PK-INIT algorithm agility
draft-ietf-krb-wg-pkinit-alg-agility-04

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware have been or will be disclosed, and any of which he or she becomes aware will be disclosed, in accordance with Section 6 of BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

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."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on February 6, 2009.

Copyright Notice

Copyright (C) The IETF Trust (2008).
Abstract

The PK-INIT defined in RFC 4556 is examined and updated to remove protocol structures tied to specific cryptographic algorithms. The affinity to SHA-1 as the checksum algorithm in the authentication request is analyzed. The PK-INIT key derivation function is made negotiable, the digest algorithms for signing the pre-authentication data and the client’s X.509 certificates are made discoverable.

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

Table of Contents

1. Introduction .................................................. 3
2. Requirements Notation ......................................... 5
3. paChecksum Agility ............................................ 6
4. CMS Digest Algorithm Agility .................................. 7
5. X.509 Certificate Signer Algorithm Agility .................. 8
6. KDF agility ..................................................... 9
7. Test vectors ................................................... 14
8. Security Considerations ....................................... 15
9. Acknowledgements ............................................... 16
10. IANA Considerations .......................................... 17
11. References .................................................... 18
  11.1. Normative References .................................... 18
  11.2. Informative References .................................. 19
Appendix A. PKINIT ASN.1 Module ................................. 20
Authors’ Addresses ............................................... 23
Intellectual Property and Copyright Statements .................. 24
1. Introduction

In August 2004, Xiaoyun Wang’s research group reported MD4 [RFC1320] collisions generated using hand calculation [WANG04] alongside attacks on later hash function designs in the MD4, MD5 [RFC1321] and SHA [RFC4634] family. Implementations based on Wang’s algorithm can find collisions in real time. These discoveries challenged the security for protocols relying on the collision resistance properties of these hashes, for example one can forge a digital signature when SHA-1 [RFC4634] is the digest algorithm. A more far reaching side effect of these recent events, however, is that it is now generally considered flawed or distasteful at least if protocols are designed to use only specific algorithms.

The Internet Engineer Task Force (IETF) called for actions to update existing protocols to provide crypto algorithm agility in order to untie protocols with specific algorithms. The idea is that if the protocol has crypto algorithm agility, and when a new vulnerability specific to an algorithm is discovered, this algorithm can be disabled via protocol negotiation quickly, thus we can avoid the wait for the multi-year standardization and implementation efforts to update the protocols. In addition, crypto agility allows deployment of new crypto algorithms to be done incrementally, rather than requiring a "flag day" on which the change must be deployed everywhere at the same time in order to maintain interoperability.

This document is to update PK-INIT [RFC4556] to provide crypto algorithm agility. Several protocol structures used in the [RFC4556] protocol are either tied to SHA-1, or require the algorithms not negotiated but rather based on local policy. The following concerns have been addressed:

- The checksum algorithm in the authentication request is hardwired to use SHA-1 [RFC4634].
- The acceptable digest algorithms for signing the authentication data are not discoverable.
- The key derivation function in Section 3.2.3 of [RFC4556] is hardwired to use SHA-1.
- The acceptable digest algorithms for signing the client X.509 certificates are not discoverable.

To accomplish these, new key derivation functions (KDFs) identifiable by object identifiers are defined. The PKINIT client provides a list of KDFs in the request and the KDC picks one in the response, thus a mutually-supported KDF is negotiated.
Furthermore, structures are defined to allow the client discover the Cryptographic Message Syntax (CMS) [RFC3852] 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", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].
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 body from being tempered with. SHA-1 is the only allowed checksum algorithm defined in [RFC4556]. This facility relies the collision resistance properties of the SHA-1 checksum [RFC4634].

When the reply key delivery mechanism is based on public key encryption as described in Section 3.2.3. 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. 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

When the KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error is returned as described in section 3.2.2 of [RFC4556], implementations conforming to this specification can OPTIONALLY send back 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.

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 [RFC3852]
   -- identifiers that identify the digest algorithms
   -- acceptable by 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

When the client’s X.509 certificate is rejected and the KDC_ERR_DIGEST_IN_SIGNED_DATA_NOT_ACCEPTED error is returned as described in section 3.2.2 of [RFC4556], conforming implementations can OPTIONALLY send a list of digest algorithms acceptable by the KDC for use by the 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 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 [RFC3852] identifiers that identify the digest algorithms
    -- that are used by the CA to sign the client’s X.509 certificate and acceptable by 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 [RFC3280].
}

The KDC fills in allowedAlgorithm field with the list of algorithm identifiers that identify digest algorithms that are used by the CA to sign the client’s X.509 certificate and acceptable by 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 [RFC3280].
6. KDF agility

Based on [RFC3766] and [X9.42], Section 3.2.3.1 of [RFC4556] defines a Key Derivation Function (KDF) that derives a Kerberos protocol key based on the secret value generated by the Diffie-Hellman key exchange. This KDF requires the use of SHA-1 [RFC4634].

New KDFs defined in this document based on [SP80056A] can be used in conjunction with any hash functions.

A new KDF is identified by an object identifier. The following KDF object identifiers are defined:

\[
\begin{align*}
\text{id-pkinit-kdf OBJECT IDENTIFIER} & \quad ::= \{ \text{id-pkinit 6} \} \\
& \quad -- \text{PKINIT KDFs} \\
\text{id-pkinit-kdf-ah-sha1 OBJECT IDENTIFIER} & \quad ::= \{ \text{id-pkinit-kdf 1} \} \\
& \quad -- \text{SP800 56A ASN.1 structured hash based KDF using SHA-1} \\
\text{id-pkinit-kdf-ah-sha256 OBJECT IDENTIFIER} & \quad ::= \{ \text{id-pkinit-kdf 2} \} \\
& \quad -- \text{SP800 56A ASN.1 structured hash based KDF using SHA-256} \\
\text{id-pkinit-kdf-ah-sha512 OBJECT IDENTIFIER} & \quad ::= \{ \text{id-pkinit-kdf 3} \} \\
& \quad -- \text{SP800 56A ASN.1 structured hash based KDF using SHA-512}
\end{align*}
\]

Where \text{id-pkinit} is defined in [RFC4556]. The \text{id-pkinit-kdf-ah-sha1} KDF is the ASN.1 structured hash based KDF (HKDF) [SP80056A] that uses SHA-1 [RFC4634] as the hash function. Similarly \text{id-pkinit-kdf-ah-sha256} and \text{id-pkinit-kdf-ah-sha512} are the ASN.1 structured HKDF using SHA-256 [RFC4634] and SHA-512 [RFC4634] respectively.

To name in input parameters, a abbreviated version of the ASN.1 version of the KDF from [SP80056A] is described below, use [SP80056A] for the full reference. follows:

1. \text{reps} = \text{keydatalen (K) / hash length (H)}
2. Initialize a 32-bit, big-endian bit string counter as 1.
3. For \text{i} = 1 to \text{reps} by 1, do the following:
   1. Compute Hash = \text{H(counter || Z || OtherInfo)}.
   2. Increment counter (modulo \text{2^32})
4. Set key = Hash1 || Hash2 || ... so that length of key is \text{K bytes}.
OtherInfo ::= SEQUENCE {
  algorithmID   AlgorithmIdentifier,
  partyUInfo     [0] OCTET STRING,
  partyVInfo     [1] OCTET STRING,
  suppPubInfo    [2] OCTET STRING OPTIONAL,
  suppPrivInfo   [3] OCTET STRING OPTIONAL
}

The input parameters for these KDFs are provided as follows:

- The key data length (K) 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 hash length (H) is 16 bytes for id-pkinit-kdf-ah-sha1, 32 bytes for id-pkinit-kdf-ah-sha256, and 64 bytes for id-pkinit-kdf-ah-sha512.

- 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 algorithm identifier (algorithmID) input parameter is the identifier of the respective KDF. For example, this is id-pkinit-kdf-ah-sha1 if the KDF is the [SP80056A] ASN.1 structured HKDF using 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 private information (suppPubInfo) is absent.

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

- The maximum hash input length is $2^{64}$ in bits.

The structure PkinitSuppPubInfo is defined as follows:
PknitSuppPubInfo ::= SEQUENCE {
    enctype           [0] Int32,
        -- The enctype of the AS reply key.
    as-REQ            [1] OCTET STRING,
        -- This contains the AS-REQ in the request.
    pk-as-rep         [2] OCTET STRING,
        -- Contains the DER encoding of the type
    ticket            [3] Ticket,
        -- This is the ticket in the KDC reply.
    ...
}

The PknitSuppPubInfo 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 ticket field is filled with the ticket in the KDC reply. The PknitSuppPubInfo 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 above ASN.1 structured [SP80056A] HKDF produces a bit string of length K in bits as the keying material, and then the AS reply key is the output of random-to-key() [RFC3961] using that returned keying material as the input.

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:
Internet-Draft          PK-INIT algorithm agility            August 2008

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. This kdf field MUST be filled by the conforming KDC if the supportedKDFs field is present in the request, and it MUST be one of the KDFs supported by the client as indicated in the request. Which KDF is chosen is a matter of the local policy on the KDC.

If the supportedKDFs field is not present in the request, the kdf
field in the reply MUST be absent.

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. In that case, it is a matter of local policy on the client whether to reject the reply when the kdf field is absent in the reply.

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

If no acceptable KDF is found, the error KDC_ERR_NO_ACCEPTABLE_KDF (82).

PKINIT introduces the following new error code:

- KDC_ERR_NO_ACCEPTABLE_KDF 82
7. Test vectors

Test vectors for the KDF.

Z: 000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
cclient: lha@SU.SE
server: krbtgt/SU.SE@SU.SE
enctype: 18
as-req: AAAAAAAAAAAAAAAAAAAA
pk-as-rep: BBBBBBBBBBBBBBBBBBB
ticket: 61353033A003020105A1071B0553552E5345A210300EA003020101A1
0730051B036C6861A311300FA003020112A208040668656A68656A
key: C76289EC4B28A691FFCE80BBB7EC8241523F99B190CF2D348F54A865812C3273
8. Security Considerations

This document describes negotiation of checksum types, key derivation functions and other cryptographic functions. If negotiation is done unauthenticated, care MUST be taken to accept only acceptable values.
9. Acknowledgements

Jeffery Hutzelman, Shawn Emery and Tim Polk reviewed the document and provided suggestions for improvements.
10. IANA Considerations

No IANA considerations.
11. References

11.1. Normative References


Hornquist Astrand & Zhu Expires February 6, 2009 [Page 18]
11.2. 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 3280.

    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
    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.

TD-CMS-DIGEST-ALGORITHMS-DATA ::= SEQUENCE OF
    AlgorithmIdentifier
    -- Contains the list of CMS algorithm [RFC3852]
    -- identifiers that identify the digest algorithms
    -- acceptable by 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 [RFC3852]
    -- identifiers that identify the digest algorithms
    -- that are used by the CA to sign the client’s
    -- X.509 certificate and acceptable by 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
-- [RFC3280].
...

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,
    -- This contains the AS-REQ in the request.
  pk-as-rep    [2] OCTET STRING,
    -- Contains the DER encoding of the type
  ticket      [3] Ticket,
    -- This is the ticket 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
Authors’ Addresses

Love Hornquist Astrand
Apple, Inc
Cupertino
USA

Email: lha@apple.com

Larry Zhu
Microsoft Corporation
One Microsoft Way
Redmond, WA  98052
USA

Email: lzhu@microsoft.com