1. Abstract

[KCRYPTO] introduced a generic framework for the inclusion of new encryption and checksum algorithms to be used in the Kerberos V5 protocol. [AES-KRB5] describes a crypto profile (per [KCRYPTO]) for AES. This document introduces a generic method for adding new crypto-systems to the GSSAPI-KerberosV5 mechanism based on crypto profiles as defined in [KCRYPTO]. It also describes the use of AES encryption for GSSAPI as an example of this new method.

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 [2].

3. Introduction

[GSSAPI-KRB5] describes the GSSAPI mechanism for Kerberos V5. It defines the format of context initiation, per-message and context deletion tokens. When adding new crypto system support, the approach taken in [GSSAPI-KRB5] is to add algorithm identifiers for each new cryptosystem.
The approach taken in this document is to use the same encryption and checksum algorithms specified by the crypto profile for the session key or subkey that is created during context negotiation. Message layouts of the per-message and context deletion tokens are revised to remove algorithm indicators and add extra information to support the generic crypto framework [KCRYPTO].

"Newer" encryption and checksum types MUST use the new token formats defined in this document. Older encryption and checksum types SHALL NOT use these new token formats. The term "newer" is more precisely defined in [KRBCLR].

Note that in this document, "AES" is used for brevity to refer loosely to either aes128-cts-hmac-sha1-96 or aes256-cts-hmac-sha1-96 as defined in [AES-KRB5].

4. Quality of Protection and Algorithm Identifiers

The GSSAPI specification [GSSAPI] provides for QOP values that can be used by the application to request a certain type of encryption or signing. A zero QOP value is used to indicate the "default" protection; applications which use the default QOP are not guaranteed to be portable across implementations or even inter-operate with different deployment configurations of the same implementation. Using an algorithm that is different from the one for which the key is defined may not be appropriate. Therefore, when the new method in this document is used, the QOP value is ignored.

The encryption and checksum algorithms in per-message and context deletion tokens are now implicitly defined by the algorithms associated with the session and subkey. Algorithms identifiers are therefore no longer needed and removed from the token headers.

5. Key Derivation

To limit the exposure of a given key, [KCRYPTO] adopted "one-way" "entropy-preserving" derived keys, for different purposes or key usages, from a base key or protocol key. This document defines four key usage values below for signing and sealing messages:

<table>
<thead>
<tr>
<th>Name</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>KG-USAGE-ACCEPTOR-SIGN</td>
<td>22</td>
</tr>
<tr>
<td>KG-USAGE-ACCEPTOR-SEAL</td>
<td>23</td>
</tr>
<tr>
<td>KG-USAGE-INITIATOR-SIGN</td>
<td>24</td>
</tr>
<tr>
<td>KG-USAGE-INITIATOR-SEAL</td>
<td>25</td>
</tr>
</tbody>
</table>

When the sender is the context acceptor, KG-USAGE-ACCEPTOR-SIGN is used as the usage number in the key derivation function for deriving keys to be used in MIC and context deletion tokens, and KG-USAGE-ACCEPTOR-SEAL is used for Wrap tokens; similarly when the sender is the context initiator, KG-USAGE-INITIATOR-SIGN is used as the usage number in the key derivation function for MIC and context deletion tokens, KG-USAGE-INITIATOR-SEAL is used for Wrap Tokens. Even if the Wrap token does not provide for confidentiality the same usage
values specified above are used.

6. Token Formats and Definitions

The new token formats defined in this document are designed to accommodate the requirements of newer crypto systems. Certain implementations of GSSAPI, such as SSPI, allow for "scatter-gather" and in-place encryption, mostly by leveraging low level details of crypto systems. The token layouts have been designed to satisfy the above requirements without incurring significant performance penalties or losing generality.

The design along with the rationale behind it, is discussed in detail in the following sections.

6.1. Sequence Number and Direction Indicators

The sequence number for any per-message or context deletion token is a 64 bit integer (expressed in big endian order). One separate byte is used as the directional indicator: the hex value FF if the sender is the context acceptor, 00 otherwise. Both the sequence number and the directional indicator are protected as specified in section 6.2.

6.2. Encryption and Checksum Operations

The encryption algorithms defined by the crypto profiles provide for integrity protection. Therefore no separate checksum is needed.

In Wrap tokens that provide for confidentiality, the "header" (the first 16 bytes of the Wrap token) is appended to the plaintext data before encryption. Hence the resulting Wrap token is ("header" | encrypt(plaintext-data | "header")), where encrypt() is the encryption operation defined in the crypto profile. In Wrap tokens that do not provide for confidentiality, the checksum is calculated over the plaintext data concatenated with the token header (the first 16 bytes of the Wrap token). Hence the resulting Wrap token is ("header" | plaintext-data | get_mic(plaintext-data | "header")), where get_mic() is the checksum operation defined in the crypto profile. The parameters for the key and the cipher-state in the encrypt() and get_mic() operations have been omitted for brevity.

The resulting Wrap tokens bind the data to the token header, including the sequence number, directional indicator, and the rotation count.

[With AEAD, Wrap tokens with confidentiality do not need to encrypt the header and the overhead can be reduced slightly]

For MIC tokens, the checksum is first calculated over the token header (the first 16 bytes of the MIC token) and then the to-be-signed plaintext data.

For context deletion tokens, the checksum is calculated over the token header (the first 16 bytes of the context deletion token).

When AES is used, the checksum algorithm is HMAC_SHA1_96 and the
checksum size is 12 bytes. Data is pre-pended with a 16 byte confounder before encryption, and no padding is needed.

6.3. RRC Field

A new field, called "RRC" (Right Rotation Count), is added to allow the data to be encrypted in place. The resulting Wrap token in the previous section, excluding the first 16 bytes of the token header, is rotated to the right by "RRC" bytes. The net result is that "RRC" bytes of trailing octets are moved toward the header.

Consider the following as an example of this rotation operation:
Assume that the RRC value is 3 and the token before the rotation is "header" | aa | bb | cc | dd | ee | ff | gg | hh}, the token after rotation would be {"header" | ff | gg | hh | aa | bb | cc | dd | ee }, where {aa | bb | cc |...| hh} is used to indicate the byte sequence.

The RRC field is expressed as a two octet integer in big endian order.

The rotation count value is chosen by the sender based on implementation details, and the receiver MUST be able to interpret all possible rotation count values.

6.4. EC Field

The decryption operation in the crypto profile may not always return the exact length of the plaintext data. An "EC"(Extra Count) field is added to the Wrap() token header to indicate the number of bytes that have been added to the end of the plaintext data before encryption. The "EC" field is a two byte integer expressed in big endian order and it should be 00 00 if confidentiality is not provided for by the Wrap tokens.

6.5. Seal Field

The Seal field in Wrap tokens is used to indicate whether confidentiality is provided for. If confidentiality is provided for by the Wrap token, this field contains the hex value FF, otherwise it contains the hex value 00.

6.6. Message Layout for Per-message and Context Deletion Tokens

The new message layouts are as follows.

MIC Token:
The Filler field is included in the checksum calculation for simplicity. This is common to both MIC and context deletion token checksum calculations.

Wrap Token:

<table>
<thead>
<tr>
<th>Byte no</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>TOK_ID</td>
<td>Identification field. Tokens emitted by GSS_Wrap() contain the hex value 05 04 in this field.</td>
</tr>
<tr>
<td>2..2</td>
<td>Direction</td>
<td>Hex value FF if the sender is the context acceptor, 00 otherwise.</td>
</tr>
<tr>
<td>3..3</td>
<td>Seal</td>
<td>Confidentiality indicator: hex value FF if confidentiality is provided for, 00 otherwise.</td>
</tr>
<tr>
<td>4..5</td>
<td>EC</td>
<td>Contains the &quot;extra count&quot; field, in big endian order as described in section 6.4.</td>
</tr>
<tr>
<td>6..7</td>
<td>RRC</td>
<td>Contains the &quot;right rotation count&quot; in big endian order, as described in section 6.3.</td>
</tr>
<tr>
<td>8..15</td>
<td>SND_SEQ</td>
<td>Sequence number field in cleartext, in big endian order.</td>
</tr>
<tr>
<td>16..</td>
<td>Data</td>
<td>Encrypted or plaintext data, as described in section 6.2, where the encryption or checksum algorithm is defined by the crypto profile for the session key or subkey.</td>
</tr>
</tbody>
</table>

Context Deletion Token:

<table>
<thead>
<tr>
<th>Byte no</th>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0..1</td>
<td>TOK_ID</td>
<td>Identification field. Tokens emitted by GSS_Delete_sec_context() contain the hex value 04 05 in this field.</td>
</tr>
<tr>
<td>2..2</td>
<td>Direction</td>
<td>Hex value FF if the sender is the context acceptor, 00 otherwise.</td>
</tr>
<tr>
<td>3..7</td>
<td>Filler</td>
<td>Contains 5 bytes of hex value FF.</td>
</tr>
<tr>
<td>8..15</td>
<td>SND_SEQ</td>
<td>Sequence number field in cleartext, in big endian order.</td>
</tr>
<tr>
<td>16..</td>
<td>SGN_CKSUM</td>
<td>Checksum of byte 0..15, where the checksum algorithm is defined by the crypto profile for the session key or subkey.</td>
</tr>
</tbody>
</table>
7. Backwards compatibility considerations

The new token formats defined in this document will only be recognized by new implementations. A MIC or wrap token generated with the algorithms defined in this document will not be recognized by an older implementation that only recognizes the algorithms defined in [GSSAPI-KRB5]. To address this, implementations can always use the explicit sign or seal algorithm in [GSSAPI-KRB5] when the key type corresponds to those algorithms. An alternative approach might be to retry sending the message with the sign or seal algorithm explicitly defined as in [GSSAPI-KRB5]. However this would require the use of a mechanism such as [SPNEGO] to securely negotiate the algorithm or the use out of band mechanism to choose appropriate algorithms. For this reason, it is RECOMMENDED that the use of the new token formats defined in this document be confined only for "newer encryption and checksum" described by a crypto profile.

8. Security Considerations

It is possible that the KDC returns a session-key type that is not supported by the GSSAPI implementation (either on the client and the server). In this case the implementation MUST not try to use the key with a supported cryptosystem. This will ensure that no security weaknesses arise due to the use of an inappropriate key with an encryption algorithm.

In addition the security problem described in [3DES] arising from the use of a service implementation with a GSSAPI mechanism supporting only DES and a Kerberos mechanism supporting both DES and Triple DES is actually a more generic one. It arises whenever the GSSAPI implementation does not support a stronger cryptosystem supported by the Kerberos mechanism. KDC administrators desiring to limit the session key types to support interoperability with such GSSAPI implementations should carefully weigh the reduction in protection offered by such mechanisms against the benefits of interoperability.

9. Acknowledgments

The authors wish to acknowledge the contributions from the following individuals:

Ken Raeburn and Nicolas Williams corrected many of our errors in the use of generic profiles and were instrumental in the creation of this draft. Sam Hartman and Ken Raeburn suggested the "floating trailer"
idea.

Sam Hartman and Nicolas Williams recommended the replacing our earlier key derivation function for directional keys with different key usage numbers for each direction as well as retaining the directional bit for maximum compatibility.

Paul Leach provided numerous suggestions and comments. Scott Field, Richard Ward, Dan Simon also provided valuable inputs on this draft.

10. References


11. Author’s Address

Larry Zhu
One Microsoft Way
Redmond, WA 98052 - USA
EMail: LZhu@microsoft.com

Karthik Jaganathan
One Microsoft Way
Redmond, WA 98052 - USA
EMail: karthikkj@microsoft.com