peer by the EAP server will be followed by the transmission of an EAP-Request of type POTP-X containing a Confirm TLV for mutual authentication, the peer MUST save either all the input parameters to the PBKDF2 computation or the keys K_MAC, K_ENC, SRK, MSK, and EMSK (recommended, since they will be used later). This is because the peer cannot be guaranteed to be able to generate the same OTP value again. For the same reason (the Confirm-TLV from the EAP server), the peer MUST also store either the hash of the contents of the sent EAP-Response or the EAP-Response itself (but see the note above about not including any User Identifier TLVs in the hash computation).

Given a set of possible OTP values, the authentication server verifies an authentication request from the peer by computing

K_MAC' | K_ENC' | MSK' | EMSK' | SRK' = PBKDF2 (otp',
  salt | pepper' | auth_id, iteration_count, key_length)

for each possible OTP value otp' and each possible pepper value pepper' , and the provided values for salt, authenticator identity, and iteration count, as well as the applicable key length (default: 176). Note: Doing the computation for each possible pepper value implements the pepper search mentioned elsewhere in this document. Note also that the EAP server may accept more than one OTP value

at a given time, e.g., due to clock drift in the token. If the given pepper length is not a multiple of 8, each tested pepper value will be padded to the left to the nearest multiple of 8, in the same manner as was done by the peer. If the server already shares a secret pepper value with this peer, then obviously there will only be one possible pepper value, and the server will find it based on the pepper_identifier provided by the peer. The server SHALL send a new EAP-Request of type POTP-X with an OTP TLV with the E bit set if the peer provided a pepper identifier unknown to the server.

For each K_MAC', the EAP server computes

mac' = MAC(K_MAC', msg_hash(msg_1', msg_2', ..., msg_n'))

where MAC is the negotiated MAC algorithm, msg_hash is the message hash algorithm defined in Section 4.9, and msg_1', msg_2', ... msg_n' are the same messages on which the peer calculated its message hash, but this time, as sent and received by the EAP server. If the first 16 octets of mac' matches the first 16 octets in the Authentication Data field of the EAP-Response in question, and the provided authenticator identity is acceptable (e.g., matches the EAP server's view of the authenticator's identity), then the peer is authenticated.

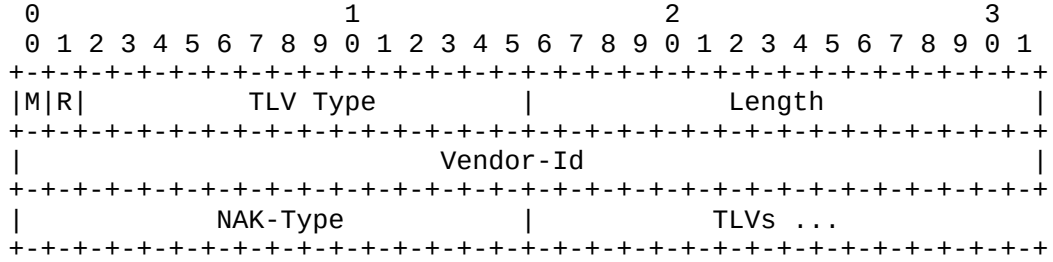
If the authentication is successful, the authentication server then attempts to authenticate itself to the peer by use of the Confirm TLV (see below). If the authentication fails, the EAP server MAY send another EAP-Request of type POTP-X containing an OTP TLV to the peer, or it MAY send an EAP-Failure message (in both cases, possibly preceded by an EAP-Request of type Notification).

4.11.4. NAK TLV

Presence of this TLV indicates that the peer did not support a received TLV with the M bit set. This TLV may occur 0, 1, or more times in an EAP-Response of type POTP-X. Each occurrence flags the non-support of a particular received TLV.

The NAK TLV MUST be supported by all peers and all EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV. Receipt of a NAK TLV by an EAP server MAY cause an authentication to fail, and the EAP server to send an EAP-Failure message to the peer.

Note: The definition of the NAK TLV herein matches the definition made in [17], and has the same type number. Field descriptions are copied from that document, with some minor modifications.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

4

Length

6 + cumulative total length of embedded TLVs

Vendor-Id

The Vendor-Id field is 4 octets, and contains the Vendor-Id of the TLV that was not supported. The high-order octet is 0 and the low-order 3 octets are the Structure of Management Information (SMI) Network Management Private Enterprise Code of the Vendor in network byte order. The Vendor-Id field MUST be zero for TLVs that are not Vendor-Specific TLVs. For Vendor-Specific TLVs, the Vendor-ID MUST be set to the SMI code.

NAK-Type

The type of the unsupported TLV. The TLV MUST have been included in the most recently received EAP message.

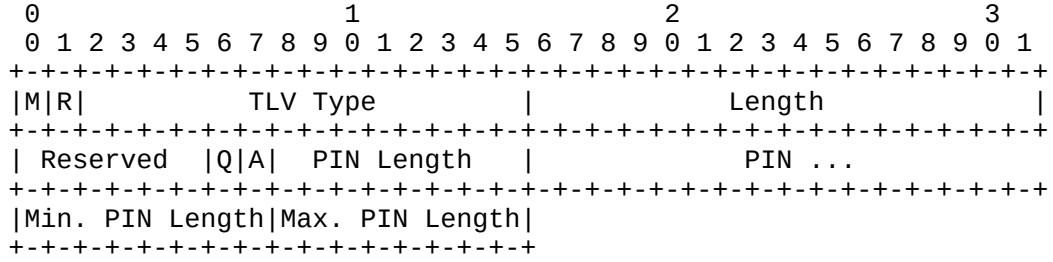
TLVs

This field contains a list of TLVs, each of which MUST NOT have the mandatory bit set. These optional TLVs can be used in the future to communicate why the offending TLV was determined to be unsupported.

4.11.5. New PIN TLV

In an EAP-Request, the New PIN TLV is used to request a new user PIN from the peer. The EAP server MAY provide a new PIN, as described below. In an EAP-Response, the New PIN TLV carries a chosen new user PIN. This TLV may be used by an EAP server when policy dictates that the peer (user) needs to change a PIN associated with the OTP Token.

This TLV type SHOULD be supported by peers and EAP servers conforming to this specification. The New PIN TLV MUST NOT be sent by an EAP server unless the peer has been authenticated. If the peer was authenticated in protected mode, then the New PIN TLV MUST NOT be present in an EAP-Request until after the exchange of the Confirm TLV (i.e., until after mutual authentication has occurred and keys are in place to protect the TLV). The New PIN TLV MUST be sent by a peer if and only if the EAP-Request that triggered the response contained a New PIN TLV, it was valid for the EAP server to send such a TLV in that request, and the TLV is supported by the peer.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

5

Length

2 + length of the PIN field (as specified in the PIN Length field) + (0, 1, or 2)

Note: The final term above is
- 0 if none of the optional Min. / Max. PIN Length fields is present in the TLV,
- 1 if only the Min. PIN Length field is present in the TLV,
- 2 if both of these optional fields are present in the TLV.

Reserved

Reserved for future use. All six bits SHALL be set to zero for this version. Recipients SHALL ignore these bits for this version of EAP-POTP.

Q

The Q bit, when set in an EAP-Request, indicates that an accompanying PIN is required, i.e., the peer (user) is not free to choose another PIN. When the Q bit is set, there MUST be an accompanying PIN and the provided PIN MUST be used in subsequent OTP generations. A peer SHALL respond with an empty POTP-X EAP-Response message if the Q bit is set but there is not any accompanying PIN. When the Q bit is not set, any provided PIN is suggested only, and the peer is free to choose another PIN, subject to local policy.

The Q bit carries no meaning, and SHALL be set to zero, in an EAP-Response.

A

This bit allows methods that distinguish between two different PIN types (e.g., decimal vs. alphanumeric) to designate whether the augmented set is to be used (when set) or not (when not set). The A bit carries no meaning, and SHALL be set to zero, in an EAP-Response.

PIN Length

This field contains an unsigned integer representing the length of the provided PIN (this implies that the maximum length of a PIN will be 255 octets).

PIN

In an EAP-Request, subject to the setting of the Q bit, the PIN field MAY be empty. If empty, the peer (user) will need to choose a PIN subject to local and (any) provided policy. When the PIN field is not empty, it MUST consist of UTF-8 encoded printable characters without a terminating NULL character.

In an EAP-Response, the PIN value SHALL consist of a UTF-8 encoded string of printable characters without a terminating NULL character.

The peer accepts a PIN suggested by the EAP server by replying with the same PIN, but MAY replace it with another one, depending on the server's setting of the Q bit. The length of the PIN is application-dependent, as are any other requirements for the PIN, e.g., allowed characters. The peer MUST be prepared to receive a repeated request for a new PIN, as described above, if the EAP server, for some reason does not accept the received PIN. Such a request MAY be preceded by an EAP-Request of type Notification (2) providing information to the user about the reason for the rejection. Mechanisms for transferring knowledge about PIN requirements from the EAP server to the peer (beyond those specified for this TLV, such as maximal and minimal PIN length) are outside the scope of this document. However, some information MAY be provided in notification messages transferred from the EAP server to the peer, as per above.

Min. PIN Length

This field MAY be present in an EAP-Request. This field MUST NOT be present in an EAP-Response. It SHALL be interpreted as an unsigned integer in network byte order representing the minimum length allowed for a new PIN.

Max. PIN Length

This field MUST NOT be present in an EAP-Request unless the Min. PIN Length field is present, in which case it MAY be present. The field MUST NOT be present in an EAP-Response. It SHALL be interpreted as an unsigned integer in network byte order representing the maximum length allowed for a new PIN. The value of this field, when present, MUST be equal to, or larger than, the value of the Min. PIN Length field.

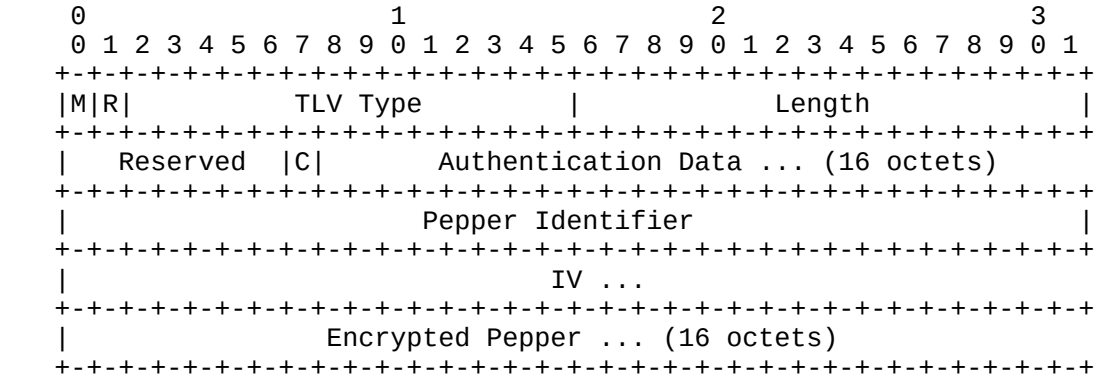
4.11.6. Confirm TLV

Presence of this TLV in a request indicates that the EAP server has successfully authenticated the peer and now attempts to authenticate itself to the peer. Presence of this TLV in a response indicates that the peer successfully authenticated the EAP server, and that calculated keys (K_MAC, K_ENC, MSK, EMSK, and SRK) now become available for use.

The Confirm TLV MUST NOT appear together with any other TLV in an EAP-Request message of type POTP-X and MUST NOT be sent unless the peer has been authenticated through an OTP TLV with the P bit set or through a Resume TLV for which the underlying session was established in protected mode. The Confirm TLV MUST be present in an EAP-Response if and only if the request that triggered the response contained a Confirm TLV, it was legal for it to do so, and the Confirm TLV authenticated the EAP server to the peer. If the peer was not able to authenticate the server, then it MUST send an empty (i.e., no TLVs present) EAP-Response of type POTP-X.

An EAP server MUST send an EAP-Success message after receiving an EAP-Response of type POTP-X containing a valid Confirm TLV, sent in response to an EAP-Request containing a Confirm TLV where the C bit was not set. A peer MUST NOT accept an EAP-Success message when it has sent an OTP TLV with the P bit set unless it has received an acceptable Confirm TLV from the EAP server.

This TLV type MUST be supported by all peers and EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

6

Length

17 or 37 + length of IV in requests, 1 in responses.

Reserved

Reserved for future use. These 7 bits SHALL be set to zero (0) for this version. Recipients SHALL ignore these bits for this version of EAP-POTP.

C

The C bit, when set in an EAP-Request, indicates that the EAP server intends to send more EAP-Requests of type POTP-X in this session, after receipt of a Confirm TLV from the peer.

The C bit carries no meaning in EAP-Responses, and MUST NOT be set within them.

Note: An EAP-Response containing a Confirm TLV, sent in response to an EAP-Request containing a Confirm TLV that did not have the C bit set, MUST be followed by an EAP-Success message from the EAP server concluding the handshake. However, when the C bit was set in an EAP-Request, the EAP server MAY send another EAP-Request (containing, for example, a New PIN TLV wrapped in a Protected TLV) rather than an EAP-Success message. Therefore, peers MUST NOT assume that the only EAP message following an EAP-Response of type POTP-X containing a Confirm TLV is EAP-Success. The C bit gives EAP servers a way to indicate their intent to follow the Confirm TLV with more requests, and allows the peer’s state machine to adapt to this.

Authentication Data

EAP-Request:

In a request, this field consists of the first 16 octets of (see also Section 4.11.3):

mac_a = MAC(K_MAC', msg_hash(trig_msg))

where

MAC is the negotiated MAC algorithm,

"K_MAC'" has been calculated as described in Section 4.11.3 or (in the case of session resumption) Section 4.11.8, and

"msg_hash" is the message hash algorithm defined in Section 4.9, and "trig_msg" the latest EAP-Response of type POTP-X received from the peer (the one which triggered this request).

Given a saved or recomputed value for K_MAC, the peer authenticates the EAP server by computing

mac'' = MAC(K_MAC, msg_hash(trig_msg'))

where "msg_hash(trig_msg')" is the peer's hash of the EAP-Response message that it sent to the server (and that the server calculated its message hash on). If the first 16 octets of mac'' matches the first 16 octets in the Authentication Data field of the EAP-Request in question, then the EAP server is authenticated.

EAP-Response:

Not used in this version, and SHALL NOT be present in EAP-Responses.

Pepper Identifier

In an EAP-Request, the truncated MAC MAY optionally be followed by an encrypted pepper and its identifier. This initial, 4-octet field identifies a pepper generated by the server.

For this version of EAP-POTP, this field SHALL NOT be present in EAP-Responses.

IV (Initialization Vector)

An initialization vector for the encryption. The length of the vector is dependent on the negotiated encryption algorithm. For example, for AES-CBC, it SHALL be 16 octets. The IV is only present if a pepper is present, and the negotiated encryption algorithm makes use of an IV. This field SHALL NOT be present in EAP-Response messages for this version of EAP-POTP.

Encrypted Pepper

When present in an EAP-Request, this will be a uniformly distributed and randomly chosen 16-octet pepper generated by the EAP server and encrypted with the negotiated encryption algorithm, using K_ENC as the encryption key and possibly (depending on the encryption algorithm) using an IV (stored in the IV field). This field MUST be present if and only if the Pepper Identifier field is present.

EAP servers are RECOMMENDED to include a freshly generated encrypted pepper (and a corresponding Pepper Identifier) in every Confirm TLV.

This field SHALL NOT be present in EAP-Response messages for this version of EAP-POTP.

When a new pepper is generated by the server and transferred in encrypted form to the peer, then this new pepper value will be stored in the EAP server upon receipt of the Confirm TLV from the peer, and SHOULD be stored with its identifier and associated with the EAP server and the current user in the peer upon receipt of the EAP-Success message. If the peer already had a pepper stored for the EAP server, it SHALL replace it with the newly received one.

4.11.7. Vendor-Specific TLV

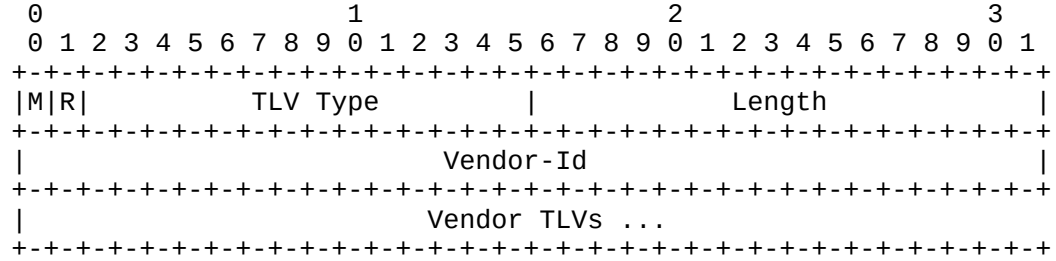
The Vendor-Specific TLV is available to allow vendors to support their own extended attributes not suitable for general usage. A Vendor-Specific TLV can contain one or more inner TLVs, referred to as Vendor TLVs. The TLV-type of a Vendor TLV will be defined by the vendor. All the Vendor TLVs inside a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

This TLV type MAY be sent by EAP servers, as well as by peers, and MUST be supported by all entities conforming to this specification. Conforming implementations may not support specific Vendor TLVs inside a Vendor-Specific TLV, however. They MAY, in this case, respond to the Vendor TLVs with a NAK TLV containing the appropriate Vendor-ID and Vendor TLV type.

The presence of a Vendor-Specific TLV in an EAP-Request or EAP-Response of type POTP-X MUST NOT violate any existing rules for coexistence of TLVs in such requests or responses. If it does, then it will result in an EAP-Failure (when the peer made the violation) or an empty EAP-POTP response (when the EAP-server made the violation). It is left to the definition of specific Vendor-Specific TLVs to further constrain when they are allowed to appear. In

particular, EAP-POTP implementations may have policies that completely disallow use of the Vendor-Specific TLV before protected mode mutual authentication has occurred (since the Protected TLV, Section 4.11.15, then can be used to protect all TLVs).

Note: This TLV type has the same definition and TLV type number as the Vendor-Specific TLV in [17], and the description of it is largely borrowed from that document.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

7

Length

4 + cumulative total length of inner Vendor TLVs

Vendor-ID

The Vendor-Id field is 4 octets. The high-order octet SHALL be set to 0, and the low-order 3 octets SHALL be set to the SMI Network Management Private Enterprise Code (see [18]) of the Vendor in network byte order.

Vendor TLVs

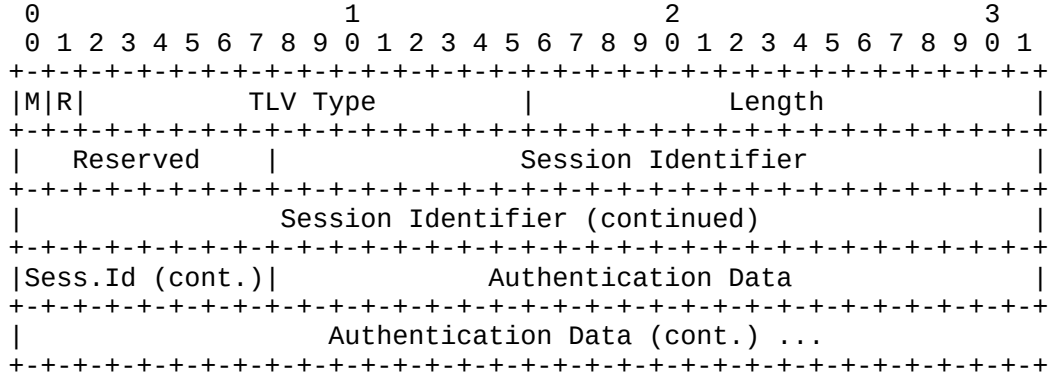
This field shall contain vendor-specific TLVs, in a format defined by the vendor. To avoid fragmentation (i.e., EAP messages longer than the minimum EAP MTU size), the field SHOULD NOT be longer than 256 octets.

To ensure interoperability when an EAP entity (peer or server) from vendor A sends a vendor-specific TLV that is not understood by the recipient EAP entity from vendor B, the vendor A entity SHALL, upon receipt of the NAK TLV from the recipient, refrain from usage of the vendor-specific TLV in question for the rest of the handshake, and MUST NOT fail the session due to the receipt of the NAK TLV for the Vendor TLV (i.e., it SHALL continue as if the vendor-specific TLV had not been sent). Additionally, all implementations conformant with this document SHOULD allow use of vendor-specific extensions to be turned off via configuration.

4.11.8. Resume TLV

The Resume TLV MAY be sent by a peer to an authentication server to attempt session resumption.

This TLV type MUST only be sent in response to an EAP-Request of type POTP-X containing a Server-Info TLV allowing session resumption. The Resume TLV MUST be supported by all EAP servers that send a Server-Info TLV allowing session resumption.



M

0 - Non-mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

8

Length

45

Reserved

Reserved for future use. This octet SHALL be set to zero (0) for this version. Recipients SHALL ignore this octet for this version of EAP-POTP.

Session Identifier

An 8-octet identifier for the session the peer is trying to resume.

Authentication Data

Upon receipt of the Server-Info TLV, and if the N bit is not set, the peer searches for any stored sessions associated with the server identified by the Server Name field. If a stored session is found, the peer generates a random, 16-octet nonce, "c_nonce", and calculates:

K_MAC | K_ENC | MSK | EMSK | SRK = PBKDF2(base_key, c_nonce | s_nonce, iteration_count, key_length)

where

"|" denotes concatenation,

"base_key" is either the current SRK for the session (if the session was created in protected mode) or the OTP used when the session was created (if the session was created in basic mode),

"c_nonce" is the generated 16-octet nonce,

"s_nonce" is the server nonce from the Server-Info TLV,

"iteration_count" is the iteration count as determined by local policy, and

"key_length" is the combined length of the desired key material, in octets. When the default algorithms are used, key_length is 176.

The iteration count need only be 1 (one) when resuming a session established in protected mode, but MUST be chosen to provide a suitable level of protection when resuming a session established in basic mode (see also Section 4.11.3).

Note: Session resumption for basic mode MUST only be carried out in a server-authenticated and protected tunnel that also provides a cryptographic binding for inner EAP methods.

The peer then calculates:

mac = MAC(K_MAC, msg_hash(resume_req))

where

"MAC" is the negotiated MAC algorithm, and

"msg_hash(resume_req) is the message hash algorithm defined in Section 4.9 applied on resume_req, the EAP server's EAP-Request of type POTP-X containing the Server-Info TLV that allowed session resumption.

The peer then places the first 16 octets of the MAC value, followed by the c_nonce value, followed by the iteration count value (as a 4-byte unsigned integer in network byte order), in the Authentication Data field. As an example, when c_nonce = 0x2b3b1b12babdebebfb43bd7bdfbeb8df and iteration_count = 1, the Authentication Data field will be populated with (in hex):

< 16 octets of mac > | 2b3b1b12babdebebfb43bd7bdfbeb8df | 00000001

The server authenticates the peer by performing the corresponding calculations. If the authentication is successful, the server MUST send an EAP-Request of type POTP-X containing a Confirm TLV to the peer. If the authentication fails, the server MUST either send an EAP-Request of type POTP-X containing an OTP TLV and a Server-Info TLV, where the Server-Info TLV indicates that session resumption is not possible, or send an EAP-Failure.

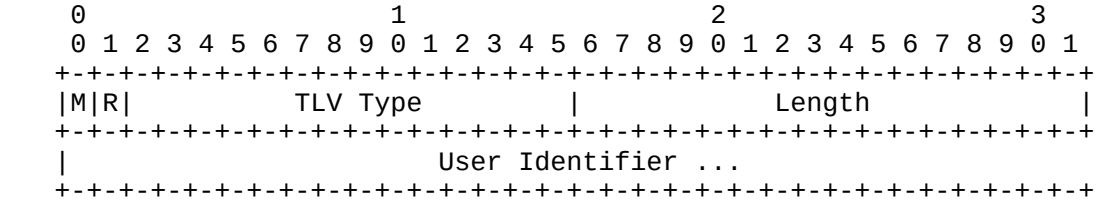
When resuming in basic mode, all calculated keys SHALL be discarded after the MAC has been calculated and verified. When resuming in protected mode, the new SRK will replace the stored SRK, and the new MSK and EMSK will be exported upon successful completion of the method.

4.11.9. User Identifier TLV

The User Identifier TLV carries an identifier, typically the username, for the holder of the OTP token used to generate the OTP.

At least one of the User Identifier TLV and the Token Key Identifier TLV SHOULD be present in the session's first EAP-Response of type POTP-X that also carries an OTP TLV unless a suitable identity has been provided in a preceding EAP-Response of type Identity (1) or is determined by some other means (see [1], Section 2). Use of the User Identifier TLV and/or the Token Key Identifier TLV is RECOMMENDED even when an EAP-Response of type Identity (1) has been sent. If a peer sends both a User Identifier TLV and a Token Key Identifier TLV, then the EAP server SHALL interpret the Token Key Identifier TLV as specifying a particular token key for the given user. The EAP server MUST respond with an EAP-Failure if it cannot find a token key for the provided user.

This TLV type is sent by peers and MUST be supported by all EAP servers conforming to this specification. The User Identifier TLV MUST NOT be present in a response that does not also carry an OTP TLV.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

9

Length

Length of User Identifier, >= 1

User Identifier

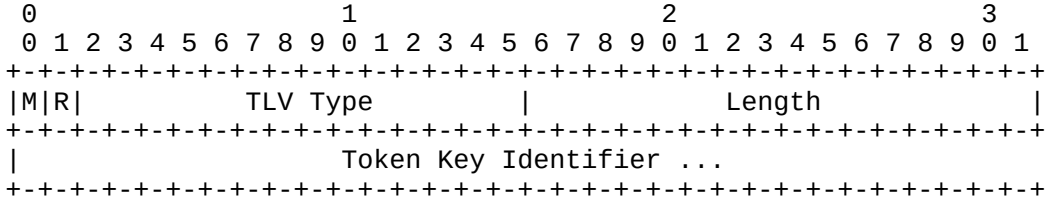
The value SHALL be an UTF-8 encoded string representing the holder of the token (MUST NOT be NULL-terminated). The string MUST be less than 128 octets in length.

4.11.10. Token Key Identifier TLV

The Token Key Identifier TLV carries an identifier for the token key used to generate the OTP.

At least one of the User Identifier TLV and the Token Key Identifier TLV SHOULD be present in the session's first EAP-Response of type POTP-X, which also carries the OTP TLV unless a suitable identity has been provided in a preceding EAP-Response of type Identity (1) or is determined by some other means (see [1], Section 2). Use of the User Identifier TLV and/or the Token Key Identifier TLV is RECOMMENDED even when an EAP-Response of type Identity (1) has been sent. If a peer sends both a User Identifier TLV and a Token Key Identifier TLV, then the EAP server SHALL interpret the Token Key Identifier TLV as specifying a particular token key for the given user. The EAP server MUST respond with an EAP-Failure if it cannot find a token key corresponding to the provided token key identifier.

This TLV type is sent by peers and MUST be supported by all EAP servers conforming to this specification. The Token Key Identifier TLV MUST NOT be present in a response that does not also carry an OTP TLV.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

10

Length

Length of Token Key Identifier, >= 1

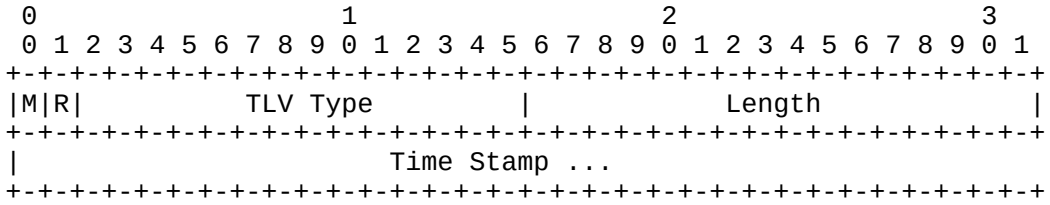
Token Key Identifier

An identifier for the OTP token key used to generate the OTP. The field MUST be less than 128 octets in length.

4.11.11. Time Stamp TLV

The Time Stamp TLV MAY be sent by peers to simplify authentications. When present, it carries the time as reported by the OTP Token.

An EAP server conformant with this specification SHOULD support (i.e., recognize) this TLV, but need not be able to process or act on it. An EAP server that does not support this TLV, but receives an EAP-Response with the TLV present, MAY ignore the value. The Time Stamp TLV MUST NOT be present in any EAP-Responses of type POTP-X other than those that also carries an OTP TLV.



M

0 - Non-mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

11

Length

Length of Time Stamp field, >= 20 (depending on precision)

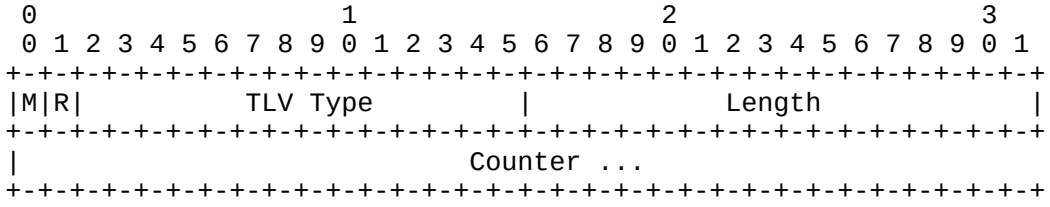
Time Stamp

The time, as reported by the OTP token, at which the OTP used for the accompanying OTP TLV was calculated. The field SHALL contain a UTF-8 encoded value of the XML simple type "dateTime", with time zone information and precision down to at least seconds, e.g., "2004-06-16T15:20:02Z".

4.11.12. Counter TLV

The Counter TLV MAY be sent by peers to simplify authentications. When present, it carries the token counter value, as reported by the OTP Token.

An EAP server conformant with this specification SHOULD support (i.e., recognize) this TLV, but need not be able to process or act on it. An EAP server that does not support this TLV, but receives an EAP-Response with the TLV present, MAY ignore the value. The Counter TLV MUST NOT be present in any EAP-Responses of type POTP-X other than those that also carries an OTP TLV.



M

0 - Non-mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

12

Length

Length of Counter field, >= 1 (depending on precision)

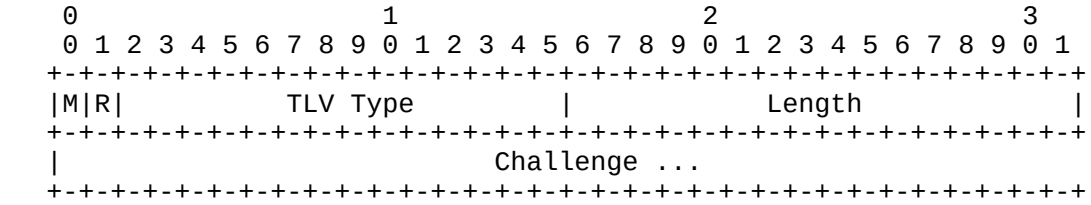
Counter

The counter value, as reported by the OTP token, at which the OTP used for the accompanying OTP TLV was calculated. The counter value SHALL be represented as an unsigned integer in network-byte order, e.g., a counter value of 1030 may be sent as the 2 octets (in hex) 04 06.

4.11.13. Challenge TLV

The Challenge TLV carries the challenge used by the token to calculate the OTP, as reported by the token to the peer. The Challenge TLV MUST be sent by a peer if and only if the challenge otherwise would be unknown to the EAP server (e.g., the token or peer modified a received challenge or generated its own challenge).

An EAP server conformant with this specification SHOULD support (i.e., recognize) this TLV, but need not be able to process or act on it. An EAP server that does not support this TLV, but receives an EAP-Response with the TLV present, MAY ignore the value. The Challenge TLV MUST NOT be present in any EAP-Responses of type POTP-X other than those that also carry an OTP TLV.



M

0 - Non-mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

16

Length

Length of Challenge field, >= 1

Challenge

The challenge value that was used to calculate the OTP used for the accompanying OTP TLV.

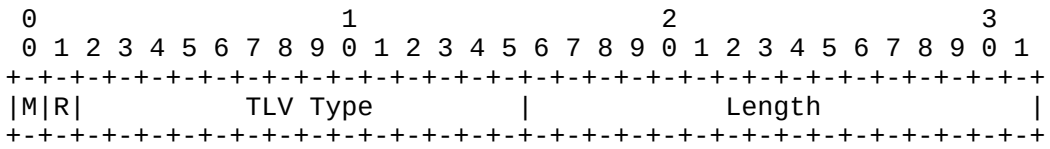
4.11.14. Keep-Alive TLV

The Keep-Alive is used to avoid EAP-POTP timeouts.

The Keep-Alive TLV MAY be sent by a peer to avoid timeouts when the peer has received an EAP-Request containing an OTP TLV or a New PIN TLV and is waiting for a response from the user.

An EAP-Request containing a Keep-Alive TLV MUST be sent by an EAP server when the server receives an EAP-Response containing a Keep-Alive TLV, and the server has an outstanding request that did not contain a Keep-Alive TLV. In this situation, the server does not need to re-transmit its latest outstanding request, but, due to the synchronous nature of EAP, it needs to send another request. Re-transmission of the latest outstanding request could be confusing for the peer since the request would get a new Identifier value. The Keep-Alive TLV MAY also be sent by an EAP server when the server detects that its processing time will exceed some locally configured threshold and may cause a network timeout. In this case, the peer MUST respond with an EAP-Response containing a Keep-Alive TLV.

This TLV type MUST be supported by all peers and all EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV. The Keep-Alive TLV MUST NOT be sent in any other situations than the ones described above. The Keep-Alive TLV MUST NOT be sent together with any other TLVs defined herein. Implementations SHOULD also follow recommendations made in Section 4.3 of [1].



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

13

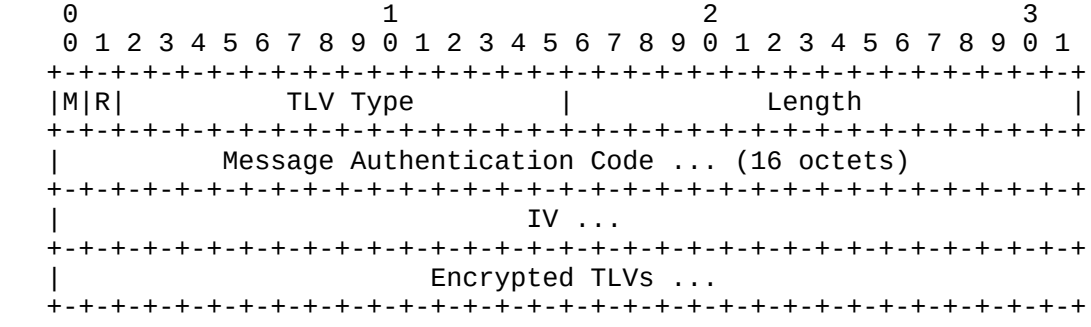
Length

0

4.11.15. Protected TLV

The Protected TLV SHALL be used to encrypt individual or multiple TLVs after successful exchange of the Confirm TLV (i.e., as soon as calculated keys have been confirmed). The Protected TLV therefore wraps "ordinary" TLVs.

This TLV type may be sent by EAP servers as well as by peers and MUST be supported by all peers conforming to this specification. It SHOULD be supported by all EAP servers conforming to this specification (it need not be supported if a server never will have a need to continue a POTP-X conversation after exchange of the Confirm TLV).



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

14

Length

>32

Message Authentication Code (MAC)

This field integrity-protects the TLV. The MAC SHALL be calculated over the IV and the Encrypted TLVs field in the following manner:

mac = MAC(K_MAC, iv | encrypted_tlvs)

where

MAC is the negotiated MAC algorithm, "iv" is the IV field's value, and "encrypted_tlvs" is the value of the Encrypted TLVs field. The first 16 octets of the MAC is placed in the Message Authentication Code field.

Recipients MUST verify the MAC. If the verification fails, the conversation SHALL be terminated (i.e., peers send an empty POTP-X EAP-Response message, and EAP servers send an EAP-Failure message possibly preceded by an EAP-Request of type Notification).

IV

An initialization vector for the encryption; see below. The length of the vector is dependent on the negotiated encryption algorithm, e.g., for AES-CBC, it shall be 16 octets. For some encryption algorithms, there may not be any initialization vector. An IV, when present, shall be randomly chosen and non-predictable.

Encrypted TLVs

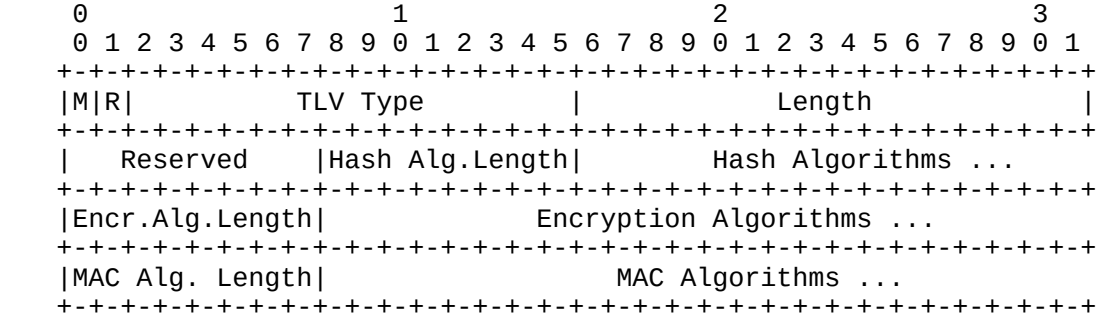
This field SHALL contain one or more encrypted POTP-X TLVs. The encryption algorithm SHALL be as negotiated; use K_ENC as the encryption key, and use the IV field as the initialization vector

(when applicable), to encrypt the concatenation of all the TLVs to be protected.

4.11.16. Crypto Algorithm TLV

The Crypto Algorithm TLV allows for negotiation of cryptographic algorithms. Cryptographic Algorithm negotiation is described in detail in Section 4.3.

This TLV MUST be present in the initial EAP-Request of type POTP-X that also carries an OTP TLV indicating protected mode, assuming the EAP server wants to negotiate use of any other algorithms than the default ones. It MAY also be present in an EAP-Request of type POTP-X that carries an OTP TLV that is sent as a result of a failed session resumption (in this case, the peer has not yet responded to this TLV), or when the Crypto Algorithm TLV was part of the initial message from the EAP server, and the client negotiated another EAP-POTP version than the highest one supported by the EAP server. The Crypto Algorithm TLV MUST NOT be present in any other EAP-Requests. Further, the Crypto Algorithm TLV MUST NOT be present in an EAP-Response of type POTP-X unless the preceding EAP-Request also contained it, and it was legal for it to do so. This TLV MUST be supported by all peers and all EAP servers conforming to this specification and MUST NOT be responded to with a NAK TLV.



M

1 - Mandatory TLV

R

Reserved for future use. This bit SHALL be set to zero (0) for this version. Recipients SHALL ignore this bit for this version of EAP-POTP.

TLV Type

15

Length

>=4 (at least one class of algorithms and one algorithm for that class needs to be present)

Reserved

Reserved for future use. This octet MUST be set to zero for this version. Recipients SHALL ignore this octet for this version of EAP-POTP.

Hash Alg. Length

The length of the Hash Algorithms field in octets.

Hash Algorithms

Each octet pair of this field represents a hash algorithm as follows. An EAP server MAY supply several suggestions for hash algorithms. Each algorithm MUST appear only once. The algorithms SHALL be supplied in order of priority. Peers MUST supply, at most, one algorithm (if none is present, the default applies). The defined values are:

Value Octet 1	Value Octet 2	Hash algorithm
0x00	0x00	Reserved
0x00	0x01	SHA-1
0x00	0x02	SHA-224
0x00	0x03	SHA-256 (default)
0x00	0x04	SHA-384
0x00	0x05	SHA-512
0x80	-	Vendor-specific (or experimental)

As indicated, values 0x8000 and higher are for proprietary vendor-specific algorithms. Values in the range 0x0006 - 0x7fff are to be assigned through IANA; see Section 7.

Encr Alg. Length

The length of the Encryption Algorithms field in octets.

Encryption Algorithms

Each octet pair of this field represents an encryption algorithm as follows. An EAP server MAY supply several suggestions for encryption algorithms. Each algorithm MUST appear only once. The algorithms SHALL be supplied in order of priority. Peers MUST supply, at most, one algorithm (if none is present, the default applies). The defined values are:

Value Octet 1	Value Octet 2	Encryption algorithm
0x00	0x00	Reserved
0x00	0x01	AES-CBC (default) with 128-bit keys and 16-octet IVs
0x00	0x02	3DES-CBC with 112-bit keys and 8-octet IVs
0x80	-	Vendor-specific

As indicated, values 0x8000 and higher are for vendor-specific proprietary algorithms. Values in the range 0x0003 - 0x7fff are to be assigned through IANA; see Section 7.

MAC Alg. Length

The length of the MAC Algorithms field in octets.

MAC Algorithms

Each octet pair of this field represents a MAC algorithm as follows. An EAP server MAY supply several suggestions for MAC algorithms. Each algorithm MUST appear only once. The algorithms SHALL be supplied in order of priority. Peers MUST supply, at most, one algorithm (if none is present, the default applies). The defined values are:

Value Octet 1	Value Octet 2	MAC algorithm
0x00	0x00	Reserved
0x00	0x01	HMAC (default)
0x80	-	Vendor-specific

As indicated, values 0x8000 and higher are for vendor-specific proprietary algorithms. Values in the range 0x0002 - 0x7fff are to be assigned through IANA; see Section 7.

When HMAC is negotiated, the hash algorithm used for HMAC SHALL be the negotiated hash algorithm.

5. EAP Key Management Framework Considerations

In line with recommendations made in [16], EAP-POTP defines the following identifiers to be associated with generated key material:

Peer-ID: The combined contents of the User Identifier TLV and the Token Key Identifier TLV.

Server-ID: The contents of the Server Identifier field of the Server-Info TLV.

Method-ID: The identifier of the established session (i.e., the contents of the Session Identifier field of the Server-Info TLV that defined the session).

6. Security Considerations

6.1. Security Claims

In conformance with RFC 3748 [1], the following security claims are made for the EAP-POTP method:

Authentication mechanism:	Generic OTP
Ciphersuite negotiation:	Yes (No in basic variant)
Mutual authentication:	Yes (No in basic variant)
Integrity protection:	Yes (No in basic variant)
Replay protection:	Yes (see below)
Confidentiality:	Only in the OTP protection variant, and then only OTP values and any information sent after exchange of the Confirm TLV
Key derivation:	Yes (No in basic variant)
Key strength:	Depends on size of OTP value, strength of underlying shared secret, strength and characteristics of OTP algorithm, pepper length, iteration count, and whether the method is used within a tunnel such as PEAPv2. For some illustrative examples, and a further discussion of this, see Appendix D.
Dictionary attack prot.:	N/A (Human-selected passwords not used)
Fast reconnect:	Yes
Crypt. binding:	N/A (EAP-POTP is not a tunnel method)
Session independence:	Yes
Fragmentation:	N/A (Packets shall not exceed MTU of 1020)
Channel binding:	Yes (No in basic variant)
Acknowledged S/F:	Yes
State Synchronization:	Yes (No in basic variant)

6.2. Passive and Active Attacks

The basic variant (i.e., when the protection of OTPs and mutual authentication is not used) of this EAP method does not provide session privacy, session integrity, server authentication, or protection from active attacks. In particular, man-in-the-middle attacks, where an attacker acts as an authenticator in order to acquire a valid OTP, are possible.

Similarly, the basic variant of this EAP method does not protect against session hijacking taking place after authentication. Nor does it, in itself, protect against replay attacks, where the attacker gains access by replaying a previous valid request, but see also the next subsection. When PIN codes are transmitted, they are sent without protection and are also subject to replay attacks.

In order to protect against these attacks, the peer MUST only use the basic variant of this method over a server-authenticated and confidentiality-protected connection. This can be achieved via use of, PEAPv2 [17], for example.

When the OTP protection variant is used, however, the EAP method provides privacy for OTPs and new PINs, negotiation of cryptographic algorithms, mutual authentication, and protection against replay attacks and protocol version downgrades. It also provides protection against man-in-the-middle attacks, not due to the infeasibility for a man-in-the-middle to solve for a valid OTP given an OTP TLV, but due to the computational expense of finding the OTP in the limited time period during which it is valid (this is mainly true for tokens, including the current time in their OTP calculations, or when a sent challenge has a certain lifetime). It should be noted, however, that a retrieved OTP, even if "old" and invalid, still may divulge some information about the user's PIN. Clearly, this is also true for the basic variant. Implementations of this EAP method, where user PINs are sent with OTPs, are therefore RECOMMENDED to ensure regular user PIN changes, regardless of whether the protected variant or the basic variant is employed.

It should also be noted that, while it is possible for a rogue access point, e.g., to clone MAC addresses, and hence mount a man-in-the-middle attack, such an access point will not be able to calculate the session keys MSK and EMSK. This demonstrates the importance of using the derived key material properly to protect a subsequent session.

Protected mode protects against version downgrade attacks due to the HMAC both parties transmit in this mode. As described, each party calculates the HMAC on sent and received EAP-POTP handshake messages. If an attacker were to modify a Version TLV, this would be reflected

in a difference between the calculated MACs (since the recipient of the Version TLV received a different value than the sender sent). Unless the attacker knows K_MAC, he cannot calculate the correct MAC, and hence the difference will be detected.

The OTP protection variant also protects against session hijacking, if the derived key material is used (directly or indirectly) to protect a subsequent session. For these reasons, use of the OTP protection variant is RECOMMENDED.

However, it should be noted that not even the OTP protection variant provides privacy for user names and/or token key identifiers. EAP-POTP MUST be used within a secure tunnel such as the one provided by PEAPv2 [17] if privacy for these parameters is required.

When resuming sessions created in the basic variant (which MUST only take place within a protected tunnel), the peer is authenticated by demonstrating knowledge of not just a valid session identifier, but also the OTP used when the session was created. Server nonces prevent replay attacks, but there still remains some likelihood of an attacker guessing the correct combination of session identifier and OTP value. Assuming OTPs with entropy about 32 bits, this means that the likelihood of succeeding with such an attack is about 1/2^48 due to the birthday paradox. Servers allowing session resumption for the basic variant MUST protect against such attacks, e.g., by keeping track of the rate of failed resumption attempts.

6.3. Denial-of-Service Attacks

An active attacker may replace the iteration count value in OTP TLVs sent by the peer to slow down an authentication server. Authentication servers SHOULD protect against this, e.g., by disregarding OTP TLVs with an iteration count value higher than some number that is preset or dynamically set (depending on load).

6.4. The Use of Pepper

As described in Section 4.8, the use of pepper will slow down an attacker's search for a matching OTP. The ability to transfer a pepper value in encrypted form from the EAP server to the peer means that, even though there may be an initial computational cost for the EAP server to authenticate the peer, subsequent authentications will be efficient, while at the same time more secure, since a pre-shared, 128-bit-long pepper value will not be easily found by an attacker. An attacker, observing an EAP-Request containing an OTP TLV calculated using a pepper chosen by the peer, may, however, depending on available resources, be able to successfully attack that particular EAP-POTP session, since it most likely will be based on a

relatively short pepper value or only an iteration count. Once the correct OTP has been found, eavesdropping on the EAP server's Confirm TLV will potentially give the attacker access to the longer, server-provided pepper for the remaining lifetime of that pepper value. For this reason, initial exchanges with EAP servers SHOULD occur in a secure environment (e.g., in a PEAPv2 tunnel offering cryptographic binding with inner EAP methods). If initial exchanges do not occur in a secure environment, the iteration count MUST be significantly higher than for messages where a pre-shared pepper is used. The lifetime of the shared pepper must also be calculated with this in mind. Finally, the peer and the EAP server MUST store the pepper value securely and associated with the user.

6.5. The Race Attack

In the case of fragmentation of EAP messages, it is possible (in the basic variant of this method) for an attacker to listen to most of an OTP, guess the remainder, and then race the legitimate user to complete the authentication. Conforming backend authentication server implementations MUST protect against this race condition. One defense against this attack is outlined below and borrowed from [14]; implementations MAY use this approach or MAY select an alternative defense. Note that the described defense relies on the user providing the identity in response to an initial Identity EAP-Request.

One possible defense is to prevent a user from starting multiple simultaneous authentication sessions. This means that once the legitimate user has initiated authentication, an attacker would be blocked until the first authentication process has completed. In this approach, a timeout is necessary to thwart a denial-of-service attack.

7. IANA Considerations

7.1. General

This document is a description of a general EAP method for OTP tokens. It also defines EAP method 32 as a profile of the general method. Extending the set of EAP-POTP TLVs or the set of EAP-POTP cryptographic algorithms shall be seen as revisions of the protocol and hence shall require an RFC that updates or obsoletes this document.

7.2. Cryptographic Algorithm Identifier Octets

A new registry for EAP-POTP cryptographic algorithm identifier octets has been created. The initial contents of this registry are as specified in Section 4.11.16.

Assignment of new values for hash algorithms, encryption algorithms, and MAC algorithms in the Crypto Algorithm TLV MUST be done through IANA with "Specification Required" and "IESG Approval" (see [9] for the meaning of these terms).

8. Intellectual Property Considerations

RSA, RSA Security, and SecurID are either registered trademarks or trademarks of RSA Security Inc. in the United States and/or other countries. The names of other products and services mentioned may be the trademarks of their respective owners.

9. Acknowledgments

This document was improved by comments from, and discussion with, a number of RSA Security employees. Simon Josefsson drafted the initial versions of an RSA SecurID EAP method while working for RSA Laboratories. The inspiration for the TLV-type of information exchange comes from [17]. Special thanks to Oliver Tavakoli of Funk Software who provided numerous useful comments and suggestions, Randy Chou of Aruba Networks for good suggestions in the session resumption area, and Jim Burns of Meetinghouse who provided inspiration for the Protected TLV. Thanks also to the IESG reviewers, Pasi Eronen, David Black, and Uri Blumenthal, for insightful comments that helped to improve the document, and to Alfred Hoenes for a thorough editorial review.

10. References

10.1. Normative References

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

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

[3] National Institute of Standards and Technology, "Secure Hash Standard", FIPS 180-2, February 2004.

[4] National Institute of Standards and Technology, "Specification for the Advanced Encryption Standard (AES)", FIPS 197, November 2001.

[5] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

[6] Kaliski, B., "PKCS #5: Password-Based Cryptography Specification Version 2.0", RFC 2898, September 2000.

[7] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

[8] Schulzrinne, H., "The tel URI for Telephone Numbers", RFC 3966, December 2004.

[9] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, October 1998.

10.2. Informative References

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

[11] The Institute of Electrical and Electronics Engineers, Inc., "IEEE Standard for Local and metropolitan area networks -- Port-Based Network Access Control", IEEE 802.1X-2001, July 2001.

[12] Kaufman, C., Ed., "Internet Key Exchange (IKEv2) Protocol", RFC 4306, December 2005.

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

[14] Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-Time Password System", STD 61, RFC 2289, February 1998.

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

[16] Aboba, B., Simon, D., Eronen, P., and H. Levkowetz, Ed., "Extensible Authentication Protocol (EAP) Key Management Framework", Work in Progress, October 2006.

[17] Palekar, A., Simon, D., Zorn, G., Salowey, J., Zhou, H., and S. Josefsson, "Protected EAP Protocol (PEAP) Version 2", Work in Progress, October 2004.

[18] Internet Assigned Numbers Authority, "Private Enterprise Numbers", December 2006.

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

Appendix A. Profile of EAP-POTP for RSA SecurID

Note: The RSA SecurID product is a hardware token card (or software emulation thereof) produced by RSA Security Inc., which is used for end-user authentication.

The EAP method type identifier for the RSA SecurID profile of EAP-POTP is 32.

Peers and EAP servers implementing the SecurID profile of EAP-POTP SHALL conform to all EAP-POTP normative requirements in this Document. In addition, the New PIN TLV and the Protected TLV MUST be supported by peers.

Appendix B. Examples of EAP-POTP Exchanges

This appendix is non-normative. In the examples, "V1", "V2", "V3", etc., stand for arbitrary values of the correct type.

B.1. Basic Mode, Unilateral Authentication

This mode should only be used within a secured tunnel. The peer identifies itself with a User Identifier TLV.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=0,C=0,N=0,T=0,E=0,R=0

EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0

OTP TLV:
P=0,C=0,N=0,T=0,E=0,R=0
Authentication Data=V1

User Identifier TLV:
User Identifier=V2

                                        <- EAP-Success
```

B.2. Basic Mode, Session Resumption

This example illustrates successful resumption of a basic mode session. It must be carried out only in a protected tunnel.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=0,C=0,N=0,T=0,E=0,R=0

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V1
                                           Server  Identifier=V2
                                           Nonce=V3
EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0

Resume TLV:
Session Identifier=V4 (indicating earlier, basic mode, session)
Authentication Data=V5

                                        <- EAP-Success
```

B.3. Mutual Authentication without Session Resumption

In this case, the peer uses the token key identifier, in addition to the user identifier. The initial EAP-Identity exchange may also provide user information, or may be restricted to only general domain information. Pepper is not used, but will be used in a subsequent session since the server provides the peer with an encrypted pepper in its Confirm TLV. Absence of the Crypto Algorithm TLV indicates use of default cryptographic algorithms.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V1
                                           Server  Identifier=V2
                                           Nonce=V3

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=0
                                           Iteration Count=V4

EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0

OTP TLV:
P=1,C=0,N=0,T=0,E=0,R=0
Pepper Length=0
Iteration Count=V4
Authentication Data=V5
```

```
   User Identifier TLV:
   User Identifier=V6

   Token Key Identifier TLV:
   Token Key Identifier=V7

                                        <- EAP-Request
                                           Type=OTP-X

                                           Confirm TLV:
                                           C=0
                                           Authentication Data=V8
                                           Pepper Identifier=V9
                                           Encrypted Pepper=V10

   EAP-Response ->
   Type=OTP-X

   Confirm TLV:
   (no data)

                                        <- EAP-Success
```

B.4. Mutual Authentication with Transfer of Pepper

The difference between this example and the previous one is that the peer makes use of an existing pepper in the PBKDF2 computation. The EAP server provides a new pepper to the peer in the Confirm TLV. Note that the peer had not been able to use a pepper in the response calculation unless it had found the existing pepper, since the server specified a maximum (new) pepper length of zero.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V1
                                           Server  Identifier=V2
                                           Nonce=V3

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=0
                                           Iteration Count=V4

EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0

OTP TLV:
P=1,C=0,N=0,T=0,E=0,R=0
Pepper Length=V5
Iteration Count=V6
Authentication Data=V7
(includes a pepper identifier)
```

```
   User Identifier TLV:
   User Identifier=V8

   Token Key Identifier TLV:
   Token Key Identifier=V9

                                        <- EAP-Request
                                           Type=OTP-X

                                           Confirm TLV:
                                           C=0
                                           Authentication Data=V10
                                           Pepper Identifier=V11
                                           Encrypted Pepper=V12

   EAP-Response ->
   Type=OTP-X

   Confirm TLV:
   (no data)

                                        <- EAP-Success
```

B.5. Failed Mutual Authentication

This example differs from the previous one in that the peer is not able to authenticate the server. Therefore, it sends an empty EAP-Response of type POTP-X, which the EAP server acknowledges by responding with an EAP-Failure. Pepper is not used.

```
   Peer                                 EAP server

                                        <- EAP-Request
                                           Type=Identity

   EAP-Response ->
   Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V1
                                           Iteration Count=V2
```

```
                                        Server-Info TLV:
                                        N=0
                                        Session Identifier=V3
                                        Server  Identifier=V4
                                        Nonce=V5

   EAP-Response ->
   Type=OTP-X

   Version TLV:
   Highest=0

   OTP TLV:
   P=1,C=0,N=0,T=0,E=0,R=0
   Pepper Length=V1
   Iteration Count=V2
   Authentication Data=V6

   User Identifier TLV:
   User Identifier=V7

   Token Key Identifier TLV:
   Token Key Identifier=V8

                                     <- EAP-Request
                                        Type=OTP-X

                                        Confirm TLV:
                                        C=0
                                        Authentication Data=V9

   EAP-Response ->
   Type=OTP-X

   (no data)

                                     <- EAP-Failure
```

B.6. Session Resumption

This example illustrates successful session resumption.

```
   Peer                              EAP server

                                     <- EAP-Request
                                        Type=Identity
```

```
   EAP-Response ->
   Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V1
                                           Iteration Count=V2

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V3
                                           Server  Identifier=V4
                                           Nonce=V5

   EAP-Response ->
   Type=OTP-X

   Version TLV:
   Highest=0

   Resume TLV:
   Session Identifier=V6 (indicating earlier, protected mode, session)
   Authentication Data=V7

                                        <- EAP-Request
                                           Type=OTP-X

                                           Confirm TLV:
                                           C=0
                                           Authentication Data=V8

   EAP-Response ->
   Type=OTP-X
   Confirm TLV:
   (no data)

                                        <- EAP-Success
```

B.7. Failed Session Resumption

This example illustrates a failed session resumption, followed by a complete mutual authentication. The user is identified through the User Identifier TLV. The client is able to reuse an older pepper. The server sends a new pepper for subsequent use in its Confirm TLV. The server suggests some non-default cryptographic algorithms, but the client only supports the default ones.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V1
                                           Iteration Count=V2

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V3
                                           Server  Identifier=V4
                                           Nonce=V5

                                           Crypto Algorithm TLV:
                                           Hash Alg. Length=V6
                                           Hash Algorithms=V7
                                           Encr. Alg. Length=V8
                                           Encr. Algorithms=V9
                                           MAC Alg. Length=V10
                                           MAC Algorithms=V11

EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0
```

```
   Resume TLV:
   Session Identifier=V12 (indicating earlier session)
   Authentication Data=V13

                                        <- EAP-Request
                                           Type=OTP-X

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V14
                                           Iteration Count=V15

                                           Server-Info TLV:
                                           N=1 (no resumption)
                                           Session Identifier=V3
                                           Server  Identifier=V4
                                           Nonce=V16

   EAP-Response ->
   Type=OTP-X

   OTP TLV:
   P=1,C=0,N=1,T=1,E=0,R=0
   Pepper Length=V17
   Iteration Count=V18
   Authentication Data=V19 (with pepper identifier)

   User Identifier TLV:
   User Identifier=V20

                                        <- EAP-Request
                                           Type=OTP-X

                                           Confirm TLV:
                                           C=0
                                           Authentication Data=V21
                                           Pepper Identifier=V22
                                           Encrypted Pepper=V23
   EAP-Response ->
   Type=OTP-X

   Confirm TLV:
   (no data)

                                        <- EAP-Success
```

B.8. Mutual Authentication, and New PIN Requested.

In this example, the user is also requested to select a new PIN. The new PIN is allowed to be alphanumeric, and must be at least 6 characters long. The user selects another PIN than the one suggested by the server. The token key is identified through a combination of the user identifier and the token key identifier. While waiting for the user input, to avoid network timeouts, the peer sends an EAP-Response containing a Keep-Alive TLV to the EAP server. The EAP server responds by sending an EAP-Request containing a Keep-Alive TLV back to the peer. Note that all TLVs exchanged after the Confirm TLV exchange are wrapped in the Protected TLV. Absence of the Crypto Algorithm TLV indicates use of default cryptographic algorithms.

```
Peer                                    EAP server

                                        <- EAP-Request
                                           Type=Identity

EAP-Response ->
Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V1
                                           Iteration Count=V2

                                           Server-Info TLV:
                                           N=0
                                           Session Identifier=V3
                                           Server  Identifier=V4
                                           Nonce=V5

EAP-Response ->
Type=OTP-X

Version TLV:
Highest=0

OTP TLV:
P=1,C=0,N=0,T=0,E=0,R=0
Pepper Length=V6
```

```
   Iteration Count=V7
   Authentication Data=V8 (with pepper identifier)

   User Identifier TLV:
   User Identifier=V9

   Token Key Identifier TLV:
   Token Key Identifier=V10

                                        <- EAP-Request
                                           Type=OTP-X

                                           Confirm TLV:
                                           C=1
                                           Authentication Data=V11

   EAP-Response ->
   Type=OTP-X

   Confirm TLV:
   (no data)

                                        <- EAP-Request
                                           Type=OTP-X

                                           Protected TLV:
                                           MAC=V12
                                           IV=V13
                                           Encrypted TLVs=V14
                                           (Contains:
                                           New PIN TLV:
                                           Q=0,A=1
                                           PIN=V15
                                           Min. PIN Length=6)

   EAP-Response ->
   Type=OTP-X

   Protected TLV:
   MAC=V16
   IV=V17
   Encrypted TLVs=V18
   (Contains:
   Keep-Alive TLV:
   (no data))

                                        <- EAP-Request
                                           Type=OTP-X
```

```
                                          Protected TLV:
                                          MAC=V19
                                          IV=V20
                                          Encrypted TLVs=V21
                                          (Contains:
                                          Keep-Alive TLV:
                                          (no data))

   EAP-Response ->
   Type=OTP-X

   Protected TLV:
   MAC=V22
   IV=V23
   Encrypted TLVs=V24
   (Contains:
   New PIN TLV:
   Q=0,A=0
   PIN=V25)

                                       <- EAP-Request
                                          Type=OTP-X

                                          Protected TLV:
                                          MAC=V26
                                          IV=V27
                                          Encrypted TLVs=V28
                                          (Contains:
                                          OTP TLV:
                                          P=1,C=0,N=0,T=0,E=0,R=0
                                          Pepper Length=V1
                                          Iteration Count=V2)

   EAP-Response ->
   Type=OTP-X

   Protected TLV
   MAC=V29
   IV=V30
   Encrypted TLVs=V31
   (Contains:
   OTP TLV:
   P=1,C=0,N=0,T=0,E=0,R=0
   Pepper Length=V6
   Iteration Count=V7
   Authentication Data=V31)
```

```
                                        <- EAP-Request
                                           Type=OTP-X

                                           Protected TLV
                                           MAC=V32
                                           IV=V33
                                           Encrypted TLVs=V34
                                           (Contains:
                                           Confirm TLV:
                                           C=0
                                           Authentication Data=V35)

   EAP-Response ->
   Type=OTP-X

   Protected TLV
   MAC=V36
   IV=V37
   Encrypted TLVs=V38
   (Contains:
   Confirm TLV:
   (no data))

                                        <- EAP-Success
```

B.9. Use of Next OTP Mode

In this example, the peer is requested to provide a second OTP to the EAP server.

```
   Peer                                 EAP server

                                        <- EAP-Request
                                           Type=Identity

   EAP-Response ->
   Type=Identity

                                        <- EAP-Request
                                           Type=OTP-X

                                           Version TLV:
                                           Highest=0,Lowest=0

                                           OTP TLV:
                                           P=1,C=0,N=0,T=0,E=0,R=0
                                           Pepper Length=V1
                                           Iteration Count=V2
```

```
                                        Server-Info TLV:
                                        N=0
                                        Session Identifier=V3
                                        Server  Identifier=V4
                                        Nonce=V5

   EAP-Response ->
   Type=OTP-X

   Version TLV:
   Highest=0

   OTP TLV:
   P=1,C=0,N=0,T=0,E=0,R=0
   Pepper Length=V6
   Iteration Count=V7
   Authentication Data=V8

   User Identifier TLV:
   User Identifier=V9

                                     <- EAP-Request
                                        Type=OTP-X

                                        OTP TLV:
                                        P=1,C=0,N=1,T=1,E=0,R=0
                                        Pepper Length=V1
                                        Iteration Count=V2

   EAP-Response ->
   Type=OTP-X

   OTP TLV:
   P=1,C=0,N=1,T=1,E=0,R=0
   Pepper Length=V6
   Iteration Count=V7
   Authentication Data=V10

                                     <- EAP-Request
                                        Type=OTP-X

                                        Confirm TLV:
                                        C=0
                                        Authentication Data=V11

   EAP-Response ->
   Type=OTP-X
```

```
   Confirm TLV:
   (no data)

                                        <- EAP-Success
```

Appendix C. Use of the MPPE-Send/Receive-Key RADIUS Attributes

C.1. Introduction

This section describes how to populate the MPPE-Send-Key and the MPPE-Receive-Key RADIUS attributes defined in [19], using an MSK established in EAP-POTP.

C.2. MPPE Key Attribute Population

Once the EAP-POTP MSK has been generated, it is used as follows to populate the MPPE-Send-Key and the MPPE-Receive-Key attributes:

Use the initial 32 octets of the MSK as the value for the "Key" sub-field in the plaintext "String" field of the MPPE-Send-Key attribute, and use the final 32 octets of the MSK as the "Key" sub-field in the plaintext "String" field of the MPPE-Receive-Key attribute (Note: "Send" and "Receive" here refer to the Authenticator; for the peer, they are reversed).

Appendix D. Key Strength Considerations

D.1. Introduction

As described in Section 6, the strength of keys generated in EAP-POTP protected mode depends on a number of factors. This appendix provides examples of actual key strengths achieved under various assumptions.

It should be noted that, while some of the examples indicate that the strength of generated keys is relatively weak, the strength applies only to those EAP-POTP sessions between a peer and an EAP server that do not share a pepper. Once a pepper, provided by an EAP server to a peer, has been established, future sessions using this pepper will provide full-strength keys.

D.2. Example 1: 6-Digit One-Time Passwords

In this example we assume the following:

OTPs are six decimal digits long;

4-digit PINs are added to generated OTPs; and

OTP hardening (iteration count and pepper searching combined) effectively adds 10 bits of entropy. One way of achieving this without use of pepper searching is to have the iteration count in PBKDF2 set to 1,000,000.

The effective key strength then becomes roughly:

log_2(10**6) + log_2(10**4) + log_2(2**10) = 43 bits

The above assumes that the entropy of the underlying shared secret is >43 bits and that there are no other weaknesses in the OTP algorithm.

D.3. Example 2: 8-Digit One-Time Passwords

In this example we assume the following:

OTPs are eight decimal digits long;

4-character alphanumeric PINs are added to generated OTPs; and

OTP hardening (iteration count and pepper searching combined) effectively adds 10 bits of entropy.

The effective key strength then becomes roughly:

log_2(10**8) + log_2(26**4) + log_2(2**10) = 55 bits

The above assumes that the entropy of the underlying shared secret is >55 bits and that there are no other weaknesses in the OTP algorithm.

Author's Address

Magnus Nystroem
RSA Security

EMail: magnus@rsa.com

Full Copyright Statement

Intellectual Property

Acknowledgement

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