

LAMPS WG  
Internet-Draft  
Intended status: Standards Track  
Expires: April 22, 2019

P. Kampanakis  
Cisco Systems  
Q. Dang  
NIST  
October 19, 2018

Internet X.509 Public Key Infrastructure: Additional Algorithm  
Identifiers for RSASSA-PSS and ECDSA using SHAKEs as Hash Functions  
draft-ietf-lamps-pkix-shake-03

## Abstract

Digital signatures are used to sign messages, X.509 certificates and CRLs (Certificate Revocation Lists). This document describes the conventions for using the SHAKE family of hash functions in the Internet X.509 as one-way hash functions with the RSA Probabilistic Signature Scheme and ECDSA signature algorithms. The conventions for the associated subject public keys are also described.

## 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 <https://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 April 22, 2019.

## Copyright Notice

Copyright (c) 2018 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to [BCP 78](#) and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

Internet-Draft

SHAKE identifiers in X.509

October 2018

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

## Table of Contents

<a href="#">1.</a>	<a href="#">Change Log</a>	<a href="#">2</a>
<a href="#">2.</a>	<a href="#">Introduction</a>	<a href="#">3</a>
<a href="#">3.</a>	<a href="#">Terminology</a>	<a href="#">4</a>
<a href="#">4.</a>	<a href="#">Identifiers</a>	<a href="#">4</a>
<a href="#">5.</a>	<a href="#">Use in PKIX</a>	<a href="#">5</a>
<a href="#">5.1.</a>	<a href="#">Signatures</a>	<a href="#">5</a>
<a href="#">5.1.1.</a>	<a href="#">RSASSA-PSS Signatures</a>	<a href="#">5</a>
<a href="#">5.1.2.</a>	<a href="#">Deterministic ECDSA Signatures</a>	<a href="#">6</a>
<a href="#">5.2.</a>	<a href="#">Public Keys</a>	<a href="#">7</a>
<a href="#">5.2.1.</a>	<a href="#">RSASSA-PSS Public Keys</a>	<a href="#">7</a>
<a href="#">5.2.2.</a>	<a href="#">ECDSA Public Keys</a>	<a href="#">8</a>
<a href="#">6.</a>	<a href="#">IANA Considerations</a>	<a href="#">8</a>
<a href="#">7.</a>	<a href="#">Security Considerations</a>	<a href="#">9</a>
<a href="#">8.</a>	<a href="#">Acknowledgements</a>	<a href="#">9</a>
<a href="#">9.</a>	<a href="#">References</a>	<a href="#">9</a>
<a href="#">9.1.</a>	<a href="#">Normative References</a>	<a href="#">9</a>
<a href="#">9.2.</a>	<a href="#">Informative References</a>	<a href="#">10</a>
<a href="#">Appendix A.</a>	<a href="#">ASN.1 module</a>	<a href="#">11</a>
	<a href="#">Authors' Addresses</a>	<a href="#">16</a>

## [1.](#) Change Log

[ EDNOTE: Remove this section before publication. ]

### o [draft-ietf-lamps-pkix-shake-03](#):

- \* Updates based on suggestions and clarifications by Jim.
- \* Added ASN.1.

### o [draft-ietf-lamps-pkix-shake-02](#):

- \* Significant reorganization of the sections to simplify the introduction, the new OIDs and their use in PKIX.
- \* Added new OIDs for RSASSA-PSS that hardcode hash, salt and MGF, according the WG consensus.

- \* Updated Public Key section to use the new RSASSA-PSS OIDs and clarify the algorithm identifier usage.
- \* Removed the no longer used SHAKE OIDs from [section 3.1](#).

- \* Consolidated subsection for message digest algorithms.
- \* Text fixes.
- o [draft-ietf-lamps-pkix-shake-01](#):
  - \* Changed titles and section names.
  - \* Removed DSA after WG discussions.
  - \* Updated shake OID names and parameters, added MGF1 section.
  - \* Updated RSASSA-PSS section.
  - \* Added Public key algorithm OIDs.
  - \* Populated Introduction and IANA sections.
- o [draft-ietf-lamps-pkix-shake-00](#):
  - \* Initial version

## [2](#). Introduction

This document describes several cryptographic algorithm identifiers for several cryptographic algorithms which use variable length output SHAKE functions introduced in [\[SHA3\]](#) which can be used with the Internet X.509 Certificate and CRL profile [\[RFC5280\]](#).

The SHA-3 family of one-way hash functions is specified in [\[SHA3\]](#). In the SHA-3 family, two extendable-output functions (SHAKEs): SHAKE128 and SHAKE256, are defined. 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. The output length, in bits, of a SHAKE is defined by the d parameter. The corresponding collision and

second preimage resistance strengths for SHAKE128 are  $\min(d/2, 128)$  and  $\min(d, 128)$  bits respectively. 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 RSASSA-PSS and ECDSA and as the hash in the mask generating function in RSASSA-PSS. In [Section 4](#), we define four new OIDs for RSASSA-PSS and ECDSA when SHAKE128 and SHAKE256 are used. The same algorithm identifiers are used for identifying a public key, and identifying a signature.

### [3.](#) Terminology

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 [[RFC2119](#)].

### [4.](#) Identifiers

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

```
id-RSASSA-PSS-SHAKE128 OBJECT IDENTIFIER ::= { TBD }
```

```
id-RSASSA-PSS-SHAKE256 OBJECT IDENTIFIER ::= { TBD }
```

[ EDNOTE: "TBD" will be specified by NIST later. ]

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

```
id-ecdsa-with-shake128 OBJECT IDENTIFIER ::= { joint-iso-ccitt(2)
    country(16) us(840) organization(1) gov(101)
    csor(3) algorithms(4) id-ecdsa-with-shake(3)
    TBD }
```

```
id-ecdsa-with-shake256 OBJECT IDENTIFIER ::= { joint-iso-ccitt(2)
```

```
country(16) us(840) organization(1) gov(101)
csor(3) algorithms(4) id-ecdsa-with-shake(3)
TBD }
```

[ EDNOTE: "TBD" will be specified by NIST later. ]

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

[Section 5.1.1](#) and [Section 5.1.2](#) 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 mask generation functions, their output lengths are  $(n - 264)$  or  $(n - 520)$  bits respectively, where  $n$  is a RSA modulus size in bits.

## [5.](#) Use in PKIX

### [5.1.](#) Signatures

Signatures can be placed in a number of different ASN.1 structures. The top level structure for an X.509 certificate, to illustrate how signatures are frequently encoded with an algorithm identifier and a location for the signature, is

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

The identifiers defined in [Section 4](#) 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](#)].

Conforming CA implementations MUST specify the algorithms explicitly by using the OIDs specified in [Section 4](#) when encoding RSASSA-PSS and ECDSA with SHAKE signatures in certificates and CRLs. Encoding rules

for RSASSA-PSS and ECDSA signature values are specified in [[RFC4055](#)] and [[RFC5480](#)] respectively.

Conforming client implementations that process RSASSA-PSS and ECDSA with SHAKE signatures when processing certificates and CRLs MUST recognize the corresponding OIDs.

#### 5.1.1. RSASSA-PSS Signatures

The RSASSA-PSS algorithm is defined in [[RFC8017](#)]. When id-RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 specified in [Section 4](#) 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.

The hash algorithm to hash a message being signed and the hash algorithm as the mask generation function "MGF(H, emLen - hLen - 1)" [[RFC8017](#)] used in RSASSA-PSS MUST be the same, SHAKE128 or SHAKE256 respectively. The output-length of the hash algorithm which hashes the message SHALL be 32 or 64 bytes respectively.

In RSASSA-PSS, a mask generation function 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 function, instead of the MGF1 algorithm that uses the hash function in multiple iterations as

specified in Section B.2.1 of [[RFC8017](#)]. In other words, the MGF is defined as

SHAKE128(mgfSeed, maskLen)

and

SHAKE256(mgfSeed, maskLen)

respectively for id-RSASSA-PSS-SHAKE128 and id-RSASSA-PSS-SHAKE256. The mgfSeed is the seed from which mask is generated, an octet string. The maskLen for SHAKE128 or SHAKE256 being used as the MGF is  $(n - 264)/8$  or  $(n - 520)/8$  bytes respectively, where  $n$  is the RSA modulus in bits. For example, when RSA modulus  $n$  is 2048, the output length of SHAKE128 or SHAKE256 as the MGF will be 223 or 191 when id-

RSASSA-PSS-SHAKE128 or id-RSASSA-PSS-SHAKE256 is used respectively.

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

#### 5.1.2. Deterministic 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 4) 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, the output size of the hash function must be explicitly determined. The output size, 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 MUST generate such signatures with a deterministically generated, non-random 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 these standards. These standards may have not specified SHAKE128 and SHAKE256 as hash algorithm options. However, SHAKE128 and SHAKE256 with output length being 32 and 64 octets respectively are substitutions for 256 and 512-bit output hash algorithms such as SHA256 and SHA512 used in the standards.

In Section 3.2 "Generation of k" of [RFC6979], HMAC is used to derive the deterministic k. Conforming implementations that generate deterministic ECDSA with SHAKE signatures in X.509 MUST use KMAC with SHAKE128 or KMAC with SHAKE256 as specified in [SP800-185] when SHAKE128 or SHAKE256 is used as the message hashing algorithm, respectively. In this situation, KMAC with SHAKE128 and KMAC with SHAKE256 have 256-bit and 512-bit outputs respectively, and the optional customization bit string S is an empty string.

## [5.2.](#) Public Keys

Certificates conforming to [\[RFC5280\]](#) can convey a public key for any public key algorithm. The certificate indicates the algorithm through an algorithm identifier. This algorithm identifier is an OID and optionally associated parameters.

In the X.509 certificate, the `subjectPublicKeyInfo` field has the `SubjectPublicKeyInfo` type, which has the following ASN.1 syntax:

```
SubjectPublicKeyInfo ::= SEQUENCE {  
    algorithm      AlgorithmIdentifier,  
    subjectPublicKey BIT STRING  
}
```

The fields in `SubjectPublicKeyInfo` have the following meanings:

- o `algorithm` is the algorithm identifier and parameters for the public key.
- o `subjectPublicKey` contains the byte stream of the public key. The algorithms defined in this document always encode the public key as an exact multiple of 8-bits.

Conforming CA implementations MUST specify the algorithms explicitly by using the OIDs specified in [Section 4](#) when encoding RSASSA-PSS and ECDSA with SHAKE public keys in certificates and CRLs. The conventions for RSASSA-PSS and ECDSA public keys algorithm identifiers are as specified in [\[RFC3279\]](#), [\[RFC4055\]](#) and [\[RFC5480\]](#) , but we include them below for convenience.

### [5.2.1.](#) RSASSA-PSS Public Keys

[\[RFC3279\]](#) defines the following OID for RSA `AlgorithmIdentifier` in the `SubjectPublicKeyInfo` with NULL parameters.

```
rsaEncryption OBJECT IDENTIFIER ::= { pkcs-1 1}
```

Additionally, when the RSA private key owner wishes to limit the use



of the public key exclusively to RSASSA-PSS, the AlgorithmIdentifiers for RSASSA-PSS defined in [Section 4](#) can be used as the algorithm field in the SubjectPublicKeyInfo sequence [[RFC5280](#)]. The identifier parameters, as explained in section [Section 4](#), MUST be absent.

Regardless of what public key algorithm identifier is used, the RSA public key, which is composed of a modulus and a public exponent, MUST be encoded using the RSAPublicKey type [[RFC4055](#)]. The output of this encoding is carried in the certificate subjectPublicKey.

```
RSAPublicKey ::= SEQUENCE {  
    modulus INTEGER, -- n  
    publicExponent INTEGER -- e  
}
```

### [5.2.2](#). ECDSA Public Keys

For ECDSA, the public key identifier defined in [[RFC5480](#)] is

```
id-ecPublicKey OBJECT IDENTIFIER ::= {  
    iso(1) member-body(2) us(840) ansi-X9-62(10045) keyType(2) 1 }
```

Additionally, the mandatory EC SubjectPublicKey is defined in [Section 2.1.1](#) and its syntax is in [Section 2.2 of \[RFC5480\]](#). We also include them here for convenience:

The id-ecPublicKey parameters MUST be present and are defined as

```
ECParameters ::= CHOICE {  
    namedCurve      OBJECT IDENTIFIER  
    -- implicitCurve NULL  
    -- specifiedCurve SpecifiedECDomain  
}
```

The ECParameters associated with the ECDSA public key in the signer's certificate SHALL apply to the verification of the signature.

## [6](#). IANA Considerations

[ EDNOTE: Update here only if there are OID allocations by IANA. ]

This document has no IANA actions.

## 7. Security Considerations

The SHAKEs are deterministic functions. Like any other deterministic functions, executing each function with the same input multiple times will produce the same output. Therefore, users should not expect unrelated outputs (with the same or different output lengths) from executing a SHAKE function with the same input multiple times. The shorter one of any 2 outputs produced from a SHAKE with the same input is a prefix of the longer one. It is a similar situation as truncating a 512-bit output of SHA-512 by taking its 256 left-most bits. These 256 left-most bits are a prefix of the 512-bit output.

Implementations must protect the signer's private key. Compromise of the signer's private key permits masquerade.

Implementations must randomly generate one-time values, such as the *k* value when generating a ECDSA signature. In addition, the generation of public/private key pairs relies on random numbers. The use of inadequate pseudo-random number generators (PRNGs) to generate such cryptographic values can result in little or no security. The generation of quality random numbers is difficult. [RFC4086] offers important guidance in this area, and [SP800-90A] series provide acceptable PRNGs.

Implementers should be aware that cryptographic algorithms may become weaker with time. As new cryptanalysis techniques are developed and computing power increases, the work factor or time required to break a particular cryptographic algorithm may decrease. Therefore, cryptographic algorithm implementations should be modular allowing new algorithms to be readily inserted. That is, implementers should be prepared to regularly update the set of algorithms in their implementations.

## 8. Acknowledgements

We would like to thank Sean Turner and Jim Schaad for his valuable contributions to this document.

## 9. References

### 9.1. 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>>.

Internet-Draft

SHAKE identifiers in X.509

October 2018

- [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>>.
- [SHA3] National Institute of Standards and Technology, "SHA-3 Standard – Permutation-Based Hash and Extendable-Output Functions FIPS PUB 202", August 2015, <<https://www.nist.gov/publications/sha-3-standard-permutation-based-hash-and-extendable-output-functions>>.

## [9.2.](#) Informative References

- [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>>.
- [RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", [BCP 106](#), [RFC 4086](#),

DOI 10.17487/RFC4086, June 2005,  
<<https://www.rfc-editor.org/info/rfc4086>>.

- [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>>.

- [SP800-185] National Institute of Standards and Technology, "SHA-3 Derived Functions: cSHAKE, KMAC, TupleHash and ParallelHash. NIST SP 800-185", December 2016,  
<<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-185.pdf>>.

- [SP800-90A] National Institute of Standards and Technology, "Recommendation for Random Number Generation Using Deterministic Random Bit Generators. NIST SP 800-90A", June 2015,  
<<http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-90Ar1.pdf>>.

- [X9.62] American National Standard for Financial Services (ANSI), "X9.62-2005 Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Standard (ECDSA)", November 2005.

## [Appendix A](#). ASN.1 module

This appendix includes the ASN.1 modules for SHAKEs in X.509. This module does not come from any existing RFC.

```
PKIXAlgsForSHAKE-2018 { iso(1) identified-organization(3) dod(6)
    internet(1) security(5) mechanisms(5) pkix(7) id-mod(0)
    id-mod-pkix1-shake-2018(TBD) }
```

DEFINITIONS EXPLICIT TAGS ::=

BEGIN

-- EXPORTS ALL;

IMPORTS

-- FROM [[RFC5912](#)]

PUBLIC-KEY, SIGNATURE-ALGORITHM, DIGEST-ALGORITHM, MAC-ALGORITHM,  
SMIME-CAPS

FROM AlgorithmInformation-2009

{ iso(1) identified-organization(3) dod(6) internet(1) security(5)  
mechanisms(5) pkix(7) id-mod(0)

Kampanakis & Dang

Expires April 22, 2019

[Page 11]

---

Internet-Draft

SHAKE identifiers in X.509

October 2018

id-mod-algorithmInformation-02(58) }

-- FROM [[RFC5912](#)]

id-RSASSA-PSS, RSAPublicKey, rsaEncryption, id-ecPublicKey,  
ECPoint, 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) }

--

-- 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 }

-- SHAKE-256

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 ::= {
    ...,
    pk-rsaSSA-PSS-SHAKE128 |
    pk-rsaSSA-PSS-SHAKE256 |
    pk-ec,
    ...
}

-- From [RFC5912] - Here so it compiles.

pk-rsa PUBLIC-KEY ::= {
    IDENTIFIER rsaEncryption
    KEY RSAPublicKey
    PARAMS TYPE NULL ARE absent
    -- Private key format not in this module --

```

```

CERT-KEY-USAGE {digitalSignature, nonRepudiation,
keyEncipherment, dataEncipherment, keyCertSign, cRLSign}
}

-- The hashAlgorithm is mda-shake128
-- The maskGenAlgorithm is mda-shake128
-- Mask Gen Algorithm is SHAKE128 with output length
-- (n - 264)/8, 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 TYPE NULL ARE absent
    -- Private key format not in this module --
    CERT-KEY-USAGE { nonRepudiation, digitalSignature,
                    keyCertSign, cRLSign }
}

```

```

-- The hashAlgorithm is mda-shake256
-- The maskGenAlgorithm is mda-shake256
-- Mask Gen Algorithm is SHAKE256 with output length
-- (n - 520)/8, 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 TYPE NULL ARE absent
    -- Private key format not in this module --
    CERT-KEY-USAGE { nonRepudiation, digitalSignature,
                    keyCertSign, cRLSign }
}

pk-ec PUBLIC-KEY ::= {
    IDENTIFIER id-ecPublicKey
    KEY ECPoint
    PARAMS TYPE ECPParameters ARE required
    -- Private key format not in this module --
    CERT-KEY-USAGE { digitalSignature, nonRepudiation, keyAgreement,
                    keyCertSign, cRLSign }
}

ECPParameters ::= CHOICE {
    namedCurve      CURVE.&id({NamedCurve})
    -- implicitCurve  NULL
    -- implicitCurve MUST NOT be used in PKIX
    -- specifiedCurve SpecifiedCurve
    -- specifiedCurve MUST NOT be used in PKIX

```

```

-- Details for specifiedCurve can be found in [X9.62]
-- Any future additions to this CHOICE should be coordinated
-- with ANSI X.9.
}

--
-- Signature Algorithms (sa-)
--
SignatureAlgs SIGNATURE-ALGORITHM ::= {
    ...,
    -- This expands SignatureAlgorithms from [RFC5912]

```

```

    sa-rsaspssWithSHAKE128 |
    sa-rsaspssWithSHAKE256 |
    sa-ecdsaWithSHAKE128 |
    sa-ecdsaWithSHAKE256
}

--
-- SMIME Capabilities (sa-)
--
SMimeCaps SMIME-CAPS ::= {
    ...,
    -- The expands SMimeCaps from [RFC5912]
    sa-rsaspssWithSHAKE128.&smimeCaps |
    sa-rsaspssWithSHAKE256.&smimeCaps |
    sa-ecdsaWithSHAKE128.&smimeCaps |
    sa-ecdsaWithSHAKE256.&smimeCaps
}

-- RSASSA-PSS with SHAKE128
sa-rsaspssWithSHAKE128 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-RSASSA-PSS-SHAKE128
    PARAMS TYPE NULL ARE absent
        -- The hashAlgorithm is mda-shake128
        -- The maskGenAlgorithm is mda-shake128
        -- Mask Gen Algorithm is SHAKE128 with output length
        -- (n - 264)/8, where n is the RSA modulus in bits.
        -- the saltLength is 32
        -- the trailerField is 1
        HASHES {mda-shake128} -- omitting mda-shake128-params
    PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE128 }
    SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE128 }
}
id-RSASSA-PSS-SHAKE128 OBJECT IDENTIFIER ::= { TBD }

-- RSASSA-PSS with SHAKE256
sa-rsaspssWithSHAKE256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-RSASSA-PSS-SHAKE256

```

```

PARAMS TYPE NULL ARE absent
    -- The hashAlgorithm is mda-shake256
    -- The maskGenAlgorithm is mda-shake256
    -- Mask Gen Algorithm is SHAKE256 with output length

```



```

-- (n - 520)/8, where n is the RSA modulus in bits.
-- the saltLength is 64
-- the trailerField is 1
    HASHES {mda-shake256} -- omitting mda-shake256-params
PUBLIC-KEYS { pk-rsa | pk-rsaSSA-PSS-SHAKE256 }
SMIME-CAPS { IDENTIFIED BY id-RSASSA-PSS-SHAKE256 }
}
id-RSASSA-PSS-SHAKE256 OBJECT IDENTIFIER ::= { TBD }

-- Deterministic ECDSA with SHAKE128
-- Generating k by using KMAC with SHAKE128 as the hash
-- [SP800-185] instead of HMAC with output length 256-bits
-- that is equal to or slightly less than the elliptic
-- curve group order. S is set to an empty string.
sa-ecdsaWithSHAKE128 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-ecdsa-with-shake128
    VALUE ECDSA-Sig-Value
    PARAMS TYPE NULL ARE absent
    HASHES { mda-shake128 }
    PUBLIC-KEYS { pk-ec }
    SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake128 }
}
id-ecdsa-with-shake128 ::= { joint-iso-itu-t(2) country(16)
                                us(840) organization(1) gov(101)
                                csor(3) nistAlgorithm(4)
                                sigAlgs(3) TBD }

-- Deterministic ECDSA with SHAKE256
-- Generating k by using KMAC with SHAKE256 as the hash
-- [SP800-185] instead of HMAC with output length 512-bits
-- truncated to equal to or slightly less than the elliptic
-- curve group order. S is set to an empty string.
sa-ecdsaWithSHAKE256 SIGNATURE-ALGORITHM ::= {
    IDENTIFIER id-ecdsa-with-shake256
    VALUE ECDSA-Sig-Value
    PARAMS TYPE NULL ARE absent
    HASHES { mda-shake256 }
    PUBLIC-KEYS { pk-ec }
    SMIME-CAPS { IDENTIFIED BY id-ecdsa-with-shake256 }
}
id-ecdsa-with-shake256 ::= { joint-iso-itu-t(2) country(16)
                                us(840) organization(1) gov(101)
                                csor(3) nistAlgorithm(4)
                                sigAlgs(3) TBD }

```

END

Authors' Addresses

Panos Kampanakis  
Cisco Systems

Email: [pkampana@cisco.com](mailto:pkampana@cisco.com)

Quynh Dang  
NIST  
100 Bureau Drive, Stop 8930  
Gaithersburg, MD 20899-8930  
USA

Email: [quynh.dang@nist.gov](mailto:quynh.dang@nist.gov)

