IEEE P802.11  
Wireless LANs

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Finding PWE in Constant Time | | | | |
| Date: 2019-07-08 | | | | |
| Author(s): | | | | |
| Name | Affiliation | Address | Phone | email |
| Dan Harkins | HPE | 3333 Scott boulevard  Santa Clara, California,  United States of America | +1 408 555 1212 |  |
|  |  |  |  |  |

Abstract

The countermeasures against side channel attack that SAE puts into the hunting and pecking loop is inefficient and fragile. Recent attacks—e.g. Dragonblood—underscore this problem. A deterministic method that is computable in constant time without repetitive looping should be used to discover PWE. Do to the minimal number of restrictions on curve type, an appealing way of hashing to an elliptic curve is the Simplified Shallue-Woestijne-Ulas (SSWU) method. For Finite Field Cryptography (FFC) groups, the branching and looping should be removed in order to generate an FFC PWE directly. In addition, it’s important to add detection of group downgrade to SAE.

**Discussion**: SAE requires use of a secret element, PWE, discovered in an agreed upon finite cyclic group. Due to the way the key exchange was initially developed, this PWE discovery cannot be done before the SAE protocol starts.

This element is deterministically discovered by repeatedly hashing the password with some additional information until the resulting hash is the abscissa of a point on the elliptic curve (for ECC) or by exponentiating the hash digest to a constant to produce an element (for FFC). Both of these techniques are prone to side channel attack. While much work has gone into countermeasures to mitigate these attacks, for instance looping 40 times regardless of how soon an abscissa is found, the countermeasures render the whole method of PWE discovery inefficient and very fragile.

When a group is rejected, another one is tried. A man-in-the-middle could exploit this to do a downgrade attack where “good” groups are rejected by the attacker until a “not so good” group is offered which is allowed to go through. There is no way to detect this attack.

**Proposal**: For ECC, use the Simplified Shallue-Woestijne-Ulas (SSWU), as defined in draft-irtf-cfrg-hash-to-curve, to directly hash-to-curve. This method will work for any Weierstrass curve which makes it ideal for use with SAE.

Since SSWU does not generate all points on the elliptic curve, the SSWU method by itself could not be used with the current SAE security proof in the random oracle model. Therefore, the SSWU method is enhanced by the following technique from Brier et al to hash to a password-based element, PT:

PT(m) := SSWU(h1(m)) + SSWU(h2(m))

Where *m* is the information being hashed, *h1*() and *h2*() are random oracles based on a hash function, and ‘+’ is the element operator (*i.e*., point arithmetic) for ECC.

For FFC groups, the results of the hash will be reduced modulo the prime to produce PT instead of skipping values that would be larger than the prime when interpreted as an integer. No looping is needed.

The secret element, PT, can be computed when the password is provisioned and retained until SAE begins at which time it can be combined with the peer’s MAC addresses to create a session-specific PWE for SAE. Such an approach not only allows for hashing-to-element to be implemented in constant time; it also helps to avoid timing attacks for implementations that cannot be completely constant time.

The Simplified SWU method is defined in draft-ietf-cfrg-hash-to-curve. Since this is an Internet-Draft and normative references to such memos are not appropriate (due to their transient nature) the algorithm from the Internet-Draft is copied here. If the draft is advanced to RFC status prior to Draft P802.11REVmd being published this can be changed to include a reference the RFC and the explicit math for SSWU and the calculations of parameter z can be removed in favor of reference to the RFC. At that time IETF downref procedures will have to be followed since we will have a normative reference to an Informational RFC (similar to what we have with RFC 2104 which describes HMAC).

These new techniques are not backwards compatible with the “hunting-and-pecking” loop in the standard and therefore must be signalled as new capabilities. This signalling is compounded by two things: SAE happens before association, and selection of PWE happens before any transactional frames have been sent. Since SAE happens before association, it’s not possible to use an AKM. So support of this is signalled by an AP using a bit in the Extended Capabilities field. Since the SAE protocol itself is not being changed the signalling to use this new PWE discovery method has to be as unintrusive as possible. Therefore, it is signalled with a new status code in the SAE Commit message—“I’m hashing directly to PWE!”. Everything else in SAE (state machine, message formats, computation, message construction and processing, etc) remains the same.

To detect a group downgrade, each peer appends rejected groups to its next Commit message. A concatenation of both sides’ lists of rejected groups (one or both can, of course, be null) is passed as salt to the key derivation function. This ensures that both sides will agree on the same key if there is no man-in-the-middle fiddling with SAE Commit messages. If there is a man-in-the-middle the SAE exchange will fail.

**Changes necessary from D2.3:**

*Instruct the editor to obtain values from ANA for <ANA-1>, <ANA-2>, and <ANA-3> and to replace all occurances of these placeholders with the real values throughout.*

*Instruct the editor to add the following line to section 2:*

**2. Normative references**

IETF RFC 5869, HMAC-based Extract and Expand Key Derivation Function, H. Krawczyk, P. Eronen May 2010

*Instruct the editor to modify table 9-42 as indicated:*

**Table 9-42—Authentication frame body**

|  |  |  |
| --- | --- | --- |
| **Order** | **Information** | **Notes** |
| 23 | Rejected Groups | The Rejected Groups element is conditionally present in certain Authentication frames as defined in Table 9-43 (Presence of fields and elements in Authentication frames) |

*Instruct editor to modify table 9-43 as indicated:*

**Table 9-43—Presence of fields and elements in Authentication frames**

|  |  |  |  |
| --- | --- | --- | --- |
| **Authentication Algorithm** | **Authentication transaction sequence number** | **Status Code** | **Presence of fields 4 onwards** |
| SAE | 1 | Any | The Scalar field is present if the Status Code field is zero or <ANA-1>.  The FFE field is present if the Status Code field is zero or <ANA-1>.  The Anti-Clogging Token field is present if status is 76 or <ANA-1> or if the Authentication frame is in response to a previous rejection with Status 76.  The Finite Cyclic Group field is present if the Status Code field is zero, 76, 77, , or <ANA-1>.  The Password Identifier element is optionally present  if the Status Code is zero, 123, or <ANA-1>.  Rejected Groups element is conditionally present if the Status Code is <ANA-1>. |

*Instruct the editor to modify table 9-52 as indicated:*

**Table 9-52—Status codes**

|  |  |  |
| --- | --- | --- |
| **Status code** | **Name** | **Meaning** |
| <ANA-1> | SAE\_HASH\_TO\_PWE | SAE authentication uses an alternate form of direct hashing, as opposed to looping, to obtain the PWE |
| <ANA-1> + 1-65535 | Reserved |  |

*Instruct the editor to modify table 9-94 as indicates:*

**Table 9-94—Element IDs**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Element** | **Element ID** | **Extended ID Extension** | **Extensible** | **Fragmentable** |
| Rejected Groups (see 9.4.2.244 (Rejected Groups element)) | 255 | <ANA-2> | No | No |
| Reserved | 255 | <ANA-2> + 1-255 |  |  |

*Instruct the editor to modify table 9-95 as indicated:*

**Table 9-95—BSS membership selector value encoding**

|  |  |  |
| --- | --- | --- |
| **Value** | **Feature** | **Interpretation** |
| 123 | SAE Hash to Element Only | Indicates that support for the direct hashing to element technique in SAE is required in order to join the BSS. |

*Instruct the editor to modify table 9-322 as indicated:*

**Table 9-322—Extended RSN Capabilities field**

|  |  |  |
| --- | --- | --- |
| **Bit** | **Information** | **Notes** |
| <ANA-3> | SAE hash-to-element | The AP supports directly hashing to obtain PWE instead of looping. See 12.4.4.2.3 and 12.4.4.3.3 |
| <ANA-3> + 1 – (8xn – 1) | Reserved |  |

*Instruct the editor to add new section 9.4.2.244, and allocate a number to replace XYZ, adjusting subsequent figure numbers as appropriate:*

**9.4.2.244 Rejected Groups element**

The Rejected Groups element lists the finite cyclic groups that have been rejected by a peer for use in SAE. The format of the Rejected Groups element is shown in Figure 9-XYZ

|  |  |  |  |
| --- | --- | --- | --- |
| Element ID | Length | Element ID Extension | Rejected Groups |

Octets: 1 1 1 variable

**Figure 9-XYZ—Rejected Groups element format**

The Element ID, Length, and Element ID Extension fields are defined in 9.4.2.1 (General).

The Rejected Groups field contains a concatenated list of unsigned integers representing Finite Cyclic Groups that have been rejected by a peer in a previous authentication attempt.

*Instruct the editor to modify section 12.4.4.2.2 as indicated:*

**12.4.4.2.2 Generation of the password element with ECC groups by looping**

When a direct form of hashing to discover PWE is not signaled by the AP, or if the SAE initiator does not signal its use in its SAE Commit message, the password element of an ECC group (PWE) shall be generated in the following random hunt-and-peck fashion.

*Instruct the editor to add the following new section and assign numbers for the new tables:*

**12.4.4.2.3 Hash-to-curve generation of the password element with ECC groups**

An SAE peer, e.g. a Mesh STA or an AP, indicates support for direct hashing to obtain an ECC password element by setting the SAE hash-to-element bit in the Extended RSN Capabilities field in all Beacon and Probe Response frames. An SAE initiator that has identified a peer that supports this technique (through receipt of Beacon or Probe Response frames) shall derive a secret element, PT, according to the following technique and indicate this by setting the status code in the SAE Commit message to “SAE\_HASH\_TO\_ELEMENT”. An SAE initiator shall not indicate support for this form of element derivation unless its peer has already signalled support for this method. If an SAE Commit message is received with status code equal to “SAE\_HASH\_TO\_ELEMENT” the peer shall generate PWE using the following technique and reply with its own SAE Commit message with status code equal to “SAE\_HASH\_TO\_ELEMENT.”

The direct hashing technique to derive an element of an ECC group is the Simplified Shallue-Woestijne-Ulas (SWU) deterministic hash-to-curve method. The SSWU method is called twice with two distinct functions to produce two points on the elliptic curve. The two points are summed to create a secret element PT.

This method works for all Weierstrass elliptic curves whose constants *a* and *b* are both not equal to zero. Other curves shall not be used with this hash-to-curve method.

This hash-to-curve method uses HKDF (RFC 5869) with the hash algorithm taken from table 12-abc based on the length of the prime of the ECC group to perform both functions. First HKDF-Extract is passed an all zero salt, the password, and optionally a password identifier to produce and intermediary password seed. The resulting seed is passed to HKDF-Expand to produce two distinct strings using different labels. Both values are reduced such that 0 < pwd-value < q, exclusive and then passed to Simple SWU to produce distinct points, P1 and P2 whose sum is PT.

**Table 12-abc – Hash algorithm based on length of prime**

|  |  |  |
| --- | --- | --- |
| **ECC prime length** | **FFC prime length** | **Hash algorithm** |
| p <= 256 | p <= 2048 | SHA-256 |
| 256 < p <= 384 | 2048 < p <= 3072 | SHA-384 |
| 384 < p | 3072 < p | SHA-512 |

This secret PT is stored until needed to generate a session-specific PWE (see 12.4.5.2 (PWE and secret generation)).

Algorithmically, this process is as follows:

*len =* *olen(p) + ceil(olen(p)/2)*

*pwd-seed = HKDF-Extract(0n, password [|| identifier])*

*pwd-value = HKDF-Expand(pwd-seed, “SAE Hash to Element u1 P1”, len)*

*u1 = pwd-value modulo p*

*P1 = SSWU(u1)*

*pwd-value = HKDF-Expand(pwd-seed, “SAE Hash to Element u2 P2”, len)*

*u2 = pwd-value modulo p*

*P2 = SSWU(u2)*

*PT = elem-op(P1, P2)*

where

* HKDF-Extract() and HKDF-Expand() are the functions defined in RFC 5869 instantiated with the hash algorithm from Table 12-abc (Hash algorithm based on prime length))
* 0n represents a *salt* of all zeros whose length is the length of the digest of the instantiated hash function
* olen() returns the length of its argument in octets
* ceil() returns the smallest integer value that is not less than the passed parameter
* [|| identifier] indicates the optional inclusion of a password identifier, if present
* SSWU(u) is a call to the Simple SWU routine passing in parameter u

The SSWU method produces two values, x1, and x2, at least one of which will represent an abscissa of a point on the curve. If x1 is the abscissa then x1 becomes the x-coordinate otherwise x2 becomes the x-coordinate. The equation of the curve with the x-coordinate produces the square of the y-coordinate which is recovered by taking the square root. The two possible results of the square root are discriminated by checking its least significant bit with the least significant bit of *u*. The result is a point on the curve.

The SSWU method takes a curve-specific parameter, z, which is determined by finding the smallest number that satisfies the following rules, given p, a, and b, from the curve’s domain parameter set:

1. z = n if
   1. n is not a quadratic residue modulo p
   2. (b/(n\*a)^3 + a \* (b/(n\*a)) + b is a quadratic residue modulo p
2. z = -n if
   1. -n is not a quadratic residue modulo p
   2. (b/(-n\*a)^3 + a \* (b/(-n\*a)) + b is a quadratic residue modulo p
3. if both n and -n satisfy the above criteria then z is n, the positive value

Values for some defined groups based on their IANA-assigned values are listed in table 12-xyz:

**Table 12-xyz—Unique curve parameter**

|  |  |  |
| --- | --- | --- |
| **Curve Name** | **IANA value** | **z** |
| NIST p256 | 19 | -2 |
| NIST p384 | 20 | -2 |
| NIST p521 | 21 | -2 |
| NIST p192 | 25 | -5 |
| NIST p224 | 26 | -11 |
| Brainpool p256 | 28 | -2 |
| Brainpool p384 | 29 | -5 |
| Brainpool p512 | 30 | 2 |

Algorithmically, the Simplified SWU method is:

*SSWU(u) {*

*m = z^2 \* u^4 + z \* u^2*

*l = CEQ(m, 0)*

*t =CSEL(l, 0, inverse(m))*

*x1 = CSEL(l, (b / (z \* a)), ((-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))*

*P = CSEL(l, (x,y), (x, p-y))*

*output P*

*}*

Where:

* p, a, and b are all defined in the domain parameter set for the curve.
* z is a curve-specific parameter from Table 12-xyz
* inverse(x) is calculated as x^(p-2) modulo p
* x is a quadratic residue if x^((p-1)/2) modulo p is zero or one
* 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.

All operatioins should be done in constant time.

Note—For curves based on a prime, p, such that p = 3 mod 4 the square root can be implemented with a single modular exponentiation of (p+1)/4, that is sqrt(w) = w(p+1)/4 modulo p.

*Instruct the editor to modify section 12.4.4.3.2 as indicated:*

**12.4.4.3.2 Generation of the password element with FFC groups by looping**

When a direct form of hashing to discover a password element is not signaled by the AP, or if the SAE initiator does not signal its use in the SAE Commit message the password element of an FFC group (**PWE**) shall be generated in the following random hunt-and-peck fashion.

*Instruct the editor to add the following new section:*

**12.4.4.3.3 Direct Generation of the password element with FFC groups**

An SAE peer indicates support for direct hashing to obtain the FFC password element by setting the SAE hash-to-PWE bit in the Extended RSN Capabilities field in all Beacon and Probe Response frames. An SAE initiator that has identified a peer that supports the following technique (through receipt of Beacon or Probe Response frames) shall derive PT according to the following technique and indicate this by setting the status code in the SAE Commit message to “SAE\_HASH\_TO\_ELEMENT”. An SAE initiator shall not indicate support for this form of PWE derivation unless its peer has already signalled support. If an SAE Commit message is received with status code equal to “SAE\_HASH\_TO\_ELEMENT” the peer shall generate PWE using the following technique and reply with its own SAE Commit message with status code equal to “SAE\_HASH\_TO\_ELEMENT.”

This direct hashing technique uses HKDF (RFC 5869) with the hash algorithm taken from table 12-abc based on the length of the prime of the FFC group.

To perform this direct hashing technique, HKDF (RFC 5869) is passed a constant salt, the password, optionally a password identifier, as an input key, a constant label “SAE Hash to Element”, and the length of the prime to produce a password value. The resulting password value shall be reduced into a range such that 1 < pwd-value < p. Then, it shall be raised to the power (p–1) / q and reduced modulo p (where *p* is the prime number and *q* is the order). This will ensure PT is a generator of order either 1 (if PT=1) or q (for all other values). The probability of PT taking the value 1 is negligible.

This secret PT is stored until needed to generate a session-specific PWE.

Algorithmically, this process is as follows:

*len =* *olen(p) + ceil(olen(p)/2)*

*pwd-seed = HKDF-Extract(0n, password [|| identifier])*

*pwd-value = HKDF-Expand(pwd-seed, “SAE Hash to Element”, len)*

*pwd-value = (pwd-value modulo (p-2)) + 2*

*PT = pwd-value(p-1)/q modulo p*

where

* HKDF-Extract() and HKDF-Expand() are defined in RFC 5869 instantiated with the hash algorithm identified Table 12-abc (Hash algorithm based on length of prime))
* 0n represents a *salt* of all zeros whose length is the length of the digest of the instantiated hash function
* olen() returns the length of its argument in octets
* ceil() is the ceiling function that returns the largest whole integer that represents the passed parameter
* [|| identifier] indicates the optional inclusion of a password identifier, if present.
* p and q are defined in the domain parameter set for the group.

This secret PT is stored until needed to generate a session-specific PWE (see 12.4.5.2 (PWE and secret generation)).

*Instruct the editor to modify section 12.4.5.2 as indicated:*

**12.4.5.2 PWE and secret generation**

Prior to beginning the protocol message exchange, the secret element PWE and two secret values are generated.

When a STA supports directly hashing to a group element (according to 12.4.4.2.3 or 12.4.4.3.3) it computes a secret element, PT, off-line at provisioning time for all groups it wishes to support with that password. Prior to initiating SAE to a STA which also supports the direct form of hashing to a group element, or upon receipt of an SAE Commit message indicating it was generated using a direct form of hashing to a group element, it shall generate PWE by hashing the two peer MAC addresses to produce a digest, reducing the digest modulo the order of the particular group, q, interpreting the reduced digest as an integer and using it with the secret element to generate PWE:

*val = H(MAX(STA-A-MAC, STA-B-MAC) || MIN(STA-A-MAC, STA-B-MAC))*

*val = val modulo (q – 1) + 1*

*PWE = scalar-op(val, PT)*

If a STA does not support a direct form of hashing to a group element it generates PWE after selecting a group , either the most preferred group if the STA is initiating SAE to a peer, or the group from a received SAE Commit message if the STA is responding to a peer. The PWE shall be generated for that group (according to 12.4.4.2.2 (Generation of the password element with ECC groups) or 12.4.4.3.2 (Generation of the password element with FFC groups), depending on whether the group is ECC or FFC, respectively) using the identities of the two STAs and the configured password.

After generation of the PWE, each STA shall generate a secret value, rand, and a temporary secret value, mask, each of which shall be chosen randomly such that 1 < rand < q and 1 < mask < q and (rand + mask ) mod q is greater than 1, where q is the (prime) order of the group. If their sum modulo q is not greater than 1, they shall both be irretrievably deleted and new values shall be randomly generated. The values rand and mask shall be random numbers produced from a quality random number drawn from a uniform distribution generator. These values shall never be reused on distinct protocol runs.

*Instruct the editor to modify section 12.4.5.4 as indicated:*

**12.4.5.4 Processing of a peer’s SAE Commit message**

If the peer’s SAE Commit message contains a password identifier, the value of that identifier shall be used in construction of the password element (PWE) for this exchange. If a password identifier is present in the peer’s SAE Commit message and there is no password with the given identifier a STA shall fail authentication.

If the peer’s SAE Commit message contains a Rejected Groups element, the list of rejected groups shall be checked to ensure that all of the groups in the list are not approved groups. If any rejected groups in the list are acceptable then processing of the SAE Commit message terminates and the STA shall reject the peer’s authenticaiton. While the rejected groups are appended to the Rejected Groups element as they are rejected (see 12.4.7.4 (Encoding and decoding of SAE Commit messages)) there is no inherent order to the groups in the list. The order in which they are sent and received shall be retained when deriving keys.

Upon receipt of a peer’s SAE Commit message both the scalar and element shall be verified.

If the scalar value is greater than 1 and less than the order, r , of the negotiated group, scalar validation succeeds; otherwise, it fails. Element validation depends on the type of group. For FFC groups, the element shall be an integer greater than 1 and less than the prime number p minus 1, (p – 1), and the scalar operation of the element and the order of the group, r , shall equal 1 modulo the prime number p . If either of these conditions does not hold, element validation fails; otherwise, it succeeds. For ECC groups, both the x- and y-coordinates of the element shall be non-negative integers less than the prime number p, and the two coordinates shall produce a valid point on the curve satisfying the group’s curve definition, not being equal to the “point at the infinity.” If either of those conditions does not hold, element validation fails; otherwise, element validation succeeds.

If either scalar validation or element validation fails, the STA shall reject the peer’s authentication. If both the scalar and element from the peer’s SAE Commit message are successfully validated, a shared secret element, K , shall be derived using the scalar and element (peer-commit-scalar and ***PEER-COMMITELEMENT*** , respectively) from the peer’s SAE Commit message and the STA’s secret value.

***K*** = scalar-op(rand , (elem-op(scalar-op(peer-commit-scalar , ***PWE***),

***PEER-COMMIT-ELEMENT***)))

If the shared secret element, ***K***, is the identity element for the negotiated group (the value one for an FFC group or the point-at-infinity for an ECC group) the STA shall reject the peer’s authentication. Otherwise, a secret value, k , shall be computed as:

k = F(K )

The entropy of k shall then be extracted using H to produce keyseed. The key derivation function from 12.7.1.6.2 (Key derivation function (KDF)) shall then be used with the hash algorithm identified by the AKM suite selector (see Table 9-151 (AKM suite selectors)) to derive a key confirmation key, KCK, and a pairwise master key, PMK, from keyseed. When used with AKMs 8 or 9, both the KCK and PMK shall be 256 bits in length, and therefore the length of keying material derived is 512. Use of other AKMs require definition of the lengths of the KCK, and the PMK. When both SAE Commit messages indicated a status code of SAE\_HASH\_TO\_ELEMENT a salt is passed to the KDF consisting of a concatenation of the Rejected Groups from each peer’s Rejected Groups element, those of the peer with the highest MAC address go first (if only one sent a Rejected Groups element then the salt will consist of that list). If neither peer sent a Rejected Groups element or the status code was not SAE\_HASH\_TO\_ELEMENT the salt shall consist of 32 octets of the value zero.

*keyseed = H(salt, k)*

*kck\_and\_pmk = KDF-Hash-512(keyseed, “SAE KCK and PMK”,*

*(commit-scalar + peer-commit-scalar) mod r)*

*KCK = L(kck\_and\_pmk, 0, 256)*

*PMK = L(kck\_and\_pmk, 256, 256)*

Where

* *salt* is either 32 octets of 0 or a list of rejected groups.
* *KDF-Hash-512* is the key derivation function defined in 12.7.1.6.2 (Key derivation function (KDF)) using the hash algorithm defined by the AKM suite selector (see Table 9-151 (AKM suite selectors)).

*Instruct the editor to modify section 12.4.7.4 as indicated:*

**12.4.7.4 Encoding and decoding of SAE Commit messages**

An SAE Commit message shall be encoded as an Authentication frame with an Authentication Algorithm

Number field set to 3, a Transaction Sequence Number of 1 and a Status Code of SUCCESS or SAE\_HASH\_TO\_ELEMENT. Status codes not equal to SUCCESS or SAE\_HASH\_TO\_ELEMENT indicate a rejection of a peer’s SAE Commit message and are described in 12.4.7.6 (Status codes).

An SAE Commit message shall consist of a Finite Cyclic Group field (9.4.1.42 (Finite Cyclic Group field)) indicating a group, a Scalar field (9.4.1.39 (Scalar field)) containing the scalar, and an FFE field containing the element (9.4.1.40 (FFE field)). If the SAE Commit message is in response to an Anti-Clogging Token field request (see 12.4.7.6 (Status codes)), the Anti-Clogging Token field is present (see 9.4.1.38 (Anti-Clogging Token field)). If a password identifier is used in generation of the password element (PWE) the Password identifier element shall be present and the identifier shall be encoded as a UTF-8 string in the Identifier portion of the element (see 9.4.2.216 (Password Identifier element)). If an SAE Commit message is being sent in response to rejection of a previous Commit message due to UNSUPPORTED\_FINITE\_CYCLIC\_GROUP, the group that was rejected shall be appended, after the rejected groups from previous attempts if applicable, to the Rejected Groups portion of the Rejected Groups element. Each rejected group shall be represented as an unsigned 16-bit integer using the bit ordering conventions of 9.2.2 (Conventions).

*Instruct the editor to modify section 12.4.7.6 as indicated:*

**12.4.7.6 Status codes**

An SAE Commit message with a status code not equal to SUCCESS or SAE\_HASH\_TO\_ELEMENT shall indicate that a peer rejects a previously sent SAE Commit message. An unsupported finite cyclic group is indicated with a status code of UNSUPPORTED\_FINITE\_CYCLIC\_GROUP, “Authentication is rejected because the offered finite cyclic group is not supported.” An unknown password identifier is indicated with a status code of UNKNOWN\_PASSWORD\_IDENTIFER, “Authentication is rejected because the password identifier is unknown.” An Anti-Clogging Token field is requested by transmitting an SAE Commit message with a status code of ANTI\_CLOGGING\_TOKEN\_REQUIRED, “Anti-Clogging Token Requested,” with the Anti-Clogging Token field occupying the Token field of the Authentication frame.

*Instruct the editor to append the following to section J.10:*

Hash to Curve technique of PE/PWE Generation

group: 19

identifier: psk4internet

password: mekmitasdigoat

MAC address 1: 43:1b:89:60:3f:be

MAC address 2: 65:aa:ea:33:2a:42

-----------------------------------------------------------------------

u1:

bbba8fe5 d1fdbfd6 a5e48228 d8a2980a ba503364 a45b8db4 391e31f7 fc7884c4

SSWU(u1)

--------

m = z^2 \* u^4 + z \* u^2:

0f01a0c3 4193bfdf b5f79668 9737535a 1cb67848 2d311d4b a5faf4ae 0a9aad6d

t = 1/m:

1742547c ea2c5a9a 4fbb909a 618b1da8 0b16c541 b2c2ab78 79921abc 4b588cfb

(-b/a) \* (1 + t):

1589bbf2 a6299889 efbb5222 a33cb8fc 70368efe 5e60aecc 11c356a7 8c83dd3e

b/(z \* a):

c6344c5b 0e4b9204 e2036071 c1913ee0 99d07ee3 5df20d2c 4b5da0a0 4eb24548

m != 0, so x1 = (-b/a)\*(1+t)

x1:

1589bbf2 a6299889 efbb5222 a33cb8fc 70368efe 5e60aecc 11c356a7 8c83dd3e

gx1:

5083c9ec f8f4e413 ce023c41 2ab37495 d88bcd09 151a31f4 a43ca439 c28d3b22

x2:

4c4388ed cf56e139 b0513ab0 38d080b1 8ede4315 72da1a13 4b2c9e51 691268cd

gx2:

6b380456 67fa9e6c be666e05 e1ae25ee c20ce5ce a2bc87f9 6a79ed31 3f7a9d21

gx1 is a quadratic residue

gx2 is not a quadratic residue

point P1:

P1.x:

1589bbf2 a6299889 efbb5222 a33cb8fc 70368efe 5e60aecc 11c356a7 8c83dd3e

P1.y:

2addcc19 01b7100d f6ff40ce 4184276f b90dcd7d bb5dc9bf d5e74cf8 b1a76170

-----------------------------------

u2:

439134db 422625fd 9ee7e636 4baf4cb2 12205aad 3c1f9d55 6ccd2eda 626d766c

SSWU(u2)

--------

m = z^2 \* u^4 + z \* u^2:

28899c6f e662d88e ac6a31a0 ccb337e1 4521ba3b 72be8e97 5b95e69c eadfc82b

t = 1/m:

753821c5 157b9f22 9130fd22 8c735d85 bb157f24 2d13ef64 a1659696 0f423806

(-b/a) \* (1 + t):

02422071 a480e52b c32a33d1 efb7e4fa 03f4f34a 602e4d8d 60ff438b b8fb0a28

b/(z \* a):

c6344c5b 0e4b9204 e2036071 c1913ee0 99d07ee3 5df20d2c 4b5da0a0 4eb24548

m != 0, so x1 = (-b/a)\*(1+t)

x1:

02422071 a480e52b c32a33d1 efb7e4fa 03f4f34a 602e4d8d 60ff438b b8fb0a28

gx1:

e7f9a3f6 95f3a517 04528bd2 e8b68f3f 56e9e912 9c106400 c3f4d776 86bcd90a

x2:

2594c3fd 5f91df67 7ac9dee6 f65642a2 0de1b417 64e71c41 855c3ea0 0ede8fd5

gx2:

a09ba929 98eb49b8 b3daf3de 205b3931 4eb12e3a 2ea36ae8 1ab31f61 1dd7facf

gx1 is a quadratic residue

gx2 is not a quadratic residue

point P2:

P2.x:

02422071 a480e52b c32a33d1 efb7e4fa 03f4f34a 602e4d8d 60ff438b b8fb0a28

P2.y:

2eb94ce8 21858e17 3ab00187 c0fa8495 d205ad0b 3879de85 ea8eb58d eebcd4f6

-----------------------------------

PE = P1 + P2:

PE.x:

b32fe479 b29e6dd1 79a31b2e f8d947ef 7b5b2a6f b2773df7 2641cd28 e0fabd85

PE.y:

88cec9ec 5ba1f34d 404b8e70 1e5abbff c1dee7a5 69c97f01 e09e1280 48af654b

val:

1a7c192c 5d51f37a ce6177c2 5337ed82 7d4ea68f a4c82af0 b416b9f8 b0878b02

PWE = val \* PE:

PWE.x:

f87ff7bd 5783d0dd c427e220 5956c698 5c59e700 e5c1f2b4 6c7ea4d8 6a46226d

PWE.y:

8111fece 71d6198c d8093e58 8e8d0bf7 0d633aea 05875e64 adfe4710 82cbab20

**References:**

Brier E., Coron JS., Icart T., Madore D., Randriam H., Tibouchi M. (2010) “Efficient Indifferentiable Hashing into Ordinary Elliptic Curves.” In: Rabin T. (eds) Advances in Cryptology – CRYPTO 2010. Lecture Notes in Computer Science, vol 6223. Springer, 2010

Shallue, A. and van de Woestijne, C, “Construction of rational points on elliptic curves over finite fields”, Lecture Notes in Computer Science, vol 4076, pages 510-524, Springer, 2006.

Ulas, M., “Rational points on certain hyperelliptic curves over finite fields.”, Polish Academy of Sciences, 55(2): 97-104, 2007

Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R., Wood, C., “Hashing to Elliptic Curves”, draft-irtf-cfrg-hash-to-curve, A work in progress, July 2019

Krawczyk, H., Eronen, P., “HMAC-based Extract and Expand Key Derivation Function (HKDF)”, RFC 5869, May 2010