Stream: Internet Engineering Task Force (IETF)
RFC: 10031
Category: Standards Track
Published: August 2026
ISSN: 2070-1721
Authors:
R. Housley *Vigil Security*
C. Bonnell *TurboLight Solutions*
J. Mandel *AKAYLA*
T. Okubo *Penguin Securities*
M. StJohns *NthPermutation Security LLC*

RFC 10031
Media Access Control (MAC) Addresses in X.509 Certificates

Abstract

This document defines a new `GeneralName.otherName` for inclusion in the X.509 Subject Alternative Name (SAN) and Issuer Alternative Name (IAN) extensions to carry an IEEE Media Access Control (MAC) address. The new name form makes it possible to bind a Layer 2 interface identifier to a public key certificate. Additionally, this document defines how constraints on this name form can be encoded and processed in the X.509 Name Constraints extension (NCE).

Status of This Memo

This is an Internet Standards Track document.

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). Further information on Internet Standards is available in Section 2 of RFC 7841.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at https://www.rfc-editor.org/info/rfc10031.

Copyright Notice

Table of Contents

1. Introduction

Deployments that use X.509 certificates to identify a device by a Media Access Control (MAC) address need a standard way to encode it in the Subject Alternative Name (SAN) extension defined in [RFC5280]. This document defines a new `otherName` form "`MACAddress`". The name form carries either a 48-bit IEEE 802 MAC address (EUI-48) or a 64-bit extended identifier (EUI-64) in an `OCTET STRING` [X680]. Additionally, the name form also can convey constraints on EUI-48 or EUI-64 values when included in the Name Constraints extension (NCE) defined in Section 4.2.1.10 of [RFC5280]. The new name form enables certificate-based authentication at Layer 2 and facilitates secure provisioning in Internet of Things (IoT) and automotive networks, in particular.

Note that while this construct may be used to carry EUI-48 or EUI-64 addresses in an Issuer Alternative Name (IAN) extension, there are probably few, if any, reasons to do so.

2. Conventions and Definitions

The key words "**MUST**", "**MUST NOT**", "**REQUIRED**", "**SHALL**", "**SHALL NOT**", "**SHOULD**", "**SHOULD NOT**", "**RECOMMENDED**", "**NOT RECOMMENDED**", "**MAY**", and "**OPTIONAL**" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

3. MACAddress otherName

In this document, "`otherName`", "`OtherName`", and "`GeneralName.otherName`" all refer to a `GeneralName.otherName` field included in a SAN or IAN. The new name form is identified by the `OBJECT IDENTIFIER` (OID) `id-on-MACAddress` (`1.3.6.1.5.5.7.8.12`) and declared below using the `OTHER-NAME` class declaration syntax. The name form has variants to convey an EUI-48 as an `OCTET STRING` consisting of 6 octets, or an EUI-64 as an `OCTET STRING` consisting of 8 octets. Constraints on EUI-48 and EUI-64 values are conveyed as `OCTET STRING`s whose lengths are twice the octet length of the identifiers. The first set of N octets (where N is the length of the address octets) define the bit pattern of the constraint that the address must match, and the second set of N octets defines the bit mask that defines the set of significant bits in the bit pattern.

The following subsections describe how to encode EUI-48 and EUI-64 values and their corresponding constraints.

3.1. Encoding a MACAddress as an Alternative Name

When the name form is included in a SAN or IAN extension as an `OtherName`, the syntax consists of exactly six or eight octets. Values are encoded with the most significant octet encoded first ("big-endian" or "left-to-right" encoding). No text representation is permitted in the certificate, as human-readable forms such as "`00-24-98-7B-19-02`" or "`0024.987B.1902`" are used only in management interfaces. When a device possesses a 48-bit MAC identifier, the Certification

Authority (CA) **MUST** encode it using a 6-octet `OCTET STRING` as the `MACAddress` value. When the device's factory identifier is a 64-bit EUI-64 or when no canonical 48-bit form exists, the CA **MUST** encode it using an 8-octet `OCTET STRING` as the `MACAddress` value.

Example: `00-24-98-7B-19-02` encodes as `OCTET STRING '0024987B1902'H`.

3.2. Encoding a MACAddress Constraint

When the name form is included in the NCE, the syntax consists of an `OCTET STRING` that is twice as long as the `OCTET STRING` representation of the address type being constrained. Within the `OCTET STRING`, two elements are encoded:

1. The first set of N octets (where N is 6 for an EUI-48 constraint or 8 for an EUI-64 constraint) contains the "value bit pattern". This bit pattern encodes the bits that the masked address must contain to be considered a match.
2. The second set of N octets encodes the "mask bit pattern" of the constraint. Each bit that is asserted in the mask bit pattern indicates that the bit in the same position in the address is constrained by the first set of N octets.

For example, a constraint that specifies that the acceptable names must all be within an Organizationally Unique Identifier (OUI) of `'00-00-5e'` for an EUI-48 address would have a value part of `'00005E000000'H`, a mask part of `'FFFFFFFF000000'H`, and would be encoded as `OCTET STRING '00005E000000FFFFFF000000'H`.

The bit patterns encoded in both the value bit pattern and mask bit pattern are encoded with the most significant bit encoded first ("big-endian" or "left-to-right" encoding).

If a bit is not asserted in the mask bit pattern, then the CA **MUST NOT** assert the corresponding bit in the value bit pattern. This rule ensures that a canonical encoding is used for a given mask bit pattern and value bit pattern.

Per Section 4.2.1.10 of [RFC5280], NCE are valid in and "**MUST** be used only in a CA certificate".

3.3. Generation and Validation Rules

The CA **MUST** ensure that `MACAddress` `otherName` values included in certificates that it issues are owned by (or are expected to be owned by) the subject device for the certificate's lifetime. The same MAC address **MUST NOT** be included in certificates issued to different devices, unless different devices share the same Layer 2 interface.

A relying party that matches a presented MAC address to a certificate **SHALL** perform a byte-for-byte comparison of the `OCTET STRING` contents.

Wildcards are not supported.

Self-signed certificates that carry a `MACAddress` `otherName` **MUST** include the address of one of the device's physical ports.

3.4. Path Processing for the Name Constraints Extension

The `MACAddress` `otherName` follows the general rules for `otherName` constraints in [RFC5280], Section 4.2.1.10. An NCE **MAY** impose `permittedSubtrees` and `excludedSubtrees` on `OtherNames` of type `id-on-MACAddress`.

In the pseudocode below, 'mask' is shorthand for the bit string formed from the mask portion of a constraint (e.g., the second set of N octets in the constraint, where N is 6 for an EUI-48 constraint or 8 for an EUI-64 constraint). Similarly, 'value' refers to the bit string formed from the first set of N octets in the constraint.

The declaration 'constraint' used below indicates an `OtherName.MACAddress` constraint value/mask pair -- with fields 'mask', 'value', and 'length'. '.length' as a field returns the byte length of the complete encoded constraint -- either 12 or 16, depending on the type of constraint. The declaration 'name' used below represents an `OtherName.MACAddress` name with fields 'value' and 'length'. The length is either 6 or 8 representing the encoded name's length.

3.4.1. Matching Rules

To determine if a name matches a given constraint, the certificate-consuming application performs the following algorithm:

1. If the name is 6 octets (representing an EUI-48 value) and the constraint is 16 octets (representing an EUI-64 constraint), then the name does not match the constraint.
2. If the name is 8 octets (representing an EUI-64 value) and the constraint is 12 octets (representing an EUI-48 constraint), then the name does not match the constraint.
3. Extract the value bit pattern from the upper (big-endian) N octets of the constraint, where N is "6" for EUI-48 identifiers and "8" for EUI-64 identifiers.
4. Extract the mask bit pattern from the lower (big-endian) N octets of the constraint, where N is "6" for EUI-48 identifiers and "8" for EUI-64 identifiers.
5. Perform an exclusive OR (XOR) operation with the value bit string extracted in step 3 and the octets of the name value.
6. Perform a bitwise AND operation with the bit string calculated in step 5 and the mask bit pattern.
7. If the result of step 6 is a bit string consisting of entirely zeros, then the name matches the constraint. Conversely, if the result of the operation is a bit string with at least one bit asserted, then the name does not match the constraint.

The algorithm can be alternatively expressed as:

```
// Returns true if 'name n' matches 'constraint c'
boolean nameMatchesConstraint (name n, constraint c) {
   return ((2 * n.length) == c.length &&
            ((c.value ^ n.value) &
             c.mask) == 0) ;
}
```

For example, a constraint of `'000000000000 030000000000'H` will be matched by any universal/unicast EUI-48 address such as `00-00-5e-00-50-34`. A constraint of `'00005E000000 FFFFFF000000'H` will be matched by any universal/unicast address with an OUI of `00-00-5E` -- i.e., it will also match `00-00-5e-00-50-34`. Note that `'00-00-5E'` is an OUI controlled by IANA (Section 1.3 of [RFC9542]).

Implementations are not required to implement this algorithm, but they **MUST** calculate an identical result to this algorithm for a given set of inputs.

3.4.2. OtherName.MACAddress Path Validation Processing

This section describes the Path Validation Processing specific to `OtherName.MACAddress` constraints. N.B., it is possible to build hierarchies of NCEs for `OtherName.MACAddress`'s that prohibit all names, even if that was not intended. For example, say that the level 1 NCE contained only a "`permitted_subtrees`" of only (`OtherName.MACAddress`) global/unicast EUI-48, and the level 2 NCE contained only a "`permitted_subtrees`" of "any address" (i.e., the initial constraint set). This would result in an empty `permitted_subtrees` set, as an "any address" constraint is not contained within a "global/unicast" constraint. The worked example is left to the reader.

The following is a utility function used to determine whether or not the set of matching addresses for one `MACAddress` constraint is a subset of the matching addresses for another constraint.

For example, given the following (using the IANA-assigned DOI), 'child' is a constraint wholly contained within 'parent':

```
constraint parent = '000000000000 000000000000'H
constraint child =  '00005E000000 FCFFFF000000'H
```

'child' is a subset of parent because 1) they are the same length (both EUI-48 constraints); and 2) the child.mask ANDed with the parent.mask equals the parent mask; and 3) the bits in the child.value under the parent.mask are set to the same values as the bits in the parent.value under the parent mask.

Note that the child mask allows for any combination of the local/universal and unicast/multicast address bits within the OUI of `00-00-5e`.

If `constraint child2 = '00005E005000 FFFFFFFFFF00'H` and 'child' are compared, 'child2' would be a subset of 'child'. 'child2' uses the same OUI as 'child' but further restricts the matching addresses to universal/unicast by turning on the `'030000000000'H` mask bits and also restricts the range of valid addresses from `00-00-5E-00-50-00` to `00-00-5E-00-50-FF`, i.e., to the 'example' range for the `00-00-5E` OUI.

```
// Both 'child' and 'parent' are OtherName.MACAddress
// constraints.
// Returns true if all addresses that match child also match
// parent; false otherwise.
// Used to calculate INTERSECTION sets for
// OtherName.MACAddress constraints.
boolean childIsSubsetOfParent (constraint c, constraint p) {
  return (
     // if the lengths are the same
     c.length == p.length &&
     // and if there are no bits set in the parent's mask that
     //   are not also set in the child's mask
     // e.g., we can add mask bits to the current set, we cannot
     //   remove them
     (c.mask & p.mask) == p.mask &&
     // and if the child's value has at least all the bits set that
     //   were set (and live) in the parent's value
     // e.g., we can't change the values of the live bits from the
     //   superior constraint
     (c.value & p.mask) == (p.value & p.mask)
    );
}
```

3.4.2.1. Initialization

Per (h) and (i) in Section 6.1.1 of [RFC5280], we need to specify NCE `OtherName.MACAddress` set values for both the initial-permitted-subtrees and for initial-excluded-subtrees. For initial-permitted-subtree, the first constraint is "accept all EUI-48 MACAddresses", and the second constraint is "accept all EUI-64 MACAddresses":

```
initial-permitted-subtrees{} += { 000000000000000000000000H,
                                  00000000000000000000000000000000H }
initial-excluded-subtrees{} += { };
```

3.4.2.2. Intersection Operation

See (g) (1) in Section 6.1.4 of [RFC5280]. As we walk down the tree from the root, the set of `permitted_subtrees` can only stay the same or shrink. At each level, we clear the set of `permitted_subtrees` and for each NCE `OtherName.MACAddress.permitted_subtree` constraint in the certificate, we look to see if there is a `permitted_subtree` constraint at the previous level that equals or encloses this new constraint. If so, we add this new constraint to the current level's set of `permitted_subtrees`. We repeat this going down the tree for the remaining CA certificates.

The intersection of the set of `OtherName.MACAddress` current `permitted_subtrees` with each certificate in the path is as follows:

```
// This logic can be used for both MACAddress and iPAddress
//   OtherName types
// Initialize -
permitted_subtrees{} (0) = initial-permitted-subtrees;

// foreach (certificate i = (1..n) in the path) {
set constraint prevSubtrees{}  =
   { the set of OtherName.MACAddress.permitted_subtrees
     from the permitted_subtree{} (i-1) variable};
constraint tempPermittedSubtrees {} = {};
constraint tempRequestedSubtrees {} =
   { the set of OtherName.MACAddress.permitted_subtrees from
     the Name Constraints extension in the current certificate };

// rst => one of the requested subtrees (from the cert)
// pst -> one of the current permitted subtrees
foreach ( constraint rst in tempRequestedSubtrees) {
    foreach ( constraint pst in prevSubtrees) {
          if (childIsSubsetOfParent (rst,
                                    pst)) {
                tempPermittedSubtrees += rst;
                break;
          }
     }
 }

permitted_subtrees{} (i) = tempPermittedSubtrees;
// } end foreach CA cert on path
```

3.4.2.3. Union Operation

See (g) (2) in Section 6.1.4 of [RFC5280]. Unlike `permitted_subtrees`, which is the intersection of the NCEs at each level, `excluded_subtrees` is the union of all constraints. Starting with an `excluded_subtrees` empty set, at each level add to that set any constraints from the CA certificates that are not already in the set, or that are not covered by a constraint already in the set.

The union of the `excluded_subtrees` set with the `OtherName.MACAddress` `excluded_subtrees` for each certificate in the path is computed as follows:

```
// Initialize
excluded_subtrees{} (0) = initial-excluded-subtrees;

// foreach (certificate i = (1..n) in the path) {
// Since we are doing a union operation we start with
// what was excluded at the previous level and try and
// add to it.
tempExcludedSubtrees {} =
  { the set of OtherName.MACAddress.excluded_subtrees from
    excluded_subtrees (i-1) };
tempRequestedSubtrees {} =
  { the set of OtherName.MACAddress.excluded_subtrees from
    the current certificate };

// note that the ordering of the loop here differs
// from the 'intersection' operation.
foreach (constraint rExcl in tempRequestedSubtrees) {
  boolean matches = false;
  foreach (constraint est in tempExcludedSubtrees) {
      // If I find a constraint in the current excluded
      // constraints that 'covers' the requested subtree,
      // I do not need to add the requested subtree
      // to the set of excluded subtrees.
      if (childIsSubsetOfParent (rExcl, est)) {
        matches = true;
        break;
      }
   }
   if (!matches) {
      tempExcludedSubtrees += rExcl;
   }
}
// } end foreach certificate in the path
excluded_subtrees{} (i) = tempExcludedSubtrees;
```

4. Security Considerations

The binding of a MAC address to a certificate is only as strong as the CA's validation process. CAs **MUST** verify that the subscriber legitimately controls or owns the asserted MAC address. The validation process **MUST** account for the possibility that MAC addresses can be spoofed.

Some systems dynamically assign or share MAC addresses. Such practices can undermine the uniqueness and accountability that this name form aims to provide.

Unlike IP addresses, MAC addresses are not typically routed across Layer 3 boundaries. Relying parties **SHOULD NOT** assume uniqueness beyond their local network unless the relying party has information that addresses are stable across network boundaries.

The Security Considerations section of [RFC5280] applies to this specification as well.

4.1. Privacy Considerations

A MAC address can uniquely identify a physical device and by extension, its user. Certificates that embed unchanging MAC addresses facilitate long-term device tracking. Deployments that use the `MACAddress` name **SHOULD** consider rotating addresses, using short-lived certificates, or employing MAC address randomization where feasible.

5. IANA Considerations

IANA has made the following assignment in the "SMI Security for PKIX Module Identifier" (`1.3.6.1.5.5.7.0`) registry:

Decimal	Description	Reference
126	id-mod-mac-address-other-name-2025	RFC 10031

Table 1

IANA has made the following assignment in the "SMI Security for PKIX Other Name Forms" (`1.3.6.1.5.5.7.8`) registry:

Decimal	Description	Reference
12	id-on-MACAddress	RFC 10031

Table 2

6. ASN.1 Module

This section contains the ASN.1 module for the MAC address; it follows the conventions established by [RFC5912].

```
<CODE BEGINS>
MACAddressOtherName-2025
  { iso(1) identified-organization(3) dod(6) internet(1)
    security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-mac-address-other-name-2025(126) }

DEFINITIONS IMPLICIT TAGS ::=
BEGIN

IMPORTS
  OTHER-NAME FROM PKIX1Implicit-2009
    { iso(1) identified-organization(3) dod(6) internet(1)
      security(5) mechanisms(5) pkix(7) id-mod(0)
      id-mod-pkix1-implicit-02(59) }

 id-pkix FROM PKIX1Explicit-2009
   { iso(1) identified-organization(3) dod(6) internet(1)
     security(5) mechanisms(5) pkix(7) id-mod(0)
     id-mod-pkix1-explicit-02(51) } ;

-- id-pkix 8 is the otherName arc
id-on  OBJECT IDENTIFIER ::= { id-pkix 8 }

-- OID for this name form
id-on-MACAddress OBJECT IDENTIFIER ::= { id-on 12 }

-- Contents of the otherName field
MACAddressOtherNames OTHER-NAME ::= { on-MACAddress, ... }

on-MACAddress OTHER-NAME ::= {
    MACAddress IDENTIFIED BY id-on-MACAddress }

MACAddress ::= OCTET STRING (SIZE (6 | 8 | 12 | 16))

END

<CODE ENDS>
```

7. MAC Address otherName Examples

7.1. EUI-48 Identifier

The following is a human-readable summary of the Subject Alternative Name extension from a certificate containing a single `MACAddress` `otherName` with value `00-24-98-7B-19-02`:

```
  SEQUENCE {
    otherName [0] {
      OBJECT IDENTIFIER id-on-MACAddress
      [0] OCTET STRING '0024987B1902'H
    }
  }
```

7.2. EUI-64 Identifier

An EUI-64 example (`AC-DE-48-00-11-22-33-44`):

```
[0] OCTET STRING 'ACDE480011223344'H
```

7.3. EUI-48 Constraint for Universal, Unicast Addresses

The first octet of a MAC address contains two flag bits. IEEE bit numbering has bit '0' as the least significant bit of the octet because that is the bit transmitted first.

- Individual(I)/Group(G) bit (bit 0 or mask 0x01): 0 = unicast, 1 = multicast. Multicast prefixes are never OUIs.
- Universal(U)/Local(L) bit (bit 1 or mask 0x02): 0 = universal (IEEE-assigned), 1 = local.

These flags let the implementations exclude multicast and local addresses but still cannot prove that a 24-bit value is an IEEE-registered OUI. 36-bit Company IDs (CIDs) share the same first 24 bits, and enterprises **MAY** deploy pseudo-OUIs. CAs **MUST** include only addresses the subscriber legitimately controls (registered OUI or CID). Before issuing a certificate that contains a `MACAddress` or a name constraint based on such a permitted set of addresses, the CA **MUST** verify that control: for example, by consulting the IEEE registry [IEEERA] or reviewing manufacturer documentation.

The following constraint definition constrains EUI-48 values to only those that are universal and unicast; locally assigned or multicast values will not match the constraint.

```
[0] OCTET STRING '000000000000 030000000000'H
```

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, DOI 10.17487/RFC5280, May 2008, <https://www.rfc-editor.org/info/rfc5280>.

[RFC5912] Hoffman, P. and J. Schaad, "New ASN.1 Modules for the Public Key Infrastructure Using X.509 (PKIX)", RFC 5912, DOI 10.17487/RFC5912, June 2010, <https://www.rfc-editor.org/info/rfc5912>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[X680] ITU-T, "Information Technology -- Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, ISO/IEC 8824-1:2021, February 2021, <https://www.itu.int/rec/T-REC-X.680>.

8.2. Informative References

[IEEERA] IEEE Standards Association, "Guidelines for Use of Extended Unique Identifier (EUI), Organizationally Unique Identifier (OUI), and Company ID (CID)", 3 August 2017, <https://standards.ieee.org/wp-content/uploads/import/documents/tutorials/eui.pdf>.

[RFC9542] Eastlake 3rd, D., Abley, J., and Y. Li, "IANA Considerations and IETF Protocol and Documentation Usage for IEEE 802 Parameters", BCP 141, RFC 9542, DOI 10.17487/RFC9542, April 2024, <https://www.rfc-editor.org/info/rfc9542>.

Acknowledgments

We thank the participants on the LAMPS Working Group mailing list for their insightful feedback and comments. In particular, the authors extend sincere appreciation to Bob Beck, David von Oheimb, Deb Cooley, Francois Rousseau, Jacqueline McCall, John Preuß Mattsson, Mahesh Jethanandani, Mohamed Boucadair, Murray Kucherawy, Sean Turner, and Tim Hollebeek for their reviews and suggestions, which greatly improved the quality of this document.

Authors' Addresses

Russ Housley
Vigil Security, LLC
Email: housley@vigilsec.com

Corey Bonnell
TurboLight Solutions, LLC
Email: corey.bonnell@turbolightsolutions.com

Joe Mandel
AKAYLA, Inc.
Email: joe@akayla.com

Tomofumi Okubo
Penguin Securities Pte. Ltd.
Email: tomofumi.okubo+ietf@gmail.com

Michael StJohns
NthPermutation Security LLC
Email: msj@nthpermutation.com