Stream: Internet Engineering Task Force (IETF)
RFC: 8692
Updates: 3279
Category: Standards Track
Published: December 2019
ISSN: 2070-1721
Authors: P. Kampanakis, *Cisco Systems*; Q. Dang, *NIST*

RFC 8692
Internet X.509 Public Key Infrastructure: Additional Algorithm Identifiers for RSASSA-PSS and ECDSA Using SHAKEs

Abstract

Digital signatures are used to sign messages, X.509 certificates, and Certificate Revocation Lists (CRLs). This document updates the "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" (RFC 3279) and describes the conventions for using the SHAKE function family in Internet X.509 certificates and revocation lists as one-way hash functions with the RSA Probabilistic signature and Elliptic Curve Digital Signature Algorithm (ECDSA) signature algorithms. The conventions for the associated subject public keys are also described.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

[RFC3279] defines cryptographic algorithm identifiers for the "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile" [RFC5280]. This document updates RFC 3279 and defines identifiers for several cryptographic algorithms that use variable-length output SHAKE functions introduced in [SHA3] which can be used with RFC 5280.

In the SHA-3 family, two extendable-output functions (SHAKEs) are defined: SHAKE128 and SHAKE256. Four other hash function instances, SHA3-224, SHA3-256, SHA3-384, and SHA3-512, are also defined but are out of scope for this document. A SHAKE is a variable-length hash function defined as SHAKE(M, d) where the output is a d-bits-long digest of message M. The corresponding collision and second-preimage-resistance strengths for SHAKE128 are min(d/2,

128) and min(d, 128) bits, respectively (see Appendix A.1 of [SHA3]). And the corresponding collision and second-preimage-resistance strengths for SHAKE256 are min(d/2, 256) and min(d, 256) bits, respectively.

A SHAKE can be used as the message digest function (to hash the message to be signed) in RSA Probabilistic Signature Scheme (RSASSA-PSS) [RFC8017] and ECDSA [X9.62] and as the hash in the mask generation function (MGF) in RSASSA-PSS.

2. Terminology

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. Identifiers

This section defines four new object identifiers (OIDs), for RSASSA-PSS and ECDSA with each of SHAKE128 and SHAKE256. The same algorithm identifiers can be used for identifying a public key in RSASSA-PSS.

The new identifiers for RSASSA-PSS signatures using SHAKEs are below.

```
  id-RSASSA-PSS-SHAKE128  OBJECT IDENTIFIER  ::=  { iso(1)
            identified-organization(3) dod(6) internet(1)
            security(5) mechanisms(5) pkix(7) algorithms(6)
            30 }

  id-RSASSA-PSS-SHAKE256  OBJECT IDENTIFIER  ::=  { iso(1)
            identified-organization(3) dod(6) internet(1)
            security(5) mechanisms(5) pkix(7) algorithms(6)
            31 }
```

The new algorithm identifiers of ECDSA signatures using SHAKEs are below.

```
  id-ecdsa-with-shake128 OBJECT IDENTIFIER  ::=  { iso(1)
            identified-organization(3) dod(6) internet(1)
            security(5) mechanisms(5) pkix(7) algorithms(6)
            32 }

  id-ecdsa-with-shake256 OBJECT IDENTIFIER  ::=  { iso(1)
            identified-organization(3) dod(6) internet(1)
            security(5) mechanisms(5) pkix(7) algorithms(6)
            33 }
```

The parameters for the four identifiers above **MUST** be absent. That is, the identifier **SHALL** be a SEQUENCE of one component: the OID.

Sections 4.1.1 and 4.1.2 specify the required output length for each use of SHAKE128 or SHAKE256 in RSASSA-PSS and ECDSA. In summary, when hashing messages to be signed, output lengths of SHAKE128 and SHAKE256 are 256 and 512 bits, respectively. When the SHAKEs are used as MGFs in RSASSA-PSS, their output length is (8*ceil((n-1)/8) - 264) or (8*ceil((n-1)/8) - 520) bits, respectively, where n is the RSA modulus size in bits.

4. Use in PKIX

4.1. Signatures

Signatures are used in a number of different ASN.1 structures. As shown in the ASN.1 representation from [RFC5280] below, in an X.509 certificate, a signature is encoded with an algorithm identifier in the signatureAlgorithm attribute and a signatureValue attribute that contains the actual signature.

```
   Certificate  ::=  SEQUENCE  {
      tbsCertificate       TBSCertificate,
      signatureAlgorithm   AlgorithmIdentifier,
      signatureValue       BIT STRING  }
```

The identifiers defined in Section 3 can be used as the AlgorithmIdentifier in the signatureAlgorithm field in the sequence Certificate and the signature field in the sequence TBSCertificate in X.509 [RFC5280]. The parameters of these signature algorithms are absent, as explained in Section 3.

Conforming Certification Authority (CA) implementations **MUST** specify the algorithms explicitly by using the OIDs specified in Section 3 when encoding RSASSA-PSS or ECDSA with SHAKE signatures in certificates and CRLs. Conforming client implementations that process certificates and CRLs using RSASSA-PSS or ECDSA with SHAKE **MUST** recognize the corresponding OIDs. Encoding rules for RSASSA-PSS and ECDSA signature values are specified in [RFC4055] and [RFC5480], respectively.

When using RSASSA-PSS or ECDSA with SHAKEs, the RSA modulus and ECDSA curve order **SHOULD** be chosen in line with the SHAKE output length. Refer to Section 6 for more details.

4.1.1. RSASSA-PSS Signatures

The RSASSA-PSS algorithm is defined in [RFC8017]. When id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 (specified in Section 3) is used, the encoding **MUST** omit the parameters field. That is, the AlgorithmIdentifier **SHALL** be a SEQUENCE of one component: id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256. [RFC4055] defines RSASSA-PSS-params that is used to define the algorithms and inputs to the algorithm. This specification does not use parameters because the hash, mask generation algorithm, trailer, and salt are embedded in the OID definition.

The hash algorithm to hash a message being signed and the hash algorithm used as the MGF in RSASSA-PSS **MUST** be the same: both SHAKE128 or both SHAKE256. The output length of the hash algorithm that hashes the message **SHALL** be 32 bytes (for SHAKE128) or 64 bytes (for SHAKE256).

The MGF takes an octet string of variable length and a desired output length as input and outputs an octet string of the desired length. In RSASSA-PSS with SHAKEs, the SHAKEs **MUST** be used natively as the MGF, instead of the MGF1 algorithm that uses the hash function in multiple iterations, as specified in Appendix B.2.1 of [RFC8017]. In other words, the MGF is defined as the SHAKE128 or SHAKE256 output of the mgfSeed for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256, respectively. The mgfSeed is the seed from which the mask is generated, an octet string [RFC8017]. As explained in Step 9 of Section 9.1.1 of [RFC8017], the output length of the MGF is emLen - hLen - 1 bytes. emLen is the maximum message length ceil((n-1)/8), where n is the RSA modulus in bits. hLen is 32 and 64 bytes for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256, respectively. Thus, when SHAKE is used as the MGF, the SHAKE output length maskLen is (8*emLen - 264) or (8*emLen - 520) bits, respectively. For example, when RSA modulus n is 2048 bits, the output length of SHAKE128 or SHAKE256 as the MGF will be 1784 or 1528 bits when id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 is used, respectively.

The RSASSA-PSS saltLength **MUST** be 32 bytes for id-RSASSA-PSS-SHAKE128 or 64 bytes for id-RSASSA-PSS-SHAKE256. Finally, the trailerField **MUST** be 1, which represents the trailer field with hexadecimal value 0xBC [RFC8017].

4.1.2. ECDSA Signatures

The Elliptic Curve Digital Signature Algorithm (ECDSA) is defined in [X9.62]. When the id-ecdsa-with-shake128 or id-ecdsa-with-shake256 (specified in Section 3) algorithm identifier appears, the respective SHAKE function (SHAKE128 or SHAKE256) is used as the hash. The encoding **MUST** omit the parameters field. That is, the AlgorithmIdentifier **SHALL** be a SEQUENCE of one component: the OID id-ecdsa-with-shake128 or id-ecdsa-with-shake256.

For simplicity and compliance with the ECDSA standard specification [X9.62], the output length of the hash function must be explicitly determined. The output length, d, for SHAKE128 or SHAKE256 used in ECDSA **MUST** be 256 or 512 bits, respectively.

Conforming CA implementations that generate ECDSA with SHAKE signatures in certificates or CRLs **SHOULD** generate such signatures with a deterministically generated, nonrandom k in accordance with all the requirements specified in [RFC6979]. They **MAY** also generate such signatures in accordance with all other recommendations in [X9.62] or [SEC1] if they have a stated policy that requires conformance to those standards. Those standards have not specified SHAKE128 and SHAKE256 as hash algorithm options. However, SHAKE128 and SHAKE256 with output length being 32 and 64 octets, respectively, can be used instead of 256- and 512-bit output hash algorithms such as SHA256 and SHA512.

4.2. Public Keys

Certificates conforming to [RFC5280] can convey a public key for any public key algorithm. The certificate indicates the public key algorithm through an algorithm identifier. This algorithm identifier is an OID with optionally associated parameters. The conventions and encoding for RSASSA-PSS and ECDSA public key algorithm identifiers are as specified in Sections 2.3.1 and 2.3.5 of [RFC3279], Section 3.1 of [RFC4055] and Section 2.1 of [RFC5480].

Traditionally, the rsaEncryption object identifier is used to identify RSA public keys. The rsaEncryption object identifier continues to identify the subject public key when the RSA private key owner does not wish to limit the use of the public key exclusively to RSASSA-PSS with SHAKEs. When the RSA private key owner wishes to limit the use of the public key exclusively to RSASSA-PSS with SHAKEs, the AlgorithmIdentifiers for RSASSA-PSS defined in Section 3 **SHOULD** be used as the algorithm field in the SubjectPublicKeyInfo sequence [RFC5280]. Conforming client implementations that process RSASSA-PSS with SHAKE public keys when processing certificates and CRLs **MUST** recognize the corresponding OIDs.

Conforming CA implementations **MUST** specify the X.509 public key algorithm explicitly by using the OIDs specified in Section 3 when encoding ECDSA with SHAKE public keys in certificates and CRLs. Conforming client implementations that process ECDSA with SHAKE public keys when processing certificates and CRLs **MUST** recognize the corresponding OIDs.

The identifier parameters, as explained in Section 3, **MUST** be absent.

5. IANA Considerations

One object identifier for the ASN.1 module in Appendix A has been assigned in the "SMI Security for PKIX Module Identifier" (1.3.6.1.5.5.7.0) registry:

Decimal	Description	References
94	id-mod-pkix1-shakes-2019	RFC 8692

Table 1

IANA has updated the "SMI Security for PKIX Algorithms" (1.3.6.1.5.5.7.6) registry [SMI-PKIX] with four additional entries:

Decimal	Description	References
30	id-RSASSA-PSS-SHAKE128	RFC 8692
31	id-RSASSA-PSS-SHAKE256	RFC 8692
32	id-ecdsa-with-shake128	RFC 8692
33	id-ecdsa-with-shake256	RFC 8692

Table 2

IANA has updated the "Hash Function Textual Names" registry [Hash-Texts] with two additional entries for SHAKE128 and SHAKE256:

Hash Function Name	OID	Reference
shake128	2.16.840.1.101.3.4.2.11	RFC 8692
shake256	2.16.840.1.101.3.4.2.12	RFC 8692

Table 3

6. Security Considerations

This document updates [RFC3279]. The Security Considerations section of that document applies to this specification as well.

NIST has defined appropriate use of the hash functions in terms of the algorithm strengths and expected time frames for secure use in Special Publications (SPs) [SP800-78-4] and [SP800-107]. These documents can be used as guides to choose appropriate key sizes for various security scenarios.

SHAKE128 with output length of 256 bits offers 128 bits of collision and preimage resistance. Thus, SHAKE128 OIDs in this specification are **RECOMMENDED** with 2048- (112-bit security) or 3072-bit (128-bit security) RSA modulus or curves with group order of 256 bits (128-bit security). SHAKE256 with a 512-bit output length offers 256 bits of collision and preimage resistance. Thus, the SHAKE256 OIDs in this specification are **RECOMMENDED** with 4096-bit RSA modulus or higher or curves with a group order of at least 512 bits, such as the NIST Curve P-521 (256-bit security). Note that we recommended a 4096-bit RSA because we would need a 15360-bit modulus for 256 bits of security, which is impractical for today's technology.

7. References

7.1. 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>.

[RFC3279] Bassham, L., Polk, W., and R. Housley, "Algorithms and Identifiers for the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 3279, DOI 10.17487/RFC3279, April 2002, <https://www.rfc-editor.org/info/rfc3279>.

[RFC4055] Schaad, J., Kaliski, B., and R. Housley, "Additional Algorithms and Identifiers for RSA Cryptography for use in the Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 4055, DOI 10.17487/RFC4055, June 2005, <https://www.rfc-editor.org/info/rfc4055>.

[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>.

[RFC5480] Turner, S., Brown, D., Yiu, K., Housley, R., and T. Polk, "Elliptic Curve Cryptography Subject Public Key Information", RFC 5480, DOI 10.17487/RFC5480, March 2009, <https://www.rfc-editor.org/info/rfc5480>.

[RFC8017] Moriarty, K., Ed., Kaliski, B., Jonsson, J., and A. Rusch, "PKCS #1: RSA Cryptography Specifications Version 2.2", RFC 8017, DOI 10.17487/RFC8017, November 2016, <https://www.rfc-editor.org/info/rfc8017>.

[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>.

[SHA3] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions", DOI 10.6028/NIST.FIPS.202, FIPS PUB 202, August 2015, <https://doi.org/10.6028/NIST.FIPS.202>.

7.2. Informative References

[Hash-Texts] IANA, "Hash Function Textual Names", , <https://www.iana.org/assignments/hash-function-text-names/>.

[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>.

[RFC6979] Pornin, T., "Deterministic Usage of the Digital Signature Algorithm (DSA) and Elliptic Curve Digital Signature Algorithm (ECDSA)", RFC 6979, DOI 10.17487/RFC6979, August 2013, <https://www.rfc-editor.org/info/rfc6979>.

[SEC1] Standards for Efficient Cryptography Group, "SEC 1: Elliptic Curve Cryptography", May 2009, <http://www.secg.org/sec1-v2.pdf>.

[SMI-PKIX] IANA, "SMI Security for PKIX Algorithms", , <https://www.iana.org/assignments/smi-numbers>.

[SP800-107] National Institute of Standards and Technology (NIST), "Recommendation for Applications Using Approved Hash Algorithms", DOI 10.6028/NIST.SP.800-107r1, Revision 1, NIST Special Publication (SP) 800-107, August 2012, <http://dx.doi.org/10.6028/NIST.SP.800-107r1>.

[SP800-78-4] National Institute of Standards and Technology (NIST), "Cryptographic Algorithms and Key Sizes for Personal Identity Verification", DOI 10.6028/NIST.SP.800-78-4, NIST Special Publication (SP) 800-78-4, May 2015, <http://dx.doi.org/10.6028/NIST.SP.800-78-4>.

[X9.62] ANSI, "Public Key Cryptography for the Financial Services Industry: the Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI X9.62, 2005.

Appendix A. ASN.1 Module

This appendix includes the ASN.1 module for SHAKEs in X.509. This module does not come from any previously existing RFC. This module references [RFC5912].

```
PKIXAlgsForSHAKE-2019 { iso(1) identified-organization(3) dod(6)
  internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
  id-mod-pkix1-shakes-2019(94) }

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL;

IMPORTS

-- FROM RFC 5912

PUBLIC-KEY, SIGNATURE-ALGORITHM, DIGEST-ALGORITHM, SMIME-CAPS
FROM AlgorithmInformation-2009
  { iso(1) identified-organization(3) dod(6) internet(1) security(5)
    mechanisms(5) pkix(7) id-mod(0)
    id-mod-algorithmInformation-02(58) }

-- FROM RFC 5912

RSAPublicKey, rsaEncryption, pk-rsa, pk-ec,
CURVE, id-ecPublicKey, ECPoint, ECParameters, ECDSA-Sig-Value
FROM PKIXAlgs-2009 { iso(1) identified-organization(3) dod(6)
     internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
     id-mod-pkix1-algorithms2008-02(56) }
;

--
-- Message Digest Algorithms (mda-)
--
DigestAlgorithms DIGEST-ALGORITHM ::= {
  -- This expands DigestAlgorithms from RFC 5912
  mda-shake128   |
  mda-shake256,
  ...
}

--
-- One-Way Hash Functions
--

-- SHAKE128
mda-shake128 DIGEST-ALGORITHM ::= {
  IDENTIFIER id-shake128  -- with output length 32 bytes.
}
id-shake128 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
                                    us(840) organization(1) gov(101)
                                    csor(3) nistAlgorithm(4)
                                    hashAlgs(2) 11 }

-- SHAKE256
mda-shake256 DIGEST-ALGORITHM ::= {
  IDENTIFIER id-shake256  -- with output length 64 bytes.
}
id-shake256 OBJECT IDENTIFIER ::= { joint-iso-itu-t(2) country(16)
```

```
                                    us(840) organization(1) gov(101)
                                    csor(3) nistAlgorithm(4)
                                    hashAlgs(2) 12 }

  --
  -- Public Key (pk-) Algorithms
  --
  PublicKeys PUBLIC-KEY ::= {
    -- This expands PublicKeys from RFC 5912
    pk-rsaSSA-PSS-SHAKE128 |
    pk-rsaSSA-PSS-SHAKE256,
    ...
  }

  -- The hashAlgorithm is mda-shake128
  -- The maskGenAlgorithm is id-shake128
  -- Mask Gen Algorithm is SHAKE128 with output length
  -- (8*ceil((n-1)/8) - 264) bits, where n is the RSA
  -- modulus in bits.
  -- The saltLength is 32.  The trailerField is 1.
  pk-rsaSSA-PSS-SHAKE128 PUBLIC-KEY ::= {
    IDENTIFIER id-RSASSA-PSS-SHAKE128
    KEY RSAPublicKey
    PARAMS ARE absent
    -- Private key format not in this module --
    CERT-KEY-USAGE { nonRepudiation, digitalSignature,
                     keyCertSign, cRLSign }
  }

  -- The hashAlgorithm is mda-shake256
  -- The maskGenAlgorithm is id-shake256
  -- Mask Gen Algorithm is SHAKE256 with output length
  -- (8*ceil((n-1)/8) - 520)-bits, where n is the RSA
  -- modulus in bits.
  -- The saltLength is 64.  The trailerField is 1.
  pk-rsaSSA-PSS-SHAKE256 PUBLIC-KEY ::= {
    IDENTIFIER id-RSASSA-PSS-SHAKE256
    KEY RSAPublicKey
    PARAMS ARE absent
    -- Private key format not in this module --
    CERT-KEY-USAGE { nonRepudiation, digitalSignature,
                     keyCertSign, cRLSign }
  }

  --
  -- Signature Algorithms (sa-)
  --
  SignatureAlgs SIGNATURE-ALGORITHM ::= {
    -- This expands SignatureAlgorithms from RFC 5912
    sa-rsassapssWithSHAKE128 |
    sa-rsassapssWithSHAKE256 |
    sa-ecdsaWithSHAKE128 |
    sa-ecdsaWithSHAKE256,
    ...
  }

  --
  -- SMIME Capabilities (sa-)
```

```
--
SMimeCaps SMIME-CAPS ::= {
  -- The expands SMimeCaps from RFC 5912
  sa-rsassapssWithSHAKE128.&smimeCaps |
  sa-rsassapssWithSHAKE256.&smimeCaps |
  sa-ecdsaWithSHAKE128.&smimeCaps |
  sa-ecdsaWithSHAKE256.&smimeCaps,
  ...
}

-- RSASSA-PSS with SHAKE128
sa-rsassapssWithSHAKE128 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-RSASSA-PSS-SHAKE128
  PARAMS ARE absent
      -- The hashAlgorithm is mda-shake128
      -- The maskGenAlgorithm is id-shake128
      -- Mask Gen Algorithm is SHAKE128 with output length
      -- (8*ceil((n-1)/8) - 264) bits, where n is the RSA
      -- modulus in bits.
      -- The saltLength is 32. The trailerField is 1
  HASHES { mda-shake128 }
  PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE128 }
  SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE128 }
}
id-RSASSA-PSS-SHAKE128  OBJECT IDENTIFIER  ::=  { iso(1)
        identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) algorithms(6)
        30 }

-- RSASSA-PSS with SHAKE256
sa-rsassapssWithSHAKE256 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-RSASSA-PSS-SHAKE256
  PARAMS ARE absent
      -- The hashAlgorithm is mda-shake256
      -- The maskGenAlgorithm is id-shake256
      -- Mask Gen Algorithm is SHAKE256 with output length
      -- (8*ceil((n-1)/8) - 520)-bits, where n is the
      -- RSA modulus in bits.
      -- The saltLength is 64. The trailerField is 1.
 HASHES { mda-shake256 }
 PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE256 }
 SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE256 }
}
id-RSASSA-PSS-SHAKE256  OBJECT IDENTIFIER  ::=  { iso(1)
        identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) algorithms(6)
        31 }

-- ECDSA with SHAKE128
sa-ecdsaWithSHAKE128 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-ecdsa-with-shake128
  VALUE ECDSA-Sig-Value
  PARAMS ARE absent
  HASHES { mda-shake128 }
  PUBLIC-KEYS { pk-ec }
  SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake128 }
}
id-ecdsa-with-shake128 OBJECT IDENTIFIER  ::=  { iso(1)
```

```
        identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) algorithms(6)
        32 }

-- ECDSA with SHAKE256
sa-ecdsaWithSHAKE256 SIGNATURE-ALGORITHM ::= {
  IDENTIFIER id-ecdsa-with-shake256
  VALUE ECDSA-Sig-Value
  PARAMS ARE absent
  HASHES { mda-shake256 }
  PUBLIC-KEYS { pk-ec }
  SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake256 }
}
id-ecdsa-with-shake256 OBJECT IDENTIFIER  ::=  { iso(1)
        identified-organization(3) dod(6) internet(1)
        security(5) mechanisms(5) pkix(7) algorithms(6)
        33 }

END
```

Acknowledgements

We would like to thank Sean Turner, Jim Schaad, and Eric Rescorla for their valuable contributions to this document.

The authors would like to thank Russ Housley for his guidance and very valuable contributions with the ASN.1 module.

Authors' Addresses

Panos Kampanakis
Cisco Systems
Email: pkampana@cisco.com

Quynh Dang
NIST
100 Bureau Drive, Stop 8930
Gaithersburg, MD 20899-8930
United States of America
Email: quynh.dang@nist.gov