EMU D.
Harkins
Internet-Draft
HPE
Intended status: Informational July 24,
2019
Expires: January 25, 2020

Improved Extensible Authentication Protocol Using Only a Password
draft-harkins-eap-pwd-prime-00.txt

Abstract

Passwords are a popular form of credential for user authentication. EAP-pwd (RFC 5931) is a popular method of performing secure password authenticaiton. EAP-pwd requires a secret element in a finite cyclic group, unfortunately the technique it uses to derive this secret is open to timing and cache attacks. This improved version, EAP-pwd', uses a different technique to derive the secret element which is resistant to timing and cache attacks.

Requirements Language

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 [1].

Status of This Memo

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

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

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

This Internet-Draft will expire on January 25, 2020.

Table of Contents

1. Introduction

EAP-pwd is a popular EAP method due to the fact that it authenticates without requiring certificates. Large federated networks sometimes have latency issues with numerous fragmented packets going between the EAP client and EAP server, a problem exacerbated by using EAP methods that require certificate-based authentication. EAP-pwd obviates this.

The technique used by EAP-pwd to obtain its secret element is susceptible to timing attacks and cache attacks that can partition

the dictionary enough to successfully determine the password. Recent work in the Crypto Forum Research Group on constant time techniques to hash a string into a point on an elliptic curve in constant time [2] provides an opportunity to address this.

2. EAP-pwd'

EAP-pwd' is an EAP method that follows the EAP-pwd specification ([3]) in all respects except for the following:

- It uses the Type code TBD-1, not 52 which is used by EAP-pwd.

- It derives PWE/pwe as defined in Section 2.1 and Section 2.2 for ECC and FFC groups, respectively, using a different technique than the "hunting and pecking" loop defined in [3].

- it defines three new random functions using HKDF instantiated with SHA-256, SHA-384, and SHA-512.

EAP-pwd' MUST be used with one of the random functions defined in this document.

The technique used by EAP-pwd' for deriving PWE/pwe can be implemented in constant time and is resistant to the side channel and timing attacks that the hunting-and-pecking loop of [3] is susceptible to. Computing the password element in EAP-pwd' is a two-step process. First, a secret element based on the password is generated using one of the two new techniques, one for ECC and one for FFC. Next the identities of the EAP server and EAP peer are combined with the secret element to create the password element used by the key exchange of [3].

The secret element can be generated at provisioning time or a run-time. Either way, the EAP server will generate the password element prior to generation of an EAP-pwd-Commit/Request and the EAP peer will generate the password element prior to generation of an EAP-pwd-Commit/Response.

2.1. Secret Element Derivation for ECC

The new technique to hash into an elliptic curve is the "Simplified Shallue-van de Woestijne-Ulas Method" from [2]. The operations to derive the secret element can be implemented in constant time.

The Simplified SWU method takes a password as input and generates 2 values-- x1 and x2-- at least one of which will be the abscissa of a point on the curve. Since this method does not map its input to the entire curve it is necessary to use a construct from [5] that uses Simplified SWU with two functions that operate as random oracles to produce two different points whose sum is the secret point S:

```
S := SSWU( h1(m) ) + SSWU( h2(m) )
```

Where m is the message to hash, h1() and h2() are random oracles based on hash functions, '+' is point addition, and SSWU() is the "Simplified SWU" hash-to-curve method.

EAP-pwd' uses HKDF ([4]) to instantiate the random oracles. The password and a label is passed to HKDF() to produce a password-seed.

The password seed is then reduced modulo the prime to produce the

input variable, u, for "Simplified SWU" which generates the first intermediate point. This process is repeated with a different label to produce the second intermediate point. Their sum is S.

The particular flavor of HKDF is the random function negotiated by EAP-pwd'.

Algorithmically, the process looks like this:

```
simplified_swu(password) {
    pwd-seed = HKDF(0^n, password,
                    "EAP-pwd' Hash to Element P1", olen(p))
    u = (pwd-seed modulo (p - 2)) + 2

    t = inverse(z^2 * u^4 + z * u^2)
    x1 = (-b/a) * (1 + t)
    gx1 = x1^3 + a * x1 + b
    x2 = z * u^2 * x1
    gx2 = x2^3 + a * x2 + b

    l = gx1 is a quadratic residue modulo p
    v = CSEL(l, gx1, gx2)
    x = CSEL(l, x1, x2)
    y = sqrt(v)

    l = CEQ(LSB(u), LSB(y))
    P1 = CSEL(l, (x,y), (x, p-y))


    pwd-seed = HKDF(0^n, password,
                    "EAP-pwd' Hash to Element P2", olen(p))
    u = (pwd-seed modulo (p - 2)) + 2

    t = inverse(z^2 * u^4 + z * u^2)
    x1 = (-b/a) * (1 + t)
    gx1 = x1^3 + a * x1 + b
    x2 = z * u^2 * x1
    gx2 = x2^3 + a * x2 + b

    l = gx1 is a quadratic residue modulo p
    v = CSEL(l, gx1, gx2)
    x = CSEL(l, x1, x2)
    y = sqrt(v)

    l = CEQ(LSB(u), LSB(y))
    P2 = CSEL(l, (x,y), (x, p-y))

    S = P1 + P2

    output S
}
```

Figure 1: Generation of the ECC Secret Point

Where:

- 0^n is a salt of all zeros whose length equals the length of the digest of the hash function that instantiates HKDF

- p is the prime, q is the order, a and b are part of the equation of the curve, and all of these are defined in the domain parameter set of the chosen curve

- z is a curve-specific parameter derived according to [2] for the chosen curve

- LSB(x) returns the least significant bit of x

- CSEL(x,y,z) operates in constant time and returns y if x is true and z otherwise

- CEQ(x,y) operates in constant time and returns true if x equals y and false otherwise

2.2. Secret Element Derivation for FFC

The new technique to hash into an FFC group is similar to the technique used in [3] but it does so without looping thereby obviating a timing attack that can partition the dictionary.

EAP-pwd' uses HKDF ([4]) to produce a password value which is exponentiated to produce a new element of the same order as the generator of the group. This new element is output.

Algorithmically, the process looks like this:

```
hash_to_ffc(password) {
    pwd-value = HKDF(0^n, password,
                     "EAP-pwd' Hash To Element",
                     olen(p))
    pwd-value = (pwd-value modulo (p - 2)) + 2

    s = pwd-value^((p-1)/q) modulo p

    output s
}
```

Figure 2: Generation of the FFC Secret Point

Where:

- 0^n is a salt of all zeros whose length equals the length of the digest of the hash function that instantiates HKDF

- p is the prime, and q is the order and are defined in the domain parameter set of the chosen group

The secret element, s, is guaranteed to have an order of either 1 or q and the probability that it is 1 is remote enough to ignore.

2.3. Fixing the Password Element

The secret element derived in Section 2.1 or Section 2.2 is used to fix EAP-pwd's Password Element prior to the generation of the EAP-pwd-Commit/Request by the EAP server and prior to generation of the EAP-pwd-Commit/Response by the EAP peer. To do this, they use the negotiated random function to hash the anti-clogging token from [3] and their identities to the length of the order of the negotiated group. This is interpreted as an integer and reduced such that it is between 1 and the order of the group, exclusive. The secret element is then operated on by this value, point multiplication for ECC and exponentiation for FFC, to produce the Password Element.

For ECC groups, this process looks like:

```
fix_PWE(S) {
    val = HKDF(peer-ID | server-ID, token, "Fixing PWE", olen(p))
    val = val modulo (q - 1) + 1

    PWE = val * S
}
```

Figure 3: Generation of PWE

Where: p is the prime, and q is the order and are defined in the domain parameter set of the chosen group.

For FFC groups, this process looks like:

```
fix_pwe(S) {
    val = HKDF(peer-ID | server-ID, token, "Fixing pwe", olen(p))
    val = val modulo (q - 1) + 1

    pwe = s^val modulo p
}
```

Figure 4: Generation of pwe

Where: p is the prime, and q is the order and are defined in the domain parameter set of the chosen group.

3. Acknowledgements

The author thanks Hugo Krawczyk and Riad Wahby.

4. IANA Considerations

IANA is insructed to assign a new EAP method type to EAP-pwd' and replace TBD-1 in this document with that value.

IANA is instructed to assign values from the Random Function registry of [3] for the following:

- TBD-2: HKDF with SHA256 as defined in [4]
- TBD-3: HKDF with SHA384 as defined in [4]
- TBD-4: HKDF with SHA512 as defined in [4]

Replacing TBD-[2-4] with the assigned values.

5. Implementation Considerations

Implementations SHOULD generate the secret element from Section 2.1 and Section 2.2 when the password is provisioned and wait to generate a session-specific password element when the EAP-pwd' protocol begins.

Implementations SHOULD offer use a random function that provides commensurate strength for the curve being negotiated. Guidance is as follows based on the length of the curve's prime, len(p):

- HKDF-SHA256 when len(p) <= 256
- HKDF-SHA384 when 256 < len(p) <= 384
- HKDF-SHA512 when 384 < len(p)

The technique to generate the secret element on an elliptic curve from Section 2.1 only works on Weierstrass curves with an equation of y^2 = x^3 + a*x + b, with a != 0 and b != 0. A different hash-to-curve technique implementable in constant time will have to be used for other curves. [2] defines curve-specific techniques to obtain a secret element for other curves. In the event that such a technique is used, the random function negotiated SHALL be HKDF based on the hash function defined in the ciphersuite of the particular hash to curve technique.

[2] describes useful utility functions that can be used to perform the operations in Figure 1 in constant time. In addition, [7] describes a useful blinding technique that can be used to determine whether number is a quadratic residue modulo a prime in constant time.

6. Security Considerations

The "hunting and pecking" loop done in [3] leaked information on how many loops it took to determine the password element. This allows an attacker to partition the dictionary by excluding possible passwords which would take a different number of loops. After a frighteningly few such partitionings it becomes possible for the attacker to eliminate enough passwords to feasibly launch active attacks to learn the password. [6] describes cache based attacks and timing attacks that are possible against [3].

The Simplified SWU hash-to-curve method with the Brier construct allows for the password element to be derived in constant time which obviates these attacks.

For implementations that cannot become completely constant time (due to, for instance, limitations in a crypto library) it is possible to limit timing attacks by generating the secret element from Section 2.1 and Section 2.2 when the password is provisioned and then generating the password element at run time.

7. References

7.1. Normative References

[1] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997, <http://xml.resource.org/public/rfc/html/rfc2119.html>.

[2] Fax-Hernandez, A., Scott, S., Sullivan, N., Wahby, R., and C. Wood, "Hashing to Elliptic Curves", draft-irtf-cfrg-hash-to-curve A work in progress, July 2019.

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

[4] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

7.2. Informative References

[5] Brier, E., "Efficient Indifferentiable Hashing into Ordinary Elliptic Curves", Advances in Cryptology--Crypto 2010 Springer-Verlag, 2010.

[6] Vanhoef, M. and E. Ronen, "Dragonblood: A Security Analysis of WPA3's SAE Handshake", Cryptology ePrint Archive Report 2019, 2019.

[7] Harkins, D., Ed., "Dragonfly Key Exchange", RFC 7664, DOI 10.17487/RFC7664, November 2015, <http://www.rfc-editor.org/info/rfc7664>.

Author's Address

Dan Harkins
Hewlett Packard Enterprise
3333 Scott boulevard
Santa Clara
United States of America

Email: dharkins@lounge.org