PKINIT Algorithm Agility
draft-ietf-kitten-pkinit-alg-agility-06

Abstract

This document updates the Public Key Cryptography for Initial Authentication in Kerberos standard (PKINIT) [RFC4556], to remove protocol structures tied to specific cryptographic algorithms. The PKINIT key derivation function is made negotiable, and the digest algorithms for signing the pre-authentication data and the client’s X.509 certificates are made discoverable.

These changes provide preemptive protection against vulnerabilities discovered in the future against any specific cryptographic algorithm, and allow incremental deployment of newer algorithms.

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 September 7, 2019.
Copyright Notice

Copyright (c) 2019 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 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

1. Introduction .................................................. 3
2. Requirements Notation ........................................ 4
3. paChecksum Agility .......................................... 4
4. CMS Digest Algorithm Agility ................................ 4
5. X.509 Certificate Signer Algorithm Agility ................. 5
6. KDF agility .................................................. 6
7. Interoperability ............................................. 11
8. Test vectors .................................................. 12
   8.1. Common Inputs ......................................... 12
   8.2. Test Vector for SHA-1, enctype 18 .................... 12
       8.2.1. Specific Inputs .................................. 12
       8.2.2. Outputs .......................................... 12
   8.3. Test Vector for SHA-256, enctype ...................... 13
       8.3.1. Specific Inputs .................................. 13
       8.3.2. Outputs .......................................... 13
   8.4. Test Vector for SHA-512, enctype ...................... 13
       8.4.1. Specific Inputs .................................. 13
       8.4.2. Outputs .......................................... 13
9. Security Considerations ..................................... 14
1. Introduction

The Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) standard [RFC4556] defines several protocol structures that are either tied to SHA-1 [RFC6234], or do not support negotiation or discovery, but are instead based on local policy:

- The checksum algorithm in the authentication request is hardwired to use SHA-1.
- The acceptable digest algorithms for signing the authentication data are not discoverable.
- The key derivation function in Section 3.2.3.1 of [RFC4556] is hardwired to use SHA-1.
- The acceptable digest algorithms for signing the client X.509 certificates are not discoverable.

In August 2004, Xiaoyun Wang’s research group reported MD4 [RFC6150] collisions generated using hand calculation [WANG04], alongside attacks on later hash function designs in the MD4, MD5 [RFC1321] and SHA [RFC6234] family. These attacks and their consequences are discussed in [RFC6194]. These discoveries challenged the security of protocols relying on the collision resistance properties of these hashes.

The Internet Engineering Task Force (IETF) called for actions to update existing protocols to provide crypto algorithm agility so that protocols support multiple cryptographic algorithms (including hash functions) and provide clean, tested transition strategies between algorithms, as recommended by BCP 201 [RFC7696].

To address these concerns, new key derivation functions (KDFs), identified by object identifiers, are defined. The PKINIT client provides a list of KDFs in the request and the Key Distribution Center (KDC) picks one in the response, thus a mutually-supported KDF is negotiated.
Furthermore, structures are defined to allow the client to discover the Cryptographic Message Syntax (CMS) [RFC5652] digest algorithms supported by the KDC for signing the pre-authentication data and signing the client X.509 certificate.

2. Requirements Notation

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. paChecksum Agility

The paChecksum defined in Section 3.2.1 of [RFC4556] provides a cryptographic binding between the client’s pre-authentication data and the corresponding Kerberos request body. This also prevents the KDC-REQ body from being tampered with. SHA-1 is the only allowed checksum algorithm defined in [RFC4556]. This facility relies on the collision resistance properties of the SHA-1 checksum [RFC6234].

When the reply key delivery mechanism is based on public key encryption as described in Section 3.2.3.2 of [RFC4556], the asChecksum in the KDC reply provides the binding between the pre-authentication and the ticket request and response messages, and integrity protection for the unauthenticated clear text in these messages. However, if the reply key delivery mechanism is based on the Diffie-Hellman key agreement as described in Section 3.2.3.1 of [RFC4556], the security provided by using SHA-1 in the paChecksum is weak, and nothing else cryptographically binds the AS request to the ticket response. In this case, the new KDF selected by the KDC as described in Section 6 provides the cryptographic binding and integrity protection.

4. CMS Digest Algorithm Agility

Section 3.2.2 of [RFC4556] is updated to add optional typed data to the KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error. When a KDC implementation conforming to this specification returns this error code, it MAY include in a list of supported CMS types signifying the digest algorithms supported by the KDC, in the decreasing preference order. This is accomplished by including a TD_CMS_DATA_DIGEST_ALGORITHMS typed data element in the error data.

\[
\text{td-cms-digest-algorithms INTEGER ::= 111}
\]
The corresponding data for the TD_CMS_DATA_DIGEST_ALGORITHMS contains the ASN.1 Distinguished Encoding Rules (DER) [X680] [X690] encoded TD-CMS-DIGEST-ALGORITHMS-DATA structure defined as follows:

TD-CMS-DIGEST-ALGORITHMS-DATA ::= SEQUENCE OF
   AlgorithmIdentifier
   -- Contains the list of CMS algorithm [RFC5652]
   -- identifiers indicating the digest algorithms
   -- acceptable to the KDC for signing CMS data in
   -- the order of decreasing preference.

The algorithm identifiers in the TD-CMS-DIGEST-ALGORITHMS identify digest algorithms supported by the KDC.

This information sent by the KDC via TD_CMS_DATA_DIGEST_ALGORITHMS can facilitate trouble-shooting when none of the digest algorithms supported by the client is supported by the KDC.

5. X.509 Certificate Signer Algorithm Agility

Section 3.2.2 of [RFC4556] is updated to add optional typed data to the KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED error. When a KDC conforming to this specification returns this error, it MAY send a list of digest algorithms acceptable to the KDC for use by the Certificate Authority (CA) in signing the client’s X.509 certificate, in the decreasing preference order. This is accomplished by including a TD_CERT_DIGEST_ALGORITHMS typed data element in the error data. The corresponding data contains the ASN.1 DER encoding of the structure TD-CERT-DIGEST-ALGORITHMS-DATA defined as follows:
td-cert-digest-algorithms INTEGER ::= 112

TD-CERT-DIGEST-ALGORITHMS-DATA ::= SEQUENCE {
  allowedAlgorithms [0] SEQUENCE OF AlgorithmIdentifier,
    -- Contains the list of CMS algorithm [RFC5652]
    -- identifiers indicating the digest algorithms
    -- that are used by the CA to sign the client’s
    -- X.509 certificate and are acceptable to the KDC
    -- in the process of validating the client’s X.509
    -- certificate, in the order of decreasing
    -- preference.
  rejectedAlgorithm [1] AlgorithmIdentifier OPTIONAL,
    -- This identifies the digest algorithm that was
    -- used to sign the client’s X.509 certificate and
    -- has been rejected by the KDC in the process of
    -- validating the client’s X.509 certificate
    -- [RFC5280].
    ...
}

The KDC fills in the allowedAlgorithm field with the list of
algorithm [RFC5652] identifiers indicating digest algorithms that are
used by the CA to sign the client’s X.509 certificate and are
acceptable to the KDC in the process of validating the client’s X.509
certificate, in the order of decreasing preference. The
rejectedAlgorithm field identifies the signing algorithm for use in
signing the client’s X.509 certificate that has been rejected by the
KDC in the process of validating the client’s certificate [RFC5280].

6. KDF agility

Section 3.2.3.1 of [RFC4556] is updated to define additional Key
Derivation Functions (KDFs) to derive a Kerberos protocol key based
on the secret value generated by the Diffie-Hellman key exchange.
Section 3.2.1 of [RFC4556] is updated to add a new field to the
AuthPack structure to indicate which new KDFs are supported by the
client. Section 3.2.3 of [RFC4556] is updated to add a new field to
the DHRrepInfo structure to indicate which KDF is selected by the KDC.

The KDF algorithm described in this document (based on [SP80056A])
can be implemented using any cryptographic hash function.

A new KDF for PKINIT usage is identified by an object identifier.
The following KDF object identifiers are defined:
id-pkinit OBJECT IDENTIFIER ::= 
    { iso(1) identified-organization(3) dod(6) internet(1) 
      security(5) kerberosv5(2) pkinit (3) }
    -- Defined in RFC 4556 and quoted here for the reader.

id-pkinit-kdf OBJECT IDENTIFIER ::= { id-pkinit kdf(6) }
    -- PKINIT KDFs

id-pkinit-kdf-ah-sha1 OBJECT IDENTIFIER ::= { id-pkinit-kdf sha1(1) }
    -- SP800-56A ASN.1 structured hash-based KDF using SHA-1

id-pkinit-kdf-ah-sha256 OBJECT IDENTIFIER ::= { id-pkinit-kdf sha256(2) }
    -- SP800-56A ASN.1 structured hash-based KDF using SHA-256

id-pkinit-kdf-ah-sha512 OBJECT IDENTIFIER ::= { id-pkinit-kdf sha512(3) }
    -- SP800-56A ASN.1 structured hash-based KDF using SHA-512

id-pkinit-kdf-ah-sha384 OBJECT IDENTIFIER ::= { id-pkinit-kdf sha384(4) }
    -- SP800-56A ASN.1 structured hash-based KDF using SHA-384

Where id-pkinit is defined in [RFC4556]. All key derivation 
functions specified above use the one-step key derivation method 
described in Section 5.8.2.1 of [SP80056A], using the ASN.1 format 
for FixedInfo, and Section 4.1 of [SP80056C], using option 1 for the 
auxiliary function H. id-pkinit-kdf-ah-sha1 uses SHA-1 [RFC6234] as 
the hash function. id-pkinit-kdf-ah-sha256, id-pkinit-kdf-ah-sha356, 
and id-pkinit-kdf-ah-sha512 use SHA-256 [RFC6234], SHA-384 ([RFC6234] 
and SHA-512 [RFC6234] respectively.

To name the input parameters, an abbreviated version of the key 
derivation method is described below.

1. reps = ceiling(L/H_outputBits)

2. Initialize a 32-bit, big-endian bit string counter as 1.

3. For i = 1 to reps by 1, do the following:
   1. Compute Hashi = H(counter || Z || OtherInfo).
   2. Increment counter (not to exceed 2^32-1)
4. Set key_material = Hash1 || Hash2 || ... so that the length of key_material is L bits, truncating the last block as necessary.

5. The above KDF produces a bit string of length L in bits as the keying material. The AS reply key is the output of random-to-key() [RFC3961] using that keying material as the input.

The input parameters for these KDFs are provided as follows:

- **H_outputBits** is 160 bits for id-pkinit-kdf-ah-sha1, 256 bits for id-pkinit-kdf-ah-sha256, 384 bits for id-pkinit-kdf-ah-sha384, and 512 bits for id-pkinit-kdf-ah-sha512.

- **max_H_inputBits** is $2^{64}$.

- The secret value (Z) is the shared secret value generated by the Diffie-Hellman exchange. The Diffie-Hellman shared value is first padded with leading zeros such that the size of the secret value in octets is the same as that of the modulus, then represented as a string of octets in big-endian order.

- The key data length (L) is the key-generation seed length in bits [RFC3961] for the Authentication Service (AS) reply key. The enctype of the AS reply key is selected according to [RFC4120].

- The algorithm identifier (algorithmID) input parameter is the identifier of the respective KDF. For example, this is id-pkinit-kdf-ah-sha1 if the KDF uses SHA-1 as the hash.

- The initiator identifier (partyUInfo) contains the ASN.1 DER encoding of the KRB5PrincipalName [RFC4556] that identifies the client as specified in the AS-REQ [RFC4120] in the request.

- The recipient identifier (partyVInfo) contains the ASN.1 DER encoding of the KRB5PrincipalName [RFC4556] that identifies the TGS as specified in the AS-REQ [RFC4120] in the request.

- The supplemental public information (suppPubInfo) is the ASN.1 DER encoding of the structure PkinitSuppPubInfo as defined later in this section.

- The supplemental private information (suppPrivInfo) is absent.

OtherInfo is the ASN.1 DER encoding of the following sequence:
OtherInfo ::= SEQUENCE {
    algorithmID   AlgorithmIdentifier,
    partyUInfo    [0] OCTET STRING,
    partyVInfo    [1] OCTET STRING,
    suppPubInfo   [2] OCTET STRING OPTIONAL,
    suppPrivInfo  [3] OCTET STRING OPTIONAL
}

The structure PkinitSuppPubInfo is defined as follows:

PkinitSuppPubInfo ::= SEQUENCE {
    enctype   [0] Int32,
        -- The enctype of the AS reply key.
    as-REQ    [1] OCTET STRING,
        -- The DER encoding of the AS-REQ [RFC4120] from the
        -- client.
    pk-as-rep [2] OCTET STRING,
        -- The DER encoding of the PA-PK-AS-REP [RFC4556] in the
        -- KDC reply.
    ...
}

The PkinitSuppPubInfo structure contains mutually-known public
information specific to the authentication exchange. The enctype
field is the enctype of the AS reply key as selected according to
[RFC4120]. The as-REQ field contains the DER encoding of the type
AS-REQ [RFC4120] in the request sent from the client to the KDC.
Note that the as-REQ field does not include the wrapping 4 octet
length field when TCP is used. The pk-as-rep field contains the DER
encoding of the type PA-PK-AS-REP [RFC4556] in the KDC reply. The
PkinitSuppPubInfo provides a cryptographic bindings between the pre-
authentication data and the corresponding ticket request and
response, thus addressing the concerns described in Section 3.

The KDF is negotiated between the client and the KDC. The client
sends an unordered set of supported KDFs in the request, and the KDC
picks one from the set in the reply.

To accomplish this, the AuthPack structure in [RFC4556] is extended
as follows:
AuthPack ::= SEQUENCE {
    pkAuthenticator   [0]  PKAuthenticator,
    clientPublicValue [1]  SubjectPublicKeyInfo OPTIONAL,
    supportedCMSTypes [2]  SEQUENCE OF AlgorithmIdentifier
        OPTIONAL,
    clientDHNonce     [3]  DHNonce OPTIONAL,
    ...,
    supportedKDFs     [4]  SEQUENCE OF KDFAlgorithmId OPTIONAL,
        -- Contains an unordered set of KDFs supported by the
        -- client.
    ...}

KDFAlgorithmId ::= SEQUENCE {
    kdf-id            [0]  OBJECT IDENTIFIER,
        -- The object identifier of the KDF
    ...
}

The new field supportedKDFs contains an unordered set of KDFs supported by the client.

The KDFAlgorithmId structure contains an object identifier that identifies a KDF. The algorithm of the KDF and its parameters are defined by the corresponding specification of that KDF.

The DHRepInfo structure in [RFC4556] is extended as follows:

DHRepInfo ::= SEQUENCE {
    dhSignedData         [0]  IMPLICIT OCTET STRING,
    serverDHNonce        [1]  DHNonce OPTIONAL,
    ...,
    kdf                  [2]  KDFAlgorithmId OPTIONAL,
        -- The KDF picked by the KDC.
    ...
}

The new field kdf in the extended DHRepInfo structure identifies the KDF picked by the KDC. If the supportedKDFs field is present in the request, a KDC conforming to this specification MUST choose one of the KDFs supported by the client and indicate its selection in the kdf field in the reply. If the supportedKDFs field is absent in the request, the KDC MUST omit the kdf field in the reply and use the key derivation function from Section 3.2.3.1 of [RFC4556]. If none of the KDFs supported by the client is acceptable to the KDC, the KDC MUST reply with the new error code KDC_ERR_NO_ACCEPTABLE_KDF:
7. Interoperability

An old client interoperating with a new KDC will not recognize a TD-CMS-DIGEST-ALGORITHMS-DATA element in a KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error, or a TD-CERT-DIGEST-ALGORITHMS-DATA element in a KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED error. Because the error data is encoded as typed data, the client will ignore the unrecognized elements.

An old KDC interoperating with a new client will not include a TD-CMS-DIGEST-ALGORITHMS-DATA element in a KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error, or a TD-CERT-DIGEST-ALGORITHMS-DATA element in a KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED error. To the client this appears just as if a new KDC elected not to include a list of digest algorithms.

An old client interoperating with a new KDC will not include the supportedKDFs field in the request. The KDC MUST omit the kdf field in the reply and use the [RFC4556] KDF as expected by the client, or reject the request if local policy forbids use of the old KDF.

A new client interoperating with an old KDC will include the supportedKDFs field in the request; this field will be ignored as an unknown extension by the KDC. The KDC will omit the kdf field in the reply and will use the [RFC4556] KDF. The client can deduce from the omitted kdf field that the KDC is not updated to conform to this specification, or that the exchange was subjected to a downgrade attack. The client MUST use the [RFC4556] KDF, or reject the reply if local policy forbids the use of the old KDF.
8. Test vectors

This section contains test vectors for the KDF defined above.

8.1. Common Inputs

Z: Length = 256 bytes, Hex Representation = (All Zeros)
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000

client: Length = 9 bytes, ASCII Representation = lha@SU.SE
server: Length = 18 bytes, ASCII Representation = krbtgt/SU.SE@SU.SE
as-req: Length = 10 bytes, Hex Representation = 
aaaaa aaaa aaaa aaaa

pk-as-rep: Length = 9 bytes, Hex Representation = 
bbbbbbbb bbbbbbbb bb

ticket: Length = 55 bytes, Hex Representation = 
61353033 A0030201 05A1071B 0553552E 5345A210 300EA003 020101A1 0730051B 036C6861 A311300F A0030201 12A20804 0668656A 68656A

8.2. Test Vector for SHA-1, enctype 18

8.2.1. Specific Inputs

algorithm-id: (id-pkinit-kdf-ah-sha1) Length = 8 bytes, Hex Representation = 2B060105 02030601

encetype: (aes256-cts-hmac-sha1-96) Length = 1 byte, Decimal Representation = 18

8.2.2. Outputs
8.3. Test Vector for SHA-256, enctype

8.3.1. Specific Inputs

algorithm-id: (id-pkinit-kdf-ah-sha256) Length = 8 bytes, Hex Representation = 2B06010502030602

enctype: (aes256-cts-hmac-sha1-96) Length = 1 byte, Decimal Representation = 18

8.3.2. Outputs

key-material: Length = 32 bytes, Hex Representation = 77EF4E48C420AE3FEC75109D7981697EED5D295C90C62564F7BFD101FA9bC1D5

key: Length = 32 bytes, Hex Representation = 77EF4E48C420AE3FEC75109D7981697EED5D295C90C62564F7BFD101FA9bC1D5

8.4. Test Vector for SHA-512, enctype

8.4.1. Specific Inputs

algorithm-id: (id-pkinit-kdf-ah-sha512) Length = 8 bytes, Hex Representation = 2B06010502030603

enctype: (des3-cbc-sha1-kd) Length = 1 byte, Decimal Representation = 16

8.4.2. Outputs

key-material: Length = 24 bytes, Hex Representation = D3C78A79D65213EF9A826F75DFB01F72362FB16FB01DAD6

key: Length = 32 bytes, Hex Representation = D3C78A79D65213EF9A826F75DFB01F72362FB16FB01DAD6
9. Security Considerations

This document describes negotiation of checksum types, key derivation functions and other cryptographic functions. If a given negotiation is unauthenticated, care must be taken to accept only secure values; to do otherwise allows an active attacker to perform a downgrade attack.

10. Acknowledgements

Jeffery Hutzelman, Shawn Emery, Tim Polk, Kelley Burgin, Ben Kaduk, and Scott Bradner reviewed the document and provided suggestions for improvements.

11. IANA Considerations

IANA is requested to update the following registrations in the Kerberos Pre-authentication and Typed Data Registry created by section 7.1 of RFC 6113 to refer to this specification. These values were reserved for this specification in the initial registrations.

TD-CMS-DIGEST-ALGORITHMS 111 [ALG-AGILITY]
TD-CERT-DIGEST-ALGORITHMS 112 [ALG-AGILITY]

12. References

12.1. Normative References


12.2. Informative References


Appendix A. PKINIT ASN.1 Module

KerberosV5-PK-INIT-Agility-SPEC {
  iso(1) identified-organization(3) dod(6) internet(1)
  security(5) kerberosV5(2) modules(4) pkinit(5) agility (1)
} DEFINITIONS EXPLICIT TAGS ::= BEGIN

IMPORTS
  AlgorithmIdentifier, SubjectPublicKeyInfo
    FROM PKIX1Explicit88 { iso (1)
      identified-organization (3) dod (6) internet (1)
      security (5) mechanisms (5) pkix (7) id-mod (0)
      id-pkix1-explicit (18) }
    -- As defined in RFC 5280.

  Ticket, Int32, Realm, EncryptionKey, Checksum
    FROM KerberosV5Spec2 { iso(1) identified-organization(3)
      dod(6) internet(1) security(5) kerberosV5(2)
      modules(4) krb5spec2(2) }
    -- as defined in RFC 4120.

  PKAuthenticator, DHNonce, id-pkinit
    FROM KerberosV5-PK-INIT-SPEC {
      iso(1) identified-organization(3) dod(6) internet(1)
      security(5) kerberosV5(2) modules(4) pkinit(5) }
    -- as defined in RFC 4556.

  id-pkinit-kdf OBJECT IDENTIFIER ::= { id-pkinit kdf(6) }
    -- PKINIT KDFs
id-pkinit-kdf-ah-sha1 OBJECT IDENTIFIER
  ::= { id-pkinit-kdf sha1(1) }
  -- SP800-56A ASN.1 structured hash-based KDF using SHA-1

id-pkinit-kdf-ah-sha256 OBJECT IDENTIFIER
  ::= { id-pkinit-kdf sha256(2) }
  -- SP800-56A ASN.1 structured hash-based KDF using SHA-256

id-pkinit-kdf-ah-sha512 OBJECT IDENTIFIER
  ::= { id-pkinit-kdf sha512(3) }
  -- SP800-56A ASN.1 structured hash-based KDF using SHA-512

id-pkinit-kdf-ah-sha384 OBJECT IDENTIFIER
  ::= { id-pkinit-kdf sha384(4) }
  -- SP800-56A ASN.1 structured hash-based KDF using SHA-384

TD-CMS-DIGEST-ALGORITHMS-DATA ::= SEQUENCE OF
  AlgorithmIdentifier
  -- Contains the list of CMS algorithm [RFC5652]
  -- identifiers indicating the digest algorithms
  -- acceptable to the KDC for signing CMS data in
  -- the order of decreasing preference.

TD-CERT-DIGEST-ALGORITHMS-DATA ::= SEQUENCE {
  allowedAlgorithms [0] SEQUENCE OF AlgorithmIdentifier,
    -- Contains the list of CMS algorithm [RFC5652]
    -- identifiers indicating the digest algorithms
    -- that are used by the CA to sign the client’s
    -- X.509 certificate and are acceptable to the KDC
    -- in the process of validating the client’s X.509
    -- certificate, in the order of decreasing
    -- preference.
  rejectedAlgorithm [1] AlgorithmIdentifier OPTIONAL,
    -- This identifies the digest algorithm that was
    -- used to sign the client’s X.509 certificate and
    -- has been rejected by the KDC in the process of
    -- validating the client’s X.509 certificate
    -- [RFC5280].
...
}

OtherInfo ::= SEQUENCE {
  algorithmID   AlgorithmIdentifier,
  partyUInfo     [0] OCTET STRING,
  partyVInfo     [1] OCTET STRING,
  suppPubInfo    [2] OCTET STRING OPTIONAL,
  suppPrivInfo   [3] OCTET STRING OPTIONAL
}
PkinitSuppPubInfo ::= SEQUENCE {
  enctype [0] Int32,
     -- The enctype of the AS reply key.
  as-REQ [1] OCTET STRING,
     -- The DER encoding of the AS-REQ [RFC4120] from the
     -- client.
  pk-as-rep [2] OCTET STRING,
     -- The DER encoding of the PA-PK-AS-REP [RFC4556] in the
     -- KDC reply.
  ...
}

AuthPack ::= SEQUENCE {
  pkAuthenticator [0] PKAuthenticator,
  clientPublicKey [1] SubjectPublicKeyInfo OPTIONAL,
  supportedCMSTypes [2] SEQUENCE OF AlgorithmIdentifier
     OPTIONAL,
  clientDHNonce [3] DHNonce OPTIONAL,
  ...
  supportedKDFs [4] SEQUENCE OF KDFAlgorithmId OPTIONAL,
     -- Contains an unordered set of KDFs supported by the
     -- client.
  ...
}

KDFAlgorithmId ::= SEQUENCE {
  kdf-id [0] OBJECT IDENTIFIER,
     -- The object identifier of the KDF
  ...
}

DHRepInfo ::= SEQUENCE {
  dhSignedData [0] IMPLICIT OCTET STRING,
  serverDHNonce [1] DHNonce OPTIONAL,
  ...
  kdf [2] KDFAlgorithmId OPTIONAL,
     -- The KDF picked by the KDC.
  ...
}

END

Authors’ Addresses
Love Hornquist Astrand
Apple, Inc
Cupertino, CA
USA

Email: lha@apple.com

Larry Zhu
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA  94065
USA

Email: larryzhu@live.com

Margaret Wasserman
Painless Security
356 Abbott Street
North Andover, MA  01845
USA

Phone: +1 781 405-7464
Email: mrw@painless-security.com
URI:  http://www.painless-security.com

Greg Hudson (editor)
MIT

Email: ghudson@mit.edu