Internet Engineering Task Force (IETF)
Request for Comments: 6124
Category: Informational
ISSN: 2070-1721

Y. Sheffer
Independent
G. Zorn
Network Zen
H. Tschofenig
Nokia Siemens Networks
S. Fluhrer
Cisco

February 2011

An EAP Authentication Method Based on the Encrypted Key Exchange (EKE) Protocol

Abstract

The Extensible Authentication Protocol (EAP) describes a framework that allows the use of multiple authentication mechanisms. This document defines an authentication mechanism for EAP called EAP-EKE, based on the Encrypted Key Exchange (EKE) protocol. This method provides mutual authentication through the use of a short, easy to remember password. Compared with other common authentication methods, EAP-EKE is not susceptible to dictionary attacks. Neither does it require the availability of public-key certificates.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

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 to the 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-EKE is an EAP method [RFC3748] that addresses the problem of password-based authenticated key exchange, using a possibly weak password for authentication and to derive an authenticated and cryptographically strong shared secret. This problem was first described by Bellovin and Merritt in [BM92] and [BM93]. Subsequently, a number of other solution approaches have been proposed, for example [JAB96], [LUC97], [BMP00], and others.

This proposal is based on the original Encrypted Key Exchange (EKE) proposal, as described in [BM92]. Some of the variants of the original EKE have been attacked, see e.g., [PA97], and improvements have been proposed. None of these subsequent improvements have been incorporated into the current protocol. However, we have used only the subset of [BM92] (namely the variant described in Section 3.1 of that paper) that has withstood the test of time and is believed secure as of this writing.

2. Terminology

This document uses Encr(Ke, ...) to denote encrypted information, and Prot(Ke, Ki, ...) to denote encrypted and integrity protected information.

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

3. Protocol

EAP is a two-party protocol spoken between an EAP peer and an EAP server (also known as "authenticator"). 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 and the EAP peer for the purpose of authentication and key derivation.

3.1. Message Flows

A successful run of EAP-EKE consists of three message exchanges: an Identity exchange, a Commit exchange, and a Confirm exchange. This is shown in Figure 1.

The peer and server use the EAP-EKE Identity exchange to learn each other’s identities and to agree upon a ciphersuite to use in the subsequent exchanges. 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 (note that the second message of the Commit exchange already plays an essential part in this liveness proof).

```
         +--------+                                     +--------+
         |        |                  EAP-EKE-ID/Request |        |
         |  EAP   |<------------------------------------|  EAP   |
         |  peer  |                                     | server |
         |  (P)   | EAP-EKE-ID/Response                 |  (S)   |
         |        |------------------------------------>|        |
         |        |                                     |        |
         |        |              EAP-EKE-Commit/Request |        |
         |        |<------------------------------------|        |
         |        |                                     |        |
         |        | EAP-EKE-Commit/Response             |        |
         |        |------------------------------------>|        |
         |        |                                     |        |
         |        |             EAP-EKE-Confirm/Request |        |
         |        |<------------------------------------|        |
         |        |                                     |        |
         |        | EAP-EKE-Confirm/Response            |        |
         |        |------------------------------------>|        |
         |        |                                     |        |
         |        |          EAP-Success                |        |
         |        |<------------------------------------|        |
         +--------+                                     +--------+
```

Figure 1: A Successful EAP-EKE Exchange

Schematically, the original exchange as described in [BM92] (and with the roles reversed) is:

```
  Server                                Peer
  ------                                ----

  Encr(Password, y_s) ->

                     <- Encr(Password, y_p), Encr(SharedSecret, Nonce_P)

  Encr(SharedSecret, Nonce_S | Nonce_P) ->

                                          <- Encr(SharedSecret, Nonce_S)
```

Where:

- o Password is a typically short string, shared between the server and the peer. In other words, the same password is used to authenticate the server to the peer, and vice versa.

- o y_s and y_p are the server’s and the peer’s, respectively, ephemeral public key, i.e., y_s = g ^ x_s (mod p) and y_p = g ^ x_p (mod p).

- o Nonce_S, Nonce_P are random strings generated by the server and the peer as cryptographic challenges.

- o SharedSecret is the secret created by the Diffie-Hellman algorithm, namely SharedSecret = g^(x_s * x_p) (mod p). This value is calculated by the server as: SharedSecret = y_p ^ x_s (mod p), and by the peer as: SharedSecret = y_s ^ x_p (mod p).

The current protocol extends the basic cryptographic protocol, and the regular successful exchange becomes:

```
  Message                  Server                        Peer
 ---------                --------                      ------
ID/Request         ID_S, CryptoProposals ->

ID/Response                                  <- ID_P, CryptoSelection

Commit/Request     Encr(Password, y_s) ->

Commit/Response        <- Encr(Password, y_p), Prot(Ke, Ki, Nonce_P)

Confirm/Request    Prot(Ke, Ki, Nonce_S | Nonce_P), Auth_S ->

Confirm/Response                     <- Prot(Ke, Ki, Nonce_S), Auth_P
```

Where, in addition to the above terminology:

- o Encr means encryption only, and Prot is encryption with integrity protection.

- o Ke is an encryption key, and Ki is an integrity-protection key.

Section 5 explains the various cryptographic values and how they are derived.

As shown in the exchange above, the following information elements have been added to the original protocol: identity values for both protocol parties (ID_S, ID_P), negotiation of cryptographic protocols, and signature fields to protect the integrity of the negotiated parameters (Auth_S, Auth_P). In addition, the shared secret is not used directly. In this initial exposition, a few details were omitted for clarity. Section 5 should be considered as authoritative regarding message and field details.

4. Message Formats

EAP-EKE defines a small number of message types, each message consisting of a header followed by a payload. This section defines the header, several payload formats, as well as the format of specific fields within the payloads.

As usual, all multi-octet strings MUST be laid out in network byte order.

4.1. EAP-EKE Header

The EAP-EKE header consists of the standard EAP header (see Section 4 of [RFC3748]), followed by an EAP-EKE exchange type. The 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      |   EKE-Exch    |             Data             ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 2: EAP-EKE Header

The Code, Identifier, Length, and Type fields are all part of the EAP header as defined in [RFC3748]. The Type field in the EAP header is 53 for EAP-EKE Version 1.

The EKE-Exch (EKE Exchange) field identifies the type of EAP-EKE payload encapsulated in the Data field. This document defines the following values for the EKE-Exch field:

- 0x00: Reserved

- 0x01: EAP-EKE-ID exchange

- 0x02: EAP-EKE-Commit exchange

- 0x03: EAP-EKE-Confirm exchange

- 0x04: EAP-EKE-Failure message

Further values of this EKE-Exch field are available via IANA registration (Section 7.7).

4.2. EAP-EKE Payloads

EAP-EKE messages all contain the EAP-EKE header and information encoded in a single payload, which differs for the different exchanges.

4.2.1. The EAP-EKE-ID Payload

```
     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
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
    | NumProposals  |   Reserved    |           Proposal           ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ...    Proposal                  |    IDType     |  Identity    ...
    +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 3: EAP-EKE-ID Payload

The EAP-EKE-ID payload contains the following fields:

NumProposals:

The NumProposals field contains the number of Proposal fields subsequently contained in the payload. In the EAP-EKE-ID/Request message, the NumProposals field MUST NOT be set to zero (0), and in the EAP-EKE-ID/Response message, the NumProposals field MUST be set to one (1). The offered proposals in the Request are listed contiguously in priority order, most preferable first. The selected proposal in the Response MUST be fully identical with one of the offered proposals.

Reserved:

This field MUST be sent as zero, and MUST be ignored by the recipient.

Proposal:

Each proposal consists of four one-octet fields, in this order:

Group Description:

This field’s value is taken from the IANA registry for Diffie-Hellman groups defined in Section 7.1.

Encryption:

This field’s value is taken from the IANA registry for encryption algorithms defined in Section 7.2.

PRF:

This field’s value is taken from the IANA registry for pseudo-random functions defined in Section 7.3.

MAC:

This field’s value is taken from the IANA registry for keyed message digest algorithms defined in Section 7.4.

IDType:

Denotes the Identity Type. This is taken from the IANA registry defined in Section 7.5. The server and the peer MAY use different identity types. All implementations MUST be able to receive two identity types: ID_NAI and ID_FQDN.

Identity:

The meaning of the Identity field depends on the values of the Code and IDType fields.

- EAP-EKE-ID/Request: server ID
- EAP-EKE-ID/Response: peer ID

The length of the Identity field is computed from the Length field in the EAP header. Specifically, its length is

eap_header_length - 9 - 4 * number_of_proposals.

This field, like all other fields in this specification, MUST be octet-aligned.

4.2.2. The EAP-EKE-Commit Payload

This payload allows both parties to send their encrypted ephemeral public key, with the peer also including a Challenge.

In addition, a small amount of data can be included by the server and/or the peer, and used for channel binding. This data is sent here unprotected, but is verified later, when it is signed by the Auth_S/Auth_P payloads of the EAP-EKE-Confirm exchange.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          DHComponent_S/DHComponent_P                          ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           PNonce_P                                            ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           CBValue (zero or more occurrences)                  ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 4: EAP-EKE-Commit Payload

DHComponent_S/DHComponent_P:

This field contains the password-encrypted Diffie-Hellman public key, which is generated as described in Section 5.1. Its size is determined by the group and the encryption algorithm.

PNonce_P:

This field only appears in the response, and contains the encrypted and integrity-protected challenge value sent by the peer. The field's size is determined by the encryption and MAC algorithms being used, since this protocol mandates a fixed nonce size for a given choice of algorithms. See Section 5.2.

CBValue:

This structure MAY be included both in the request and in the response, and MAY be repeated multiple times in a single payload. See Section 4.5. Each structure contains its own length. The field is neither encrypted nor integrity protected, instead it is protected by the AUTH payloads in the Confirm exchange.

4.2.3. The EAP-EKE-Confirm Payload

Using this payload, both parties complete the authentication by generating a shared temporary key, authenticating the entire protocol, and generating key material for the EAP consumer protocol.

```
 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          PNonce_PS/PNonce_S                                   ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Auth_S/Auth_P                                        ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
~                                                               ~
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 5: EAP-EKE-Confirm Payload

PNonce_PS/PNonce_S:

This field ("protected nonce") contains the encrypted and integrity-protected response to the other party’s challenge; see Sections 5.3 and 5.4. Similarly to the PNonce_P field, this field’s size is determined by the encryption and MAC algorithms.

Auth_S/Auth_P:

This field signs the preceding messages, including the Identity and the negotiated fields. This prevents various possible attacks, such as algorithm downgrade attacks. See Section 5.3 and Section 5.4. The size is determined by the pseudo-random function negotiated.

4.2.4. The EAP-EKE-Failure Payload

The EAP-EKE-Failure payload format is defined as follows:

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                          Failure-Code                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 6: EAP-EKE-Failure Payload

The payload’s size is always exactly 4 octets.

The following Failure-Code values are defined:

Value	Name	Meaning
0x00000000	Reserved	
0x00000001	No Error	This code is used for failure acknowledgement, see below.
0x00000002	Protocol Error	A failure to parse or understand a protocol message or one of its payloads.
0x00000003	Password Not Found	A password could not be located for the identity presented by the other protocol party, making authentication impossible.
0x00000004	Authentication Failure	Failure in the cryptographic computation, most likely caused by an incorrect password or an inappropriate identity type.
0x00000005	Authorization Failure	While the password being used is correct, the user is not authorized to connect.
0x00000006	No Proposal Chosen	The peer is unwilling to select any of the cryptographic proposals offered by the server.

Additional values of this field are available via IANA registration, see Section 7.8.

When the peer encounters an error situation, it MUST respond with EAP-EKE-Failure. The server MUST reply with an EAP-Failure message to end the exchange.

When the server encounters an error situation, it MUST respond with EAP-EKE-Failure. The peer MUST send back an EAP-EKE-Failure message containing a "No Error" failure code. Then the server MUST send an EAP-Failure message to end the exchange.

Implementation of the "Password Not Found" code is not mandatory. For security reasons, implementations MAY choose to return "Authentication Failure" also in cases where the password cannot be located.

4.3. Protected Fields

Several fields are encrypted and integrity-protected. They are denoted Prot(...). Their general structure is as follows:

```
    0                   1                   2                   3
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |         Initialization Vector (IV) (optional)                 |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                          Encrypted Data                       ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~                                                               ~
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   ~               |            Random Padding                     |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
   |                  Integrity Check Value (ICV)                  |
   +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 7: Protected Field Structure

The protected field is a concatenation of three octet strings:

- An optional IV, required when the encryption algorithm/mode necessitates it, e.g., for CBC encryption. The content and size of this field are determined by the selected encryption algorithm. In the case of CBC encryption, this field is a random octet string having the same size as the algorithm’s block size.
- The original data, followed if necessary by random padding. This padding has the minimal length (possibly zero) required to complete the length of the encrypted data to the encryption algorithm’s block size. The original data and the padding are encrypted together.

- ICV, a Message Authentication Code (MAC) cryptographic checksum of the encrypted data, including the padding. The checksum is computed over the encrypted, rather than the plaintext, data. Its length is determined by the MAC algorithm negotiated.

We note that because of the requirement for an explicit ICV, this specification does not support authenticated encryption algorithms. Such algorithms may be added by a future extension.

4.4. Encrypted Fields

Two fields are encrypted but are not integrity protected. They are denoted Encr(...). Their format is identical to a protected field (Section 4.3), except that the Integrity Check Value is omitted.

4.5. Channel Binding Values

This protocol allows higher-level protocols to transmit limited opaque information between the peer and the server. This information is integrity protected but not encrypted, and may be used to ensure that protocol participants are identical at different protocol layers. See Section 7.15 of [RFC3748] for more information on the rationale behind this facility.

EAP-EKE neither validates nor makes any use of the transmitted information. The information MUST NOT be used by the consumer protocol until it is verified in the EAP-EKE-Confirm exchange (specifically, until it is integrity protected by the Auth_S, Auth_P payloads). Consequently, it MUST NOT be relied upon in case an error occurs at the EAP-EKE level.

An unknown Channel Binding Value SHOULD be ignored by the recipient.

Some implementations may require certain values to be present, and will abort the protocol if they are not. Such policy is out of scope of the current protocol.

Each Channel Binding Value is encoded using a TLV 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
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          CBType               |           Length              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|          Value                                                 ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Figure 8: Channel Binding Value

CBType:

This is the Channel Binding Value’s type. This document defines the value 0x0000 as reserved. Other values are available for IANA allocation, see Section 7.6.

Length:

This field is the total length in octets of the structure, including the CBType and Length fields.

This facility should be used with care, since EAP-EKE does not provide for message fragmentation. EAP-EKE is not a tunneled method and should not be used as a generic transport; specifically, implementors should refrain from using the Channel Binding facility to transmit posture information, in the sense of [RFC5209].

5. Protocol Sequence

This section describes the sequence of messages for the Commit and Confirm exchanges, and lists the cryptographic operations performed by the server and the peer.

5.1. EAP-EKE-Commit/Request

The server computes:

$$y_s = g ^ {x_s} \pmod p,$$

where x_s is a randomly chosen number in the range 2 .. p-1. The randomly chosen number is the ephemeral private key, and the calculated value is the corresponding ephemeral public key. The server and the peer MUST both use a fresh, random value for x_s and the corresponding x_p on each run of the protocol.

The server computes and transmits the encrypted field (Section 4.4)

```
temp = prf(0+, password)

key = prf+(temp, ID_S | ID_P)

DHComponent_S = Encr(key, y_s).
```

See Section 6.1 for the prf+ notation. The first argument to "prf" is a string of zero octets whose length is the output size of the base hash algorithm, e.g., 20 octets for HMAC-SHA1; the result is of the same length. The first output octets of prf+ are used as the encryption key for the negotiated encryption algorithm, according to that algorithm’s key length.

Since the PRF function is required to be an application of the HMAC operator to a hash function, the above construction implements HKDF as defined in [RFC5869].

When using block ciphers, it may be necessary to pad y_s on the right, to fit the encryption algorithm’s block size. In such cases, random padding MUST be used, and this randomness is critical to the security of the protocol. Randomness recommendations can be found in [RFC4086]; also see [NIST.800-90.2007] for additional recommendations on cryptographic-level randomness. When decrypting this field, the real length of y_s is determined according to the negotiated Diffie-Hellman group.

If the password needs to be stored on the server, it is RECOMMENDED to store a randomized password value as a password-equivalent, rather than the cleartext password. We note that implementations may choose the output of either of the two steps of the password derivation. Using the output of the second step, where the password is salted by the identity values, is more secure; however, it may create an operational issue if identities are likely to change. See also Section 8.5.

This protocol supports internationalized, non-ASCII passwords. The input password string SHOULD be processed according to the rules of the [RFC4013] profile of [RFC3454]. A password SHOULD be considered a "stored string" per [RFC3454], and unassigned code points are therefore prohibited. The output is the binary representation of the processed UTF-8 [RFC3629] character string. Prohibited output and unassigned code points encountered in SASLprep preprocessing SHOULD cause a preprocessing failure and the output SHOULD NOT be used.

5.2. EAP-EKE-Commit/Response

The peer computes:

```
y_p = g ^ x_p (mod p)
```

Then computes:

```
temp = prf(0+, password)

key = prf+(temp, ID_S | ID_P)

DHComponent_P = Encr(key, y_p)
```

formatted as an encrypted field (Section 4.4).

Both sides calculate

```
SharedSecret = prf(0+, g ^ (x_s * x_p) (mod p))
```

The first argument to "prf" is a string of zero octets whose length is the output size of the base hash algorithm, e.g., 20 octets for HMAC-SHA1; the result is of the same length. This extra application of the pseudo-random function is the "extraction step" of [RFC5869]. Note that the peer needs to compute the SharedSecret value before sending out its response.

The encryption and integrity protection keys are computed:

```
Ke | Ki = prf+(SharedSecret, "EAP-EKE Keys" | ID_S | ID_P)
```

And the peer generates the Protected Nonce:

```
PNonce_P = Prot(Ke, Ki, Nonce_P),
```

where Nonce_P is a randomly generated binary string. The length of Nonce_P MUST be the maximum of 16 octets, and half the key size of the negotiated prf (rounded up to the next octet if necessary). The peer constructs this value as a protected field (Section 4.3), encrypted using Ke and integrity protected using Ki with the negotiated encryption and MAC algorithm.

The peer now sends a message that contains the two generated fields.

The server MUST verify the correct integrity protection of the received nonce, and MUST abort the protocol if it is incorrect, with an "Authentication Failure" code.

5.3. EAP-EKE-Confirm/Request

The server constructs:

PNonce_PS = Prot(Ke, Ki, Nonce_P | Nonce_S),

as a protected field, where Nonce_S is a randomly generated string, of the same size as Nonce_P.

It computes:

Ka = prf+(SharedSecret, "EAP-EKE Ka" | ID_S | ID_P | Nonce_P | Nonce_S)

whose length is the preferred key length of the negotiated prf (see Section 5.2). It then constructs:

Auth_S = prf(Ka, "EAP-EKE server" | EAP-EKE-ID/Request | EAP-EKE-ID/Response | EAP-EKE-Commit/Request | EAP-EKE-Commit/Response).

The messages are included in full, starting with the EAP header, and including any possible future extensions.

This construction of the Auth_S (and Auth_P) value implies that any future extensions MUST NOT be added to the EAP-EKE-Confirm/Request or EAP-EKE-Confirm/Response messages themselves, unless these extensions are integrity-protected in some other manner.

The server now sends a message that contains the two fields.

The peer MUST verify the correct integrity protection of the received nonces and the correctness of the Auth_S value, and MUST abort the protocol if either is incorrect, with an "Authentication Failure" code.

5.4. EAP-EKE-Confirm/Response

The peer computes Ka, and generates:

PNonce_S = Prot(Ke, Ki, Nonce_S)

as a protected field. It then computes:

Auth_P = prf(Ka, "EAP-EKE peer" | EAP-EKE-ID/Request | EAP-EKE-ID/Response | EAP-EKE-Commit/Request | EAP-EKE-Commit/Response)

The peer sends a message that contains the two fields.

The server MUST verify the correct integrity protection of the received nonce and the correctness of the Auth_P value, and MUST abort the protocol if either is incorrect, with an "Authentication Failure" code.

5.5. MSK and EMSK

Following the last message of the protocol, both sides compute and export the shared keys, each 64 bytes in length:

```
MSK | EMSK = prf+(SharedSecret, "EAP-EKE Exported Keys" | ID_S |
ID_P | Nonce_P | Nonce_S)
```

When the RADIUS attributes specified in [RFC2548] are used to transport keying material, then the first 32 bytes of the MSK correspond to MS-MPPE-RECV-KEY and the second 32 bytes to MS-MPPE-SEND-KEY. In this case, only 64 bytes of keying material (the MSK) are used.

At this point, both protocol participants MUST discard all intermediate cryptographic values, including x_p, x_s, y_p, y_s, Ke, Ki, Ka, and SharedSecret. Similarly, both parties MUST immediately discard these values whenever the protocol terminates with a failure code or as a result of timeout.

6. Cryptographic Details

6.1. Generating Keying Material

Keying material is derived as the output of the negotiated pseudo-random function (prf) algorithm. Since the amount of keying material needed may be greater than the size of the output of the prf algorithm, we will use the prf iteratively. We denote by "prf+" the function that outputs a pseudo-random stream based on the inputs to a prf as follows (where "|" indicates concatenation):

```
prf+ (K, S) = T1 | T2 | T3 | T4 | ...
```

where:

```
T1 = prf(K, S | 0x01)

T2 = prf(K, T1 | S | 0x02)

T3 = prf(K, T2 | S | 0x03)

T4 = prf(K, T3 | S | 0x04)
```

continuing as needed to compute all required keys. The keys are taken from the output string without regard to boundaries (e.g., if the required keys are a 256-bit Advanced Encryption Standard (AES) key and a 160-bit HMAC key, and the prf function generates 160 bits, the AES key will come from T1 and the beginning of T2, while the HMAC key will come from the rest of T2 and the beginning of T3).

The constant concatenated to the end of each string feeding the prf is a single octet. In this document, prf+ is not defined beyond 255 times the size of the prf output.

6.2. Diffie-Hellman Groups

Many of the commonly used Diffie-Hellman groups are inappropriate for use in EKE. Most of these groups use a generator that is not a primitive element of the group. As a result, an attacker running a dictionary attack would be able to learn at least 1 bit of information for each decrypted password guess.

Any MODP Diffie-Hellman group defined for use in this protocol MUST have the following properties to ensure that it does not leak a usable amount of information about the password:

1. The generator is a primitive element of the group.

2. The most significant 64 bits of the prime number are 1.

3. The group's order p is a "safe prime", i.e., (p-1)/2 is also prime.

The last requirement is related to the strength of the Diffie-Hellman algorithm, rather than the password encryption. It also makes it easy to verify that the generator is primitive.

Suitable groups are defined in Section 7.1.

6.3. Mandatory Algorithms

To facilitate interoperability, the following algorithms are mandatory to implement:

- ENCR_AES128_CBC (encryption algorithm)

- PRF_HMAC_SHA1 (pseudo-random function)

- MAC_HMAC_SHA1 (keyed message digest)

- DHGROUP_EKE_14 (DH-group)

7. IANA Considerations

IANA has allocated the EAP method type 53 from the range 1-191, for "EAP-EKE Version 1".

Per this document, IANA created the registries described in the following sub-sections. Values (other than private-use ones) can be added to these registries per Specification Required [RFC5226], with two exceptions: the Exchange and Failure Code registries can only be extended per RFC Required [RFC5226].

7.1. Diffie-Hellman Group Registry

This section defines an IANA registry for Diffie-Hellman groups.

This table defines the initial contents of this registry. The Value column is used when negotiating the group. Additional groups may be defined through IANA allocation. Any future specification that defines a non-MODP group MUST specify its use within EAP-EKE and MUST demonstrate the group’s security in this context.

Name	Value	Description
Reserved	0	
DHGROUP_EKE_2	1	The prime number of the 1024-bit Group 2 [RFC5996], with the generator 5 (decimal)
DHGROUP_EKE_5	2	The prime number of the 1536-bit Group 5 [RFC3526], g=31
DHGROUP_EKE_14	3	The prime number of the 2048-bit Group 14 [RFC3526], g=11
DHGROUP_EKE_15	4	The prime number of the 3072-bit Group 15 [RFC3526], g=5
DHGROUP_EKE_16	5	The prime number of the 4096-bit Group 16 [RFC3526], g=5
Available for allocation via IANA	6-127	
Reserved for Private Use	128-255	

7.2. Encryption Algorithm Registry

This section defines an IANA registry for encryption algorithms:

Name	Value	Definition
Reserved	0	
ENCR_AES128_CBC	1	AES with a 128-bit key, CBC mode
	2-127	Available for allocation via IANA
	128-255	Reserved for Private Use

7.3. Pseudo-Random Function Registry

This section defines an IANA registry for pseudo-random function algorithms:

Name	Value	Definition
Reserved	0	
PRF_HMAC_SHA1	1	HMAC SHA-1, as defined in [RFC2104]
PRF_HMAC_SHA2_256	2	HMAC SHA-2-256 [SHA]
	3-127	Available for allocation via IANA
	128-255	Reserved for Private Use

A pseudo-random function takes two parameters K and S (the key and input string respectively), and, to be usable in this protocol, must be defined for all lengths of K between 0 and 65,535 bits (inclusive).

Any future pseudo-random function MUST be based on the HMAC construct, since the security of HKDF is only known for such functions.

7.4. Keyed Message Digest (MAC) Registry

This section defines an IANA registry for keyed message digest algorithms:

Name	Value	Key Length (Octets)	Definition
Reserved	0		
MAC_HMAC_SHA1	1	20	HMAC SHA-1, as defined in [RFC2104]
MAC_HMAC_SHA2_256	2	32	HMAC SHA-2-256
Reserved	3-127		Available for allocation via IANA
Reserved	128-255		Reserved for Private Use

7.5. Identity Type Registry

This section defines an IANA registry for identity types:

Name	Value	Definition
Reserved	0	
ID_OPAQUE	1	An opaque octet string
ID_NAI	2	A Network Access Identifier, as defined in [RFC4282]
ID_IPv4	3	An IPv4 address, in binary format
ID_IPv6	4	An IPv6 address, in binary format
ID_FQDN	5	A fully qualified domain name, see note below
ID_DN	6	An LDAP Distinguished Name formatted as a string, as defined in [RFC4514]
	7-127	Available for allocation via IANA
	128-255	Reserved for Private Use

An example of an ID_FQDN is "example.com". The string MUST NOT contain any terminators (e.g., NULL, CR, etc.). All characters in the ID_FQDN are ASCII; for an internationalized domain name, the syntax is as defined in [RFC5891], for example "xn--tmonesimerkki-bfbb.example.net".

7.6. EAP-EKE Channel Binding Type Registry

This section defines an IANA registry for the Channel Binding Type registry, a 16-bit long code. The value 0x0000 has been defined as Reserved. All other values up to and including 0xfeff are available for allocation via IANA. The remaining values up to and including 0xffff are available for Private Use.

7.7. Exchange Registry

This section defines an IANA registry for the EAP-EKE Exchange registry, an 8-bit long code. Initial values are defined in Section 4.1. All values up to and including 0x7f are available for allocation via IANA. The remaining values up to and including 0xff are available for private use.

7.8. Failure-Code Registry

This section defines an IANA registry for the Failure-Code registry, a 32-bit long code. Initial values are defined in Section 4.2.4. All values up to and including 0xfeffffff are available for allocation via IANA. The remaining values up to and including 0xffffffff are available for private use.

8. Security Considerations

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 paragraphs.

Resistance to Passive Attack: A passive 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.

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.

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.

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

[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:

1. Mechanism: password.

2. Claims:

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

   * Forward secrecy: compromise of the password does not reveal the secret keys (MSK and EMSK) from earlier runs of the protocol.

   * Replay protection: an attacker is unable to replay messages from a previous exchange either to 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.

   * Key derivation: a shared secret is 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, and the protocol is designed so that the attacker does not gain any confirmation of her guess by observing the decrypted y_s or y_p value (see below). 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 attacks and does not enable compromise of subsequent or prior MSKs or EMSKs from either passive or active attacks.

- Denial-of-service resistance: it is possible for an attacker to cause a server to allocate state and consume CPU. 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-EKE-ID/Request.

- 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-EKE, the peer and server both agree on the MSK and EMSK values. 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 generated shared secret are all combined to make the authentication element that must be shared between the peer and server for the exchange to complete. This is REQUIRED by [RFC4017].

- Fragmentation: this protocol does not define a technique for fragmentation and reassembly.

- 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.

3. Key strength: the strength of the resulting key depends on the finite cyclic group chosen. Sufficient key strength is REQUIRED by [RFC4017]. Clearly, "sufficient" strength varies over time, depending on computation power assumed to be available to potential attackers.

4. Key hierarchy: MSKs and EMSKs are derived from the secret values generated during the protocol run, using a negotiated pseudo-random function.

5. Vulnerabilities (note that none of these are REQUIRED by [RFC4017]):

   * Protected ciphersuite negotiation: the ciphersuite proposal made by the server is not protected from tampering by an active attacker. However, if a proposal was modified by an active attacker, it would result in a failure to confirm the message sent by the other party, since the proposal is bound by each side into its Confirm message, and the protocol would fail as a result. Note that this assumes that none of the proposed ciphersuites enables an attacker to perform real-time cryptanalysis.

   * Confidentiality: none of the messages sent in this protocol are encrypted, though many of the protocol fields are.

   * Integrity protection: protocol messages are not directly integrity protected; however, the ID and Commit exchanges are integrity protected through the Auth payloads exchanged in the Confirm exchange.

   * Channel binding: this protocol enables 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-EKE-ID exchange is not protected. An attacker will see the server’s identity in the EAP-EKE-ID/Request and see the peer’s identity in EAP-EKE-ID/Response. See also Section 8.4.

8.1. Cryptographic Analysis

When analyzing the Commit exchange, it should be noted that the base security assumptions are different from "normal" cryptology. Normally, we assume that the key has strong security properties, and that the data may have few or none. Here, we assume that the key has weak security properties (the attacker may have a list of possible keys), and hence we need to ensure that the data has strong properties (indistinguishable from random). This difference may mean that conventional wisdom in cryptology might not apply in this case. This also imposes severe constraints on the protocol, e.g., the mandatory use of random padding and the need to define specific finite groups.

8.2. Diffie-Hellman Group Considerations

It is fundamental to the dictionary attack resistance that the Diffie-Hellman public values y_s and y_p are indistinguishable from a random string. If this condition is not met, then a passive attacker can do trial-decryption of the encrypted DHComponent_P or DHComponent_S values based on a password guess, and if they decrypt to a value that is not a valid public value, they know that the password guess was incorrect.

For MODP groups, Section 6.2 gives conditions on the group to make sure that this criterion is met. For other groups (for example, Elliptic Curve groups), some other means of ensuring this must be employed. The standard way of expressing Elliptic Curve public values does not meet this criterion, as a valid Elliptic Curve X coordinate can be distinguished from a random string with probability of approximately 0.5.

A future document might introduce a group representation, and/or a slight modification of the password encryption scheme, so that Elliptic Curve groups can be accommodated. [BR02] presents several alternative solutions for this problem.

8.3. Resistance to Active Attacks

An attacker, impersonating either the peer or the server, can always try to enumerate all possible passwords, for example by using a dictionary. To counter this likely attack vector, both peer and server MUST implement rate-limiting mechanisms. We note that locking out the other party after a small number of tries would create a trivial denial-of-service opportunity.

8.4. Identity Protection, Anonymity, and Pseudonymity

By default, the EAP-EKE-ID exchange is unprotected, and an eavesdropper can observe both parties’ identities. A future extension of this protocol may support anonymity, e.g., by allowing the server to send a temporary identity to the peer at the end of the exchange, so that the peer can use that identity in subsequent exchanges.

EAP-EKE differs in this respect from tunneled methods, which typically provide unconditional identity protection to the peer by encrypting the identity exchange, but reveal information in the server certificate. It is possible to use EAP-EKE as the inner method in a tunneled EAP method in order to achieve this level of identity protection.

8.5. Password Processing and Long-Term Storage

This document recommends that a password-equivalent (a hash of the password) be stored instead of the cleartext password. While this solution provides a measure of security, there are also tradeoffs related to algorithm agility:

- Each stored password must identify the hash function that was used to compute the stored value.
- Complex deployments and migration scenarios might necessitate multiple stored passwords, one per each algorithm.
- Changing the algorithm can require, in some cases, that the users manually change their passwords.

The reader is referred to Section 10 of [RFC3629] for security considerations related to the parsing and processing of UTF-8 strings.

9. Acknowledgements

Much of this document was unashamedly picked from [RFC5931] and [EAP-SRP], and we would like to acknowledge the authors of these documents: Dan Harkins, Glen Zorn, James Carlson, Bernard Aboba, and Henry Haverinen. We would like to thank David Jacobson, Steve Bellovin, Russ Housley, Brian Weis, Dan Harkins, and Alexey Melnikov for their useful comments. Lidar Herooty and Idan Ofrat implemented this protocol and helped us improve it by asking the right questions, and we would like to thank them both.

10. References

10.1. Normative References

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

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

[RFC2548] Zorn, G., "Microsoft Vendor-specific RADIUS Attributes", RFC 2548, March 1999.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3526] Kivinen, T. and M. Kojo, "More Modular Exponential (MODP) Diffie-Hellman groups for Internet Key Exchange (IKE)", RFC 3526, May 2003.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003.

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

[RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.

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

[RFC4514] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): String Representation of Distinguished Names", RFC 4514, June 2006.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5996] Kaufman, C., Hoffman, P., Nir, Y., and P. Eronen, "Internet Key Exchange Protocol Version 2 (IKEv2)", RFC 5996, September 2010.

[SHA] National Institute of Standards and Technology, U.S. Department of Commerce, "Secure Hash Standard", NIST FIPS 180-3, October 2008.

10.2. Informative References

[BM92]	Bellovin, S. and M. Merritt, "Encrypted Key Exchange: Password-Based Protocols Secure Against Dictionary Attacks", Proc. IEEE Symp. on Research in Security and Privacy , May 1992.
[BM93]	Bellovin, S. and M. Merritt, "Augmented Encrypted Key Exchange: A Password-Based Protocol Secure against Dictionary Attacks and Password File Compromise", Proc. 1st ACM Conference on Computer and Communication Security , 1993.
[BMP00]	Boyko, V., MacKenzie, P., and S. Patel, "Provably Secure Password Authenticated Key Exchange Using Diffie-Hellman", Advances in Cryptology, EUROCRYPT 2000 , 2000.
[BR02]	Black, J. and P. Rogaway, "Ciphers with Arbitrary Finite Domains", Proc. of the RSA Cryptographer’s Track (RSA CT ’02), LNCS 2271 , 2002.
[EAP-SRP]	Carlson, J., Aboba, B., and H. Haverinen, "EAP SRP-SHA1 Authentication Protocol", Work in Progress, July 2001.
[JAB96]	Jablon, D., "Strong Password-Only Authenticated Key Exchange", ACM Computer Communications Review Volume 1, Issue 5, October 1996.
[LUC97]	Lucks, S., "Open Key Exchange: How to Defeat Dictionary Attacks Without Encrypting Public Keys", Proc. of the Security Protocols Workshop LNCS 1361, 1997.
[NIST.800-90.2007]	National Institute of Standards and Technology, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators (Revised)", NIST SP 800-90, March 2007.
[PA97]	Patel, S., "Number Theoretic Attacks On Secure Password Schemes", Proceedings of the 1997 IEEE Symposium on Security and Privacy , 1997.
[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.
[RFC5209]	Sangster, P., Khosravi, H., Mani, M., Narayan, K., and J. Tardo, "Network Endpoint Assessment (NEA): Overview and Requirements", RFC 5209, June 2008.
[RFC5226]	Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
[RFC5869]	Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, May 2010.
[RFC5931]	Harkins, D. and G. Zorn, "Extensible Authentication Protocol (EAP) Authentication Using Only a Password", RFC 5931, August 2010.

Authors' Addresses

Yaron Sheffer
Independent

EMail: yaronf.ietf@gmail.com

Glen Zorn
Network Zen
227/358 Thanon Sanphawut
Bang Na, Bangkok 10260
Thailand

Phone: +66 (0) 87-040-4617
EMail: gwz@net-zen.net

Hannes Tschofenig
Nokia Siemens Networks
Linnoitustie 6
Espoo 02600
Finland

Phone: +358 (50) 4871445
EMail: Hannes.Tschofenig@gmx.net
URI: http://www.tschofenig.priv.at

Scott Fluhrer
Cisco Systems.
1414 Massachusetts Ave.
Boxborough, MA 01719
USA

EMail: sfluhrer@cisco.com