Internet Engineering Task Force (IETF)
Request for Comments: 8636
Updates: 4556
Category: Standards Track
ISSN: 2070-1721

L. Hornquist Astrand
Apple, Inc
L. Zhu
Oracle Corporation
M. Cullen
Painless Security
G. Hudson
MIT

July 2019

Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) Algorithm Agility

Abstract

This document updates the Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) standard (RFC 4556) 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 in any specific cryptographic algorithm and allow incremental deployment of newer algorithms.

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/rfc8636.

Table of Contents

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 [WANG04], alongside attacks on later hash functions including MD5 [RFC1321] and SHA-1 [RFC6234]. 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 action 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 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 integrity protection for the unauthenticated clear text in these messages and the binding between the pre-authentication and the ticket request and response 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 Authentication Service (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 a list of supported CMS types signifying the digest algorithms supported by the KDC in decreasing order of preference. This is accomplished by including a TD_CMS_DATA_DIGEST_ALGORITHMS typed data element in the error data.

td-cms-digest-algorithms INTEGER ::= 111

The corresponding data for the TD_CMS_DATA_DIGEST_ALGORITHMS contains the TD-CMS-DIGEST-ALGORITHMS-DATA structure, which is ASN.1 Distinguished Encoding Rules (DER) [X680] [X690] encoded and is 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
        -- decreasing order of preference.
```

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

This information sent by the KDC via TD_CMS_DATA_DIGEST_ALGORITHMS can facilitate troubleshooting 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 certification authority (CA) in signing the client's X.509 certificate in decreasing order of preference. 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 TD-CERT-DIGEST-ALGORITHMS-DATA structure 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 decreasing order of
                    -- 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 decreasing order of 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 DHRepInfo 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], choosing the ASN.1 format for FixedInfo, and Section 4.1 of [SP80056C], choosing 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 ticket-granting server (TGS) as specified in the AS-REQ [RFC4120] in the request.

- The supplemental public information (suppPubInfo) is the ASN.1 DER encoding of the PkinitSuppPubInfo structure, 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 PkinitSuppPubInfo structure 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 AS-REQ type [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 when TCP is used. The pk-as-rep field contains the DER encoding of the PA-PK-AS-REP [RFC4556] type in the KDC reply. The PkinitSuppPubInfo provides a cryptographic binding 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 supportedKDFs field 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 kdf field 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:

o KDC_ERR_NO_ACCEPTABLE_KDF 100

If the client fills the supportedKDFs field in the request but the kdf field in the reply is not present, the client can deduce that the KDC is not updated to conform with this specification, or that the exchange was subjected to a downgrade attack. It is a matter of local policy on the client whether to reject the reply when the kdf field is absent in the reply; if compatibility with non-updated KDCs is not a concern, the reply should be rejected.

Implementations conforming to this specification MUST support id-pkinit-kdf-ah-sha256.

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 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000 000000000 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 =
AAAAAAAA AAAAAAAA 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

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

8.2.2. Outputs

key-material: Length = 32 bytes, Hex Representation =
E6AB38C9 413E035B B079201E D0B6B73D 8D49A814 A737C04E E6649614 206F73AD

key: Length = 32 bytes, Hex Representation =
E6AB38C9 413E035B B079201E D0B6B73D 8D49A814 A737C04E E6649614 206F73AD

8.3. Test Vector for SHA-256, enctype 18

8.3.1. Specific Inputs

algorithm-id: (id-pkinit-kdf-ah-sha256) Length = 8 bytes, Hex Representation = 2B060105 02030602

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

8.3.2. Outputs

key-material: Length = 32 bytes, Hex Representation = 77EF4E48 C420AE3F EC75109D 7981697E ED5D295C 90C62564 F7BFD101 FA9bC1D5

key: Length = 32 bytes, Hex Representation = 77EF4E48 C420AE3F EC75109D 7981697E ED5D295C 90C62564 F7BFD101 FA9bC1D5

8.4. Test Vector for SHA-512, enctype 16

8.4.1. Specific Inputs

algorithm-id: (id-pkinit-kdf-ah-sha512) Length = 8 bytes, Hex Representation = 2B060105 02030603

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

8.4.2. Outputs

key-material: Length = 24 bytes, Hex Representation = D3C78A79 D65213EF E9A826F7 5DFB01F7 2362FB16 FB01DAD6

key: Length = 32 bytes, Hex Representation = D3C78A79 D65213EF E9A826F7 5DFB01F7 2362FB16 FB01DAD6

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.

The discovery method described in Section 4 uses a Kerberos error message, which is unauthenticated in a typical exchange. An attacker may attempt to downgrade a client to a weaker CMS type by forging a KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error. It is a matter of

local policy whether a client accepts a downgrade to a weaker CMS type and whether the KDC accepts the weaker CMS type. A client may reasonably assume that the real KDC implements all hash functions used in the client's X.509 certificate, and so the client may refuse attempts to downgrade to weaker hash functions.

The discovery method described in Section 5 also uses a Kerberos error message. An attacker may attempt to downgrade a client to a certificate using a weaker signing algorithm by forging a KDC_ERR_DIGEST_IN_CERT_NOT_ACCEPTED error. It is a matter of local policy whether a client accepts a downgrade to a weaker certificate and whether the KDC accepts the weaker certificate. This attack is only possible if the client device possesses multiple client certificates of varying strengths.

In the KDF negotiation method described in Section 6, the client supportedKDFs value is protected by the signature on the signedAuthPack field in the request. If this signature algorithm is vulnerable to collision attacks, an attacker may attempt to downgrade the negotiation by substituting an AuthPack with a different or absent supportedKDFs value, using a PKINIT freshness token [RFC8070] to partially control the legitimate AuthPack value. A client that is performing anonymous PKINIT [RFC8062] does not sign the AuthPack, so an attacker can easily remove the supportedKDFs value in this case. Finally, the kdf field in the DHRepInfo of the KDC response is unauthenticated and could be altered or removed by an attacker, although this alteration will likely result in a decryption failure by the client rather than a successful downgrade. It is a matter of local policy whether a client accepts a downgrade to the old KDF and whether the KDC allows the use of the old KDF.

The paChecksum field, which binds the client pre-authentication data to the Kerberos request body, remains fixed at SHA-1. If an attacker substitutes a different request body using an attack against SHA-1 (a second preimage attack is likely required as the attacker does not control any part of the legitimate request body), the KDC will not detect the substitution. Instead, if a new KDF is negotiated, the client will detect the substitution by failing to decrypt the reply.

An attacker may attempt to impersonate the KDC to the client via an attack on the hash function used in the dhSignedData signature, substituting the attacker's subjectPublicKey for the legitimate one without changing the hash value. It is a matter of local policy which hash function the KDC uses in its signature and which hash functions the client will accept in the KDC signature. A KDC may reasonably assume that the client implements all hash functions used in the KDF algorithms listed the supportedKDFs field of the request.

10. IANA Considerations

IANA has made the following assignments in the Kerberos "Pre-authentication and Typed Data" registry created by Section 7.1 of RFC 6113.

TD-CMS-DIGEST-ALGORITHMS	111	[RFC8636]
TD-CERT-DIGEST-ALGORITHMS	112	[RFC8636]

11. References

11.1. 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>.

[RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <https://www.rfc-editor.org/info/rfc3961>.

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <https://www.rfc-editor.org/info/rfc4120>.

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, DOI 10.17487/RFC4556, June 2006, <https://www.rfc-editor.org/info/rfc4556>.

[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>.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, DOI 10.17487/RFC5652, September 2009, <https://www.rfc-editor.org/info/rfc5652>.

[RFC6234] Eastlake 3rd, D. and T. Hansen, "US Secure Hash Algorithms (SHA and SHA-based HMAC and HKDF)", RFC 6234, DOI 10.17487/RFC6234, May 2011, <https://www.rfc-editor.org/info/rfc6234>.

[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>.

[SP80056A] Barker, E., Chen, L., Roginsky, A., Vassilev, A., and R. Davis, "Recommendation for Pair-Wise Key-Establishment Schemes Using Discrete Logarithm Cryptography", NIST Special Publications 800-56A, Revision 3, DOI 10.6028/NIST.SP.800-56Ar3, April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Ar3.pdf>.

[SP80056C] Barker, E., Chen, L., and R. Davis, "Recommendation for Key-Derivation Methods in Key-Establishment Schemes", NIST Special Publications 800-56C, Revision 1, DOI 10.6028/NIST.SP.800-56Cr1, April 2018, <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-56Cr1.pdf>.

[X680] ITU-T, "Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation", ITU-T Recommendation X.680, August 2015, <https://www.itu.int/rec/T-REC-X.680-201508-I/en>.

[X690] ITU-T, "Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)", ITU-T Recommendation X.690, August 2015, <https://www.itu.int/rec/T-REC-X.690-201508-I/en>.

11.2. Informative References

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, DOI 10.17487/RFC1321, April 1992, <https://www.rfc-editor.org/info/rfc1321>.

[RFC6150] Turner, S. and L. Chen, "MD4 to Historic Status", RFC 6150, DOI 10.17487/RFC6150, March 2011, <https://www.rfc-editor.org/info/rfc6150>.

[RFC6194] Polk, T., Chen, L., Turner, S., and P. Hoffman, "Security Considerations for the SHA-0 and SHA-1 Message-Digest Algorithms", RFC 6194, DOI 10.17487/RFC6194, March 2011, <https://www.rfc-editor.org/info/rfc6194>.

[RFC7696] Housley, R., "Guidelines for Cryptographic Algorithm Agility and Selecting Mandatory-to-Implement Algorithms", BCP 201, RFC 7696, DOI 10.17487/RFC7696, November 2015, <https://www.rfc-editor.org/info/rfc7696>.

[RFC8062] Zhu, L., Leach, P., Hartman, S., and S. Emery, Ed., "Anonymity Support for Kerberos", RFC 8062, DOI 10.17487/RFC8062, February 2017, <https://www.rfc-editor.org/info/rfc8062>.

[RFC8070] Short, M., Ed., Moore, S., and P. Miller, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT) Freshness Extension", RFC 8070, DOI 10.17487/RFC8070, February 2017, <https://www.rfc-editor.org/info/rfc8070>.

[WANG04] Wang, X., Lai, X., Feng, D., Chen, H., and X. Yu, "Cryptanalysis of the Hash Functions MD4 and RIPEMD", Advances in Cryptology - EUROCRYPT 2005, DOI 10.1007/11426639_1, August 2004.

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
           -- decreasing order of 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 decreasing order of
              -- 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,
          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
          ...
   }

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

Acknowledgements

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

Authors' Addresses

Love Hornquist Astrand
Apple, Inc
Cupertino, CA
United States of America

Email: lha@apple.com

Larry Zhu
Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
United States of America

Email: larryzhu@live.com

Margaret Cullen
Painless Security
4 High St, Suite 134
North Andover, MA 01845
United States of America

Phone: +1 781-405-7464
Email: margaret@painless-security.com

Greg Hudson
MIT

Email: ghudson@mit.edu