Internet-Draft K. Raeburn
Kerberos Working Group MIT
Updates: RFC 1964 August 13, 2003
Document: draft-ietf-krb-wg-gss-crypto-00.txt expires February 13, 2004

General Kerberos Cryptosystem Support for the Kerberos 5 GSSAPI Mechanism

Abstract

This document describes an update to the Kerberos 5 mechanism for GSSAPI to allow the use of Kerberos-defined cryptosystems directly in GSSAPI messages, without requiring further changes for each new encryption or checksum algorithm that complies with the Kerberos crypto framework specifications.

1. Introduction

Kerberos defines an encryption and checksum framework [KCRYPTO] that provides for complete specification and enumeration of cryptosystem specifications in a general way, to be used within Kerberos and associated protocols. The GSSAPI Kerberos 5 mechanism definition [GSSAPI-KRB5] sets up a framework for enumerating encryption and checksum types, independently of how such schemes may be used in Kerberos, thus requiring updates for any new encryption or checksum algorithm not directly compatible with those already defined.

This document describes an update to [GSSAPI-KRB5] to allow the use of any Kerberos-defined cryptosystems directly in GSSAPI messages, without requiring further changes for each new encryption or checksum algorithm that complies with the framework specifications, and without making assumptions concerning block sizes or other characteristics of the underlying encryption operations.

2. Conventions Used in This Document

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 RFC 2119.

3. New Algorithm Identifiers

Two new sealing algorithm numbers and one new signing algorithm number are defined, for use in MIC, Wrap and Delete tokens.

type	name	octet values
seal	KERBEROS5-ENCRYPT	00 01
sign	KERBEROS5-CHECKSUM	00 01
sign	NONE	ff ff

The KERBEROS5-ENCRYPT algorithm encrypts using the Kerberos encryption type [KCRYPTO] indicated by the encryption key type (using the session key or initiator's subkey, as described in [GSSAPI-KRB5]), with a key usage value KG_USAGE_SEAL, defined below. All Kerberos encryption types provide for integrity protection, and specify any padding that might be required; neither needs to be done at the GSS mechanism level when KERBEROS5-ENCRYPT is used. When KERBEROS5-ENCRYPT is used as the seal algorithm, the sign algorithm MUST be NONE.

The signing algorithm value NONE MUST be used only with a sealing algorithm that incorporates integrity protection; currently, KERBEROS5-ENCRYPT is the only such sealing algorithm.

The KERBEROS5-CHECKSUM signing algorithm MAY be used in other cases. The contents of the SGN_CKSUM field are determined by computing a Kerberos checksum [KCRYPTO], using the session key or subkey, and a key usage value of KG_USAGE_SIGN. The Kerberos checksum algorithm to be used is the required-to-implement checksum algorithm associated with the encryption key type. It should be noted that the checksum input data in this case is not the same as the "to-be-signed data" described in section 1.2.1.1 of [GSSAPI-KRB5]; see below.

The encryption or checksum output incorporated in the MIC and Wrap tokens is the octet string output from the corresponding operation in [KCRYPTO]; it should not be confused with the EncryptedData or Checksum object in [KrbClar].

For purposes of key derivation, we add two new usage values to the list defined in [KrbClar]; one for signing messages, and one for sealing messages:

name	value
KG_USAGE_SEAL	22
KG_USAGE_SIGN	23

4. Adjustments to Previous Definitions

4.1. Quality of Protection

The GSSAPI specification [GSSAPI] says that a zero QOP value indicates the "default". The original specification for the Kerberos 5 mechanism says that a zero QOP value (or a QOP value with the appropriate bits clear) means DES encryption.

Since the quality of protection cannot be improved without fully reauthenticating with a stronger key type, the QOP value is now ignored.

4.2. Message Layout

The definitions of the MIC and Wrap tokens in [GSSAPI-KRB5] assumed an 8-byte checksum size, and a CBC-mode block cipher with an 8-byte block size, without integrity protection. In the crypto framework described in [KCRYPTO], integrity protection is built into the encryption operations. CBC mode is not assumed, and indeed there may be no initial vector to supply. While the operations are performed on messages of specific sizes, the underlying cipher may be a stream cipher.

We modify the message definitions such that the portions after the first 8 bytes (which specify the token identification and the signing and sealing algorithms) are defined by the algorithms chosen. The remaining bytes must convey sequence number and direction information, and must protect the integrity of the token id and algorithm indicators. For the DES-based algorithms specified in [GSSAPI-KRB5], the definition for the remaining data is backwards compatible.

The revised message descriptions are thus as follows:

MIC token

Byte #	Name	Description
0..1	TOK_ID	Identification field (01 01).
2..3	SGN_ALG	Integrity algorithm indicator.
4..7	Filler	Contains ff ff ff ff
8..N		Dependent on SGN_ALG.
If SGN_ALG is 0000, 0100, 0200:		
8..15	SND_SEQ	Sequence number/direction field, encrypted.
16..23	SGN_CKSUM	Checksum of bytes 0..7 and application data, as described in [GSSAPI-KRB5].
If SGN_ALG is 0001:		
8..15	SND_SEQ	Sequence number/direction field, NOT encrypted.
16..N	SGN_CKSUM	Checksum of bytes 0..15 and application data, with key usage KG_USAGE_SIGN.

Suggestions from Microsoft: Moving to 64-bit sequence numbers would be better for long sessions with many messages. Using the direction flag to perturb the encryption or integrity protection is safer than simply including a flag which a buggy but mostly working implementation might fail to check.

I am considering changing to use 64-bit sequence numbers, and omitting the direction flag from the transmitted cleartext data. How it would factor into the encrypted Wrap token, I haven't figured out yet.

- Change the key usage values based on the direction? It's suggested in [KCRYPTO], perhaps not strongly enough, that the key usage numbers should perturb the key, but DES ignores them, although DES shouldn't use this extension.

- Add a direction flag byte in encrypted data? Either depends on an implementor remembering to add the check. Adding it to checksummed data requires that the implementor get it right.

- Generate one or two new keys using PRF and random-to-key operations, using different keys for each direction? Pulling the DK function out of the simplified profile is probably not a good

way to do this.

The filler bytes are included in the checksum calculation for simplicity. There is no security benefit from including them.

In the Wrap token, the initial bytes, sequence number and direction are incorporated into the data to be encrypted. In most cases, this is likely to be more efficient in terms of space and computing power than using unencrypted sequence number and direction fields, adding a checksum, and doing the additional work to authenticate that the checksum and encrypted data are part of the same message. (The framework in [KCRYPTO] has no support for integrity protection of a block of data only some of which is encrypted, except by treating the two portions independently and using some additional means to ensure that the two parts continue to be associated with one another.)

The length is also included, as a 4-byte value in network byte order, because the decryption operation in the Kerberos crypto framework does not recover the exact length of the original input. Thus, messages with application data larger than 4 gigabytes are not supported.

[Q: Should this length be 8 bytes? ASN.1 wrapper?]

```
                             Wrap token
    Byte #      Name         Description
    -------------------------------------------------------------
      0..1      TOK_ID       Identification field (02 01).
      2..3      SGN_ALG      Integrity algorithm indicator.
      4..5      SEAL_ALG     Sealing algorithm indicator.
      6..7      Filler       Contains ff ff
      8..last                Dependent on SEAL_ALG and SGN_ALG.

    If SEAL_ALG is 0000:
      8..15     SND_SEQ      Encrypted sequence number field.
     16..23     SGN_CKSUM    Checksum of plaintext padded data,
                             calculated according to algorithm
                             specified in SGN_ALG field.  (RFC
                             1964)
     24..last   Data         Encrypted padded data.

    If SEAL_ALG is 0001 and SGN_ALG is ffff:
      8..last   Data         Encrypted bytes 0..5, 2-byte
                             direction flag, sequence number,
                             4-byte data length, and data.
```

```
    If SEAL_ALG is ffff, and SGN_ALG is 0000, 0100, 0200:
      8..15     SND_SEQ      Encrypted sequence number field.
     16..23     SGN_CKSUM    Checksum of plaintext padded data,
                             as in RFC 1964.
     24..last   Data         plaintext padded data

    If SEAL_ALG if ffff, and SGN_ALG is 0001:
      8..15     SND_SEQ      Sequence number/direction field,
                             NOT encrypted.
     16..N      SGN_CKSUM    Checksum of bytes 0..15 and
                             application data, with key usage
                             KG_USAGE_SIGN.
    N+1..last   Data         plaintext data
```

The direction flag, as in [GSSAPI-KRB5], is made up of bytes indicating the party sending the token: 00 for the context initiator, or hex FF for the context acceptor. In the KERBEROS5-ENCRYPT case, only two bytes are used, and they replace the fixed filler bytes of the token header, which need no protection; this reduces slightly the redundancy of the data transmitted.

The context-deletion token is essentially a MIC token with no user data and a different TOK_ID value. Thus, its modification is straightforward.

```
                       Context deletion token
       Byte #    Name         Description
       -------------------------------------------------------
         0..1    TOK_ID       Identification field (01 02).
         2..3    SGN_ALG      Integrity algorithm indicator.
         4..7    Filler       Contains ff ff ff ff

       If SGN_ALG is 0000, 0100, 0200:
         8..15   SND_SEQ      Sequence number/direction
                              field, encrypted.
        16..23   SGN_CKSUM    Checksum of bytes 0..7, as
                              described in [GSSAPI-KRB5].

       If SGN_ALG is 0001:
         8..15   SND_SEQ      Sequence number/direction
                              field, NOT encrypted.
        16..N    SGN_CKSUM    Checksum of bytes 0..15, with
                              key usage KG_USAGE_SIGN.
```

5. Backwards Compatibility Considerations

The context initiator should request of the KDC credentials using session-key cryptosystem types supported by that implementation; if the only types returned by the KDC are not supported by the mechanism implementation, it should indicate a failure. This may seem obvious, but early implementations of both Kerberos and the GSSAPI Kerberos mechanism supported only DES keys, so the cryptosystem compatibility question was easy to overlook.

Under the current mechanism, no negotiation of algorithm types occurs, so server-side (acceptor) implementations cannot request that clients not use algorithm types not understood by the server. However, administration of the server's Kerberos data (e.g., the service key) has to be done in communication with the KDC, and it is from the KDC that the client will request credentials. The KDC could therefore be given the task of limiting session keys for a given service to types actually supported by the Kerberos and GSSAPI software on the server.

This does have a drawback for cases where a service principal name is used both for GSSAPI-based and non-GSSAPI-based communication (most notably the "host" service key), if the GSSAPI implementation does not understand (for example) AES but the Kerberos implementation does. It means that AES session keys cannot be issued for that service principal, which keeps the protection of non-GSSAPI services weaker than necessary.

It would also be possible to have clients attempt to get DES session keys before trying to get AES session keys, and have the KDC refuse to issue the DES keys only for the most critical of services, for which DES protection is considered inadequate. However, that would eliminate the possibility of connecting with the more secure cryptosystem to any service that can be accessed with the weaker cryptosystem. We thus recommend the former approach, putting the burden on the KDC administration and gaining the best protection possible for GSSAPI services, possibly at the cost of weaker protection of non-GSSAPI Kerberos services sharing service principal names with GSSAPI services that have not been updated to support this extension.

[optional:]

This mechanism extension MUST NOT be used with the DES encryption key types described in [KCRYPTO], which ignore the key usage values.

6. Implementation Note

At least two implementations have been done of extensions to the RFC 1964 mechanism for specific non-DES encryption types. These are not standards-track extensions, but implementors may wish to implement them as well for compatibility with existing products. No guidance is provided as to when an implementation may wish to use these non-standard extensions instead of the extension specified in this document.

7. Security Considerations

Various tradeoffs arise regarding the mixing of new and old software, or GSSAPI-based and non-GSSAPI Kerberos authentication. They are discussed in section 5.

Remember to check direction flag. Key usage numbers and direction checks? Considerations depend on the approach taken....

8. Acknowledgements

Larry Zhu...

9. Normative References

[GSSAPI]
Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January, 2000.

[GSSAPI-KRB5]
Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June, 1996.

[KCRYPTO]
draft-ietf-krb-wg-crypto-XX -> RFC xxxx

[KrbClar]
draft-ietf-krb-wg-kerberos-clarifications-XX -> RFC xxxx

[RFC2026]
RFC 2026 ...

[RFC2119]
RFC 2119 ...

10. Author's Address

Kenneth Raeburn
Massachusetts Institute of Technology
77 Massachusetts Avenue
Cambridge, MA 02139

Document Change History

Version -XX:

Split up Abstract and create a real Introduction. Fix RFC 2026 reference in Status section. Added Conventions, Acknowledgements and Implementation Note sections. Updated References with more placeholders. Capitalize some uses of "must" etc.

Fill in case of Wrap token without integrity protection, using KERBEROS5-CHECKSUM for SGN_ALG. Fix descriptions of which message layout is used for which algorithms.

Remove discussion of authenticated encryption with additional data.

Add discussion of 64-bit sequence numbers and data length, and alternate handling of the direction flag.

Version -XX sent in early 2003 to Kerberos working group:

Initial revision.