Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: December 31, 2009

D. Harkins
Aruba Networks
G. Zorn
NetCube

June 29, 2009

EAP Authentication Using Only A Password
draft-harkins-emu-eap-pwd-04

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79. This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

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

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

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

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

This Internet-Draft will expire on December 31, 2009.

Copyright Notice

Copyright (c) 2009 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents in effect on the date of publication of this document (http://trustee.ietf.org/license-info). Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Abstract

This memo describes an Extensible Authentication Protocol (EAP) method, EAP-pwd, which uses a shared password for authentication. The password may be a low-entropy one and may be drawn from some set of possible passwords, like a dictionary, which is available to an attacker.

Table of Contents

1. Introduction 4
  1.1. Background 4
  1.2. Keyword Definitions 4
  1.3. Requirements 4
    1.3.1. Resistance to Passive Attack 4
    1.3.2. Resistance to Active Attack 5
    1.3.3. Resistance to Dictionary Attack 5
    1.3.4. Forward Secrecy 5
2. Specification of EAP-pwd 5
  2.1. Notation 5
    2.1.1. Prime Modulus Groups 6
    2.1.2. Elliptic Curve Groups 7
  2.2. Assumptions 8
  2.3. Instantiating the Random Function 8
  2.4. Key Derivation Function 9
  2.5. Random Numbers 9
  2.6. Protocol 10
    2.6.1. Overview 10
    2.6.2. Message Flows 10
    2.6.3. Fixing the Password Element 12
      2.6.3.1. Elliptic Curve PWE 12
      2.6.3.2. Prime Modulus pwe 13
    2.6.4. Message Construction 14
      2.6.4.1. Elliptic Curve Groups 14
      2.6.4.2. Prime Modulus Groups 15
    2.6.5. Message Processing 16
      2.6.5.1. EAP-pwd-ID Exchange 16
      2.6.5.2. EAP-pwd-Commit Exchange 18
      2.6.5.3. EAP-pwd-Confirm Exchange 19
  2.7. Management of EAP-pwd Keys 19
3. Packet Formats 20

3.1. EAP-pwd Header 20
3.2. EAP-pwd Payloads 22
3.2.1. EAP-pwd-ID 22
3.2.2. EAP-pwd-Commit 23
3.2.3. EAP-pwd-Confirm 24
3.3. Representation of Field Elements 25
3.3.1. Prime Modulus Groups 25
3.3.2. Elliptic Curve Groups 25
4. Fragmentation 25
5. IANA Considerations 26
6. Security Considerations 27
6.1. Resistance to Passive Attack 28
6.2. Resistance to Active Attack 28
6.3. Resistance to Dictionary Attack 28
6.4. Forward Secrecy 30
6.5. Random Functions 31
7. Security Claims 31
8. Acknowledgements 34
9. References 34
9.1. Normative References 34
9.2. Informative References 35
Authors' Addresses 36

1. Introduction

1.1. Background

The predominant access method for the Internet today is that of a human using a username and password to authenticate to a computer enforcing access control. Proof of knowledge of the password authenticates the human and computer.

Typically these passwords are not stored on a user's computer for security reasons and must be entered each time the human desires network access. Therefore the passwords must be ones that can be repeatedly entered by a human with a low probability of error. They will likely not possess high-entropy and it may be assumed that an adversary with access to a dictionary will have the ability to guess a user's password. It is therefore desirable to have a robust authentication method that is secure even when used with a weak password in the presence of a strong adversary.

EAP-pwd is an EAP method that addresses the problem of password-based authenticated key exchange-- using a possibly weak password for authentication to derive an authenticated and cryptographically strong shared secret. This problem was first described by Bellovin and Merritt in [BM92] and [BM93]. There have been a number of subsequent suggestions ([JAB96], [LUC97], [BMP00], and others) for password-based authenticated key exchanges.

1.2. Keyword Definitions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.3. Requirements

Any protocol that claims to solve the problem of password-authenticated key exchange must be resistant to active, passive and dictionary attack and have the quality of forward secrecy. These characteristics are discussed further in the following sections.

1.3.1. Resistance to Passive Attack

A passive, or benign, attacker is one that merely relays messages back and forth between the peer and server, faithfully, and without modification. The contents of the messages are available for inspection, but that is all. To achieve resistance to passive attack, such an attacker must not be able to obtain any information about the password or anything about the resulting shared secret from

watching repeated runs of the protocol. Even if a passive attacker is able to learn the password, she will not be able to determine any information about the resulting secret shared by the peer and server.

1.3.2. Resistance to Active Attack

An active attacker is able to modify, add, delete, and replay messages sent between protocol participants. For this protocol to be resistant to active attack, the attacker must not be able to obtain any information about the password or the shared secret by using any of its capabilities. In addition, the attacker must not be able to fool a protocol participant into thinking that the protocol completed successfully.

It is always possible for an active attacker to deny delivery of a message critical in completing the exchange. This is no different than dropping all messages and is not an attack against the protocol.

1.3.3. Resistance to Dictionary Attack

For this protocol to be resistant to dictionary attack any advantage an adversary can gain must be directly related to the number of interactions she makes with an honest protocol participant and not through computation. The adversary will not be able to obtain any information about the password except whether a single guess from a single protocol run is correct or incorrect.

1.3.4. Forward Secrecy

Compromise of the password must not provide any information about the secrets generated by earlier runs of the protocol.

2. Specification of EAP-pwd

2.1. Notation

The following notation is used in this memo:

peer-ID
    The peer's identity, the peer NAI [RFC4282].

server-ID
    A string that identifies the server to the peer.

password
    The password shared between the peer and server.

y = H(x)
    The binary string x is given to a function H which produces an output y.

a | b
    denotes concatenation of string a with string b.

g^x mod p
    indicates multiplication of the value "g" with itself "x" times, modulo the value "p".

inv(Q)
    indicates the inverse of an element, Q, from a finite field.

len(x)
    indicates the length in bits of the string x.

chop(x, y)
    is reduction of string x, being at least y bits in length, to y bits.

LSB(x)
    returns the least-significant bit of the bitstring "x".

CipherSuite
    an encoding of a finite cyclic group to use with EAP-pwd, the definition of function H, and a PRF, in that order.

MSK
    The Master Session Key exported by EAP-pwd. This is a high-entropy secret 512 bits in length.

EMSK
    The Extended Master Session Key exported by EAP-pwd. This is a high-entropy secret 512 bits in length.

This protocol uses a finite cyclic group in which the discrete logarithm problem is known to be hard. Groups can be either based on exponentiation modulo a prime or on an elliptic curve.

2.1.1. Prime Modulus Groups

Groups formed by a prime modulus have a generator, g, a prime modulus p, optionally an order r. The group operation is exponentiation modulus the prime:

y = g^x mod p

the generator taken to the x-th power modulo the prime returns an element in the group.

If the order of the generator of the group is part of the group definition that value MUST be used as the order of the group, r, when an order is called for, otherwise the order, r, MUST be computed as the prime minus one divided by two-- (p-1)/2.

The inverse function for a prime modulus group is defined such that the product of an element and its inverse modulo the group prime equals one (1). In other words,

(q * inv(q)) mod p = 1

2.1.2. Elliptic Curve Groups

Elliptic curve over GF(2^m) SHALL NOT be used by EAP-pwd. While such groups exist in the IANA registry used by EAP-pwd their use in EAP-pwd is not defined.

Elliptic curves over GF(p) are defined by a curve equation, y^2 = x^3 + ax + b, for a defined "a" and "b". Groups formed by an elliptic curve over GF(p) have a generator G, a prime p, a cofactor f, and an order r. The group operation is multiplication of a point on the curve by itself a number of times:

Y = x*P

the point P is multiplied x-times to produce another point on the curve, Y.

The convention for this memo to represent a point on a curve is to use an upper-case letter while a scalar is indicated with a lower-case letter.

Elliptic curve groups require a mapping function, q = F(Q), to convert a group element to an integer. The mapping function used in this memo returns the x-coordinate of the point it is passed.

The inverse function for an elliptic group is defined such that the sum of an element and its inverse is the "point at infinity", denoted here as "O". In other words,

Q + inv(Q) = O

2.2. Assumptions

In order to see how the protocol addresses the requirements above (see Section 1.3) it is necessary to state some assumptions under which the protocol can be evaluated. They are:

1. Function H maps a binary string of indeterminate length onto a fixed binary string that is x bits in length. That is H: {0,1}^* --> {0,1}^x.

2. Function H is a "random oracle" (see [RANDOR]). Given knowledge of the input to H an adversary is unable to distinguish the output of H from a random data source.

3. Function H is a one-way function. Given the output of H it is computationally infeasible for an adversary to determine the input.

4. For any given input to function H each of the 2^x possible outputs are equally probable.

5. The discrete logarithm problem for the chosen finite cyclic group is hard. That is, given g, p and y = g^x mod p it is computationally infeasible to determine x. Similarly for an elliptic curve group given the curve definition, a generator G, and Y = x * G it is computationally infeasible to determine x.

6. There exists a pool of passwords from which the password shared by the peer and server is drawn. This pool can consist of words from a dictionary, for example. Each password in this pool has an equal probability of being the shared password. All potential attackers have access to this pool of passwords.

2.3. Instantiating the Random Function

The protocol described in this memo uses a random function, H. As noted in Section 2.2 this is a "random oracle" as defined in [RANDOR]. At first glace one may view this as a hash function. As noted in [RANDOR], though, hash functions are too structured to be used directly as a random oracle. But they can be used to instantiate the random oracle. [RANDOR] suggests several ways to instantiate a random function with a hash function, one of those is used here.

The random function, H, in this memo is instantiated by concatenating the function's input with itself and running it through SHA-256 (see [FIPS.180-2.2002]). In other words,

```
H(x) = SHA-256(x | x)
```

2.4. Key Derivation Function

The keys output by this protocol, MSK and EMSK, are 512 bits in length each. The shared secret that results from the successful termination of this protocol is only 256 bits. Therefore it is necessary to stretch the shared secret using a key derivation function (KDF).

The KDF used in this protocol has a counter-mode with feed-back construction using a generic pseudo-random function (PRF), according to [SP800-108]. The specific value of the PRF is specified along with the random function and finite cyclic group when the server sends the first EAP-pwd packet to the peer.

The KDF takes a key to stretch, a label to bind into the key, and an indication of the desired length of the output. Algorithmically it is:

```
KDF(key, label, length) {
  i = 1
  L = length
  res = PRF(key, i | label | L)
  K(1) = res
  while (len(res) < length)
  do
    i = i + 1
    K(i) = PRF(key, K(i-1) | i | label | L)
    res = res | K(i)
  done
  return chop(res, length)
}

where "i" and "L" are 16-bits in length
```

Figure 1: Key Derivation Function

2.5. Random Numbers

The security of EAP-pwd relies upon each side, the peer and server, producing quality secret random numbers. A poor random number chosen by either side in a single exchange can compromise the shared secret from that exchange and open up the possibility of dictionary attack.

Producing quality random numbers without specialized hardware entails using a cryptographic mixing function (like a strong hash function) to distill entropy from multiple, uncorrelated sources of information

and events. A very good discussion of this can be found in [RFC4086].

2.6. Protocol

2.6.1. Overview

EAP is a two-party protocol spoken between an EAP peer and an authenticator. For scaling purposes the functionality of the authenticator that speaks EAP is frequently broken out into a stand-alone EAP server. In this case the EAP peer communicates with an EAP server through the authenticator with the authenticator merely being a passthrough.

An EAP method defines the specific authentication protocol being used by EAP. This memo defines a particular method and therefore defines the messages sent between the EAP server (or the "EAP server" functionality in an authenticator if it is not broken out) and the EAP peer for the purpose of authentication and key derivation.

2.6.2. Message Flows

EAP-pwd defines three message exchanges, an Identity exchange, a Commit exchange and a Confirm exchange. A successful authentication is shown in Figure 2.

The peer and server use the Identity exchange to discover each other's identities and to agree upon a ciphersuite to use in the subsequent exchanges; in addition, the EAP Server uses the EAP-pwd-ID/Request message to inform the client of any password preprocessing that may be required. In the Commit exchange the peer and server exchange information to generate a shared key and also to bind each other to a particular guess of the password. In the Confirm exchange the peer and server prove liveness and knowledge of the password by generating and verifying verification data.

```
           +--------+                                     +--------+
           |        |                  EAP-pwd-ID/Request |        |
           |  EAP   |<------------------------------------|  EAP   |
           |  peer  |                                     | server |
           |        | EAP-pwd-ID/Response                 |        |
           |        |------------------------------------>|        |
           |        |                                     |        |
           |        |              EAP-pwd-Commit/Request |        |
           |        |<------------------------------------|        |
           |        |                                     |        |
           |        | EAP-pwd-Commit/Response             |        |
           |        |------------------------------------>|        |
           |        |                                     |        |
           |        |             EAP-pwd-Confirm/Request |        |
           |        |<------------------------------------|        |
           |        |                                     |        |
           |        | EAP-pwd-Confirm/Response            |        |
           |        |------------------------------------>|        |
           |        |                                     |        |
           |        |          EAP-Success                |        |
           |        |<------------------------------------|        |
           +--------+                                     +--------+
```

Figure 2: A Successful EAP-pwd Exchange

The components of the EAP-pwd-* messages are as follows:

EAP-pwd-ID/Request
    Server_ID, Ciphersuite

EAP-pwd-ID/Response
    Peer_ID, Ciphersuite

EAP-pwd-Commit/Request
    Scalar_S, Element_S

EAP-pwd-Commit/Response
    Scalar_P, Element_P

EAP-pwd-Confirm/Request
    Confirm_S

EAP-pwd-Confirm/Response
    Confirm_P

2.6.3. Fixing the Password Element

Once the EAP-pwd-ID exchange is completed the peer and server use each other's identities and the agreed upon ciphersuite to fix an element in the finite cyclic group called the Password Element (PWE or pwe, for an element in an ellipic curve group or a prime modulus group, respectively). The resulting element must be selected in a determinstic fashion using the password but must result in selection of an element that will not leak any information on the password to an attacker. From the point-of-view of an attacker who does not know the password, PWE will be a random element in a finite cyclic group.

To properly fix the Password Element both parties must have a common view of the string "password". Therefore, if a password preprocessing algorithm was negotiated during the EAP-pwd-ID exchange the client MUST perform the specified password pre-processing prior to fixing the Password Element.

The actual fixing of the Password Element depends on the type of finite cyclic group being used.

2.6.3.1. Elliptic Curve PWE

For a finite cyclic group based on an elliptic curve it is necessary to use an iterative hunt-and-peck technique to fix the password element.

First, an 8-bit counter is incremented to the value one (1). Then, the agreed upon random function is used to generate a password seed:

```
pwd-seed = H(peer-ID | server-ID | password | counter)
```

Then, the pwd-seed is expanded using the KDF from the agreed-upon Ciphersuite out to the length of the prime of the curve, modulo the prime of the curve.

```
pwd-value = KDF(pwd-seed, "EAP-pwd Hunting And Pecking", len(p))

pwd-value = pwd-value mod p
```

The pwd-value is used as the x-coordinate, x, with the equation for the elliptic curve to solve for a y-coordinate, y. If there is no solution to the quadratic equation the counter is incremented, a new pwd-seed is generated and the hunting and pecking continues. If a solution is found then an ambiguity exists as there are technically two solutions to the equation and pwd-seed is used to unambiguously select one of them. If the low-order bit of pwd-seed is equal to the low-order bit of y then a candidate PWE is defined as the point (x,

y); if the low-order bit of pwd-seed differs from the low-order bit of y then a candidate PWE is defined as the point (x, p - y), where p is the prime over which the curve is defined. The order of the candidate PWE is then checked to make sure it can safely be used as a generator in the protocol. If the co-factor of the curve multiplied by the candidate PWE equals the point-at-infinity then the candidate PWE is discarded, the counter is incremented, a new pwd-seed is generated and the hunting and pecking continues. If it does not equal the point-at-infinity the candidate PWE becomes the PWE and hunting and pecking terminates. (Note: the point multiplied by the co-factor does not become PWE, it is only used to determine the order of the group defined with PWE as a generator).

Algorithmically, the process looks like this:

```
found = 0
counter = 1
do {
  pwd-seed = H(peer-ID | server-ID | password | counter)
  pwd-value = KDF(pwd-seed, "EAP-pwd Hunting And Pecking", len(p))
  x = pwd-value mod p
  if ( (y = sqrt(x^3 + ax + b)) != FAIL)
  then
    if (LSB(y) == LSB(pwd-seed))
    then
      PWE = (x,y)
    else
      PWE = (x, p-y)
    fi
    P = f*PWE
    if (P != "O")
      found = 1
    fi
  fi
  counter = counter + 1
} while (found == 0)
```

Figure 3: Fixing PWE for Elliptic Curves

2.6.3.2. Prime Modulus pwe

For a finite cyclic group based on exponentiation of a generator, g, modulo a large prime p it is not necessary to hunt-and-peck to find pwe. A pwe can be computed in a sub-field of the group directly using the prime, p, and order r.

First, the agreed upon random function is used to generate a password seed:

pwd-seed = H(peer-ID | server-ID | password)

Then the pwd-seed is expanded using the KDF to the length of the prime, modulo the prime.

pwd-value = KDF(pwd-seed, "EAP-pwd Affixing the PWE", len(p))

pwd-value = pwd-value mod p

The pwe is then computed by exponentiating the pwd-value to the value ((p-1)/r) modulo the prime.

pwe = pwd-value ^ ((p-1)/r) mod p

2.6.4. Message Construction

After the EAP-pwd Identity exchange the construction of the components of each message depends on the finite cyclic group from the ciphersuite.

2.6.4.1. Elliptic Curve Groups

Assuming an elliptic curve group with order r:

```
Server: EAP-pwd-Commit/Request
   - choose two random numbers, 1 < s_rand, s_mask < r
   - compute Element_S = inv(s_mask * PWE)
   - compute Scalar_S = (s_rand + s_mask) mod r

 Element_S and Scalar_S are used to construct EAP-pwd-Commit/Request

Peer: EAP-pwd-Commit/Response
   - choose two random numbers, 1 < p_rand, p_mask < r
   - compute Element_P = inv(p_mask * PWE)
   - compute Scalar_P = (p_rand + p_mask) mod r

 Element_P and Scalar_P are used to construct EAP-pwd-Commit/Response

Server: EAP-pwd-Confirm/Request
   - compute KS = s_rand * (Scalar_P * PWE + Element_P)
   - compute ks = F(KS)
   - compute Confirm_S = H(ks | Element_S | Scalar_S |
                           Element_P | Scalar_P | Ciphersuite)

 Confirm_S is used to construct EAP-pwd-Confirm/Request

Peer: EAP-pwd-Confirm/Response
   - compute KP = p_rand * (Scalar_S * PWE + Element_S)
   - compute kp = F(KP)
   - compute Confirm_P = H(kp | Element_P | Scalar_P |
                           Element_S | Scalar_S | Ciphersuite)

 Confirm_P is used to construct EAP-pwd-Confirm/Response

The EAP Server computes the shared secret as:
  MK = H(ks | F(Element_S+Element_P) | (Scalar_S+Scalar_P) mod r)

The EAP Peer computes the shared secet as:
  MK = H(kp | F(Element_P+Element_S) | (Scalar_P+Scalar_S) mod r)
```

The MSK and EMSK are derived from MK per Section 2.7.

2.6.4.2. Prime Modulus Groups

When using a finite cyclic group based on exponentiation of a generator (g) modulo a prime (p), a subgroup order (r) may or may not be specified. If it is not specified r is set to p-1 for use in this protocol. Also, there is no mapping function needed when using such a group.

```
Server: EAP-pwd-Commit/Request
   - choose two random numbers, 1 < s_rand, s_mask < r
   - compute Element_S = inv(pwe^s_mask mod p)
   - compute Scalar_S = (s_rand + s_mask) mod r

 Element_S and Scalar_S are used to construct EAP-pwd-Commit/Request

Peer: EAP-pwd-Commit/Response
   - choose random two numbers, 1 < p_rand, p_mask < r
   - compute Element_P = inv(pwe^p_mask mod p)
   - compute Scalar_P = (p_rand + p_mask) mod r

 Element_P and Scalar_P are used to construct EAP-pwd-Commit/Response

Server: EAP-pwd-Confirm/Request
   - compute ks = ((pwe^Scalar_P mod p) * Element_P)^s_rand mod p
   - compute Confirm_S = H(ks | Element_S | Scalar_S |
                           Element_P | Scalar_P | Ciphersuite)

 Confirm_S is used to construct EAP-pwd-Confirm/Request

Peer: EAP-pwd-Confirm/Response
   - compute kp = ((pwe^Scalar_S mod p) * Element_S)^p_rand mod p
   - compute Confirm_P = H(kp | Element_P | Scalar_P |
                           Element_S | Scalar_S | Ciphersuite)

 Confirm_P is used to construct EAP-pwd-Confirm/Request

The EAP Server computes the shared secret as:
  MK = H(ks | (Element_S + Element_P) mod r |
         (Scalar_S + Scalar_P) mod r)

The EAP Peer computes the shared secet as:
  MK = H(kp | (Element_P + Element_S) mod r |
         (Scalar_P + Scalar_S) mod r)
```

The MSK and EMSK derived from MK per Section 2.7.

2.6.5. Message Processing

2.6.5.1. EAP-pwd-ID Exchange

Although EAP provides an Identity method to determine the identity of the peer, the value in the Identity Response may have been truncated or obfuscated to provide privacy or decorated for routing purposes [RFC3748], making it inappropriate for usage by the EAP-pwd method. Therefore, the EAP-pwd-ID exchange is defined for the purpose of

exchanging identities between the peer and server.

The EAP-pwd-ID/Request contains the following quantities:

- a ciphersuite

- a representation of the server's identity in UTF-8

- an anti-clogging token

- a password pre-processing method

The ciphersuite specifies the finite cyclic group, random function and PRF selected by the server for use in the subsequent authentication exchange.

The value of the anti-clogging token MUST be unpredictable and SHOULD NOT be from a source of random entropy. The purpose of the anti-clogging token is to provide the server an assurance that the peer constructing the EAP-pwd-ID/Response is genuine and not part of a flooding attack.

A actual plaintext value of the user's password may not be accessible to the EAP server; for example, passwords may be stored in a hashed form. For this reason, EAP-pwd allows the server to communicate a password pre-processing method to the client so that the two sides can be synchronized.

The EAP-pwd-ID/Request is constructed according to Section 3.2.1 and is transmitted to the peer.

Upon receipt of an EAP-pwd-ID/Request, the peer determines whether the ciphersuite and pre-processing method are acceptable. If not, the peer MUST respond with an EAP-NAK. If acceptable, the peer responds to the EAP-pwd-ID/Request, constructed according to Section 3.2.1, that acknowledging the Ciphersuite and token and adding its identity. After sending the EAP-pwd-ID/Response, the peer has the identity of the server (from the Request), its own identity (it encoded in the Response), optionally a password preprocessing algorithm, and it can compute the password element as specified in Section 2.6.3. The password element is stored in state allocated for this exchange.

The EAP-pwd-ID/Response acknowledges the Ciphersuite from the Request, acknowledges the anti-clogging token from the Request providing a demonstration of "liveness" on the part of the peer, and contains the identity of the peer. Upon receipt of the Response, the server verifies that the Ciphersuite acknowledged by the peer is the

same as that sent in the Request and that the token added by the peer in the Response is the same as the one the server sent in the Request. If Ciphersuites or tokens differ, the server MUST respond with an EAP-Failure message. If the Ciphersuites are the same, the server now knows its own identity (it encoded in the Request) and the peer's identity (from the Response) and can compute the password element according to Section 2.6.3. The server stores the password element in state it has allocated for this exchange. The server then initiates an EAP-pwd-Commit exchange.

2.6.5.2. EAP-pwd-Commit Exchange

The server begins the EAP-pwd-Confirm exchange by choosing two random numbers between 1 and r (where r is described in Section 2.1 according to the group established in Section 2.6.5.1). It then computes Element_S and Scalar_S as defined in Section 2.6.4 and constructs an EAP-pwd-Commit/Request according to Section 3.2.2. Element_S and Scalar_S are added to the state allocated for this exchange and the EAP-pwd-Commit/Request is transmitted to the peer.

Upon receipt of the EAP-pwd-Commit/Request, the peer validates the length of the entire payload based upon the expected lengths of Element_S and Scalar_S (which are fixed according to the agreed-upon group). If the length is incorrect, the peer MUST terminate the exchange and free up any state allocated. If the length is correct, the peer chooses two random numbers between 1 and r (where r is described in Section 2.1 according to the group established in Section 2.6.5.1). It then computes Element_P and Scalar_P, constructs an EAP-pwd-Commit/Response according to Section 3.2.2 and transmits the EAP-pwd-Commit/Response to the server. The peer also computes kp from p_rand, Element_S, Scalar_S and the password element according to Section 2.6.4 and stores kp, Element_P and Scalar_P in state allocated for this exchange.

Upon receipt of the EAP-pwd-Commit/Response, the server validates the length of the entire payload based upon the expected lengths of Element_P and Scalar_P (which are fixed according to the agreed-upon group). If the length is incorrect, the server MUST respond with an EAP-Failure message and it MUST terminate the exchange and free up any state allocated. If the length is correct, the server checks whether Scalar_P equals Scalar_S and Element_P equals Element_S. If this is true it indicates a reflection attack and the server MUST respond with an EAP-Failure and terminate the exchange freeing up all allocated state. If the scalars and elements are not equal, the server computes kp from s_rand, Element_P, Scalar_P and the password element according to Section 2.6.4. The server stores ks in the state it has allocated for this exchange and initiates an EAP-pwd-Confirm Exchange.

2.6.5.3. EAP-pwd-Confirm Exchange

The server computes Confirm_S according to Section 2.6.4 and constructs an EAP-pwd-Confirm/Request according to Section 3.2.3 and sends the EAP-pwd-Confirm/Request to the peer.

Upon receipt of an EAP-pwd-Confirm/Request, the peer validates the length of the entire payload based upon the expected length of Confirm_S (whose length is fixed by the agreed-upon random function). If the length is incorrect, the peer MUST terminate the exchange and free up any state allocated. If the length is correct, the peer verifies that Confirm_S is the value it expects based on the value of kp. If the value of Confirm_S is incorrect, the peer MUST terminate the exchange and free up any state allocated. If the value of Confirm_S is correct, the peer computes Confirm_P, constructs an EAP-pwd-Confirm/Response according to Section 3.2.3 and sends it off to the server. The peer then computes MK (according to Section 2.6.4) and the MSK and EMSK (according to Section 2.7) and stores these keys in state allocated for this exchange. The peer SHOULD export the MSK and EMSK at this time in anticipation of a secure association protocol by the lower-layer to create session keys. Alternately, the peer can wait until an EAP-success messsage from the server before exporting the MSK and EMSK.

Upon receipt of an EAP-pwd-Confirm/Response, the server validates the length of the entire payload based upon the expected length of Confirm_P (whose length is fixed by the agreed-upon random function). If the length is incorrect, the server MUST respond with an EAP-Failure message and it MUST terminate the exchange and free up any state allocated. If the length is correct, the server verifies that Confirm_P is the value it expects based on the value of ks. If the value of Confirm_P is incorrect, the server MUST respond with an EAP-Failure message. If the value of Confirm_P is correct, the server computes MK (according to Section 2.6.4) and the MSK and EMSK (according to Section 2.7). It exports the MSK and EMSK and responds with an EAP-success Request. The server SHOULD free up state allocated for this exchange.

2.7. Management of EAP-pwd Keys

[RFC5247] recommends each EAP method define how to construct a Method-ID and Session-ID to identify a particular EAP session between a peer and server. This information is constructed thusly:

```
Method-ID = H(Ciphersuite | Scalar_P | Scalar_S)

Session-ID = Type-Code | Method-ID
```

where Ciphersuite, Scalar_P and Scalar_S are from the specific exchange being identified; H is the random function specified in the Ciphersuite; and, Type-Code is the code assigned by IANA for EAP-pwd: TBD1.

The authenticated key exchange of EAP-pwd generates a shared and authenticated key, MK. The size of MK is dependent on the random function, H, asserted in the Ciphersuite. EAP-pwd must export two 512-bit keys, MSK and EMSK. Regardless of the value of len(MK) implementations MUST invoke the KDF defined in Section 2.4 to construct the MSK and EMSK. The MSK and EMSK are derived thusly:

    MSK | EMSK = KDF(MK, Session-ID, 1024)

[RFC4962] mentions the importance of naming keys, particularly when key caching is being used. To faciliate such an important optimization, names are assigned thusly:

- o EMSK-name = Session-ID | 'E' | 'M'| 'S' | 'K'

- o MSK-name = Session-ID | 'M'| 'S' | 'K'

where 'E' is a single octet of value 0x45, 'M' is a single octet of value 0x4d, 'S' is a single octet of value 0x53, and 'K' is a single octet of value 0x4b.

This naming scheme allows for key management applications to quickly and accurately identify keys for a particular session or all keys of a particular type.

3. Packet Formats

3.1. EAP-pwd Header

The EAP-pwd header has the following structure:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Code      |  Identifier   |            Length             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|     Type      |L|M|  PWD-Exch |         Total-Length          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                             Data...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: EAP-pwd Header

Code

Either 1 (for Request) or 2 (for Response); see [RFC3748].

Identifier

The Identifier field is one octet and aids in matching responses with requests. The Identifier field MUST be changed on each Request packet.

Length

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

Type

TBD1 - EAP-pwd

L and M bits

The L bit (Length included) is set to indicate the presence of the two-octet Total-Length field, and MUST be set for the first fragment of a fragmented EAP-pwd message or set of messages.

The M bit (more fragments) is set on all but the last fragment.

PWD-Exch

The PWD-Exch field identifies the type of EAP-pwd payload excapsulated in the Data field. This document defines the following values for the PWD-Exch field:

* 0x01 : EAP-pwd-ID exchange

* 0x02 : EAP-pwd-Commit exchange

* 0x03 : EAP-pwd-Confirm exchange

All other values of the PWD-Exch field are reserved to IANA.

Total-Length

The Total-Length field is two octets in length, and is present only if the L bit is set. This field provides the total length of the EAP-pwd message or set of messages that is being fragmented.

3.2. EAP-pwd Payloads

EAP-pwd payloads all contain the EAP-pwd header and encoded information. Encoded information is comprised of sequences of data. Payloads in the EAP-pwd-ID exchange also include a ciphersuite statement indicating what finite cyclic group to use, what cryptographic primitive to use for H, and what PRF to use for deriving keys.

3.2.1. EAP-pwd-ID

The Group Description, Random Function, and PRF together, and in that order, comprise the Ciphersuite included in the calculation of the peer's and server's confirm messages.

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|       Group Description       | Random Func'n |      PRF      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                              Token                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|      Prep     |                  Identity...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
~                                                               ~
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: EAP-pwd-ID Payload

The Group Description field value is taken from the IANA registry for DIffie-Hellman groups created by IKE [RFC2409].

This document defines the following value for the Random Function field:

o 0x01 : Function defined in this memo in Section 2.3

All other values of the Random Function field are reserved to IANA.

The PRF field has the following value:

- o 0x01 : HMAC-SHA256 [RFC4634]

All other values of the PRF field are reserved to IANA.

The Token field contains an unpredictable value assigned by the server in an EAP-pwd-ID/Request and acknowledged by the peer in an EAP-pwd-ID/Response (see Section 2.6.5).

The Prep field represents the password pre-processing algorithm to be used by the client prior to generating the password seed (see Section 2.6.3). This document defines the following values for the Prep field:

- o 0x00 : None

- o 0x01 : Microsoft

If the value of the Prep field is equal to 0x01, the plaintext password is processed to produce the NtPasswordHashHash [RFC3079]. All other values of the Prep field are reserved to IANA.

The Identity field depends on the value of PWD-Exch.

- o EAP-pwd-ID/Request : Server_ID

- o EAP-pwd-ID/Response : Peer_ID

The length of the identity is computed from the Length field in the EAP header.

3.2.2. EAP-pwd-Commit

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                           Element                             ~
    |                                                               |
    ~                               +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                               |                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+                               ~
    |                                                               |
    ~                            Scalar             +-+-+-+-+-+-+-+-+
    |                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: EAP-pwd-Commit Payload

The Element and Scalar fields depend on the value of PWD-Exch.

- o EAP-pwd-Commit/Request : Element_S, Scalar_S

- o EAP-pwd-Commit/Response : Element_P, Scalar_P

The Element is encoded according to Section 3.3. The length of the Element is inferred by the finite cyclic group from the agreed-upon Ciphersuite. The length of the scalar can then be computed from the Length in the EAP header.

3.2.3. EAP-pwd-Confirm

```
     0                   1                   2                   3
     0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    |                                                               |
    ~                            Confirm                            ~
    |                                                               |
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: EAP-pwd-Confirm Payload

The Confirm field depends on the value of PWD-Exch.

- o EAP-pwd-Confirm/Request : Confirm_S

- o EAP-pwd-Confirm/Response : Confirm_P

The length of the Confirm field computed from the Length in the EAP header.

3.3. Representation of Field Elements

Payloads in the EAP-pwd-Commit exchange contain elements from the finite cyclic group. To ensure interoperability field elements MUST be represented according to one of the following two techniques, depending on the type of group.

3.3.1. Prime Modulus Groups

Field elements in a prime modulus group are integers less than the prime modulus. Each element MUST have a bit length equal to the bit length of the prime modulus. This length is enforced, if necessary, by prepending the integer with zeros until the required length is achieved.

3.3.2. Elliptic Curve Groups

Elliptic curve field elements are points on the elliptic curve and consist of two components, an x-coordinate and a y-coordinate. Each component MUST have a bit length equal to the field size of the group. This length is enforced, if necessary, by prepending the component with zeros until the required length is achieved.

The field element is represented in a payload by the x-coordinate followed by the y-coordinate. Therefore the field element in the payload MUST be twice the size of the field size defined in the group.

4. Fragmentation

EAP [RFC3748] is a request-response protocol. The server sends requests and the peer responds. These request and response messages are assumed to be limited to at most 1020 bytes. Messages in EAP-pwd can be larger than 1020 bytes and therefore require support for fragmentation and reassembly.

Implementations MUST establish a fragmentation threshold that indicates the maximum size of an EAP-pwd payload. When an implementation knows the maximum transmission unit (MTU) of its lower-layer, it SHOULD calculate the fragmentation threshold from that value. In lieu of knowledge of the lower-layer's MTU the fragmentation threshold MUST be set to 1020 bytes.

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

fragment offset field as is provided in IPv4.

EAP-pwd fragmentation support is provided through addition of flags within the EAP-Response and EAP-Request packets, as well as a Total-Length field of two octets. Flags include the Length included (L) and More fragments (M) bits. The L flag is set to indicate the presence of the two octet Total-Length field, and MUST be set for the first fragment of a fragmented EAP-pwd message or set of messages. The M flag is set on all but the last fragment. The Total-Length field is two octets, and provides the total length of the EAP-pwd message or set of messages that is being fragmented; this simplifies buffer allocation.

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

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

5. IANA Considerations

This memo contains new numberspaces to be managed by IANA. The policies used to allocate numbers are described in [RFC5226]. This memo requires IANA to allocate a new EAP method type for EAP-pwd.

This memo also requires IANA to create new registries for PWD-Exch messages, random functions, PRFs, password pre-processing methods and error codes and to add the message numbers, random function, PRF, pre-processing method and error codes specified in this memo to those registries, respectively.

The following is the initial PWD-Exch message registry layout:

o 0x01 : EAP-pwd-ID exchange

o 0x02 : EAP-pwd-Commit exchange

o 0x03 : EAP-pwd-Confirm exchange

The PWD-Exch field is 6 bits long and all other values are available through assignment by IANA. IANA is instructed to assign values based on "IETF Review" (see [RFC5226]).

The following is the initial Random Function registry layout:

o 0x01 : Function defined in this memo in Section 2.3

The Random Function field is 8 bits long and all other values are available through assignment by IANA. IANA is instructed to assign values based on "Specification Required" and "Expert Review" (see [RFC5226]) to ensure that new random functions have received the proper vetting.

The following is the initial PRF registry layout:

o 0x01 : HMAC-SHA256 as defined in [RFC4634]

The PRF field is 8 bits long and all other values are available through assignment by IANA. IANA is instructed to assign values based on "IETF Review" (see [RFC5226]).

The following is the initial layout for the password preprocessing method registry:

o 0x00 : None

o 0x01 : Microsoft

The Prep field is 8 bits long and all other values are available through assignment by IANA. IANA is instructed to assign values based on "First Come First Served" (see [RFC5226]).

6. Security Considerations

In Section 1.3 several security properties were presented that motivated the design of this protocol. This section will address how well they are met.

6.1. Resistance to Passive Attack

A passive attacker will see Scalar_P, Element_P, Scalar_S, and Element_S. She can guess at passwords to compute the password element but will not know s_rand or p_rand and therefore will not be able to compute MK.

The secret random value of the peer (server) is effectively hidden by adding p_mask (s_mask) to p_rand (s_rand) modulo the order of the group. If the order is "r", then there are approximately "r" distinct pairs of numbers that will sum to the value Scalar_P (Scalar_S). Attempting to guess the particular pair is just as difficult as guessing the secret random value p_rand (s_rand), the probability of a guess is 1/(r - i) after "i" guesses and for a large value of r this exhaustive search technique is computationally infeasible. An attacker would do better by determining the discrete logarithm of Element_P (Element_S) using an algorithm like Baby-Step giant-step (see [APPCRY]), which runs on the order of the square root of r group operations (e.g. a group with order 2^160 that would be 2^80 exponentiations or point multiplications). Based on the assumptions made on the finite cyclic group made in Section 2.2 that is also computationally infeasible.

6.2. Resistance to Active Attack

An active attacker can launch her attack after an honest server has sent EAP-pwd-Commit/Request to an honest peer. This would result in the peer sending EAP-pwd-Commit/Response. In this case the active attack has been reduced to that of a passive attacker since p_rand and s_rand will remain unknown. The active attacker could forge a value of Confirm_P (Confirm_S) and send it to the EAP server (EAP peer) in the hope that it will be accepted but due to the assumptions on H made in Section 2.2 that is computationally infeasible.

The active attacker can launch her attack by forging EAP-pwd-Commit/Request and sending it to the peer. This will result in the peer responding with EAP-pwd-Commit/Response. The attacker can then attempt to compute ks but since she doesn't know the password this is infeasible. It can be shown that an attack by receiving EAP-pwd-Commit/Request from an honest server and forging EAP-pwd-Commit/Response is an identical attack with equal infeasibility.

6.3. Resistance to Dictionary Attack

An active attacker can wait until an honest server sends EAP-pwd-Commit/Request and then forge EAP-pwd-Commit/Response and send it to the server. The server will respond with EAP-pwd-Confirm/Request. Now the attacker can attempt to launch a dictionary attack. She can

guess at potential passwords, compute the password element and compute kp using her p_rand, Scalar_S and Element_S from the EAP-pwd-Commit/Request and the candidate password element from her guess. She will know if her guess is correct when she is able to verify Confirm_S in EAP-pwd-Confirm/Request.

But the attacker committed to a password guess with her forged EAP-pwd-Commit/Response when she computed Element_P. That value was used by the server in his computation of ks which was used when he constructed Confirm_S in EAP-pwd-Confirm/Request. Any guess of the password that differs from the one used in the forged EAP-pwd-Commit/Response could not be verified as correct since the attacker has no way of knowing whether it is correct. She is able to make one guess and one guess only per attack. This means that any advantage she can gain-- guess a password, if it fails exclude it from the pool of possible passwords and try again-- is solely through interaction with an honest protocol peer.

The attacker can commit to the guess with the forged EAP-pwd-Commit/Response and then run through the dictionary, computing the password element and ks using her forged Scalar_P and Element_P. She will know she is correct if she can compute the same value for Confirm_S that the server produced in EAP-pwd-Confirm/Request. But this requires the attacker to know s_rand which we noted, above, was not possible.

The password element PWE/pwe is chosen using a method described in Section 2.6.3. Since this is an element in the group there exists a scalar value, q, such that:

    PWE = q * G, for an elliptic curve group

    pwe = g^q mod p, for an modular exponentiation group

Knowledge of q can be used to launch a dictionary attack. For the sake of brevity, the attack will be demonstrated assuming an elliptic curve group. The attack works thusly:

The attacker waits until an honest server sends EAP-pwd-Commit/Request. The attacker then generates a random Scalar_P and a random p_mask and computes Element_P = p_mask * G. The attacker sends the bogus Scalar_P and Element_P to the server and obtains Confirm_S in return. Note that the server is unable to detect that Element_P was calculated incorrectly.

The attacker now knows that:

    KS = (Scalar_P * q + p_mask) * s_rand * G

and

s_rand * G = Scalar_P * G - ((1/q) mod r * -Element_P)

Since Scalar_P, p_mask, G, and Element_P are all known the attacker can run through the dictionary, making a password guess, computing PWE using the technique in Section 2.6.3, determine q, and then use the equations above to compute KS and see if it can verify Confirm_S. But to determine q for a candidate PWE the attacker needs to perform a discrete logarithm which was assumed to be computationally infeasible in Section 2.2. Therefore this attack is also infeasible.

The best advantage an attacker can gain in a single active attack is to determine whether a single guess at the password was correct. Therefore her advantage is solely through interaction and not computation, which is the definition for resistance to dictionary attack.

Resistance to dictionary attack means that the attacker must launch an active attack to make a single guess at the password. If the size of the dictionary from which the password was extracted was D, and each password in the dictionary has an equal probability of being chosen, then the probability of success after a single guess is 1/D. After X guesses, and removal of failed guesses from the pool of possible passwords, the probability becomes 1/(D-X). As X grows so does the probability of success. Therefore it is possible for an attacker to determine the password through repeated brute-force, active, guessing attacks. This protocol does not presume to be secure against this and implementations SHOULD ensure the size of D is sufficiently large to prevent this attack. Implementations SHOULD also take countermeasures, for instance refusing authentication attempts for a certain amount of time, after the number of failed authentication attempts reaches a certain threshold. No such threshold or amount of time is recommended in this memo.

6.4. Forward Secrecy

The MSK and EMSK are extracted from MK which is derived from doing group operations with s_rand, p_rand, and the password scalar value. The peer and server choose random values with each run of the protocol. So even if an attacker is able to learn the password, she will not know the random values used by either the peer or server from an earlier run and will therefore be unable to determine MK, or the MSK or EMSK. This is the definition of Forward Secrecy.

6.5. Random Functions

The protocol described in this memo uses a function referred to as a "random oracle" (as defined in [RANDOR]). A significant amount of care must be taken to instantiate a random oracle out of handy cryptographic primitives. Section 6 of [RANDOR] provides guidance on how to do this using hash functions. The random function, H, defined in this memo in Section 2.3 uses one of the suggested constructs-- a hash algorithm with doubled input.

This protocol can use any properly instantiated random oracle. To ensure that any new value for H will use a properly instantiated random oracle IANA has been instructed (in Section 5) to only allocate values from the Random Function registry after being vetted by an expert.

The security of this protocol depends on the finite cyclic group used and the infeasibility of performing a discrete logarithm. A few of the defined groups that can be used with this protocol have a security estimate less than 128 bits, many do not though, and to prevent the random function from being the gating factor (or a target for attack) any new random function MUST map its input to a target of at least 128 bits and SHOULD map its input to a target of at least 256 bits.

7. Security Claims

[RFC3748] requires that documents describing new EAP methods clearly articulate the security properties of the method. In addition, for use with wireless LANs [RFC4017] mandates and recommends several of these. The claims are:

a. mechanism: password.

b. claims:

  * mutual authentication: the peer and server both authenticate each other by proving possession of a shared password. This is REQUIRED by [RFC4017].

  * foward secrecy: compromise of the password does not reveal the secret keys-- MK, MSK, or EMSK-- from earlier runs of the protocol.

  * replay protection: an attacker is unable to replay messages from a previous exchange either learn the password or a key derived by the exchange. Similarly the attacker is unable to

induce either the peer or server to believe the exchange has successfully completed when it hasn't. Reflection attacks are foiled because the server ensures that the scalar and element supplied by the peer do not equal its own.

* key derivation: keys are derived by performing a group operation in a finite cyclic group (e.g. exponentiation) using secret data contributed by both the peer and server. An MSK and EMSK are derived from that shared secret. This is REQUIRED by [RFC4017]

* dictionary attack resistance: an attacker can only make one password guess per active attack. The advantage she can gain is through interaction not through computation. This is REQUIRED by [RFC4017].

* session independence: this protocol is resistant to active and passive attack and does not enable compromise of subsequent or prior MSKs or EMSKs from either passive or active attack.

* Denial of Service Resistance: it is possible for an attacker to cause a server to allocate state and consume CPU generating Scalar_S and Element_S. Such an attack is gated, though, by the requirement that the attacker first obtain connectivity through a lower-layer protocol (e.g. 802.11 authentication followed by 802.11 association, or 802.3 "link-up") and respond to two EAP messages--the EAP-ID/Request and the EAP-pwd-ID/Request. The EAP-pwd-ID exchange further includes an anti-clogging token that provides a level of assurance to the server that the peer is, at least, performing a rudimentary amount of processing and not merely spraying packets. This prevents distributed denial of service attacks and also requires the attacker to announce, and commit to, a lower-layer identity (such as a MAC address).

* Man-in-the-Middle Attack Resistance: this exchange is resistant to active attack, which is a requirement for launching a man-in-the-middle attack. This is REQUIRED by [RFC4017].

* shared state equivalence: upon completion of EAP-pwd the peer and server both agree on MK, MSK, EMSK, Method-ID, and Session-ID. The peer has authenticated the server based on the Server-ID and the server has authenticated the peer based on the Peer-ID. This is due to the fact that Peer-ID, Server-ID, and the shared password are all combined to make

the password element which must be shared between the peer and server for the exchange to complete. This is REQUIRED by [RFC4017].

* fragmentation: this protocol defines a technique for fragmentation and reassembly in Section 4.

* resistance to "Denning-Sacco" attack: learning keys distributed from an earlier run of the protocol, such as the MSK or EMSK, will not help an adversary learn the password.

c. key strength: the strength of the resulting key depends on the finite cyclic group chosen. For example, [RFC5114] defines new groups available for use with this protocol. Using groups from [RFC5114] the strength can vary from 80 bits (for the 1024-bit MODP with 160-bit Prime Subgroup) to 256 bits (for the 521-bit Random ECP Group). Other groups can be defined and the strength of those groups depends on their definition. This is REQUIRED by [RFC4017].

d. key hierarchy: MSKs and EMSKs are derived from the MK using the KDF defined in Section 2.4 as described in Section 2.6.4.

e. vulnerabilities (note that none of these are REQUIRED by [RFC4017]):

* protected ciphersuite negotiation: the ciphersuite offer made by the server is not protected from tampering by an active attacker. Downgrade attacks are prevented, though, since this is not a "negotiation" with a list of acceptable ciphersuites. If a Ciphersuite was modified by an active attacker it would result in a failure to confirm the message sent by the other party, since the Ciphersuite is bound by each side into its confirm message, and the protocol would fail as a result.

* confidentiality: none of the messages sent in this protocol are encrypted.

* integrity protection: messages in the EAP-pwd-Commit exchange are not integrity protected.

* channel binding: this protocol does not enable the exchange of integrity-protected channel information that can be compared with values communicated via out-of-band mechanisms.

* fast reconnect: this protocol does not provide a fast reconnect capability.

* cryptographic binding: this protocol is not a tunneled EAP method and therefore has no cryptographic information to bind.

* identity protection: the EAP-pwd-ID exchange is not protected. An attacker will see the server's identity in the EAP-pwd-ID/Request and see the peer's identity in EAP-pwd-ID/Response.

8. Acknowledgements

The authors would like to thank Scott Fluhrer for discovering the "password as exponent" attack that was possible in the initial version of this memo and for his very helpful suggestions on the techniques for fixing the PWE/pwe to prevent it. The authors would also like to thank Hideyuki Suzuki for his insight in discovering an attack against a previous version of the underlying key exchange protocol. Scott Kelly suggested adding the anti-clogging token to the ID exchange to prevent distributed denial of service attacks. Dorothy Stanley provided valuable suggestions to improve the quality of this memo. The fragmentation method used was taken from [RFC5216].

9. References

9.1. Normative References

[FIPS.180-2.2002] National Institute of Standards and Technology, "Secure Hash Standard", FIPS PUB 180-2, August 2002, <http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>.

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

[RFC3748] Aboba, B., Blunk, L., Vollbrecht, J., Carlson, J., and H. Levkowetz, "Extensible Authentication Protocol (EAP)", RFC 3748, June 2004.

[RFC4282] Aboba, B., Beadles, M., Arkko, J., and P. Eronen, "The Network Access Identifier", RFC 4282, December 2005.

[SP800-108] Chen, L., "Recommendations for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, April 2008.

9.2. Informative References

[APPCRY] Menezes, A., van Oorshot, P., and S. Vanstone, "Handbook of Applied Cryptography", CRC Press Series on Discrete Mathematics and Its Applications, 1996.

[BM92] Bellovin, S. and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attack", Proceedings of the IEEE Symposium on Security and Privacy, Oakland, 1992.

[BM93] Bellovin, S. and M. Merritt, "Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise", Proceedings of the 1st ACM Conference on Computer and Communication Security, ACM Press, 1993.

[BMP00] Boyko, V., MacKenzie, P., and S. Patel, "Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman", Eurocrypt 2000, 2000.

[JAB96] Jablon, D., "Strong Password-Only Authenticated Key Exchange", ACM SIGCOMM Computer Communication Review Volume 1, Issue 5, October 1996.

[LUC97] Lucks, S., "Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys", Proceedings of the Security Protocols Workshop, LNCS 1361. Springer-Verlag, Berlin, 1997.

[RANDOR] Bellare, M. and P. Rogaway, "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols", Proceedings of the 1st ACM Conference on Computer and Communication Security, ACM Press, 1993, <http://www.cs.ucsd.edu/~mihir/papers/ro.pdf>.

[RFC2409] Harkins, D. and D. Carrel, "The Internet Key Exchange (IKE)", RFC 2409, November 1998.

[RFC3079] Zorn, G., "Deriving Keys for use with Microsoft Point-to-Point Encryption (MPPE)", RFC 3079, March 2001.

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

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4634] Eastlake, D. and T. Hansen, "US Secure Hash Algorithms (SHA and HMAC-SHA)", RFC 4634, July 2006.

[RFC4962] Housley, R. and B. Aboba, "Guidance for Authentication, Authorization, and Accounting (AAA) Key Management", BCP 132, RFC 4962, July 2007.

[RFC5114] Lepinski, M. and S. Kent, "Additional Diffie-Hellman Groups for Use with IETF Standards", RFC 5114, January 2008.

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

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

[RFC5247] Aboba, B., Simon, D., and P. Eronen, "Extensible Authentication Protocol (EAP) Key Management Framework", RFC 5247, August 2008.

Authors' Addresses

Dan Harkins
Aruba Networks
1322 Crossman Avenue
Sunnyvale, CA 94089-1113
United States of America

Email: dharkins@arubanetworks.com

Glen Zorn
NetCube
77/440 Soi Phoomjit
Rama IV Road
Phrakanong Klongtoie
Bangkok 10110
Thailand

Email: gwz@netcube.com