Internet Engineering Task Force (IETF)
Request for Comments: 6617
Category: Experimental
ISSN: 2070-1721

D. Harkins
Aruba Networks

June 2012

# Secure Pre-Shared Key (PSK) Authentication for the Internet Key Exchange Protocol (IKE)

## Abstract

This memo describes a secure pre-shared key (PSK) authentication method for the Internet Key Exchange Protocol (IKE). It is resistant to dictionary attack and retains security even when used with weak pre-shared keys.

## Status of This Memo

This document is not an Internet Standards Track specification; it is published for examination, experimental implementation, and evaluation.

This document defines an Experimental Protocol for the Internet community. This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc6617.

Table of Contents

## 1. Introduction

[RFC5996] allows for authentication of the IKE peers using a pre-shared key. This exchange, though, is susceptible to dictionary attack and is therefore insecure when used with weak pre-shared keys, such as human-memorizable passwords. To address the security issue, [RFC5996] recommends that the pre-shared key used for authentication "contain as much unpredictability as the strongest key being negotiated". That means any non-hexadecimal key would require over 100 characters to provide enough strength to generate a 128-bit key suitable for AES. This is an unrealistic requirement because humans have a hard time entering a string over 20 characters without error. Consequently, pre-shared key authentication in [RFC5996] is used insecurely today.

A pre-shared key authentication method built on top of a zero-knowledge proof will provide resistance to dictionary attack and still allow for security when used with weak pre-shared keys, such as user-chosen passwords. Such an authentication method is described in this memo.

Resistance to dictionary attack is achieved when an adversary gets one, and only one, guess at the secret per active attack (see, for example, [BM92], [BMP00], and [BPR00]). Another way of putting this is that any advantage the adversary can realize is through interaction and not through computation. This is demonstrably different than the technique from [RFC5996] of using a large, random number as the pre-shared key. That can only make a dictionary attack less likely to succeed; it does not prevent a dictionary attack. Furthermore, as [RFC5996] notes, it is completely insecure when used with weak keys like user-generated passwords.

### 1.1. 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].

## 2. Usage Scenarios

[RFC5996] describes usage scenarios for IKEv2. These are:

1. "Security Gateway to Security Gateway Tunnel": The endpoints of the IKE (and IPsec) communication are network nodes that protect traffic on behalf of connected networks. Protected traffic is between devices on the respective protected networks.

2. "Endpoint-to-Endpoint Transport": The endpoints of the IKE (and IPsec) communication are hosts according to [RFC4301]. Protected traffic is between the two endpoints.

3. "Endpoint to Security Gateway Tunnel": One endpoint connects to a protected network through a network node. The endpoints of the IKE (and IPsec) communication are the endpoint and network node, but the protected traffic is between the endpoint and another device on the protected network behind the node.

The authentication and key exchange process described in this memo is suitable for all the usage scenarios described in [RFC5996]. In the "Security Gateway to Security Gateway Tunnel" scenario and the "Endpoint-to-Endpoint Transport" scenario, it provides a secure method of authentication without requiring a certificate. For the "Endpoint to Security Gateway Tunnel" scenario, it provides for secure username+password authentication that is popular in remote-access VPN situations.

## 3. Terms and Notation

The following terms and notations are used in this memo:

PSK
: A shared, secret, and potentially low-entropy word, phrase, code, or key used as a credential to mutually authenticate the peers.

a = prf(b, c)
: The string "b" and "c" are given to a pseudo-random function (prf) to produce a fixed-length output "a".

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

[a]b
: indicates a string consisting of the single bit "a" repeated "b" times.

len(a)
: indicates the length in bits of the string "a".

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

element
: one member of a finite cyclic group.

scalar
a quantity that can multiply an element.

The convention for this memo to represent an element in a finite cyclic group is to use an upper-case letter or acronym, while a scalar is indicated with a lowercase letter or acronym.

## 4. Discrete Logarithm Cryptography

This protocol uses Discrete Logarithm Cryptography to achieve authentication. Each party to the exchange derives ephemeral public and private keys with respect to a particular set of domain parameters (referred to here as a "group"). Groups can be either based on finite field cryptography (modular exponentiation (MODP) groups) or elliptic curve cryptography (ECP groups).

This protocol uses the same group as the IKE exchange in which it is being used for authentication, with the exception of characteristic-two elliptic curve groups (EC2N). Use of such groups is undefined for this authentication method, and an IKE exchange that negotiates one of these groups MUST NOT use this method of authentication.

For each group, the following operations are defined:

- "scalar operation" -- takes a scalar and an element in the group to produce another element -- Z = scalar-op(x, Y).
- "element operation" -- takes two elements in the group to produce a third -- Z = element-op(X, Y).
- "inverse operation" -- takes an element and returns another element such that the element operation on the two produces the identity element of the group -- Y = inverse(X).

### 4.1. Elliptic Curve Cryptography (ECP) Groups

The key exchange defined in this memo uses fundamental algorithms of ECP groups as described in [RFC6090].

Domain parameters for ECP elliptic curves used for Secure PSK Authentication include:

- A prime, p, determining a prime field GF(p). The cryptographic group will be a subgroup of the full elliptic curve group that consists of points on an elliptic curve -- elements from GF(p) that satisfy the curve’s equation -- together with the "point at infinity" (denoted here as "0") that serves as the identity element.

- Elements a and b from GF(p) that define the curve's equation. The point (x,y) is on the elliptic curve if and only if y^2 = x^3 + a*x + b.

- A prime, r, which is the order of, or number of elements in, a subgroup generated by an element G.

The scalar operation is multiplication of a point on the curve by itself a number of times. The point Y is multiplied x-times to produce another point Z:

```
Z = scalar-op(x, Y) = x*Y
```

The element operation is addition of two points on the curve. Points X and Y are summed to produce another point Z:

```
Z = element-op(X, Y) = X + Y
```

The inverse function is defined such that the sum of an element and its inverse is "0", the point-at-infinity of an elliptic curve group:

```
Q + inverse(Q) = "0"
```

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.

scalar-op(x, Y) can be viewed as x iterations of element-op() by defining:

```
Y = scalar-op(1, Y)
```

```
Y = scalar-op(x, Y) = element-op(Y, scalar-op(x-1, Y)), for x > 1
```

A definition of how to add two points on an elliptic curve (i.e., element-op(X, Y)) can be found in [RFC6090].

Note: There is another ECP domain parameter, a cofactor, h, that is defined by the requirement that the size of the full elliptic curve group (including "0") be the product of h and r. ECP groups used for Secure PSK Authentication MUST have a cofactor of one (1). At the time of publication of this memo, all ECP groups in [IKEV2-IANA] had a cofactor of one (1).

### 4.2. Finite Field Cryptography (MODP) Groups

Domain parameters for MODP groups used for Secure PSK Authentication include:

- A prime, p, determining a prime field GF(p), the integers modulo p.
- A prime, r, which is the multiplicative order, and thus also the size, of the cryptographic subgroup of GF(p)* that is generated by an element G.

The scalar operation is exponentiation of a generator modulo a prime. An element Y is taken to the x-th power modulo the prime, thereby returning another element, Z:

```
Z = scalar-op(x, Y) = Y^x mod p
```

The element operation is modular multiplication. Two elements, X and Y, are multiplied modulo the prime, thereby returning another element, Z:

```
Z = element-op(X, Y) = (X * Y) mod p
```

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

```
(Q * inverse(Q)) mod p = 1
```

Unlike ECP groups, MODP groups do not require a mapping function to convert an element into an integer. However, for the purposes of notation in protocol definition, the function F, when used below, shall just return the value that was passed to it, i.e., F(i) = i.

Some MODP groups in [IKEV2-IANA] are based on safe primes, and the order is not included in the group's domain parameter set. In this case only, the order, r, MUST be computed as the prime minus one divided by two -- (p-1)/2. If an order is included in the group's domain parameter set, that value MUST be used in this exchange when an order is called for. If a MODP group does not include an order in its domain parameter set and is not based on a safe prime, it MUST NOT be used with this exchange.

## 5. Random Numbers

As with IKE itself, the security of the Secure PSK Authentication method relies upon each participant in the protocol 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 a dictionary attack.

Producing quality random numbers without specialized hardware entails using a cryptographic mixing function (like a strong hash function) to mix entropy from multiple, uncorrelated sources of information and events. A very good discussion of this can be found in [RFC4086].

## 6. Using Passwords and Raw Keys For Authentication

The PSK used as an authentication credential with this protocol can be either a character-based password or passphrase, or it could be a binary or hexadecimal string. Regardless, however, this protocol requires both the Initiator and Responder to have identical binary representations of the shared credential.

If the PSK is a character-based password or passphrase, there are two types of pre-processing that SHALL be employed to convert the password or passphrase into a hexadecimal string suitable for use with Secure PSK Authentication. If a PSK is already a hexadecimal or binary string, it SHALL be used directly as the shared credential without any pre-processing.

The first step of pre-processing is to remove ambiguities that may arise due to internationalization. Each character-based password or passphrase MUST be pre-processed to remove that ambiguity by processing the character-based password or passphrase according to the rules of the SASLprep [RFC4013] profile of [RFC3454]. The password or passphrase SHALL be considered a "stored string" per [RFC3454], and unassigned code points are therefore prohibited. The output SHALL be the binary representation of the processed UTF-8 character string. Prohibited output and unassigned codepoints encountered in SASLprep pre-processing SHALL cause a failure of pre-processing, and the output SHALL NOT be used with Secure PSK Authentication.

The next pre-processing step for character-based passwords or passphrases is to effectively obfuscate the string. This is done in an attempt to reduce exposure of stored passwords in the event of server compromise, or compromise of a server’s database of stored passwords. The step involves taking the output of the SASLprep [RFC4013] profile of [RFC3454] and passing it, as the key, with the

ASCII string "IKE Secure PSK Authentication", as the data, to HMAC-SHA256(). The output of this obfuscation step SHALL become the shared credential used with Secure PSK Authentication.

Note: Passwords tend to be shared for multiple purposes, and compromise of a server or database of stored plaintext passwords can be used, in that event, to mount multiple attacks. The obfuscation step is merely to hide the password in the event of server compromise or compromise of the database of stored passwords. Advances in distributed computing power have diminished the effectiveness of performing multiple prf iterations as a technique to prevent dictionary attacks, so no such behavior is proscribed here. Mutually consenting implementations can agree to use a different password obfuscation method; the one described here is for interoperability purposes only.

If a device stores passwords for use at a later time, it SHOULD pre-process the password prior to storage. If a user enters a password into a device at authentication time, it MUST be pre-processed upon entry and prior to use with Secure PSK Authentication.

## 7. Assumptions

The security of the protocol relies on certain assumptions. They are:

1. The pseudo-random function, prf, defined in [RFC5996], acts as an "extractor" (see [RFC5869]) by distilling the entropy from a secret input into a short, fixed string. The output of prf is indistinguishable from a random source.

2. 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.

3. The pre-shared key is drawn from a finite pool of potential keys. Each possible key in the pool has equal probability of being the shared key. All potential adversaries have access to this pool of keys.

## 8. Secure PSK Authentication Message Exchange

The key exchange described in this memo is based on the "Dragonfly" key exchange, which has also been defined for use in 802.11 wireless networks (see [SAE]) and as an Extensible Authentication Protocol (EAP) method (see [RFC5931]). "Dragonfly" is patent-free and

royalty-free. It SHALL use the same pseudo-random function (prf) and the same Diffie-Hellman group that are negotiated for use in the IKE exchange that "Dragonfly" is authenticating.

A pseudo-random function that uses a block cipher is NOT RECOMMENDED for use with Secure PSK Authentication due to its poor job operating as an "extractor" (see Section 7). Pseudo-random functions based on hash functions using the HMAC construct from [RFC2104] SHOULD be used.

To perform Secure PSK Authentication, each side must generate a shared and secret element in the chosen group based on the pre-shared key. This element, called the Secret Key Element, or SKE, is then used in the "Dragonfly" authentication and key exchange protocol. "Dragonfly" consists of each side exchanging a Commit payload and then proving knowledge of the resulting shared secret.

The Commit payload contributes ephemeral information to the exchange and binds the sender to a single value of the pre-shared key from the pool of potential pre-shared keys. An authentication payload (AUTH) proves that the pre-shared key is known and completes the zero-knowledge proof.

### 8.1. Negotiation of Secure PSK Authentication

The Initiator indicates its desire to use Secure PSK Authentication by adding a Notify payload of type SECURE_PASSWORD_METHODS (see [RFC6467]) to the first message of the IKE_SA_INIT exchange and by including 3 in the notification data field of the Notify payload, indicating Secure PSK Authentication.

The Responder indicates its acceptance to perform Secure PSK Authentication by adding a Notify payload of type SECURE_PASSWORD_METHODS to its response in the IKE_SA_INIT exchange and by adding the sole value of 3 to the notification data field of the Notify payload.

If the Responder does not include a Notify payload of type SECURE_PASSWORD_METHODS in its IKE_SA_INIT response, the Initiator MUST terminate the exchange, and it MUST NOT fall back to the PSK authentication method of [RFC5996]. If the Initiator only indicated its support for Secure PSK Authentication (i.e., if the Notify data field only contained 3) and the Responder replies with a Notify payload of type SECURE_PASSWORD_METHODS and a different value in the Notify data field, the Initiator MUST terminate the exchange.

### 8.2. Fixing the Secret Element, SKE

The method of fixing SKE depends on the type of group, either MODP or ECP. The function "prf+" from [RFC5996] is used as a key derivation function.

Fixing SKE involves an iterative hunting-and-pecking technique using the prime from the negotiated group's domain parameter set and an ECP- or MODP-specific operation depending on the negotiated group. This technique requires the pre-shared key to be a binary string; therefore, any pre-processing transformation (see Section 6) MUST be performed on the pre-shared key prior to fixing SKE.

To thwart side-channel attacks that attempt to determine the number of iterations of the hunting-and-pecking loop that are used to find SKE for a given password, a security parameter, k, is used to ensure that at least k iterations are always performed.

Prior to beginning the hunting-and-pecking loop, an 8-bit counter is set to the value one (1). Then the loop begins. First, the pseudo-random function is used to generate a secret seed using the counter, the pre-shared key, and two nonces (without the fixed headers) exchanged by the Initiator and the Responder (see Section 8.6):

```
ske-seed = prf(Ni | Nr, psk | counter)
```

Then, the ske-seed is expanded using prf+ to create an ske-value:

```
ske-value = prf+(ske-seed, "IKE SKE Hunting And Pecking")
```

where len(ske-value) is the same as len(p), the length of the prime from the domain parameter set of the negotiated group.

If the ske-seed is greater than or equal to the prime, p, the counter is incremented, a new ske-seed is generated, and the hunting-and-pecking continues. If ske-seed is less than the prime, p, it is passed to the group-specific operation to select the SKE or fail. If the group-specific operation fails, the counter is incremented, a new ske-seed is generated, and the hunting-and-pecking continues. This process continues until the group-specific operation returns the password element. After the password element has been chosen, a random number is used in place of the password in the ske-seed calculation, and the hunting-and-pecking continues until the counter is greater than the security parameter, k.

#### 8.2.1. ECP Operation to Select SKE

The group-specific operation for ECP groups uses ske-value, ske-seed, and the equation of the curve to produce SKE. First, ske-value is used directly as the x-coordinate, x, with the equation of the elliptic curve, with parameters a and b from the domain parameter set of the curve, to solve for a y-coordinate, y.

Note: A method of checking whether a solution to the equation of the elliptic curve is to see whether the Legendre symbol of (x^3 + ax + b) equals one (1). If it does, then a solution exists; if it does not, then there is no solution.

If there is no solution to the equation of the elliptic curve, then the operation fails, the counter is incremented, a new ske-value and ske-seed are selected, and the hunting-and-pecking continues. If there is a solution then, y is calculated as the square root of (x^3 + ax + b) using the equation of the elliptic curve. In this case, an ambiguity exists as there are technically two solutions to the equation, and ske-seed is used to unambiguously select one of them. If the low-order bit of ske-seed is equal to the low-order bit of y, then a candidate SKE is defined as the point (x,y); if the low-order bit of ske-seed differs from the low-order bit of y then a candidate SKE is defined as the point (x, p-y) where p is the prime from the negotiated group’s domain parameter set. The candidate SKE becomes the SKE, and the ECP-specific operation completes successfully.

Algorithmically, the process looks like this:

```
found = 0
counter = 1
v = psk
do {
  ske-seed = prf(Ni | Nr, v | counter)
  ske-value = prf+(ske-seed, "IKE SKE Hunting And Pecking")
  if (ske-value < p)
  then
    x = ske-value
    if ( (y = sqrt(x^3 + ax + b)) != FAIL)
    then
      if (found == 0)
      then
        if (LSB(y) == LSB(ske-seed))
        then
          SKE = (x,y)
        else
          SKE = (x, p-y)
        fi
        found = 1
        v = random()
      fi
    fi
  fi
  counter = counter + 1
} while ((found == 0) || (counter <= k))
```

where FAIL indicates that there is no solution to sqrt(x^3 + ax + b).

Figure 1: Fixing SKE for ECP Groups

Note: For ECP groups, the probability that more than "n" iterations of the hunting-and-pecking loop are required to find SKE is roughly (1-(r/2p))^n, which rapidly approaches zero (0) as "n" increases.

### 8.2.2. MODP Operation to Select SKE

The group-specific operation for MODP groups takes ske-value, the prime, p, and order, r, from the group's domain parameter set to directly produce a candidate SKE by exponentiating the ske-value to the value ((p-1)/r) modulo the prime. If the candidate SKE is greater than one (1), the candidate SKE becomes the SKE, and the MODP-specific operation completes successfully. Otherwise, the MODP-specific operation fails (and the hunting-and-pecking continues).

Algorithmically, the process looks like this:

```
found = 0
counter = 1
v = psk
do {
  ske-seed = prf(Ni | Nr, v | counter)
  ske-value = prf+(ske-seed, "IKE SKE Hunting And Pecking")
  if (ske-value < p)
  then
    ELE = ske-value ^ ((p-1)/r) mod p
    if (ELE > 1)
    then
      if (found == 0)
        SKE = ELE
        found = 1
        v = random()
      fi
    fi
  fi
  counter = counter + 1
} while ((found == 0) || (counter <= k))
```

Figure 2: Fixing SKE for MODP Groups

Note: For MODP groups, the probability that more than "n" iterations of the hunting-and-pecking loop are required to find SKE is roughly ((m-p)/p)^n, where m is the largest unsigned number that can be expressed in len(p) bits, which rapidly approaches zero (0) as "n" increases.

### 8.3. Encoding and Decoding of Group Elements and Scalars

The payloads used in the Secure PSK Authentication method contain elements from the negotiated group and scalar values. To ensure interoperability, scalars and field elements MUST be represented in payloads in accordance with the requirements in this section.

#### 8.3.1. Encoding and Decoding of Scalars

Scalars MUST be represented (in binary form) as unsigned integers that are strictly less than r, the order of the generator of the agreed-upon cryptographic group. The binary representation of each scalar MUST have a bit length equal to the bit length of the binary representation of r. This requirement is enforced, if necessary, by prepending the binary representation of the integer with zeros until the required length is achieved.

Scalars in the form of unsigned integers are converted into octet strings and back again using the technique described in [RFC6090].

### 8.3.2. Encoding and Decoding of ECP Elements

Elements in ECP groups are points on the negotiated elliptic curve. Each such element MUST be represented by the concatenation of two components, an x-coordinate and a y-coordinate.

Each of the two components, the x-coordinate and the y-coordinate, MUST be represented (in binary form) as an unsigned integer that is strictly less than the prime, p, from the group's domain parameter set. The binary representation of each component MUST have a bit length equal to the bit length of the binary representation of p. This length requirement is enforced, if necessary, by prepending the binary representation of the integer with zeros until the required length is achieved.

The unsigned integers that represent the coordinates of the point are converted into octet strings and back again using the technique described in [RFC6090].

Since the field element is represented in a payload by the x-coordinate followed by the y-coordinate, it follows, then, that the length of the element in the payload MUST be twice the bit length of p.

### 8.3.3. Encoding and Decoding of MODP Elements

Elements in MODP groups MUST be represented (in binary form) as unsigned integers that are strictly less than the prime, p, from the group's domain parameter set. The binary representation of each group element MUST have a bit length equal to the bit length of the binary representation of p. This length requirement is enforced, if necessary, by prepending the binary representation of the integer with zeros until the required length is achieved.

The unsigned integer that represents a MODP element is converted into an octet string and back using the technique described in [RFC6090].

### 8.4. Message Generation and Processing

#### 8.4.1. Generation of a Commit

Before a Commit payload can be generated, the SKE must be fixed using the process described in Section 8.2.

A Commit payload has two components, a scalar and an element. To generate a Commit payload, two random numbers, a "private" value and a "mask" value, are generated (see Section 5). Their sum modulo the order of the group, r, becomes the scalar component:

```
scalar = (private + mask) mod r
```

If the scalar is not greater than one (1), the private and mask values MUST be thrown away, and new values randomly generated. If the scalar is greater than one (1), the inverse of the scalar operation with the mask and SKE becomes the element component.

```
Element = inverse(scalar-op(mask, SKE))
```

The Commit payload consists of the scalar followed by the element, and the scalar and element are encoded in the Commit payload according to Section 8.3.

#### 8.4.2. Processing of a Commit

Upon receipt of a peer’s Commit payload, the scalar and element MUST be validated. The processing of an element depends on the type, either an ECP element or a MODP element.

##### 8.4.2.1. Validation of an ECP Element

Validating a received ECP element involves: 1) checking whether the two coordinates, x and y, are both greater than zero (0) and less than the prime defining the underlying field; and 2) checking whether the x- and y-coordinates satisfy the equation of the curve (that is, that they produce a valid point on the curve that is not "0"). If either of these conditions are not met, the received element is invalid; otherwise, the received element is valid.

##### 8.4.2.2. Validation of a MODP Element

A received MODP element is valid if: 1) it is between one (1) and the prime, p, exclusive; and 2) if modular exponentiation of the element by the group order, r, equals one (1). If either of these conditions are not true, the received element is invalid; otherwise, the received element is valid.

#### 8.4.2.3. Commit Processing Steps

Commit payload validation is accomplished by the following steps:

1. The length of the Commit payload is checked against its anticipated length (the anticipated length of the scalar plus the anticipated length of the element, for the negotiated group). If it is incorrect, the Commit payload is invalidated; otherwise, processing continues.

2. The peer's scalar is extracted from the Commit payload according to Section 8.3.1 and checked to ensure it is between one (1) and r, the order of the negotiated group, exclusive. If it is not, the Commit payload is invalidated; otherwise, processing continues.

3. The peer's element is extracted from the Commit payload according to Section 8.3.2 and checked in a manner that depends on the type of group negotiated. If the group is ECP, the element is validated according to Section 8.4.2.1. If the group is MODP, the element is validated according to Section 8.4.2.2. If the element is not valid, then the Commit payload is invalidated; otherwise, the Commit payload is validated.

4. The Initiator of the IKE exchange has an added requirement to verify that the received element and scalar from the Commit payload differ from the element and scalar sent to the Responder. If they are identical, it signifies a reflection attack, and the Commit payload is invalidated.

If the Commit payload is invalidated, the payload MUST be discarded and the IKE exchange aborted.

### 8.4.3. Authentication of the Exchange

After a Commit payload has been generated and a peer's Commit payload has been processed, a shared secret used to authenticate the peer is derived. Using SKE, the "private" value generated as part of Commit payload generation, and the peer's scalar and element from the peer's Commit payload, named here peer-scalar and Peer-Element, respectively, a preliminary shared secret, skey, is generated as:

```
skey = F(scalar-op(private,
                   element-op(Peer-Element,
                              scalar-op(peer-scalar, SKE))))
```

For the purposes of subsequent computation, the bit length of skey SHALL be equal to the bit length of the prime, p, used in either a MODP or ECP group. This bit length SHALL be enforced, if necessary, by prepending zeros to the value until the required length is achieved.

A shared secret, ss, is then computed from skey and the nonces exchanged by the Initiator (Ni) and Responder (Nr) (without the fixed headers) using prf():

ss = prf(Ni | Nr, skey | "Secure PSK Authentication in IKE")

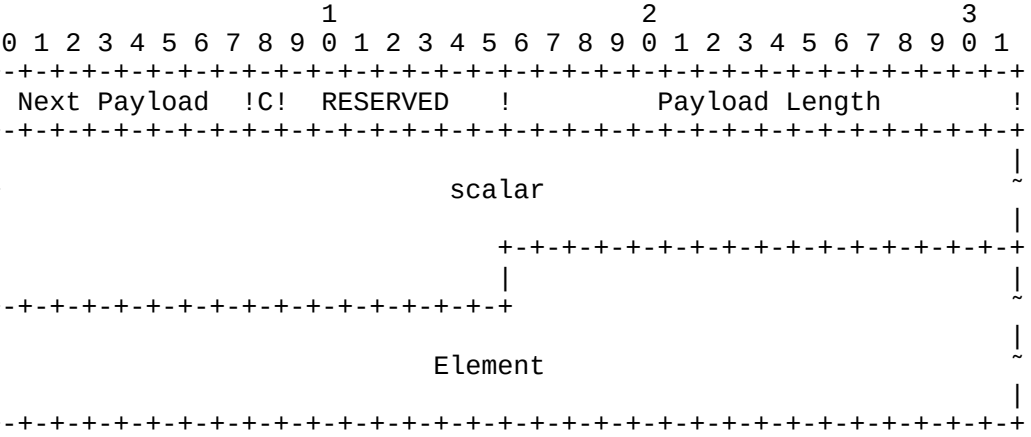
The shared secret, ss, is used in an AUTH authentication payload to prove possession of the shared secret and therefore knowledge of the pre-shared key.

## 8.5. Payload Format

### 8.5.1. Commit Payload

[RFC6467] defines a Generic Secure Password Method (GSPM) payload that is used to convey information that is specific to a particular secure password method. This memo uses the GSPM payload as a Commit payload to contain the scalar and element used in the Secure PSK Authentication exchange:

The Commit payload is defined as follows:




The scalar and element SHALL be encoded in the Commit payload according to Section 8.3.

## 8.6. IKEv2 Messaging

Secure PSK Authentication modifies the IKE_AUTH exchange by adding one additional round trip to exchange Commit payloads to perform the Secure PSK Authentication exchange and by changing the calculation of the AUTH payload data to bind the IKEv2 exchange to the outcome of the Secure PSK Authentication exchange (see Figure 3).

```
 Initiator                                Responder
-----------                              -----------

IKE_SA_INIT:

 HDR, SAi1, KEi, Ni,
      N(SPM-SPSK)  -->

                                 <--    HDR, SAr1, KEr, Nr,
                                             N(SPM-SPSK)

IKE_AUTH:

 HDR, SK {IDi, COMi, [IDr,]
          SAi2, TSi, TSr}        -->
                                 <--    HDR, SK {IDr, COMr}
 HDR, SK {AUTHi}                 -->
                                 <--    HDR, SK {AUTHr, SAr2, TSi, TSr}
```

where N(SPM-SPSK) indicates the Secure Password Methods Notify payloads used to negotiate the use of Secure PSK Authentication (see Section 8.1), COMi and AUTHi are the Commit payload and AUTH payload, respectively, sent by the Initiator, and COMr and AUTHr are the Commit payload and AUTH payload, respectively, sent by the Responder.

Figure 3: Secure PSK in IKEv2

When doing Secure PSK Authentication, the AUTH payloads SHALL be computed as

```
AUTHi = prf(ss, <InitiatorSignedOctets> | COMi | COMr)

AUTHr = prf(ss, <ResponderSignedOctets> | COMr | COMi)
```

where "ss" is the shared secret derived in Section 8.4.3, COMi and COMr are the entire Commit payloads (including the fixed headers) sent by the Initiator and Responder, respectively, and <InitiatorSignedOctets> and <ResponderSignedOctets> are defined in

[RFC5996]. The Authentication Method indicated in both AUTH payloads SHALL be "Generic Secure Password Authentication Method", value 12, from [IKEV2-IANA].

## 9. IANA Considerations

IANA has assigned the value 3 for "Secure PSK Authentication" from the Secure Password Authentication Method registry in [IKEV2-IANA].

## 10. Security Considerations

Both the Initiator and Responder obtain a shared secret, "ss" (see Section 8.4.3), based on a secret group element and their own private values contributed to the exchange. If they do not share the same pre-shared key, they will be unable to derive the same secret group element, and if they do not share the same secret group element, they will be unable to derive the same shared secret.

Resistance to dictionary attack means that the adversary must launch an active attack to make a single guess at the pre-shared key. If the size of the pool from which the key was extracted was d and each key in the pool 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 keys, the probability becomes 1/(d-x). As x grows, so does the probability of success. Therefore, it is possible for an adversary to determine the pre-shared key through repeated brute-force, active, guessing attacks. This authentication method does not presume to be secure against this, and implementations SHOULD ensure the value 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.

An active attacker can impersonate the Responder of the exchange and send a forged Commit payload after receiving the Initiator’s Commit payload. The attacker then waits until it receives the authentication payload from the Responder. Now the attacker can attempt to run through all possible values of the pre-shared key, computing SKE (see Section 8.2), computing "ss" (see Section 8.4.3), and attempting to recreate the Confirm payload from the Responder.

But, by sending a forged Commit payload the attacker commits to a single guess of the pre-shared key. That value was used by the Responder in his computation of "ss", which was used in the authentication payload. Any guess of the pre-shared key that differs from the one used in the forged Commit payload would result in each

side using a different secret element in the computation of "ss" and therefore the authentication payload could not be verified as correct, even if a subsequent guess, while running through all possible values, was correct. The attacker gets one guess, and one guess only, per active attack.

An attacker, acting as either the Initiator or Responder, can take the element from the Commit payload received from the other party, reconstruct the random "mask" value used in its construction, and then recover the other party's "private" value from the scalar in the Commit payload. But this requires the attacker to solve the discrete logarithm problem, which we assumed was intractable (Section 7).

Instead of attempting to guess at pre-shared keys, an attacker can attempt to determine SKE and then launch an attack, but SKE is determined by the output of the pseudo-random function, prf, which is assumed to be indistinguishable from a random source (Section 7). Therefore, each element of the finite cyclic group will have an equal probability of being the SKE. The probability of guessing SKE will be 1/r, where r is the order of the group. This is the same probability of guessing the solution to the discrete logarithm, which is assumed to be intractable (Section 7). The attacker would have a better chance of success at guessing the input to prf, i.e., the pre-shared key, since the order of the group will be many orders of magnitude greater than the size of the pool of pre-shared keys.

The implications of resistance to dictionary attack are significant. An implementation can provision a pre-shared key in a practical and realistic manner -- i.e., it MAY be a character string, and it MAY be relatively short -- and still maintain security. The nature of the pre-shared key determines the size of the pool, D, and countermeasures can prevent an adversary from determining the secret in the only possible way: repeated, active, guessing attacks. For example, a simple four-character string using lowercase English characters, and assuming random selection of those characters, will result in D of over four hundred thousand. An adversary would need to mount over one hundred thousand active, guessing attacks (which will easily be detected) before gaining any significant advantage in determining the pre-shared key.

If an attacker knows the number of hunting-and-pecking loops that were required to determine SKE, it is possible to eliminate passwords from the pool of potential passwords and increase the probability of successfully guessing the real password. MODP groups will require more than "n" loops with a probability based on the value of the prime -- if m is the largest unsigned number that can be expressed in len(p) bits, then the probability is ((m-p)/p)^n -- which will typically be very small for the groups defined in [IKEV2-IANA]. ECP

groups will require more than one "n" loop with a probability of roughly (1-(r/2p))^n. Therefore, a security parameter, k, is defined that will ensure that at least k loops will always be executed regardless of whether SKE is found in less than k loops. There is still a probability that a password would require more than k loops, and a side-channel attacker could use that information to his advantage, so selection of the value of k should be based on a trade-off between the additional workload to always perform k iterations and the potential of providing information to a side-channel attacker. It is important to note that the possibility of a successful side-channel attack is greater against ECP groups than MODP groups, and it might be appropriate to have separate values of k for the two.

For a more detailed discussion of the security of the key exchange underlying this authentication method, see [SAE] and [RFC5931].

## 11. Acknowledgements

The author would like to thank Scott Fluhrer and Hideyuki Suzuki for their insight in discovering flaws in earlier versions of the key exchange that underlies this authentication method and for their helpful suggestions in improving it. Thanks to Lily Chen for useful advice on the hunting-and-pecking technique to "hash into" an element in a group and to Jin-Meng Ho for a discussion on countering a small sub-group attack. Rich Davis suggested several checks on received messages that greatly increase the security of the underlying key exchange. Hugo Krawczyk suggested using the prf as an extractor.

## 12. References

### 12.1. Normative References

[IKEV2-IANA] IANA, "IKEv2 Parameters", <http://www.iana.org/assignments/ikev2-parameters>.

[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.

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

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

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

[RFC6090] McGrew, D., Igoe, K., and M. Salter, "Fundamental Elliptic Curve Cryptography Algorithms", RFC 6090, February 2011.

[RFC6467] Kivinen, T., "Secure Password Framework for Internet Key Exchange Version 2 (IKEv2)", RFC 6467, December 2011.

## 12.2. Informative References

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

[BMP00] Boyko, V., MacKenzie, P., and S. Patel, "Provably Secure Password-Authenticated Key Exchange Using Diffie-Hellman", Proceedings of Eurocrypt 2000, LNCS 1807 Springer-Verlag, 2000.

[BPR00] Bellare, M., Pointcheval, D., and P. Rogaway, "Authenticated Key Exchange Secure Against Dictionary Attacks", Advances in Cryptology -- Eurocrypt ’00, Lecture Notes in Computer Science Springer-Verlag, 2000.

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

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[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.

[SAE] Harkins, D., "Simultaneous Authentication of Equals: A Secure, Password-Based Key Exchange for Mesh Networks", Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications Volume 00, 2008.

## Author's Address

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

EMail: dharkins@arubanetworks.com