# IEEE Std 1609.2-2016 Guidance Note 10: Hash function when calculating implicit certificates

William Whyte, Security Innovation

Version 0.1, 2017-03-07

## 1 Introduction

This note identifies an area in IEEE Std 1609.2-2016 that may cause confusion to implementers and to specifiers of Secure Data Exchange Entities (SDEEs, i.e. any entity that makes use of WAVE Security Services as defined in 1609.2). The note clarifies the relevant text, which could create confusion about the hash process used to calculate implicit certificates. The clarification is that when calculating the hash value Hn used in the standard SEC4 to generate implicit certificates, no right shift should be performed. This means that the way that 1609.2 uses SEC4 is not strictly compliant with SEC4 and the revised 1609.2 text provided in this guidance note should be taken to supersede the relevant text in SEC4.

## 2 Summary of issue

### 2.1 Background

The problem arises from inconsistencies in how a hash output, which is an octet string, gets converted to an integer for use in the Elliptic Curve Digital Signature Algorithm (ECDSA) and in Elliptic Curve Qu-Vanstone (ECQV, or “implicit”) certificates.

Note that to the best of our knowledge there is no ambiguity in the specification of the inputs to hash functions in 1609.2. The issues arise in the use of the hash output.

In ECDSA all integers are taken modulo n, which is a 256-bit long number, and all hashes are output from SHA-256, which makes them also 256 bits long. That means that some outputs of SHA-256 are larger than n. ECDSA doesn't care about this: it just treats the output h as a 256-bit number and reduces mod n if necessary.

However, in the specification of ECQV in SEC 4, which is the reference cited by 1609.2 for ECQV certificates, the hash is treated differently: it's reduced to a 255-bit number so it's always less than n, and it's reduced not by simply setting the top bit to 0, but by right-shifting the entire thing by one bit. SEC 4 refers to this right-shifted hash as Hn. This hash is used to derive the public key from implicit certificate by the formula:

Public Key = Hn * (Reconstruction value from the implicit certificate) + (Public key of the CA)

(The exact details of the input to Hn are not germane to this guidance note, as noted above, so those details are omitted in the interest of compactness.)

1609.2 explictly refers to the hash in SEC 4 in two places:

- 5.3.2 says that the output from a SHA-256 hash specified in 1609.2 shall be treated as the hash input to Hn, i.e. it implies that the hash should be right-shifted.
- 6.4.8 talks about the "value H(CertU)", rather than about "Hn(CertU)". "H(CertU)" isn't a string that appears in SEC4. The implication is that .2 does not expect the right-shift to happen.

There is no cryptographic reason to prefer one approach over the other out of standard-ECDSA and SEC4. Both approaches introduce a significant bias; the algorithm is clearly not vulnerable to any bias in these values, so either approach is okay. Everyone who's implemented ECDSA would be expecting the mod-n approach rather than the right-shift approach, and this inconsistency in approach is clearly ripe for misinterpretation (as has already happened).

### 2.2 Impact

The impact is that if two entities calculate the hash differently, they will calculate the public and (if appropriate) private keys differently. For example:

- If vehicle 1 has certificates issued by CA A, and they calculate the hash differently, then the private key calculated by vehicle 1 will not correspond to the public key derived from the certificate (because the private key calculation involves both the device's and the CA's calculation of Hn and these will be inconsistent).
- If vehicle 1 has certificates issued by CA A, and vehicle 2 calculates the hash differently from CA A, then vehicle 2 will not be able to verify messages sent by vehicle 1.

This inconsistency would fragment the deployed system into two non-interoperable populations. Therefore it is critical to issue guidance to ensure that all implementations implement 1609.2 consistently.

## 3 Proposed resolution

The proposed resolution is that **when the value Hn is used to calculate implicit certificates for use in the 1609.2 ecosystem, the right-shift shall NOT be applied**.

### 3.1 Proposed changes to 1609.2

Changes are in red font and underlined.

#### 3.1.1 Clause 5.3.2

a) In this standard, an implicit certificate is encoded as an ImplicitCertificate, as defined in 6.4.5, encoded with the Canonical Octet Encoding Rules (COER). All references to “the certificate CertU” in SEC 4 should be taken as referring to the encoded ImplicitCertificate except in the instance the implicit certificate is hashed to an integer modulus n; this case is addressed in item b) below.

b) When an implicit certificate is hashed to an integer modulo n, the input is not simply the implicit certificate CertU but the information specified below. This affects the following steps in SEC 4:

   1) Section 3.4, Action 7

   2) Section 3.5, Action 4

   3) Section 3.6, Action 2

   4) Section 3.7, Action 4

   5) Section 3.8, Action 4

   The encoded data input to the hash function is Hash (ToBeSignedCertificate from the subordinate certificate as specified in 6.4.8, canonicalized as specified in 6.4.3) || Hash (Entirety of issuer certificate, canonicalized as specified in 6.4.3).

c) SHA-256 shall be used as the Hash algorithm H

d) The output of the hash function is not converted to an integer mod n using the mechanism specified in SEC 4, section 2.3. Instead, the hash function is converted to an integer by taking the 256-bit output from SHA-256, converting that bit string to an octet string using the Bit String To Octet String Conversion Primitive of SEC 1, and then converting that octet string to an integer using the Octet String To Integer Conversion Primitive of SEC 1.

### 3.1.2 Clause 6.4.8

For both implicit and explicit certificates, when the certificate is hashed to create or recover the public key (in the case of an implicit certificate) or to generate or verify the signature (in the case of an explicit certificate), the hash is Hash (*Data input*) || Hash (*Signer identifier input*), where:

— *Data input* is the COER encoding of `toBeSigned`, canonicalized as described above.

— *Signer identifier input* depends on the verification type, which in turn depends on the choice indicated by `issuer`. If the choice indicated by `issuer` is `self`, the verification type is *self-signed* and the *signer identifier input* is the empty string. If the choice indicated by `issuer` is not `self`, the verification type is *certificate* and the *signer identifier input* is the COER encoding of the canonicalization per 6.4.3 of the certificate indicated by `issuer`.

In other words, for implicit certificates, the value H (CertU) in SEC 4, section 3, is for purposes of this standard taken to be H [H (canonicalized ToBeSignedCertificate from the subordinate certificate) || H (canonicalized entirety of issuer Certificate)]. See 5.3.2 for further discussion, including material differences between this standard and SEC4 regarding how the hash function output is converted from a bit string to an integer.

## 4 Change log

V0.1, 2017-03-17: Initial draft

V0.2, 2017-03-17: Corrected title to refer to 1609.2-2016 rather than 1609.2-2017