

Kerberos Working Group
Internet-Draft
Expires: April 8, 2012

G. Hudson, Ed.
MIT Kerberos Consortium
October 6, 2011

Camellia Encryption for Kerberos 5
draft-ietf-krb-wg-camellia-cts-00

Abstract

This document specifies two encryption types and two corresponding checksum types for the Kerberos cryptosystem suite. The new types use the Camellia block cipher in CBC-mode with ciphertext stealing and the CMAC algorithm for integrity protection.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and 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/lid-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 April 8, 2012.

Copyright Notice

Copyright (c) 2011 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 (<http://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 BSD License.

1. Introduction

The Camellia block cipher, described in [RFC3713], has a 128-bit block size and a 128-bit, 192-bit, or 256-bit key size, similar to AES. This document specifies Kerberos encryption and checksum types for Camellia using 128-bit or 256-bit keys. The new types conform to the framework specified in [RFC3961], but do not use the simplified profile.

Like the simplified profile, the new types use key derivation to produce keys for encryption, integrity protection, and checksum operations. Instead of the [RFC3961] section 5.1 key derivation function, the new types use a key derivation function from the family specified in [SP800-108].

The new types use the CMAC algorithm for integrity protection and checksum operations. As a consequence, they do not rely on a hash algorithm except when generating keys from strings.

Like the AES encryption types [RFC3962], the new encryption types use CBC mode with ciphertext stealing to avoid the need for padding. They also use the same PBKDF2 algorithm for key generation from strings, with a modification to the salt string to ensure that different keys are generated for Camellia and AES encryption types.

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].

2. Protocol Key Representation

The Camellia key space is dense, so we use random octet strings directly as keys. The first bit of the Camellia bit string is the high bit of the first byte of the random octet string.

3. Key Generation from Strings

We use a variation on the key generation algorithm specified in [RFC3962] section 4.

First, to ensure that different long-term keys are used with Camellia and AES, we prepend the enctype name to the salt string, separated by a null byte. The enctype name is "camellia128-cts-cmac" or "camellia256-cts-cmac" (without the quotes).

Second, the final key derivation step uses the algorithm described in Section 4 instead of the key derivation algorithm used by the simplified profile.

Third, if no string-to-key parameters are specified, the default number of iterations is raised to 32768.

```
saltp = enctype-name | 0x00 | salt
tkey = random2key(PBKDF2-HMAC-SHA1(passphrase, saltp,
                                iter_count, keylength))
key = KDF-FEEDBACK-CMAC(tkey, "kerberos")
```

4. Key Derivation

We use a key derivation function from the family specified in [SP800-108] section 5.2, "KDF in Feedback Mode". The PRF parameter of the key derivation function is CMAC with Camellia-128 or Camellia-256 as the underlying block cipher; this PRF has an output size of 128 bits. A block counter is used with a length of 4 bytes, represented in big-endian order. The length of the output key in bits (denoted as k) is also represented as a four-byte string in big-endian order. The label input to the KDF is the usage constant supplied to the key derivation function, and the context is unused.

```
n = ceiling(k / 128)
K0 = zeros
Ki = CMAC(key, K(i-1) | i | constant | 0x00 | k)
DR(key, constant) = k-truncate(K1 | K2 | ... | Kn)
KDF-FEEDBACK-CMAC(key, constant) = random-to-key(DR(key, constant))
```

The constants used for key derivation are the same as those used in the simplified profile.

5. CMAC Checksum Algorithm

For integrity protection and checksums, we use the CMAC function defined in [SP800-38B], with Camellia-128 or Camellia-256 as the underlying block cipher.

6. Kerberos Algorithm Protocol Parameters

The following parameters apply to the encryption types `camellia128-cts-cmac`, which uses a 128-bit protocol key, and `camellia256-cts-cmac`, which uses a 256-bit protocol key.

Protocol key format: as defined in Section 2.

Specific key structure: three protocol format keys: { Kc, Ke, Ki }.

Required checksum mechanism: as defined in Section 7.

Key generation seed length: the key size (128 or 256 bits).

String-to-key function: as defined in Section 3.

Default string-to-key parameters: 00 00 80 00.

Random-to-key function: identity function.

Key-derivation function: as indicated below, with usage represented as four octets in big-endian order.

Kc = KDF-FEEDBACK-CMAC(base-key, usage | 0x99)

Ke = KDF-FEEDBACK-CMAC(base-key, usage | 0xAA)

Ki = KDF-FEEDBACK-CMAC(base-key, usage | 0x55)

Cipher state: a 128-bit CBC initialization vector.

Initial cipher state: all bits zero.

Encryption function: as follows, where E() is Camellia encryption in CBC-CTS mode, with the next-to-last block used as the CBC-style ivec, or the last block if there is only one.

```
conf = Random string of 128 bits
(C, newstate) = E(Ke, conf | plaintext, oldstate)
M = CMAC(Ki, conf | plaintext)
ciphertext = C | M
```

Decryption function: as follows, where D() is Camellia decryption in CBC-CTS mode, with the ivec treated as in E().

```
(C, M) = ciphertext
(P, newIV) = D(Ke, C, oldstate)
if (M != CMAC(Ki, P)) report error
newstate = newIV
```

Pseudo-random function: as follows.

```
Kp = KDF-FEEDBACK-CMAC(protocol-key, "prf")
PRF = CMAC(Kp, octet-string)
```

7. Checksum Parameters

The following parameters apply to the checksum types `cmac-camellia128` and `cmac-camellia256`, which are the associated checksum for `camellia128-cts-cmac` and `camellia256-cts-cmac` respectively.

Associated cryptosystem: Camellia-128 or Camellia-256 as appropriate for the checksum type.

`get_mic`: `CMAC(Kc, message)`.

`verify_mic`: `get_mic` and compare.

8. Assigned Numbers

Encryption types

Type name	etype value	key size
<code>camellia128-cts-cmac</code>	TBD	128
<code>camellia256-cts-cmac</code>	TBD	256

Checksum types

Type name	sumtype value	length
<code>cmac-camellia128</code>	TBD	128
<code>cmac-camellia256</code>	TBD	128

9. Security Considerations

[CRYPTOENG] chapter 4 discusses weaknesses of the CBC cipher mode. An attacker who can observe enough messages generated with the same key to encounter a collision in ciphertext blocks could recover the XOR of the plaintexts of the previous blocks. Observing such a collision becomes likely as the number of blocks observed approaches

2^{64} . This consideration applies to all previously standardized Kerberos encryption types and all uses of CBC encryption with 128-bit block ciphers in other protocols. Kerberos deployments can mitigate this concern by rolling over keys often enough to make observing 2^{64} messages unlikely.

Because the new checksum types are deterministic, an attacker could pre-compute checksums for a known plain-text message in 2^{64} randomly chosen protocol keys. The attacker could then observe checksums legitimately computed in different keys until a collision with one of the pre-computed keys is observed; this becomes likely after the number of observed checksums approaches 2^{64} . Observing such a collision allows the attacker to recover the protocol key. This consideration applies to most previously standardized Kerberos checksum types and most uses of 128-bit checksums in other protocols.

Kerberos deployments should not migrate to the Camellia encryption types simply because they are newer, but should use them only if business needs require the use of Camellia, or if a serious flaw is discovered in AES which does not apply to Camellia.

The security considerations described in [RFC3962] section 8 regarding the string-to-key algorithm also apply to the Camellia encryption types.

At the time of writing this document, there are no known weak keys for Camellia, and no security problem has been found on Camellia (see [NESSIE], [CRYPTREC], and [LNCS5867]).

10. IANA Considerations

Assign two Kerberos Encryption Type Numbers for camellia128-cts-cmac and camellia256-cts-cmac.

Assign two Kerberos Checksum Type Numbers for cmac-camellia128 and camellia256.

11. Test Vectors

Sample results for string-to-key conversion:

```
Iteration count = 1
Pass phrase = "password"
Salt = "ATHENA.MIT.EDUraeburn"
128-bit Camellia key:
  57 D0 29 72 98 FF D9 D3 5D E5 A4 7F B4 BD E2 4B
```

256-bit Camellia key:

B9 D6 82 8B 20 56 B7 BE 65 6D 88 A1 23 B1 FA C6
82 14 AC 2B 72 7E CF 5F 69 AF E0 C4 DF 2A 6D 2C

Iteration count = 2

Pass phrase = "password"

Salt = "ATHENA.MIT.EDUraeburn"

128-bit Camellia key:

73 F1 B5 3A A0 F3 10 F9 3B 1D E8 CC AA 0C B1 52

256-bit Camellia key:

83 FC 58 66 E5 F8 F4 C6 F3 86 63 C6 5C 87 54 9F
34 2B C4 7E D3 94 DC 9D 3C D4 D1 63 AD E3 75 E3

Iteration count = 1200

Pass phrase = "password"

Salt = "ATHENA.MIT.EDUraeburn"

128-bit Camellia key:

8E 57 11 45 45 28 55 57 5F D9 16 E7 B0 44 87 AA

256-bit Camellia key:

77 F4 21 A6 F2 5E 13 83 95 E8 37 E5 D8 5D 38 5B
4C 1B FD 77 2E 11 2C D9 20 8C E7 2A 53 0B 15 E6

Iteration count = 5

Pass phrase = "password"

Salt=0x1234567878563412

128-bit Camellia key:

00 49 8F D9 16 BF C1 C2 B1 03 1C 17 08 01 B3 81

256-bit Camellia key:

11 08 3A 00 BD FE 6A 41 B2 F1 97 16 D6 20 2F 0A
FA 94 28 9A FE 8B 27 A0 49 BD 28 B1 D7 6C 38 9A

Iteration count = 1200

Pass phrase = (64 characters)

"XX"

Salt="pass phrase equals block size"

128-bit Camellia key:

8B F6 C3 EF 70 9B 98 1D BB 58 5D 08 68 43 BE 05

256-bit Camellia key:

11 9F E2 A1 CB 0B 1B E0 10 B9 06 7A 73 DB 63 ED
46 65 B4 E5 3A 98 D1 78 03 5D CF E8 43 A6 B9 B0

Iteration count = 1200

Pass phrase = (65 characters)

"XX"

Salt = "pass phrase exceeds block size"

128-bit Camellia key:

57 52 AC 8D 6A D1 CC FE 84 30 B3 12 87 1C 2F 74

256-bit Camellia key:

```
61 4D 5D FC 0B A6 D3 90 B4 12 B8 9A E4 D5 B0 88
B6 12 B3 16 51 09 94 67 9D DB 43 83 C7 12 6D DF
```

Iteration count = 50

Pass phrase = g-clef (0xf09d849e)

Salt = "EXAMPLE.COMpianist"

128-bit Camellia key:

```
CC 75 C7 FD 26 0F 1C 16 58 01 1F CC 0D 56 06 16
```

256-bit Camellia key:

```
16 3B 76 8C 6D B1 48 B4 EE C7 16 3D F5 AE D7 0E
20 6B 68 CE C0 78 BC 06 9E D6 8A 7E D3 6B 1E CC
```

Sample results for key derivation:

128-bit Camellia key:

```
57 D0 29 72 98 FF D9 D3 5D E5 A4 7F B4 BD E2 4B
```

Kc value for key usage 2 (constant = 0x0000000299):

```
D1 55 77 5A 20 9D 05 F0 2B 38 D4 2A 38 9E 5A 56
```

Ke value for key usage 2 (constant = 0x00000002AA):

```
64 DF 83 F8 5A 53 2F 17 57 7D 8C 37 03 57 96 AB
```

Ki value for key usage 2 (constant = 0x0000000255):

```
3E 4F BD F3 0F B8 25 9C 42 5C B6 C9 6F 1F 46 35
```

256-bit Camellia key:

```
B9 D6 82 8B 20 56 B7 BE 65 6D 88 A1 23 B1 FA C6
```

```
82 14 AC 2B 72 7E CF 5F 69 AF E0 C4 DF 2A 6D 2C
```

Kc value for key usage 2 (constant = 0x0000000299):

```
E4 67 F9 A9 55 2B C7 D3 15 5A 62 20 AF 9C 19 22
```

```
0E EE D4 FF 78 B0 D1 E6 A1 54 49 91 46 1A 9E 50
```

Ke value for key usage 2 (constant = 0x00000002AA):

```
41 2A EF C3 62 A7 28 5F C3 96 6C 6A 51 81 E7 60
```

```
5A E6 75 23 5B 6D 54 9F BF C9 AB 66 30 A4 C6 04
```

Ki value for key usage 2 (constant = 0x0000000255):

```
FA 62 4F A0 E5 23 99 3F A3 88 AE FD C6 7E 67 EB
```

```
CD 8C 08 E8 A0 24 6B 1D 73 B0 D1 DD 9F C5 82 B0
```

Sample encryptions (all using the default cipher state):

Plaintext: (empty)

128-bit Camellia key:

```
1D C4 6A 8D 76 3F 4F 93 74 2B CB A3 38 75 76 C3
```

Random confounder:

```
B6 98 22 A1 9A 6B 09 C0 EB C8 55 7D 1F 1B 6C 0A
```

Ciphertext:

```
C4 66 F1 87 10 69 92 1E DB 7C 6F DE 24 4A 52 DB
```

```
0B A1 0E DC 19 7B DB 80 06 65 8C A3 CC CE 6E B8
```

Plaintext: 1

Random confounder:

6F 2F C3 C2 A1 66 FD 88 98 96 7A 83 DE 95 96 D9

128-bit Camellia key:

50 27 BC 23 1D 0F 3A 9D 23 33 3F 1C A6 FD BE 7C

Ciphertext:

84 2D 21 FD 95 03 11 C0 DD 46 4A 3F 4B E8 D6 DA

88 A5 6D 55 9C 9B 47 D3 F9 A8 50 67 AF 66 15 59

B8

Plaintext: 9 bytes

Random confounder:

A5 B4 A7 1E 07 7A EE F9 3C 87 63 C1 8F DB 1F 10

128-bit Camellia key:

A1 BB 61 E8 05 F9 BA 6D DE 8F DB DD C0 5C DE A0

Ciphertext:

61 9F F0 72 E3 62 86 FF 0A 28 DE B3 A3 52 EC 0D

0E DF 5C 51 60 D6 63 C9 01 75 8C CF 9D 1E D3 3D

71 DB 8F 23 AA BF 83 48 A0

Plaintext: 13 bytes

Random confounder:

19 FE E4 0D 81 0C 52 4B 5B 22 F0 18 74 C6 93 DA

128-bit Camellia key:

2C A2 7A 5F AF 55 32 24 45 06 43 4E 1C EF 66 76

Ciphertext:

B8 EC A3 16 7A E6 31 55 12 E5 9F 98 A7 C5 00 20

5E 5F 63 FF 3B B3 89 AF 1C 41 A2 1D 64 0D 86 15

C9 ED 3F BE B0 5A B6 AC B6 76 89 B5 EA

Plaintext: 30 bytes bytes bytes bytes byt

Random confounder:

CA 7A 7A B4 BE 19 2D AB D6 03 50 6D B1 9C 39 E2

128-bit Camellia key:

78 24 F8 C1 6F 83 FF 35 4C 6B F7 51 5B 97 3F 43

Ciphertext:

A2 6A 39 05 A4 FF D5 81 6B 7B 1E 27 38 0D 08 09

0C 8E C1 F3 04 49 6E 1A BD CD 2B DC D1 DF FC 66

09 89 E1 17 A7 13 DD BB 57 A4 14 6C 15 87 CB A4

35 66 65 59 1D 22 40 28 2F 58 42 B1 05 A5

Plaintext: (empty)

Random confounder:

3C BB D2 B4 59 17 94 10 67 F9 65 99 BB 98 92 6C

256-bit Camellia key:

B6 1C 86 CC 4E 5D 27 57 54 5A D4 23 39 9F B7 03

1E CA B9 13 CB B9 00 BD 7A 3C 6D D8 BF 92 01 5B

Ciphertext:

03 88 6D 03 31 0B 47 A6 D8 F0 6D 7B 94 D1 DD 83

7E CC E3 15 EF 65 2A FF 62 08 59 D9 4A 25 92 66

Plaintext: 1

Random confounder:

DE F4 87 FC EB E6 DE 63 46 D4 DA 45 21 BB A2 D2

256-bit Camellia key:

1B 97 FE 0A 19 0E 20 21 EB 30 75 3E 1B 6E 1E 77

B0 75 4B 1D 68 46 10 35 58 64 10 49 63 46 38 33

Ciphertext:

2C 9C 15 70 13 3C 99 BF 6A 34 BC 1B 02 12 00 2F

D1 94 33 87 49 DB 41 35 49 7A 34 7C FC D9 D1 8A

12

Plaintext: 9 bytes

Random confounder:

AD 4F F9 04 D3 4E 55 53 84 B1 41 00 FC 46 5F 88

256-bit Camellia key:

32 16 4C 5B 43 4D 1D 15 38 E4 CF D9 BE 80 40 FE

8C 4A C7 AC C4 B9 3D 33 14 D2 13 36 68 14 7A 05

Ciphertext:

9C 6D E7 5F 81 2D E7 ED 0D 28 B2 96 35 57 A1 15

64 09 98 27 5B 0A F5 15 27 09 91 3F F5 2A 2A 9C

8E 63 B8 72 F9 2E 64 C8 39

Plaintext: 13 bytes

Random confounder:

CF 9B CA 6D F1 14 4E 0C 0A F9 B8 F3 4C 90 D5 14

256-bit Camellia key:

B0 38 B1 32 CD 8E 06 61 22 67 FA B7 17 00 66 D8

8A EC CB A0 B7 44 BF C6 0D C8 9B CA 18 2D 07 15

Ciphertext:

EE EC 85 A9 81 3C DC 53 67 72 AB 9B 42 DE FC 57

06 F7 26 E9 75 DD E0 5A 87 EB 54 06 EA 32 4C A1

85 C9 98 6B 42 AA BE 79 4B 84 82 1B EE

Plaintext: 30 bytes

Random confounder:

64 4D EF 38 DA 35 00 72 75 87 8D 21 68 55 E2 28

256-bit Camellia key:

CC FC D3 49 BF 4C 66 77 E8 6E 4B 02 B8 EA B9 24

A5 46 AC 73 1C F9 BF 69 89 B9 96 E7 D6 BF BB A7

Ciphertext:

0E 44 68 09 85 85 5F 2D 1F 18 12 52 9C A8 3B FD

8E 34 9D E6 FD 9A DA 0B AA A0 48 D6 8E 26 5F EB

F3 4A D1 25 5A 34 49 99 AD 37 14 68 87 A6 C6 84

57 31 AC 7F 46 37 6A 05 04 CD 06 57 14 74

Sample checksums:

Plaintext: abcdefghijk
Checksum type: cmac-camellia128
128-bit Camellia key:
1D C4 6A 8D 76 3F 4F 93 74 2B CB A3 38 75 76 C3
Key usage: 7
Checksum:
11 78 E6 C5 C4 7A 8C 1A E0 C4 B9 C7 D4 EB 7B 6B

Plaintext: ABCDEFGHIJKLMNOPQRSTUVWXYZ
Checksum type: cmac-camellia128
128-bit Camellia key:
50 27 BC 23 1D 0F 3A 9D 23 33 3F 1C A6 FD BE 7C
Key usage: 8
Checksum:
D1 B3 4F 70 04 A7 31 F2 3A 0C 00 BF 6C 3F 75 3A

Plaintext: 123456789
Checksum type: cmac-camellia256
256-bit Camellia key:
B6 1C 86 CC 4E 5D 27 57 54 5A D4 23 39 9F B7 03
1E CA B9 13 CB B9 00 BD 7A 3C 6D D8 BF 92 01 5B
Key usage: 9
Checksum:
87 A1 2C FD 2B 96 21 48 10 F0 1C 82 6E 77 44 B1

Plaintext: !@#\$%^&*(!@#\$%^&*(!@#\$%^&*()
Checksum type: cmac-camellia256
256-bit Camellia key:
32 16 4C 5B 43 4D 1D 15 38 E4 CF D9 BE 80 40 FE
8C 4A C7 AC C4 B9 3D 33 14 D2 13 36 68 14 7A 05
Key usage: 10
Checksum:
3F A0 B4 23 55 E5 2B 18 91 87 29 4A A2 52 AB 64

12. References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3713] Matsui, M., Nakajima, J., and S. Moriai, "A Description of the Camellia Encryption Algorithm", RFC 3713, April 2004.
- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.
- [RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[SP800-38B]

Dworkin, M., "Recommendation for Block Cipher Modes of Operation: The CMAC Mode for Authentication", NIST Special Publication 800-38B, October 2009.

[SP800-108]

Chen, L., "Recommendation for Key Derivation Using Pseudorandom Functions", NIST Special Publication 800-108, October 2009.

[CRYPTOENG]

Schneier, B., "Cryptography Engineering", March 2010.

[CRYPTREC]

Information-technology Promotion Agency (IPA),
"Cryptography Research and Evaluation Committees",
<<http://www.ipa.go.jp/security/enc/CRYPTREC/index-e.html>>.

[LNCS5867]

Mala, H., Shakiba, M., and M. Dakhil-alian, "New Results on Impossible Different Cryptanalysis of Reduced Round Camellia-128", LNCS 5867, November 2009,
<<http://www.springerlink.com/content/e55783u422436g77/>>.

[NESSIE]

The NESSIE Project, "New European Schemes for Signatures, Integrity, and Encryption",
<<http://www.cosic.esat.kuleuven.be/nessie/>>.

Appendix A. Notes to RFC Editor

Change the "TBD" entries in Section 8 to the values assigned by IANA.

Remove this section.

Author's Address

Greg Hudson (editor)
MIT Kerberos Consortium

Email: ghudson@mit.edu

Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

S. Sorce, Ed.
Red Hat
T. Yu, Ed.
T. Hardjono, Ed.
MIT Kerberos Consortium
Oct 31, 2011

A Generalized PAC for Kerberos V5
draft-ietf-krb-wg-general-pac-01

Abstract

This draft proposes a generalized authorization structure for the Kerberos V5 protocol. Such an authorization structure would allow for greater interoperability among directory services and other related Kerberos services across differing realms.

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 <http://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 May 3, 2012.

Copyright Notice

Copyright (c) 2011 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 (<http://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.

Table of Contents

1. Introduction	3
2. Requirements Language	3
3. Use-Case: Cross-Realm Directory Services	3
4. A Generalized Authorization Structure for Kerberos V5	4
4.1. Attributes	4
4.2. PAD-Realm	5
4.3. PAD-Principal	5
4.4. PAD-DNS-Domain	5
4.5. PAD-Short-Domain	5
4.6. PAD-UDID	6
4.7. PAD-Posix-Username	6
4.8. PAD-Posix-UID	6
4.9. PAD-Posix-GID	6
4.10. PAD-Posix-Gecos	6
4.11. PAD-Posix-Homedir	6
4.12. PAD-Posix-Shell	7
4.13. PAD-Fullname	7
4.14. PAD-AlternateNames	7
4.15. PAD-Groups	7
4.16. PAD Mapped Attributes	8
4.17. RFC2307 references for Directory Services backed KDCs	8
4.17.1. PAD-Posix-Username as 'uid'	8
4.17.2. PAD-Posix-UID as 'uidNumber'	9
4.17.3. PAD-Posix-GID as 'gidNumber'	9
4.17.4. PAD-Posix-Gecos as 'gecos'	9
4.17.5. PAD-Posix-Homedir as 'homeDirectory'	9
4.17.6. PAD-Posix-Shell as 'loginShell'	9
5. Encoding	9
5.1. PAD Format	9
6. Data Structures and Extensions	11
6.1. SignedPrincipalAuthorizationData	11
6.2. GSS-API Authenticator Extension	13
7. Assigned numbers	14
8. Timeouts Considerations	14
9. IANA Considerations	15
10. Security Considerations	15
11. Acknowledgements	15
12. References	15
12.1. Normative References	15
12.2. Informative References	16
Appendix A. Additional Stuff	16
Authors' Addresses	16

1. Introduction

There is an increasing need today for Kerberos to support the delivery and processing of authorization information pertaining to the principals seeking access to the servers. Kerberos today is used extensively for authentication to directory services within the Enterprise. In many cases, a directory service is implemented as a distributed database system organized across multiple realms. As such, when a client in one realm seeks access to a directory service component located within a different realm, information regarding both the identity of the client and the permissions associated with that client must be communicated across the realms. Currently there does not exist a common and standardized structure in Kerberos (V5) for conveying access control or authorization information.

This draft proposes a general authorization structure for Kerberos that identifies a base set of common data elements or fields within the authorization structure, as well as the format of that structure. We refer to this data structure as the Principal Authorization Data (PAD) structure in order to distinguish it from existing structures, such as the Privilege Attribute Certificate defined by Microsoft in [MS-PAC].

2. Requirements Language

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 [RFC2119].

3. Use-Case: Cross-Realm Directory Services

In this section we discuss one of the primary use-case scenarios for the Principal Authorization Data (PAD) structure within Kerberos V5. In this use-case a client principal is seeking to access a service in a different realm. Since the remote service does not have authorization information regarding the client, it needs to obtain it either from querying the directory service in its own realm or the directory service located in the client's realm. It is here that a common PAD structure becomes necessary and invaluable in order to achieve a high-degree of interoperability between directory services in distinct realms.

In this use-case a client principal C1 in realm R1 is seeking access to services (or servers) located in a different realm R2. In accessing local service S1 in realm R1 the client must first be authenticated by KDC1 in that realm. A directory service (e.g.

LDAP) called D1 is used in realm R1 to perform authorization of the client, after the client has been authenticated by KDC1.

When the client principal later seeks to access services or resources S2 in realm R2, following the usual Kerberos flow the client must first obtain a cross-realm TGT from KDC1 (in realm R1) and then present it to KDC2 (in realm R2) in order to obtain a service-ticket for S2. However, one immediate issue is the fact that service S2 does not have authorization information regarding the permissions or privileges of client C1 in realm R1. The service S2 could query its own directory service D2 to obtain authorization information pertaining to client C1. In the absence of such information in D2, the service S2 could then perform a cross-realm query to the directory services D1 operating in realm R1.

However, this cross-realm query from S2 to D1 is not only inefficient, but it also implies knowledge of multiple heterogeneous systems by all actors. Two different realms may rely on completely different infrastructures for user information storage, ranging from different LDAP implementations with different schema conventions to NIS, SQL databases, flat files, and so on. Every service in the realm R2 would have to know what information system is in use in R1, how to reach it, how to read and eventually how to map data from it. Moreover security related aspects on the authentication of S2 by the directory D1, the authorization of S2 to make such a query, the protection of responses from D1 to S2, and so on, would have to be addressed.

This use-case illustrates the need for a common PAD structure to address this cross-realm authorization problem. In particular, the PAD structure for the cross-realm access to remote services needs to be contained or carried within cross-realm TGTs and service-tickets. Such a PAD structure needs to carry enough authorization information such that a decision can be made by service S2 in realm R2 regarding the access request originating from the client principal C1 within realm R1.

4. A Generalized Authorization Structure for Kerberos V5

4.1. Attributes

The following attributes are defined in this document:

- o PAD-Realm
- o PAD-Principal

- o PAD-DNS-Domain
- o PAD-Short-Domain
- o PAD-UDID
- o PAD-Posix-Username
- o PAD-Posix-UID
- o PAD-Posix-GID
- o PAD-Posix-Gecos
- o PAD-Posix-Homedir
- o PAD-Posix-Shell
- o PAD-Fullname
- o PAD-AlternateNames
- o PAD-Groups

These are each defined and discussed further below.

4.2. PAD-Realm

The full Realm Name of the Realm the authorization information belongs to.

4.3. PAD-Principal

The name of the principal. Joined with the PAD-Realm component it MUST match the full principal name of the owner of the ticket.

4.4. PAD-DNS-Domain

The DNS Domain name associated to the Realm.

4.5. PAD-Short-Domain

A short domain name that uniquely identifies, within the set of trusted realms, the domain the principal belongs to. The short Domain name is useful for representation purposes in the OS. A KDC is allowed to change this field during validation. This may be done to resolve name conflicts in large trust relationships.

4.6. PAD-UDID

A UDID is a Unique Domain Identifier. Ideally it universally identifies the domain as the one the following local identifiers belongs to. This is used to differentiate between local identifiers belonging to different domains/realms.

The UDID size can be dependent on the specific Domain type and implementation. However it SHOULD be not less than 96 bits long so that chances of conflicts are relatively low. A 96 bit long identifier allows to construct a 128bit account identifier by concatenating the UDID to the local account Identifier (32bit quantity in POSIX).

For the purpose of this document the UDID is a completely opaque number and implementations SHOULD not try to perform any enforcement on the format of this number on receiving it.

4.7. PAD-Posix-Username

This is the user name that correspond to the kerberos principal, this is the name that SHOULD be used by the OS to represent the user. The OS may decide to prefix or suffix this name with the PAD-Domain or PAD-Realm names in case of name conflicts with local accounts.

4.8. PAD-Posix-UID

This is the UID Number associated to the user. This number is local to the domain identified by PAD-Domain-UUID.

4.9. PAD-Posix-GID

This is the Primary GID Number associated to the user. This number is local to the domain identified by PAD-Domain-UUID.

4.10. PAD-Posix-Gecos

The Gecos field for the User associated to the Principal if available. Can be omitted. If not available PAD-Fullname can be used instead.

4.11. PAD-Posix-Homedir

The home directory path relative to the local system, if available. If not available local defined defaults apply.

4.12. PAD-Posix-Shell

The default shell for the user, defined as the path of the binary relative to the local filesystem, if available. If not available local defined defaults apply.

4.13. PAD-Fullname

The full name of the user if available.

4.14. PAD-AlternateNames

Alternate names can be used by application to identify a user by means that differ from the user principal. Names are in string form and utf8 encoded [UTF-8]. In order to allow applications to recognize the name type without guesswork, alternate names are prefixed with a string followed by the colon ':' character and the name, without any space or other separation character. The following Alternate names are currently recognized: EMAIL, OS, OPENID, OAUTH It is allowed to include multiple alternate names of the same type. The order in which they are provided represent the priority within the same name type, if applications need to choose between names.

(TODO: need discussion on whether these needs labeled prefixes or explicit attributes for each alternate representation etc...)

4.15. PAD-Groups

This is a structured attribute and defines the groups the principal is member of.

The first value in the structure represents the domain UDID and is optional. If missing the domain UDID is assumed to be the one defined in the PAD-UDID attribute.

Then an array of values that define the groups as follows. Each group value includes 3 subvalues:

- o (1) Name: This is the name of the group.
- o (2) Type: Optional, type of group
- o (3) ID: group ID.

If the type is missing it is assumed that the group is of type "Posix Group" and the following ID is required and represents the gid number. The type is represented through a simplified OID like type where only 2 levels are defined. 0.0 Is reserved for posix groups, and the 0

prefix is reserved to official RFX use. Additional Prefixes can be assigned to organizations that request it for their purposes. Assignment TBD.

Multiple PAD-Groups attributes can be present at the same time. A trusting KDC can augment the original user's set of groups by adding a new PAD-Groups structure that contains groups local to the trusting domain. In this case the domain UDID is required. The domain UDID is used for gid number conflict resolution when the PAC is transmitted between services of different realms.

PAD-Groups are optional attributes and the KDC, upon PAC revalidation, may decide to remove the original attributes that do not belong to the KDC security domain in order to save space or to censor information to avoid disclosing data to services.

4.16. PAD Mapped Attributes

In POSIX, users and groups ID are not universally unique, and different Realms (even different machines within an authorization realm actually) may have overlapping and conflicting IDs. If this is the case, a trusting KDC may decide to re-map IDs coming from a foreign Realm to help services with uid/gid mapping and avoid ID conflicts that can lead to serious security issues. The original IDs are generally preserved.

If multiple PAD buffers are received and one of them contains a PAD-UDID that is recognized by the application to be the local security domain identifier, then only the mapped attributes in this buffer SHOULD be used for authorization purposes.

4.17. RFC2307 references for Directory Services backed KDCs

A few attributes contain the keyword 'Posix' in their name. These attributes are usually represented by RFC2307 in Directory Services. If the primary store for these attributes is a Directory the following equivalence with RFC2307 defined attributes can be used.

4.17.1. PAD-Posix-Username as 'uid'

The PAD-Posix-Username is the User ID, and its syntax is equivalent to the attribute named 'uid' in RFC 2307. This attribute is defined in RFC 4519 (2.39). The attribute is defined as multivalued in RFC 4519 but in this context only a single value is allowed. To define aliases refer to the attribute PAD-AlternateNames.

4.17.2. PAD-Posix-UID as 'uidNumber'

The PAD-Posix-UID is the User's Unique Identifier Number, and its syntax is equivalent to the attribute named 'uidNumber' in RFC 2307.

4.17.3. PAD-Posix-GID as 'gidNumber'

The PAD-Posix-GID is the User's Primary Group Identifier Number, and its syntax is equivalent to the attribute named 'gidNumber' in RFC 2307.

4.17.4. PAD-Posix-Gecos as 'gecos'

The PAD-Posix-Gecos is the User's Common Name, although, traditionally, this field has been used to convey additional information beyond the user's full name. Its syntax is equivalent to the attribute named 'gecos' in RFC 2307.

4.17.5. PAD-Posix-Homedir as 'homeDirectory'

The PAD-Posix-Homedir is the User's LOCAL home directory. Its syntax is equivalent to the attribute named 'homeDirectory' in RFC 2307.

4.17.6. PAD-Posix-Shell as 'loginShell'

The PAD-Posix-Shell is the User's preferred login shell. Its syntax is equivalent to the attribute named 'loginShell' in RFC 2307.

5. Encoding

The Kerberos protocol is defined in [RFC4120] using Abstract Syntax Notation One (ASN.1) [X680]. As such, this specification also uses the ASN.1 syntax for specifying both the abstract layout of the PAD attributes, as well as their encodings.

5.1. PAD Format

The information carried in the PAD needs to be augmented by some control information and packaged in a way that makes it possible to devise future extensions.

Additional information needed to validate the PAD:

- o The expiration time (must be the same as the ticket expiration time).

- o The principal name (must be the same principal that owns the ticket).
- o The KDC signature (for re-validation purposes).
- o The Service Signature (in order to trust the PAD has not been tampered with).
- o Optional Trusted Service Key Signature (for use by trusted services on a host)
- o Optional PUBKEY KDC Signature

This information is needed to validate the PAD and make sure it is not modified, outdated, or contains information for a different principal.

In order to make the PAD extensible and at the same time always verifiable we propose that the PAD is embedded in a ASN.1 structure that can contain multiple optional buffers identified by numbers (how to assign numbers TBD).

Buffer number 0 is an ASN.1 structure that includes all attributes described in paragraph 4. This buffer is itself optional.

The whole structure with all its buffers is what is signed with the KDC and the service keys.

The final structure to be included in AD-IF-RELEVANT container and looks loosely like the following diagram.

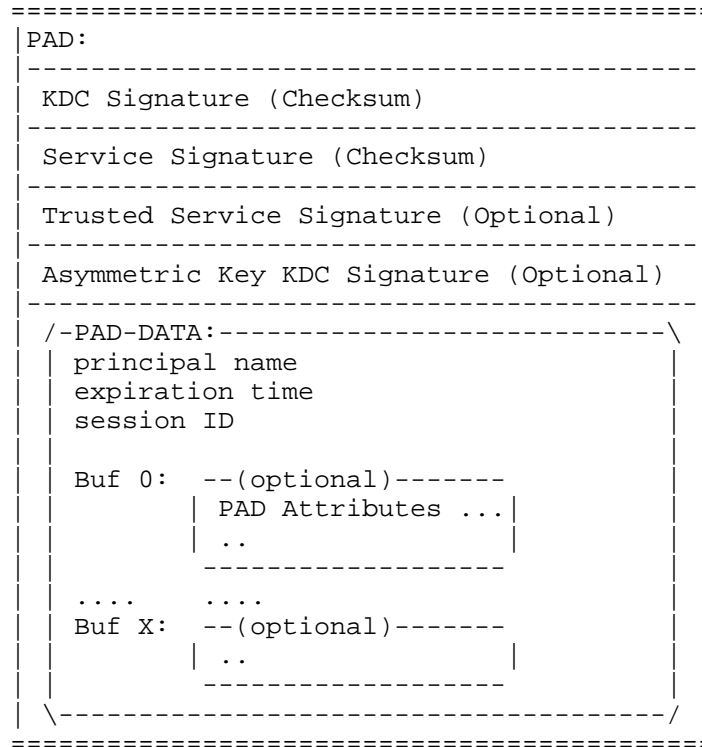


Figure 1: PAD Format

6. Data Structures and Extensions

6.1. SignedPrincipalAuthorizationData


```
AD-PAD ::= SEQUENCE {  
    kdc-signature      [0] Checksum,  
    svc-signature      [1] Checksum,  
    trusted-svc-signature [2] PAD-OPT-Checksum OPTIONAL,  
    pubkey-signature   [2] PAD-OPT-Checksum OPTIONAL,  
    pad-data           [3] PAD-DATA  
}
```

```
PAD-OPT-Checksum ::= SEQUENCE {  
    Identifier [0] Name,  
    Signature  [1] Checksum  
}
```

kdc-signature

A cryptographic checksum computed over the encoding of the pad-data field, keyed with the krbtgt key.
Checksum type TBD.

svc-signature

A cryptographic checksum computed over the encoding of the pad-data field, keyed with the service long term key.
Checksum type TBD.

Trusted-svc-signature

A principal name and a cryptographic checksum computed over the encoding of the pad-data field, keyed with the long term key of the principal name specified in the Name field. Unless otherwise explicitly administratively configured, the key SHOULD be found by substituting the service name component of the principal name of the service with 'host'.

If the service is 'host' this checksum is redundant and can be omitted.

If the resulting host/<name>@REALM or the administratively configured service is not found in the KDC database this checksum can be omitted.

Checksum type TBD.

pubkey-signature

A name identifying the asymmetric key-pair used.

A checksum computed over the encoding of the pad-data field using the Private Key identified in the Name field.

If an asymmetric key is not available this checksum MUST be omitted.

Signature type TBD.

```
PAD-DATA          ::=SEQUENCE {  
    p-realm        [0] Realm,  
    p-name         [1] PrincipalName,  
    expiration     [2] Date,  
    session-id     [3] TBD,  
    elements       [4] SEQUENCE OF AuthorizationData  
}
```

p-realm, p-name

The realm and name of the principal the authorization data elements apply to.

expiration

The Expiration Date of the Authorization Data. Normally this is the same as the original TGT expiration date.

session-id

A random number that uniquely ties any following ticket this PAD Data is associated to with the original TGT Released to the user

elements

A sequence of authorization data elements issued by the KDC.

The AD-PAD data is intended to provide a means for a Kerberos principal credentials to carry authorization data that the receiving service can use to perform authorization decisions.

The KDC signature is required to allow the KDC to validate the data without having to recompute the contents at every TGS request.

The SVC signature is required so that the Service can verify that the authorization data has been validated by the KDC.

Both the Trusted Service Checksum and the asymmetric KDC Signature are useful to verify the PAD authenticity on the same host, when the PAD is received by a less trusted service and passed to a more trusted service on the same host without the need for additional roundtrips to the KDC.

The ad-type for AD-SIGNED-PAD is (TBD).

6.2. GSS-API Authenticator Extension

The Authenticator Checksum as defined in RFC 4121 limit the size of delegated credentials in the KRB_CRED message to a size of 64KiB.

In order to be able to transfer larger messages an extension is defined. This extension is used instead of the Dlght/Deleg fields, and the Dlght and Deleg fields MUST not be included when this extension is appended to the authenticator.

The extension SHALL have the following format which is drafted according to [draft-ietf-krb-wg-gss-cb-hash-agility]:

Octet	Name	Description
0..3	ExtN	A 16bit value identifying the extension. Represented in big-endian order; Contains the hex value 0xFFFFFFFF.
4..7	Length	The length of the Extended Delegation field. Represented in big-endian order;
8..N	Data	A KRB_CRED message (N = Length + 8)

A new flag GSS_C_EXT_DELEG_FLAG with Value X is also defined. This flag is used instead of GSS_C_DELEG_FLAG when the delegated credentials are larger than 64KiB and cannot fit in the standard Deleg field.

Implementors SHOULD use this Extensions and this flag only if the KRB_CRED message is larger than 64KiB and use the standard Deleg field otherwise.

7. Assigned numbers

TBD

8. Timeouts Considerations

Current implementations depend on very strict timeouts on obtaining AS Replies. In popular implementations the client will timeout if it doesn't receive a reply within 1 second. Adding authorization data may involve lookups to external (to the KDC) data sources. Implementors should consider whether the current timeout is still reasonable in light of the additional processing KDCs may be required to do.

9. IANA Considerations

TBD.

10. Security Considerations

Although it is anticipated that the PAD structure itself will be carried within a ticket and thereby protected using the existing encryption methods on that ticket, there are a number of issues that have bearings on the security of the entire Kerberos realm as a whole. Some of these issues are as follows:

- o UID and GID Collisions: There is always the possibility of collision of numbers repressing a UID and a GID. This problem can be remedied to a large degree by realms using an appropriate range selection policy and algorithms.
- o When collisions are detected the KDC or, alternatively, the receiving Service MUST be able to remap IDs so that they do not conflict with locally defined IDs
- o Transit-domain issues: The PAC must be signed by the KDC that is attaching it to a ticket with 2 different signatures. The service signature so that the service can verify its KDC validated the contents. The KDC signature, so that the OS can ask the KDC to confirm the PAD has not been modified by a less trusted service. An optional asymmetric key signature is also allowed if Keys are available in order to avoid additional roundtrips. For cross-realm tickets the "service" signature is made with the cross-realm key. When a KDC receives a PAD it is allowed to modify it in any way. It can filter out information or add information (like group memberships defined locally). A KDC may also decide to change information in different ways depending on what service it is targeted to.

11. Acknowledgements

TBD.

12. References

12.1. Normative References

- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.

- [RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

12.2. Informative References

- [MIT-Athena] Steiner, J., Neuman, B., and J. Schiller, "Kerberos: An Authentication Service for Open Network Systems. In Proceedings of the Winter 1988 Usenix Conference. February.", 1988.
- [MS-PAC] Microsoft, "Microsoft MS-PAC: Privilege Attribute Certificate Data Structure (v20100711)", July 2010.
- [POSIX] The Open Group, "Portable Operating System Interface (POSIX.1-2008)", 2008.
- [RFC1510] Kohl, J. and B. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, September 1993.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2307] Howard, L., "An Approach for Using LDAP as a Network Information Service", RFC 2307, March 1998.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, July 2003.
- [X.690] ISO, "ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER) - ITU-T Recommendation X.690 (ISO/IEC International Standard 8825-1:1998)", 1997.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Simo Sorce (editor)
Red Hat

Email: ssorce@redhat.com

Tom Yu (editor)
MIT Kerberos Consortium

Email: tlyu@mit.edu

Thomas Hardjono (editor)
MIT Kerberos Consortium

Email: hardjono@mit.edu

Kerberos WORKING GROUP
Internet-Draft
Updates: 4120 (if approved)
Intended status: Standards Track
Expires: May 3, 2012

S. Hartman, Ed.
Painless Security
K. Raeburn
MIT
L. Zhu
Microsoft Corporation
October 31, 2011

Kerberos Principal Name Canonicalization and KDC-Generated Cross-Realm
Referrals
draft-ietf-krb-wg-kerberos-referrals-13

Abstract

The memo documents a method for a Kerberos Key Distribution Center (KDC) to respond to client requests for Kerberos tickets when the client does not have detailed configuration information on the realms of users or services. The KDC will handle requests for principals in other realms by returning either a referral error or a cross-realm TGT to another realm on the referral path. The clients will use this referral information to reach the realm of the target principal and then receive the ticket. This memo also provides a mechanism for verifying that a request has not been tampered with in transit.

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 <http://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 May 3, 2012.

Copyright Notice

Copyright (c) 2011 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 (<http://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	4
2. Conventions Used in This Document	5
3. Requesting a Referral	5
4. Realm Organization Model	6
4.1. Trust Assumptions	6
5. Enterprise Principal Name Type	7
6. Name Canonicalization	8
7. Client Referrals	10
8. Server Referrals	11
9. Cross Realm Routing	12
10. Caching Information	12
11. Negotiation of FAST and Detecting Modified Requests	12
12. Number Assignments	14
13. IANA Considerations	14
14. Security Considerations	14
14.1. Shared-password case	16
14.2. Preauthentication data	17
15. Acknowledgments	17
16. References	18
16.1. Normative References	18
16.2. Informative References	18
Appendix A. Compatibility with Earlier Implementations of Name Canonicalization	18
Appendix B. Document history [REMOVE BEFORE PUBLICATION]	19
Authors' Addresses	20

1. Introduction

Current implementations of the Kerberos AS and TGS protocols, as defined in [RFC4120], use principal names constructed from a known user or service name and realm. A service name is typically constructed from a name of the service and the DNS host name of the computer that is providing the service. Many existing deployments of Kerberos use a single Kerberos realm where all users and services would be using the same realm. However in an environment where there are multiple Kerberos realms, the client needs to be able to determine what realm a particular user or service is in before making an AS or TGS request. Traditionally this requires client configuration to make this possible.

When having to deal with multiple realms, users are forced to know what realm they are in before they can obtain a ticket granting ticket (TGT) with an AS request. However, in many cases the user would like to use a more familiar name that is not directly related to the realm of their Kerberos principal name. A good example of this is an RFC 822 style email name. This document describes a mechanism that would allow a user to specify a user principal name that is an alias for the user's Kerberos principal name. In practice this would be the name that the user specifies to obtain a TGT from a Kerberos KDC. The user principal name no longer has a direct relationship with the Kerberos principal or realm. Thus the administrator is able to move the user's principal to other realms without the user having to know that it happened.

Once a user has a TGT, they would like to be able to access services in any Kerberos realm for which there is an authentication path from the realm of their principal. To do this requires that the client be able to determine what realm the target service principal is in before making the TGS request. Current implementations of Kerberos typically have a table that maps DNS host names to corresponding Kerberos realms. The user-supplied host name or its domain component is looked up in this table (often using the result of some form of host name lookup performed with insecure DNS queries, in violation of [RFC4120]). The corresponding realm is then used to complete the target service principal name. Even if insecure DNS queries were not used, managing this table is problematic.

This traditional mechanism requires that each client have very detailed configuration information about the hosts that are providing services and their corresponding realms. Having client side configuration information can be very costly from an administration point of view-- especially if there are many realms and computers in the environment.

This memo proposes a solution for these problems and simplifies administration by minimizing the configuration information needed on each computer using Kerberos. Specifically it describes a mechanism to allow the KDC to handle canonicalization of names, provide for principal aliases for users and services and allow the KDC to determine the trusted realm authentication path by being able to generate referrals to other realms in order to locate principals.

Two kinds of KDC referrals are introduced in this memo:

1. Client referrals, in which the client doesn't know which realm contains a user account.
2. Server referrals, in which the client doesn't know which realm contains a server account.

These two types of referrals introduce new opportunities for an attacker. In order to avoid these attacks, a mechanism is provided to protect the integrity of the request between the client and KDC. This mechanism complements the Flexible Authentication through Secure Tunnels (FAST) facility provided in [RFC6113]. A mechanism is provided to negotiate the availability of FAST. Among other benefits this can be used to protect errors generated by the referral process.

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 [RFC2119].

3. Requesting a Referral

In order to request referrals as defined in later sections, the Kerberos client MUST explicitly request the canonicalize KDC option (bit 15) [RFC4120] for the AS-REQ or TGS-REQ. This flag indicates to the KDC that the client is prepared to receive a reply that contains a principal name other than the one requested.

```
KDCOptions ::= KerberosFlags
               -- canonicalize (15)
               -- other KDCOptions values omitted
```

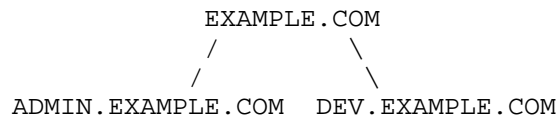
The client should expect, when sending names with the "canonicalize" KDC option, that names in the KDC's reply MAY be different than the name in the request. A referral TGT is a cross realm TGT that is returned with the server name of the ticket being different from the

server name in the request [RFC4120].

4. Realm Organization Model

This memo assumes that the world of principals is arranged on multiple levels: the realm, the enterprise, and the world. A KDC may issue tickets for any principal in its realm or cross-realm tickets for realms with which it has a direct cross-realm relationship. The KDC also has access to a trusted name service that can resolve any name from within its enterprise into a realm closer along the authentication path to the service. This trusted name service removes the need to use an un-trusted DNS lookup for name resolution.

For example, consider the following configuration, where lines indicate cross-realm relationships:



In this configuration, all users in the EXAMPLE.COM enterprise could have principal names such as `alice@EXAMPLE.COM`, with the same realm portion. In addition, servers at EXAMPLE.COM should be able to have DNS host names from any DNS domain independent of what Kerberos realm their principals reside in.

4.1. Trust Assumptions

Two realms participate in any cross-realm relationship: an issuing realm issues a cross-realm ticket and a consuming realm uses this ticket. There is a degree of trust of the issuing realm by the consuming realm implicit in this relationship. Whenever a service in the consuming realm permits an authentication path containing the issuing realm, that service trusts the issuing realm to accurately represent the identity of the authenticated principal and any information about the transited path. If the consuming realm's KDC sets the transited policy checked flag, the KDC is making the same trust assumption a service would.

This trust is transitive across a multi-hop authentication path. The service's realm trusts each hop along the authentication path closer to the client to accurately represent the authenticated identity and to accurately represent transited information. Any KDC along this path could impersonate the client.

KDC signed or issued authorization data often implies additional

trust. The implications of such trust from a security and operational standpoint is an ongoing topic of discussion during the development of this specification. As such, such discussion is out of scope for this memo.

Administrators have several tools to limit trust caused by cross-realm relationships. A service or KDC can control what authentication paths are acceptable. For example if a given realm is not permitted on the authentication path for a particular client then that realm cannot affect trust placed in that client principal. Consuming realms can exercise significant control by deciding what principals to place on an access-control list. If no client using a given issuing realm in authentication paths is permitted to access a resource, then that issuing realm is not trusted in access decisions regarding that resource.

Creating a cross-realm relationship implies relatively little inherent trust in the issuing realm. Significant trust only applies as principals dependent on that issuing realm are given access to resources. However, two deployment constraints may imply significantly greater trust is implied by the initial cross-realm relationship. First, a number of realms provide access to any principal to some resources. Access decisions involving these resources involve a degree of trust in all issuing realms in the transited graph. Secondly, many realms do not significantly constrain what principals users of that realm may grant access. In these realms, creating a cross-realm relationship delegates the decision to trust that realm to users of the consuming realm. In this situation, creating the cross-realm relationship is the primary trust decision point under the administrator's control.

5. Enterprise Principal Name Type

The NT-ENTERPRISE type principal name contains one component, a string of realm-defined content, which is intended to be used as an alias for another principal name in some realm in the enterprise. It is used for conveying the alias name, not for the real principal names within the realms, and thus is only useful when name canonicalization is requested.

The intent is to allow unification of email and security principal names. For example, all users at EXAMPLE.COM may have a client principal name of the form "joe@EXAMPLE.COM" even though the principals are contained in multiple realms. This global name is again an alias for the true client principal name, which indicates what realm contains the principal. Thus, accounts "alice" in the realm DEV.EXAMPLE.COM and "bob" in ADMIN.EXAMPLE.COM may log on as

"alice@EXAMPLE.COM" and "bob@EXAMPLE.COM".

This utilizes a new principal name type, as the KDC-REQ message only contains a single client realm field, and the realm portion of this name corresponds to the Kerberos realm with which the request is made. Thus, the entire name "alice@EXAMPLE.COM" is transmitted as a single component in the client name field of the AS-REQ message, with a name type of NT-ENTERPRISE [RFC4120] (and the local realm name). The KDC will recognize this name type and then transform the requested name into the true principal name if the client account resides in the local realm. The true principal name can have a name type different from the requested name type. Typically the true principal name will be a NT-PRINCIPAL [RFC4120].

6. Name Canonicalization

A service or account may have multiple principal names. For example, if a host is known by multiple names, host-based services on it may be known by multiple names in order to prevent the client from needing a secure directory service to determine the correct hostname to use. In order that the host should not need to be updated whenever a new alias is created, the KDC may provide the mapping information to the client in the credential acquisition process.

If the "canonicalize" KDC option is set, then the KDC MAY change the client and server principal names and types in the AS response and ticket returned from those in the request. Names MUST NOT be changed in the response to a TGS request, although it is common for KDCs to maintain a set of aliases for service principals. Regardless of which alias a client requests, the same service key is used. However, in the TGS request, the client receives a ticket for whichever alias is requested. Services MUST NOT make distinctions based on which alias is in the issued ticket because the service name in a ticket is not cryptographically protected and can be changed by parties other than the KDC.

For example the AS request may specify a client name of "bob@EXAMPLE.COM" as an NT-ENTERPRISE name with the "canonicalize" KDC option set and the KDC will return with a client name of "104567" as a NT-UID.

(It is assumed that the client discovers whether the KDC supports the NT-ENTERPRISE name type via out of band mechanisms.)

See Section 11 for a mechanism to detect modification of the request between the client and KDC. However for best protection, Flexible Authentication through Secure Tunneling (FAST) [RFC6113] or another

mechanism that protects the entire KDC exchange SHOULD be used. Clients MAY reject responses from a KDC where the client or server name is changed if the KDC does not support such a mechanism. Clients SHOULD reject an AS response that changes the server name unless the response is protected by such a mechanism or the new server name is one explicitly expected by the client. For example, many clients permit the realm name to be changed in an AS response even if the response is not protected. See Section 14 for a discussion of the tradeoffs in allowing unprotected responses.

In order to permit authorization decisions to be made based on aliases as well as the canonicalized form of a principal name, the KDC MAY include the following authorization data element, wrapped in AD-KDC-ISSUED, in the initial credentials and copy it from a ticket-granting ticket into additional credentials:

```
AD-LOGIN-ALIAS ::= SEQUENCE { -- ad-type number TBD --  
    login-aliases  [0] SEQUENCE(1..MAX) OF PrincipalName,  
}
```

The login-aliases field lists one or more of the aliases the principal is known by.

In addition to permitting authorization based on aliases, this permits user-to-user exchanges where the party receiving the authenticator knows the other party only by an alias. The recipient of such an authenticator SHOULD check the AD-LOGIN-ALIAS names, if present, in addition to the normal client name field, against the identity of the party with which it wishes to authenticate; either should be allowed to match. (Note that this is not backwards compatible with [RFC4120]; if the server side of the user-to-user exchange does not support this extension, and does not know the true principal name, authentication may fail if the alias is sought in the client name field.)

The use of AD-KDC-ISSUED authorization data elements in cross-realm cases has not been well explored at this writing; hence we will only specify the inclusion of this data in the one-realm case. The alias information SHOULD be dropped in the general cross-realm case. However, a realm MAY implement a policy of accepting and re-signing (wrapping in a new AD-KDC-ISSUED element) alias information provided by certain trusted realms, in the cross-realm ticket-granting service.

The canonical principal name for an alias MUST not be in the form of a ticket-granting service name, as (in a case of server name canonicalization) that would be construed as a case of cross-realm referral, described below.

7. Client Referrals

The simplest form of ticket referral is for a user requesting a ticket using an AS-REQ. In this case, the client machine will send the AS-REQ to a convenient realm trusted to map principals, for example the realm of the client machine. In the case of the name `alice@EXAMPLE.COM`, the client MAY optimistically choose to send the request to `EXAMPLE.COM`. The realm in the AS-REQ is always the name of the realm that the request is for as specified in [RFC4120].

The KDC will try to lookup the name in its local account database. If the account is present in the realm of the request, it SHOULD return a KDC reply structure with the appropriate ticket.

If the account is not present in the realm specified in the request and the "canonicalize" KDC option is set, the KDC may look up the client principal name using some kind of name service or directory service. If this lookup is unsuccessful, it MUST return the error `KDC_ERR_C_PRINCIPAL_UNKNOWN` [RFC4120]. If the lookup is successful, it MUST return an error `KDC_ERR_WRONG_REALM` [RFC4120] and in the error message the `crealm` field will contain either the true realm of the client or another realm that MAY have better information about the client's true realm. The client MUST NOT use the `cname` returned in this error message.

If the client receives a `KDC_ERR_WRONG_REALM` error, it will issue a new AS request with the same client principal name used to generate the first referral to the realm specified by the `realm` field of the Kerberos error message corresponding to the first request. (The client realm name will be updated in the new request to refer to this new realm.) The client SHOULD repeat these steps until it finds the true realm of the client. To avoid infinite referral loops, an implementation should limit the number of referrals. A suggested limit is 5 referrals before giving up.

Since the same client name is sent to the referring and referred-to realms, both realms must recognize the same client names. In particular, the referring realm cannot (usefully) define principal name aliases that the referred-to realm will not know.

The true principal name of the client, returned in AS-REQ, can be validated in a subsequent TGS message exchange where its value is communicated back to the KDC via the authenticator in the PA-TGS-REQ `padata` [RFC4120]. However, this requires trusting the referred-to realm's KDCs. Clients should limit the referral mappings they will accept to realms trusted via some local policy. Some possible factors that might be taken into consideration for such a policy might include:

- o Any realm indicated by the local KDC, if the returned KRB-ERROR message is protected by some additional means, for example FAST
- o A list of realms configured by an administrator
- o Any realm accepted by the user when explicitly prompted

There is currently no provision for changing the client name in a client referral response.

8. Server Referrals

The primary difference in server referrals is that the KDC returns a referral TGT rather than an error message as is done in the client referrals.

If the "canonicalize" flag in the KDC options is set and the KDC doesn't find the principal locally, either as a regular principal or as an alias for another local principal, the KDC MAY return a cross-realm ticket granting ticket to the next hop on the trust path towards a realm that may be able to resolve the principal name.

The client will use this referral information to request a chain of cross-realm ticket granting tickets until it reaches the realm of the server, and can then expect to receive a valid service ticket.

However an implementation should limit the number of referrals that it processes to avoid infinite referral loops. A suggested limit is 5 referrals before giving up.

The client may cache the mapping of the requested name to the name of the next realm to use and the principal name to ask for. (See Section 10.)

Here is an example of a client requesting a service ticket for a service in realm DEV.EXAMPLE.COM where the client is in ADMIN.EXAMPLE.COM.

```
+NC = Canonicalize KDCOption set
C: TGS-REQ sname=http/foo.dev.example.com +NC to ADMIN.EXAMPLE.COM
S: TGS-REP sname=krbtgt/EXAMPLE.COM@ADMIN.EXAMPLE.COM
C: TGS-REQ sname=http/foo.dev.example.com +NC to EXAMPLE.COM
S: TGS-REP sname=krbtgt/DEV.EXAMPLE.COM@EXAMPLE.COM
C: TGS-REQ sname=http/foo.dev.example.com +NC to DEV.EXAMPLE.COM
S: TGS-REP sname=http/foo.dev.example.com@DEV.EXAMPLE.COM
```

Note that any referral or alias processing of the server name in user-to-user authentication should use the same data as client name canonicalization or referral. Otherwise, the name used by one user

to log in may not be useable by another for user-to-user authentication to the first.

9. Cross Realm Routing

RFC 4120 permits a KDC to return a closer referral ticket when a cross-realm TGT is requested. This specification extends this behavior when the canonicalize flag is set. When this flag is set, a KDC MAY return a TGT for a realm closer to the service for any service as discussed in the previous section. When a client follows such a referral, it including the realm of the referred-to realm in the generated request.

10. Caching Information

It is possible that the client may wish to get additional credentials for the same service principal, perhaps with different authorization-data restrictions or other changed attributes. The return of a server referral from a KDC can be taken as an indication that the requested principal does not currently exist in the local realm. Clearly, it would reduce network traffic if the clients could cache that information and use it when acquiring the second set of credentials for a service, rather than always having to re-check with the local KDC to see if the name has been created locally.

When the TGT expires, the previously returned referral from the local KDC should be considered invalid, and the local KDC must be asked again for information for the desired service principal name. (Note that the client may get back multiple referral TGTs from the local KDC to the same remote realm, with different lifetimes. The lifetime information SHOULD be properly associated with the requested service principal names. Simply having another TGT for the same remote realm does not extend the validity of previously acquired information about one service principal name.)

Accordingly, KDC authors and maintainers should consider what factors (e.g., DNS alias lifetimes) they may or may not wish to incorporate into credential expiration times in cases of referrals.

11. Negotiation of FAST and Detecting Modified Requests

Implementations of this specification MUST support the FAST negotiation mechanism described in this section. This mechanism provides detection of KDC requests modified by an attacker when those requests result in a reply instead of an error. In addition, this

mechanism provides a secure way to detect if a KDC supports FAST.

Clients conforming to this specification MUST send a new pre-authentication data of type PA-REQ-ENC-PA-REP (TBD1) in all AS requests and MAY send this padata type in TGS requests. The value of this padata item SHOULD be empty and MUST be ignored by a receiving KDC. Sending this padata item indicates support for this negotiation mechanism. KDCs conforming to this specification must always set the ticket flag enc-pa-rep(15) in all the issued tickets. This ticket flag indicates KDC support for the mechanism.

The KDC response is extended to support an additional field containing encrypted pre-authentication data.

```
EncKDCRepPart ::= SEQUENCE {  
    key                [0] EncryptionKey,  
    last-req           [1] LastReq,  
    nonce              [2] UInt32,  
    key-expiration     [3] KerberosTime OPTIONAL,  
    flags              [4] TicketFlags,  
    authtime           [5] KerberosTime,  
    starttime          [6] KerberosTime OPTIONAL,  
    endtime            [7] KerberosTime,  
    renew-till         [8] KerberosTime OPTIONAL,  
    srealm             [9] Realm,  
    sname              [10] PrincipalName,  
    caddr              [11] HostAddresses OPTIONAL,  
    encrypted-pa-data [12] SEQUENCE OF PA-DATA OPTIONAL  
}
```

The The encrypted-pa-data element MUST be absent unless either the canonicalize KDC option is set or the PA-REQ-ENC-PA-REP padata item is sent.

If the PA-REQ-ENC-PA-REP padata item is sent in the request, then the KDC MUST include a PA-REQ-ENC-PA-REP padata item in the encrypted-pa-data item of any generated KDC reply. The PA-REQ-ENC-PA-REP pa-data value contains the checksum computed over the type AS-REQ or TGS-REQ in the request. The checksum key is the reply key and the checksum type is the required checksum type for the encryption type of the reply key, and the key usage number is KEY_USAGE_AS_REQ (56). If the KDC supports FAST, then the KDC MUST include a padata of type PA-FX-FAST in any encrypted-pa-data sequence it generates. The value for this padata item should be empty.

A client MUST reject a response for which it sent PA-REQ-ENC-PA-REP if the ENC-PA-REP ticket flag is set and the PA-REQ-ENC-PA-REP padata item is absent or the checksum is not successfully verified.

12. Number Assignments

Most number registries in the Kerberos protocol have not been turned over to IANA for management at the time of this writing, hence this is not an "IANA Considerations" section.

Various values do need assigning for this draft:

AD-LOGIN-ALIAS

13. IANA Considerations

In the Kerberos pre-authentication and typed data registry at <http://www.iana.org/assignments/kerberos-parameters/kerberos-parameters.xhtml#pre-authentication>, the PA-REQ-ENC-PA-REP pa-data item should be registered. Because of existing implementations the value 149 is strongly preferred.

14. Security Considerations

For the AS exchange case, it is important that the logon mechanism not trust a name that has not been used to authenticate the user. For example, the name that the user enters as part of a logon exchange may not be the name that the user authenticates as, given that the KDC_ERR_WRONG_REALM error may have been returned. The relevant Kerberos naming information for logon (if any), is the client name and client realm in the service ticket targeted at the workstation that was obtained using the user's initial TGT. That is, rather than trusting the client name in the AS response, a workstation SHOULD perform an AP-REQ authentication against itself as a service and use the client name in the ticket issued for its service by the KDC.

How the client name and client realm is mapped into a local account for logon is a local matter, but the client logon mechanism MUST use additional information such as the client realm and/or authorization attributes from the service ticket presented to the workstation by the user, when mapping the logon credentials to a local account on the workstation.

Not all fields in an RFC 4120 KDC reply are protected. None of the fields in an RFC 4120 AS request are protected and some information in a TGS request may not be protected. The referrals mechanism creates several opportunities for attack because of these unprotected fields. FAST [RFC6113] can be used to completely mitigate these issues by protecting both the KDC request and response. However, FAST requires that a client obtain an armor ticket before

authenticating. Not all realms permit all clients to obtain armor tickets. Also, while it is expected to be uncommon, a client might wish to use name canonicalization while obtaining an armor ticket. The mechanism in Section 11 detects modification of the request between the KDC and client, mitigating some attacks.

There is a wide deployed base of implementations that use name canonicalization or server referrals that uses neither the negotiation mechanism nor FAST. So, implementations may be faced with only the limited protection afforded by RFC 4120, by the negotiation mechanism discussed in this document, or by FAST. All three situations are important to consider from a security standpoint.

An attacker cannot mount a downgrade attack against a client. The negotiation mechanism described in this document is securely indicated by the presence of a ticket flag. So, a client will detect if the facility was available but not used. It is possible for an attacker to strip the indication that a client supports the negotiation facility. The client will learn from the response that this happened, but the KDC will not learn that the client is attacked. So, for a single round-trip Kerberos exchange, the KDC may believe the exchange was successful when the client detects an attack. Packet loss or client failure can produce a similar result; this is not a significant vulnerability. The negotiation facility described in this document securely indicates the presence of FAST, so if a client wishes to use FAST when it is available, an attacker cannot force the client to downgrade away from FAST. An attacker MAY be able to prevent a client from obtaining an armor ticket, for example by responding to a request for anonymous PKINIT with an error response.

If FAST is used, then the communications between the client and KDC are protected. However name canonicalization places a new responsibility for mapping principals onto the KDC. This can increase the number of KDCs involved in an authentication which adds additional trusted third parties to the exchange.

If only the negotiation mechanism is used, then the request from the client to the KDC is protected, but not all of the response is protected. In particular, the client name is not protected; the ticket is also not protected. An attacker can potentially modify these fields. Modification of the client name will result in a denial of service. When the client attempts to authenticate to a service (including the TGS), it constructs an AP-REQ message. This message includes a client name which MUST match the client name in the ticket according to RFC 4120. Thus if the client name is changed, the resulting ticket will fail when used. This is

undesirable because the authentication is separated from the later failure, which may confuse problem determination. If the ticket is replaced with another ticket, then later authentication to a service will fail because the client will not know the session key for the other ticket. If the ticket is simply modified, then authentication to a service will fail as with RFC 4120. More significant attacks are possible if a KDC violates the requirements of RFC 4120 and issues two tickets with the same session key or if a service violates the requirements of RFC 4120 and does not check the client name against that in the ticket.

There is an additional attack possible when FAST is not used against KDC_ERR_WRONG_REALM. Since this is an error response not an AS response, it is not protected by the negotiation mechanism. Thus, an attacker may be able to convince a client to authenticate to a realm other than the one intended. If an attacker is off-path this may give the attacker an advantage in attacking the client's credentials. Also, see the discussion of shared passwords below.

More serious attacks are possible if no protection beyond RFC 4120 is used. In this case, neither the client name nor the service name is protected between the client and KDC. In the general case, if an attacker changes the client name, then authentication will fail because the client will not have the right credentials (password, certificate, or other) to authenticate as the user selected by the attacker. However, see the discussion of shared passwords below. Changing the server name can be a very significant attack. For example if a user is authenticating in order to send some confidential information, then the attacker could gain this information by directing the user to a server under the attacker's control. The server name in the response is protected by RFC 4120, but not the one in the request. Fortunately, users are typically authenticating to the "krbtgt" service in an AS exchange. Clients that permit changes to the server name when no protection beyond RFC 4120 is in use SHOULD carefully restrict what service names are acceptable. One critical case to consider is the password changing service. When a user authenticates to change their password they use an AS authentication directly to the password changing service. Clients MUST restrict service name changes sufficiently that the client ends up talking to the correct password changing service.

14.1. Shared-password case

A special case to examine is when the user is known (or correctly suspected) to use the same password for multiple accounts. A man-in-the-middle attacker can either alter the request on its way to the KDC, changing the client principal name, or reply to the client with a response previously sent by the KDC in response to a request from

the attacker. The response received by the client can then be decrypted by the user, though if the default "salt" generated from the principal name is used to produce the user's key, a PA-ETYPE-INFO or PA-ETYPE-INFO2 preauth record may need to be added or altered by the attacker to cause the client software to generate the key needed for the message it will receive. None of this requires the attacker to know the user's password, and without further checking, could cause the user to unknowingly use the wrong credentials.

In normal [RFC4120] operation, a generated AP-REQ message includes in the Authenticator field a copy of the client's idea of its own principal name. If this differs from the name in the KDC-generated Ticket, the application server will reject the message.

With client name canonicalization as described in this document, the client may get its principal name from the response from the KDC. Using the wrong credentials may provide an advantage to an attacker. For example if a client uses one principal for administrative operations and one for less privileged operation, an attacker may coerce a client into using the wrong privilege to either cause some later operation to succeed or fail.

14.2. Preauthentication data

In cases of credential renewal, forwarding, or validation, if credentials are sent to the KDC that are not an initial ticket-granting ticket for the client's home realm, the encryption key used to protect the TGS exchange is one known to a third party (namely, the service for which the credential was issued). Consequently, in such an exchange, the protection described earlier may be compromised by the service. This is not generally believed to be a problem. If it is, some form of explicit TGS armor could be added to FAST.

15. Acknowledgments

John Brezak, Mike Swift, and Jonathan Trostle wrote the initial version of this document.

Karthik Jaganathan contributed to earlier versions.

Sam Hartman's work on this document was funded by the MIT Kerberos Consortium.

16. References

16.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
- [RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011.

16.2. Informative References

- [RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.
- [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, May 2008.
- [XPR] Trostle, J., Kosinovsky, I., and M. Swift, "Implementation of Crossrealm Referral Handling in the MIT Kerberos Client", Network and Distributed System Security Symposium, February 2001.

Appendix A. Compatibility with Earlier Implementations of Name Canonicalization

The Microsoft Windows 2000 and Windows 2003 releases included an earlier form of name-canonicalization [XPR]. Here are the differences:

- 1) Windows include an additional encrypted padata element. The preauth data type definition in the encrypted preauth data is as follows:

```
PA-SVR-REFERRAL-INFO          20

PA-SVR-REFERRAL-DATA ::= SEQUENCE {
    referred-name    [1] PrincipalName OPTIONAL,
    referred-realm   [0] Realm
}
```

The referred-principal is never sent. The referred-realm is included in TGS replies and includes the realm name of the realm to which the client is referred. This information is redundant with the realm in the second component of the returned TGT.

- 2) When PKINIT ([RFC4556]) is used, the NT-ENTERPRISE client name is encoded as a Subject Alternative Name (SAN) extension [RFC5280] in the client's X.509 certificate. The type of the otherName field for this SAN extension is AnotherName [RFC5280]. The type-id field of the type AnotherName is id-ms-sc-logon-upn (1.3.6.1.4.1.311.20.2.3) and the value field of the type AnotherName is a KerberosString [RFC4120]. The value of this KerberosString type is the single component in the name-string [RFC4120] sequence for the corresponding NT-ENTERPRISE name type.

In Microsoft's current implementation through the use of global catalogs any domain in one forest is reachable from any other domain in the same forest or another trusted forest with 3 or less referrals. A forest is a collection of realms with hierarchical trust relationships: there can be multiple trust trees in a forest; each child and parent realm pair and each root realm pair have bidirectional transitive direct trusts between them.

While we might want to permit multiple aliases to exist and even be reported in AD-LOGIN-ALIAS, the Microsoft implementation permits only one NT-ENTERPRISE alias to exist, so this question had not previously arisen.

Appendix B. Document history [REMOVE BEFORE PUBLICATION]

- 13 Better reflect that we are not solving the gnufpt.raeburn.org use case. Clean up other references to information in padata. Fix the Microsoft appendix based on discussions with them
- 12 Refactor to take advantage of FAST and new protected negotiation mechanism instead of providing our own. Simplify significantly based on this. Remove the true principal name support for now pending discussion in the WG. Add the new protected negotiation mechanism.
- 11 Changed title. Better protection on server referral preauth data. Support server name canonicalization. Rename ReferralInfo to ClientReferralInfo. Disallow alias mapping to a TGT principal. Explain why no name change in client referrals. Add empty IANA Considerations. Add some notes on preauth data protection during renewal etc.

- 10 Separate enterprise principal names into a separate section. Add a little wording to suggest server principal name canonicalization might be allowed; not fleshed out. Advise against AD-KDC-ISSUED in cronn-realm cases. Advise policy checks on returned client referral info, since there's no security. List number assignments. Add security analysis of shared-password case. No longer plan to remove Microsoft appendix. Add referral-valid-until field.
- 09 Changed to EXAMPLE.COM instead of using Morgan Stanley's domain. Rewrote description of existing practice. (Don't name the lookup table consulted. Mention that DNS "canonicalization" is contrary to [RFC4120].) Noted Microsoft behavior should be moved out into a separate document. Changed some second-person references in the introduction to identify the proper parties. Changed PA-CLIENT-CANONICALIZED to use a separate type for the actual referral data, add an extension marker to that type, and change the checksum key from the "returned session key" to the "AS reply key". Changed AD-LOGIN-ALIAS to contain a sequence of names, to be contained in AD-KDC-ISSUED instead of AD-IF-RELEVANT, and to drop the no longer needed separate checksum. Attempt to clarify the cache lifetime of referral information.
- 08 Moved Microsoft implementation info to appendix. Clarify lack of local server name canonicalization. Added optional authz-data for login alias, to support user-to-user case. Added requested-principal-name to ServerReferralData. Added discussion of caching information, and referral TGT lifetime.
- 07 Re-issued with new editor. Fixed up some references. Started history.

Authors' Addresses

Sam hartman (editor)
Painless Security

Email: hartmans-ietf@mit.edu

Kenneth Raeburn
Massachusetts Institute of Technology

Email: raeburn@mit.edu

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

Email: lzhu@microsoft.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 30, 2012

G. Richards
RSA, The Security Division of
EMC
October 28, 2011

OTP Pre-authentication
draft-ietf-krb-wg-otp-preauth-20

Abstract

The Kerberos protocol provides a framework authenticating a client using the exchange of pre-authentication data. This document describes the use of this framework to carry out One Time Password (OTP) authentication.

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 <http://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 April 30, 2012.

Copyright Notice

Copyright (c) 2011 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 (<http://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	4
1.1.	Scope	4
1.2.	Overall Design	4
1.3.	Conventions Used in this Document	5
2.	Usage Overview	5
2.1.	OTP Mechanism Support	5
2.2.	Pre-Authentication	5
2.3.	PIN Change	6
2.4.	Re-Synchronization	7
3.	Pre-Authentication Protocol Details	7
3.1.	Initial Client Request	7
3.2.	KDC Challenge	8
3.3.	Client Response	10
3.4.	Verifying the pre-auth Data	14
3.5.	Confirming the Reply Key Change	15
3.6.	Reply Key Generation	16
4.	OTP Kerberos Message Types	18
4.1.	PA-OTP-CHALLENGE	18
4.2.	PA-OTP-REQUEST	23
4.3.	PA-OTP-PIN-CHANGE	26
5.	IANA Considerations	28
6.	Security Considerations	28
6.1.	Man-in-the-Middle	28
6.2.	Reflection	29
6.3.	Denial of Service	29
6.4.	Replay	30
6.5.	Brute Force Attack	30
6.6.	FAST Facilities	31
7.	Acknowledgments	31
8.	References	31
8.1.	Normative References	31
8.2.	Informative References	32
	Appendix A. ASN.1 Module	33
	Appendix B. Examples of OTP Pre-Authentication Exchanges	35
	B.1. Four Pass Authentication	36
	B.2. Two Pass Authentication	38
	B.3. PIN Change	39
	B.4. Resynchronization	40
	Author's Address	42

1. Introduction

1.1. Scope

This document describes a Flexible Authentication Secure Tunneling (FAST) [RFC6113] factor that allows One-Time Password (OTP) values to be used in the Kerberos V5 [RFC4120] pre-authentication in a manner that does not require use of the user's Kerberos password. The system is designed to work with different types of OTP algorithms such as time-based OTPs [RFC2808], counter-based tokens [RFC4226] and challenge-response systems such as [RFC2289]. It is also designed to work with tokens that are electronically connected to the user's computer via means such as a USB interface.

This FAST factor provides the following facilities (as defined in [RFC6113]): client-authentication, replacing-reply-key and KDC-authentication. It does not provide the strengthening-reply-key facility.

This proposal is partially based upon previous work on integrating single-use authentication mechanisms into Kerberos [HoReNeZo04].

1.2. Overall Design

This proposal supports 4-pass and 2-pass variants. In the 4-pass system, the client sends the KDC an initial AS-REQ and the KDC responds with a KRB-ERROR containing padata that includes a random nonce. The client then encrypts the nonce and returns it to the KDC in a second AS-REQ. Finally, the KDC returns the AS-REP. In the 2-pass variant, the client encrypts a timestamp rather than a nonce from the KDC and the encrypted data is sent to the KDC in the initial AS-REQ. The two-pass system can be used in cases where the client can determine in advance that OTP pre-authentication is supported by the KDC, which OTP key should be used and the encryption parameters required by the KDC.

In both systems, in order to create the message sent to the KDC, the client must generate the OTP value and two keys: the classic Reply Key used to decrypt the KDC's reply and a key to encrypt the data sent to the KDC. In most cases, the OTP value will be used in the key generation but in order to support algorithms where the KDC cannot obtain the value (e.g. [RFC2289]), the system also supports the option of including the OTP value in the request along with the encrypted nonce. In addition, in order to support situations where the KDC is unable to obtain the plaintext OTP value, the system also supports the use of hashed OTP values in the key derivation.

The preauth data sent from the client to the KDC is sent within the

encrypted data provided by the FAST padata type of the AS-REQ. The KDC then obtains the OTP value, generates the same keys and verifies the pre-authentication data by decrypting the nonce. If the verification succeeds then it confirms knowledge of the Reply Key by using it to encrypt data in the AS-REP.

1.3. 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 [RFC2119].

This document assumes familiarity with the Kerberos pre-authentication framework [RFC6113] and so freely uses terminology and notation from this document.

The word padata is used as shorthand for pre-authentication data.

2. Usage Overview

2.1. OTP Mechanism Support

As described above, this document describes a generic system for supporting different OTP mechanisms in Kerberos pre-authentication. To ensure interoperability, all implementations of this specification SHOULD provide a mechanism (e.g. a provider interface) to add or remove support for a particular OTP mechanism.

2.2. Pre-Authentication

The approach uses pre-authentication data in AS-REQ, AS-REP and KRB-ERROR messages.

In the 4-pass system, the client begins by sending an initial AS-REQ to the KDC that may contain pre-authentication data such as the standard Kerberos password data. The KDC will then determine, in an implementation dependent fashion, whether OTP authentication is required and if it is, it will respond with a KRB-ERROR message containing a PA-OTP-CHALLENGE (see Section 4.1) in the PA-DATA.

The PA-OTP-CHALLENGE will contain a KDC generated nonce, a list of hash algorithm identifiers and an iteration count if hashed OTP values are used (see Section 3.6) and OPTIONAL information on how the OTP should be generated by the client. The client will then generate the OTP value and two keys: a Client Key to encrypt the KDC's nonce and a Reply Key used to decrypt the KDC's reply.

As described in section 5.4.1 of [RFC6113], the FAST system uses an Armor Key to set up an encrypted tunnel for use by FAST factors. As described in Section 3.6 of this document, the Client Key and Reply Key will be generated from the Armor Key and the OTP value unless the OTP algorithm does not allow the KDC to obtain the OTP value. If hash algorithm identifiers were included in the PA-OTP-CHALLENGE then the client will use the hash of the OTP value rather than the plaintext value in the key generation. Both keys will have the same encryption type as the Armor Key.

The generated Client Key will be used to encrypt the nonce received from the KDC. The encrypted value along with optional information on how the OTP was generated are then sent to the KDC in a PA-OTP-REQUEST (see Section 4.2) encrypted within the armored-data of a PA-FX-FAST-REQUEST PA-DATA element of a second AS-REQ.

In the 2-pass system, the client sends the PA-OTP-REQUEST in the initial AS-REQ instead of sending it in response to a PA-OTP-CHALLENGE returned by the KDC. Since no challenge is received from the KDC, the client includes an encrypted timestamp in the request rather than the encrypted KDC nonce.

In both cases, on receipt of a PA-OTP-REQUEST, the KDC generates the keys in the same way as the client, and uses the generated Client Key to verify the pre-authentication by decrypting the encrypted data sent by the client (either nonce or timestamp). If the validation succeeds then the KDC will authenticate itself to the client and confirm that the Reply Key has been updated by using the generated Reply Key in the AS-REP response.

2.3. PIN Change

Most OTP tokens involve the use of a PIN in the generation of the OTP value. This PIN value will be combined with the value generated by the token to produce the final OTP value that will be used in this protocol.

If, following successful validation of a PA-OTP-REQUEST in an AS-REQ, the KDC determines that the user's PIN has expired and needs to change then it SHOULD respond KRB-ERROR of type KDC_ERR_PIN_EXPIRED. It MAY include formatting information on the PIN in a PA-OTP-PIN-CHANGE (see Section 4.3) encrypted within the armored data of the PA-FX-FAST-REPLY PA-DATA element.

KDC_ERR_PIN_EXPIRED

96

If the PIN change is to be handled by a PIN-change service then it is assumed that authentication to that service will succeed if the PIN

has expired.

If the user's PIN has not expired but has been changed then the KDC MAY return the new value to the client in a PA-OTP-PIN-CHANGE encrypted within the armored data of the PA-FX-FAST-REPLY PA-DATA element of the AS-REP. Similarly, if PIN change is not required then the KDC MAY return a PA-OTP-PIN-CHANGE to inform the client of the current PIN's expiration time.

2.4. Re-Synchronization

It is possible with time and event-based tokens that the OTP server will lose synchronization with the current token state. For example, event-based tokens may drift since the counter on the token is incremented every time the token is used but the counter on the server is only incremented on an authentication. Similarly, the clocks on time-based tokens may drift.

Methods to recover from this type of situation are OTP algorithm specific but may involve the client sending a sequence of OTP values to allow the server to further validate the correct position in its search window (see section 7.4 of [RFC4226] for an example).

If, when processing a PA-OTP-REQUEST, the pre-authentication validation fails for this reason then the KDC MAY return a KRB-ERROR message. The KRB-ERROR message MAY contain a PA-OTP-CHALLENGE in the PA-DATA with a single otp-tokenInfo representing the token used in the initial authentication attempt but with "nextOTP" flag set. If this flag is set then the client SHOULD re-try the authentication using an OTP value generated using the token in the "state" after that used in the failed authentication attempt. For example, using the next time interval or counter value.

3. Pre-Authentication Protocol Details

3.1. Initial Client Request

In the 4-pass mode, the client begins by sending an initial AS-REQ, possibly containing other pre-authentication data. If the KDC determines that OTP-based pre-authentication is required and the request does not contain a PA-OTP-REQUEST then it will respond as described in Section 3.2.

If the client has all the necessary information, it MAY use the 2-pass system by constructing a PA-OTP-REQUEST as described in Section 3.3 and including it in the initial request.

3.2. KDC Challenge

If the user is required to authenticate using an OTP then the KDC SHALL respond to the initial AS-REQ with a KRB-ERROR, as described in section 2.2 of [RFC6113], with a PA-OTP-CHALLENGE contained within the enc-fast-rep of the armored-data of a PA-FX-FAST-REPLY encrypted under the current Armor Key as described in [RFC6113].

If the OTP mechanism is to be carried out as an individual mechanism then the PA-OTP-CHALLENGE SHALL be carried within the padata of the KrbFastResponse. Alternatively, if the OTP mechanism is required as part of an authentication set then the PA-OTP-CHALLENGE SHALL be carried within a PA-AUTHENTICATION-SET-ELEM as described in section 5.3 of [RFC6113].

The PA-OTP-CHALLENGE SHALL contain a nonce value to be returned encrypted in the client's PA-OTP-REQUEST. This nonce string MUST contain a randomly chosen component at least as long as the armor key length. (See [RFC4086] for an in-depth discussion of > randomness.) In order to allow it to maintain any state necessary to verify the returned nonce, the KDC SHOULD use the mechanism described in section 5.2 of [RFC6113].

The KDC MAY use the otp-service field to assist the client in locating the OTP token to be used by identifying the purpose of the authentication. For example, the otp-service field could assist a user in identifying the token to be used when a user has multiple OTP tokens that are used for different purposes. If the token is a connected device, then these values SHOULD be an exact octet-level match for the values present on the target token.

The KDC SHALL include a sequence of one or more otp-tokenInfo elements containing information on the token or tokens that the user can use for the authentication and how the OTP value is to be generated using those tokens. If a single otp-tokenInfo element is included then only a single token is acceptable by the KDC and any OTP value generated by the client MUST be generated according to the information contained within that element. If more than one otp-tokenInfo element is included then the OTP value MUST be generated according to the information contained within one of those elements.

The KDC MAY include the otp-vendor field in an otp-tokenInfo to identify the vendor of the token that can be used in the authentication request in order to assist the client in locating that token.

If the KDC is able to obtain the OTP values for the token then the OTP value SHOULD be used in the key generation as described in

Section 3.6 and so the KDC SHOULD set the "must-encrypt-nonce" flag in the otp-tokenInfo. If the KDC is unable to obtain the OTP values for the token then the "must-encrypt-nonce" flag MUST NOT be set. If the flag is not set then the OTP value will be returned by the client in the otp-value field of the PA-OTP-REQUEST and so if returning of OTP values in this way does not conform to KDC policy then the KDC SHOULD NOT include the otp-tokenInfo for that token in the PA-OTP-CHALLENGE.

If the KDC requires that hashed OTPs be used in the key generation as described in Section 3.6 (for example, it is only able to obtain hashed OTP values for the token) then it MUST include the supported hash algorithms in order of preference in the supportedHashAlg of the otp-KeyInfo and the minimum value of the iteration count in the iterationCount element.

Since the OTP mechanism described in this document is replacing the Reply Key, the classic shared-key system cannot be relied upon to allow the client to verify the KDC. Therefore, as described in section 3.4 of [RFC6113], some other mechanism must be provided to support this. If the OTP value is used in the Reply Key generation then the client and KDC have a shared key and KDC-authentication is provided by the KDC using the Reply Key generated from the OTP value. However, if the OTP value is sent in the otp-value element of the PA-OTP-REQUEST then there is no such shared key and the OTP mechanism does not provide KDC-authentication. Therefore, if the OTP mechanism is not being used in an environment where KDC-authentication is being provided by other means (e.g. by the use of host key armor) then the KDC MUST NOT include any otp-tokenInfo elements in the PA-OTP-CHALLENGE that do not have the "must-encrypt-nonce" flag set.

If the OTP for a token is to be generated using a server generated challenge then the value of the challenge SHALL be included in the otp-challenge field of the otp-tokenInfo for that token. If the token is a connected device and the OTP is to be generated by combining the challenge with the token's current state (e.g. time) then the "combine" flag SHALL be set within the otp-tokenInfo containing the challenge.

If the KDC can determine which OTP token key (the seed value on the token used to generate the OTP) is to be used, then the otp-tokenID field MAY be included in the otp-tokenInfo to pass that value to the client.

The otp-algID field MAY be included in an otp-tokenInfo to identify the algorithm that should be used in the OTP calculation for that token. For example, it could be used when a user has been issued with multiple tokens that support different algorithms.

If the KDC can determine that an OTP token that can be used by the user does not require the client to collect a PIN then it SHOULD set the "do-not-collect-pin" flag in the otp-tokenInfo representing that token. If the KDC can determine that the token requires the client to collect a PIN then it SHOULD set the "collect-pin" flag. If the KDC is unable to determine whether the client should collect a PIN or not then the "collect-pin" and "do-not-collect-pin" flags MUST NOT be set.

If the KDC requires the PIN of an OTP token to be returned to it separately then it SHOULD set the "separate-pin-required" flag in the otp-KeyInfo representing that token.

If the KDC requires that the OTPs generated by the token have a Luhn check digit appended, as defined in [ISOIEC7812], then it MUST set the "check-digit" flag. This flag only applies if the format of the OTP is decimal and so the otp-format field, if present, MUST have the value of "decimal".

Finally, in order to support connected tokens that can generate OTP values of varying lengths or formats, the KDC MAY include the desired length and format of the OTP in the otp-length and otp-format fields of an otp-tokenInfo.

3.3. Client Response

The client response SHALL be sent to the KDC as a PA-OTP-REQUEST included within the enc-fast-req of the armored-data within a PA-FX-FAST-REQUEST encrypted under the current Armor Key as described in [RFC6113].

In order to generate its response, the client MUST generate an OTP value. If the PA-OTP-CHALLENGE contained one or more otp-tokenInfo elements then the OTP value MUST be based on the information contained within one of those elements.

The otp-service, otp-vendor, otp-tokenID, otp-length, otp-format and otp-algID elements of the PA-OTP-CHALLENGE are provided by the KDC to assist the client in locating the correct token to use but the use of the above fields will depend on the type of token.

If the token is a disconnected device, then the values of otp-service and otp-vendor MAY be displayed to the user in order to help the user select the correct token and the values of otp-algID, otp-tokenID, otp-length and otp-format MAY be ignored.

If the token is a connected device, then these values, if present, SHOULD be used by the client to locate the correct token. When the

token is connected, clients MUST support matching based on a binary comparison of the otp-vendor and otp-service strings when comparing the values against those present on the token. Clients MAY have other comparisons including normalization insensitive comparisons to try and find the right token. The values of otp-vendor and otp-service MAY be displayed to prompt the user if the correct token is not found.

If the "nextOTP" flag is set in the otp-tokenInfo from the PA-OTP-CHALLENGE, then the OTP value MUST be generated from the next token state than that used in the previous PA-OTP-REQUEST for that token. The "nextOTP" flag MUST also be set in the new PA-OTP-REQUEST.

If the "collect-pin" flag is set then the token requires a PIN to be collected by the client. If the "do-not-collect-pin" flag is set in the otp-tokenInfo from the PA-OTP-CHALLENGE, then the token represented by the otp-tokenInfo does not require a PIN to be collected by the client as part of the OTP value. If neither of the "collect-pin" nor "do-not-collect-pin" flags are set then PIN requirements of the token are unspecified. If both flags are set then client SHALL regard the request as invalid.

If the "separate-pin-required" flag is set then any PIN collected by the client MUST be included as a UTF-8 string in the otp-pin of the PA-OTP-REQUEST.

If the token is a connected device, then how the PIN is used to generate the OTP value will depend on the type of device. However, if the token is a disconnected device, then it will depend on the "separate-pin-required" flag. If the flag is not set then the OTP value MUST be generated by appending the PIN with the value from the token entered by the user and, if the flag is set, then the OTP value MUST be the value from the token.

The clients SHOULD NOT normalize the PIN value or any OTP value collected from the user or returned by a connected token in any way.

If the "check-digit" flag is set then any OTP values SHOULD be decimal and have a Luhn check digit appended [ISOIEC7812]. If the token is disconnected then the Client MAY ignore this flag but if the token is connected then the Client MUST enforce it. The Client MUST regard the request as invalid if otp-format is present and set to any value other than "decimal".

If an otp-challenge is present in the otp-tokenInfo selected by the client from the PA-OTP-CHALLENGE then the OTP value for the token MUST be generated based on a challenge if the token is capable of accepting a challenge. The client MAY ignore the provided challenge

if and only if the token is not capable of including a challenge in the OTP calculation.

If the "combine" flag is not set in the otp-tokenInfo of the PA-OTP-CHALLENGE then the OTP SHALL be calculated based only the challenge and not the internal state (e.g. time or counter) of the token. If the "combine" flag is set then the OTP SHALL be calculated using both the internal state and the provided challenge if that value is obtainable by the client. If the flag is set but otp-challenge is not present then the client SHALL regard the request as invalid.

If token is a connected device then the use of the challenge will depend on the type of device but will involve passing the challenge and the value of the "combine" flag in a token-specific manner to the token, along with a PIN if collected and the values of otp-length and otp-format if specified, in order to obtain the OTP value. If the token is disconnected then the challenge MUST be displayed to the user and the value of the "combine" flag MAY be ignored by the client.

If the OTP value was generated using a challenge that was not sent by the KDC then the challenge SHALL be included in the otp-challenge of the PA-OTP-REQUEST. If the OTP was generated by combining a challenge (either received from the KDC or generated by the client) with the token state then the "combine" flag SHALL be set in the PA-OTP-REQUEST.

If the "must-encrypt-nonce" flag is set in the otp-tokenInfo then the OTP value MUST be used to generate the Client Key and Reply Key as described in Section 3.6 and MUST NOT be included in the otp-value field of the PA-OTP-REQUEST. If the flag is not set then the OTP value MUST be included in the otp-value field of the PA-OTP-REQUEST and MUST NOT be used in the key derivation. In this case, the Client Key and Reply Key SHALL be the same as the Armor Key as described in Section 3.6 and so if the returning of OTP values in this way does not conform to local policy on the client (for example, if KDC-Authentication is required and is not being provided by other means) then it SHOULD NOT use the token for authentication.

If the supportedHashAlg and iterationCount elements are included in the otp-tokenInfo then the client MUST use hashed OTP values in the generation of the Reply Key and Client Key as described in Section 3.6. The client MUST select the first algorithm from the list that it supports and the AlgorithmIdentifier [RFC5280] selected MUST be placed in the hashAlg element of the PA-OTP-REQUEST. However, if none of the algorithm identifiers conform to local policy restrictions then the authentication attempt MUST NOT proceed using that token. If the value of iterationCount does not conform to local

policy on the client then the client MAY use a larger value but MUST NOT use a lower value. The value of the iteration count used by the client MUST be returned in the PA-OTP-REQUEST sent to the KDC.

If hashed OTP values are used then the nonce generated by the client MUST be as long as the longest key length of the symmetric key types that the it supports and MUST be chosen randomly. (See [RFC4086].) The nonce MUST be included in the PA-OTP-REQUEST along with the hash algorithm and iteration count used in the nonce, hashAlg and iterationCount fields of the PA-OTP-REQUEST. These fields MUST NOT be included if hashed OTP values were not used. It is RECOMMENDED that the iteration count used by the client be chosen in such a way that it is computationally infeasible/unattractive for an attacker to brute-force search for the given OTP.

The PA-OTP-REQUEST returned by the client SHOULD include information on the generated OTP value reported by the OTP token when available to the client. The otp-time and otp-counter fields of the PA-OTP-REQUEST SHOULD be used to return the time and counter values used by the token if available to the client. The otp-format field MAY be used to report the format of the generated OTP. This field SHOULD be used if a token can generate OTP values in multiple formats. The otp-algID field SHOULD be used by the client to report the algorithm used in the OTP calculation and the otp-tokenID SHOULD be used to report the identifier of the OTP token key used if the information is known to the client.

If the PA-OTP-REQUEST is being sent in response to a PA-OTP-CHALLENGE that contained an otp-vendor field in the selected otp-tokenInfo then the otp-vendor field of the PA-OTP-REQUEST MUST be set to the same value. If no otp-vendor field was provided by the KDC then the field SHOULD be set to the vendor identifier of the token if known to the client.

The generated Client Key is used by the client to encrypt data to be included in the encData of the PA-OTP-REQUEST to allow the KDC to authenticate the user. The key usage for this encryption is KEY_USAGE_OTP_REQUEST.

- o If the PA-OTP-REQUEST is being generated in response to a PA-OTP-CHALLENGE returned by the KDC then the client SHALL encrypt a PA-OTP-ENC-REQUEST containing the value of nonce from the PA-OTP-CHALLENGE using the same encryption type as the Armor Key.
- o If the PA-OTP-REQUEST is not in response to a PA-OTP-CHALLENGE then the client SHALL encrypt a PA-ENC-TS-ENC containing the current time as in the encrypted timestamp pre-authentication mechanism [RFC4120].

If the client is working in 2-pass mode and so is not responding to an initial KDC challenge then the values of the iteration count and hash algorithms cannot be obtained from that challenge. The client SHOULD use any values obtained from a previous PA-OTP-CHALLENGE or, if no values are available, it MAY use initial configured values.

3.4. Verifying the pre-auth Data

The KDC validates the pre-authentication data by generating the Client Key and Reply Key in the same way as the client and using the generated Client Key to decrypt the value of encData from the PA-OTP-REQUEST. The generated Reply Key is used to encrypt data in the AS-REP.

If the otp-value field is included in the PA-OTP-REQUEST then the KDC MUST use that value unless the OTP method is required to support KDC-authentication (see Section 3.2). If the otp-value is not included in the PA-OTP-REQUEST then the KDC will need to generate or obtain the OTP value.

If the otp-pin field is present in the PA-OTP-REQUEST then the PIN value has been provided by the client. The KDC SHOULD SASLprep [RFC4013] the value before comparison.

It should be noted that it is anticipated that, as improved string comparison technologies are standardized, the processing done by the KDC will change but efforts will be made to maintain as much compatibility with SASLprep as possible.

If the otp-challenge field is present, then the OTP was calculated using that challenge. If the "combine" flag is also set, then the OTP was calculated using the challenge and the token's current state.

It is RECOMMENDED that the KDC acts upon the values of otp-time, otp-counter, otp-format, otp-algID and otp-tokenID if they are present in the PA-OTP-REQUEST. If the KDC receives a request containing these values but cannot act upon them then they MAY be ignored.

The KDC generates the Client Key and Reply Key as described in Section 3.6 from the OTP value using the nonce, hash algorithm and iteration count if present in the PA-OTP-REQUEST. The KDC MUST fail the request with KDC_ERR_INVALID_HASH_ALG if the KDC requires hashed OTP values and the hashAlg field was not present in the PA-OTP-REQUEST or if the value of this field does not conform to local KDC policy. Similarly, the KDC MUST fail the request with KDC_ERR_INVALID_ITERATION_COUNT if the value of the iterationCount included in the PA-OTP-REQUEST does not conform to local KDC policy or is less than that specified in the PA-OTP-CHALLENGE. In addition,

the KDC MUST fail the authentication request with KDC_ERR_PIN_REQUIRED if it requires a separate PIN to the OTP value and an otp-pin was not included in the PA-OTP-REQUEST

KDC_ERR_INVALID_HASH_ALG	94
KDC_ERR_INVALID_ITERATION_COUNT	95
KDC_ERR_PIN_REQUIRED	97

The generated Client Key is then used to decrypt the encData from the PA-OTP-REQUEST. If the client response was sent as a result of a PA-OTP-CHALLENGE then the decrypted data will be a PA-OTP-ENC-REQUEST and the client authentication MUST fail with KDC_ERR_PREAUTH_FAILED if the nonce value from the PA-OTP-ENC-REQUEST is not the same as the nonce value sent in the PA-OTP-CHALLENGE. If the response was not sent as a result of a PA-OTP-CHALLENGE then the decrypted value will be a PA-ENC-TS-ENC and the authentication process will be the same as with classic encrypted timestamp pre-authentication [RFC4120]

The KDC MUST fail the request with KDC_ERR_ETYPE_NOSUPP if the encryption type used by the client in the encData does not conform to KDC policy.

If authentication fails due to the hash algorithm, iteration count or encryption type used by the client then the KDC SHOULD return a PA-OTP-CHALLENGE with the required values in the error response. If the authentication fails due to the token state on the server no longer being synchronized with the token used then the KDC MAY return a PA-OTP-CHALLENGE with the "nextOTP" flag set as described in Section 2.4.

If, during the authentication process, the KDC determines that the user's PIN has been changed then it SHOULD include a PA-OTP-PIN-CHANGE in the response as described in Section 2.3 containing the new PIN value. The KDC MAY also include the new PIN's expiration time and the expiration time of the OTP account within the last-req field of the PA-OTP-PIN-CHANGE. (These fields can be used by the KDC to handle cases where the account related to the user's OTP token has a different expiration time to the user's Kerberos account.) If the KDC determines that the user's PIN or OTP account are about to expire, it MAY return a PA-OTP-PIN-CHANGE with that information. Finally, if the KDC determines that the user's PIN has expired then it SHOULD return a KRB-ERROR of type KDC_ERR_PIN_EXPIRED as described in Section 2.3

3.5. Confirming the Reply Key Change

If the pre-authentication data was successfully verified, then, in order to support mutual authentication, the KDC SHALL respond to the

client's PA-OTP-REQUEST by using the generated Reply Key to encrypt the data in the AS-REP.

The client then uses its generated Reply Key to decrypt the encrypted data and MUST NOT continue with the authentication process if decryption is not successful.

3.6. Reply Key Generation

In order to authenticate the user, the client and KDC need to generate two encryption keys:

- o The Client Key to be used by the client to encrypt and by the KDC to decrypt the encData in the PA-OTP-REQUEST.
- o The Reply Key to be used in the standard manner by the KDC to encrypt data in the AS-REP.

The method used to generate the two keys will depend on the OTP algorithm.

- o If the OTP value is included in the otp-value of the PA-OTP-REQUEST then the two keys SHALL be the same as the Armor Key (defined in [RFC6113]).
- o If the OTP value is not included in the otp-value of the PA-OTP-REQUEST then the two keys SHALL be derived from the Armor Key and the OTP value as described below.

If the OTP value is not included in the PA-OTP-REQUEST, then the Reply Key and Client Key SHALL be generated using the KRB-FX-CF2 algorithm from [RFC6113] as follows:

```
Client Key = KRB-FX-CF2(K1, K2, O1, O2)
Reply Key = KRB-FX-CF2(K1, K2, O3, O4)
```

The octet string parameters, O1, O2, O3 and O4, shall be the ASCII string "OTPComb1", "OTPComb2", "OTPComb3" and "OTPComb4" as shown below:

```
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x31}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x32}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x33}
{0x4f, 0x54, 0x50, 0x43, 0x6f, 0x6d, 0x62, 0x34}
```

The first input key, K1, SHALL be the Armor Key and so, as described in section 5.1 of [RFC6113], the encetypes of the generated Client Key and Reply Key will be the same as the enctype of Armor Key. The

second input key, K2, shall be derived from the OTP value using string-to-key (defined in [RFC3961]) as follows.

If the hash of the OTP value is to be used then K2 SHALL be derived as follows:

- o An initial hash value, H, is generated:

$$H = \text{hash}(\text{realm}|\text{nonce}|\text{OTP})$$

Where:

- * "|" denotes concatenation
 - * hash is the hash algorithm selected by the client.
 - * realm is the name of the server's realm as carried in the realm field of the AS-REQ. (Not including the tag and length from the DER encoding.)
 - * nonce is the value of the random nonce value generated by the client and carried in the nonce field of the PA-OTP-REQUEST. (Not including the tag and length from the DER encoding.)
 - * If the OTP format is decimal, hexadecimal or alphanumeric, then OTP is the value of the OTP generated as described in Section 3.3 with SASLprep [RFC4013] applied, otherwise it is the unnormalized OTP value.
- o The initial hash value is then hashed iterationCount-1 times to produce a final hash value, H'. (Where iterationCount is the value from the PA-OTP-REQUEST.)

$$H' = \text{hash}(\text{hash}(\dots(\text{iterationCount}-1 \text{ times})\dots(H)))$$

- o The value of K2 is then derived from the Base64 [RFC2045] encoding of this final hash value.

$$K2 = \text{string-to-key}(\text{Base64}(H')|"Krb-preAuth")$$

If the hash value is not used, then K2 SHALL be derived from the base64 encoding of the OTP value.

$$K2 = \text{string-to-key}(\text{Base64}(\text{OTP})|"Krb-preAuth")$$

The enctype used for string-to-key SHALL be that of the Armor Key and the salt and any additional parameters for string-to-key MAY be provided by the KDC in the PA-OTP-CHALLENGE. If the salt and string-to-key parameters are not provided then the default values defined for the particular enctype SHALL be used.

If the strengthen-key is present in KrbFastResponse, then it is combined with the Reply Key to generate the final AS-REQ as described

in [RFC6113]. The strengthen-key does not influence the Client Key.

4. OTP Kerberos Message Types

4.1. PA-OTP-CHALLENGE

The padata-type PA-OTP-CHALLENGE is returned by the KDC to the client in the enc-fast-rep of a PA-FX-FAST-REPLY in the PA-DATA of a KRB-ERROR when OTP pre-authentication is required. The corresponding padata-value field contains the Distinguished Encoding Rules (DER) [X.680] [X.690] encoding of a PA-OTP-CHALLENGE containing a server generated nonce and information for the client on how to generate the OTP.

PA-OTP-CHALLENGE 141

```

PA-OTP-CHALLENGE ::= SEQUENCE {
    nonce                [0] OCTET STRING,
    otp-service           [1] UTF8String OPTIONAL,
    otp-tokenInfo         [2] SEQUENCE (SIZE(1..MAX)) OF
                                OTP-TOKENINFO,
    salt                  [3] KerberosString OPTIONAL,
    s2kparams              [4] OCTET STRING OPTIONAL,
    ...
}

```

```

OTP-TOKENINFO ::= SEQUENCE {
    flags                 [0] OTPFlags,
    otp-vendor            [1] UTF8String OPTIONAL,
    otp-challenge         [2] OCTET STRING (SIZE(1..MAX))
                                OPTIONAL,
    otp-length            [3] Int32 OPTIONAL,
    otp-format            [4] OTPFormat OPTIONAL,
    otp-tokenID           [5] OCTET STRING OPTIONAL,
    otp-algID             [6] AnyURI OPTIONAL,
    supportedHashAlg      [7] SEQUENCE OF AlgorithmIdentifier
                                OPTIONAL,
    iterationCount        [8] Int32 OPTIONAL,
    ...
}

```

```

OTPFormat ::= INTEGER {
    decimal(0),
    hexadecimal(1),
    alphanumeric(2),
    binary(3),
    base64(4)
}

```

```

OTPFlags ::= KerberosFlags
-- reserved(0),
-- nextOTP(1),
-- combine(2),
-- collect-pin(3),
-- do-not-collect-pin(4),
-- must-encrypt-nonce (5),
-- separate-pin-required (6),
-- check-digit (7)

```


nonce

A KDC-supplied nonce value to be encrypted by the client in the PA-OTP-REQUEST. This nonce string **MUST** contain a randomly chosen component at least as long as the armor key length.

otp-service

Use of this field is **OPTIONAL**, but **MAY** be used by the KDC to assist the client to locate the appropriate OTP tokens to be used. For example, this field could be used when a user has multiple OTP tokens for different purposes.

otp-tokenInfo

This element **MUST** be included and it is a sequence of one or more OTP-TOKENINFO objects containing information on the token or tokens that the user can use for the authentication and how the OTP value is to be generated using those tokens. If a single OTP-TOKENINFO object is included then only a single token is acceptable by the KDC and any OTP value generated by the client **MUST** be generated according to the information contained within that element. If more than one OTP-TOKENINFO object is included then the OTP value **MUST** be generated according to the information contained within one of those objects.

flags

If the "nextOTP" flag is set then the OTP **SHALL** be based on the next token "state" rather than the one used in the previous authentication. As an example, for a time-based token, this means the next time slot and for an event-based token, this could mean the next counter value. If the "nextOTP" flag is set then there **MUST** only be a single otp-tokenInfo element in the PA-OTP-CHALLENGE.

The "combine" flag controls how the challenge included in otp-challenge shall be used. If the flag is set then OTP **SHALL** be calculated using the challenge from otp-challenge and the internal token state (e.g. time or counter). If the "combine" flag is not set then the OTP **SHALL** be calculated based only on the challenge. If the flag is set and otp-challenge is not present then the request **SHALL** be regarded as invalid.

If the "do-not-collect-pin" flag is set then the token represented by the current otp-tokenInfo does not require a PIN to be collected as part of the OTP. If the "collect-pin" flag is set then the token requires a PIN. If neither flag is set then whether or not a PIN is required is unspecified. The flags are mutually exclusive and so both flags **MUST NOT** be set, or the client **MUST** regard the request as invalid.

If the "must-encrypt-nonce" flag is set then the OTP value MUST NOT be included in the otp-value field of the PA-OTP-REQUEST but instead MUST be used in the generation of the Reply Key and Client Key as described in Section 3.6.

If the "separate-pin-required" flag is set then the PIN collected by the client SHOULD NOT be used in the generation of the OTP value and SHOULD be returned in the otp-pin field of the PA-OTP-REQUEST.

The "check-digit" flag controls whether or not the OTP values generated by the token need to include a Luhn check digit [ISOIEC7812]. If the token is disconnected then the Client MAY ignore this flag but if this flag is set and the token is connected then the OTP MUST be decimal with a check digit appended.

otp-vendor

Use of this field is OPTIONAL, but MAY be used by the KDC to identify the vendor of the OTP token to be used.

otp-challenge

The otp-challenge is used by the KDC to send a challenge value for use in the OTP calculation. The challenge is an OPTIONAL octet string that SHOULD be uniquely generated for each request in which it is present. When the challenge is not present, the OTP will be calculated on the current token state only. The client MAY ignore a provided challenge if and only if the OTP token the client is interacting with is not capable of including a challenge in the OTP calculation. In this case, KDC policies will determine whether to accept a provided OTP value or not.

otp-length

Use of this field is OPTIONAL, but MAY be used by the KDC to specify the desired length of the generated OTP. For example, this field could be used when a token is capable of producing OTP values of different lengths. If the format of the OTP is 'decimal', 'hexidecimal' or 'alphanumeric' then this value indicates the desired length in digits/characters, if the OTP format is 'binary' then this value indicates the desired length in octets and if the OTP format is 'base64' then this value indicates the desired length of the unencoded OTP value in octets.

otp-format

Use of this field is OPTIONAL, but MAY be used by the KDC to specify the desired format of the generated OTP value. For example, this field could be used when a token is capable of producing OTP values of different formats.

otp-tokenID

Use of this field is OPTIONAL, but MAY be used by the KDC to identify which token key should be used for the authentication. For example, this field could be used when a user has been issued multiple token keys by the same server.

otp-algID

Use of this field is OPTIONAL, but MAY be used by the KDC to identify the algorithm to use when generating the OTP. The value of this field MUST be a URI [RFC3986] and SHOULD be obtained from the PSKC algorithm registry [RFC6030].

supportedHashAlg

If present then a hash of the OTP value MUST be used in the key derivation rather than the plain text value. Each AlgorithmIdentifier identifies a hash algorithm that is supported by the KDC in decreasing order of preference. The client MUST select the first algorithm from the list that it supports. Support for SHA-256 by both the client and KDC is REQUIRED. The AlgorithmIdentifier selected by the client MUST be placed in the hashAlg element of the PA-OTP-REQUEST.

iterationCount

The minimum value of the iteration count to be used by the client when hashing the OTP value. This value MUST be present if supportedHashAlg is present and otherwise MUST NOT be present. If the value of this element does not conform to local policy on the client then the client MAY use a larger value but MUST NOT use a lower value. The value of the iteration count used by the client MUST be returned in the PA-OTP-REQUEST sent to the KDC.

salt

The salt value to be used in string-to-key when used to calculate the keys as described in Section 3.6.

s2kparams

Any additional parameters required by string-to-key as described in Section 3.6.

4.2. PA-OTP-REQUEST

The padata-type PA-OTP-REQUEST is sent by the client to the KDC in the KrbFastReq padata of a PA-FX-FAST-REQUEST that is included in the PA-DATA of an AS-REQ. The corresponding padata-value field contains the DER encoding of a PA-OTP-REQUEST.

The message contains pre-authentication data encrypted by the client using the generated Client Key and optional information on how the OTP was generated. It may also, optionally, contain the generated OTP value.

```

PA-OTP-REQUEST      142

PA-OTP-REQUEST ::= SEQUENCE {
    flags             [0]  OTPFlags,
    nonce             [1]  OCTET STRING                OPTIONAL,
    encData           [2]  EncryptedData,
                        -- PA-OTP-ENC-REQUEST or PA-ENC-TS-ENC
                        -- Key usage of KEY_USAGE_OTP_REQUEST
    hashAlg           [3]  AlgorithmIdentifier          OPTIONAL,
    iterationCount    [4]  Int32                        OPTIONAL,
    otp-value         [5]  OCTET STRING                OPTIONAL,
    otp-pin           [6]  UTF8String                  OPTIONAL,
    otp-challenge     [7]  OCTET STRING (SIZE(1..MAX))  OPTIONAL,
    otp-time          [8]  KerberosTime                OPTIONAL,
    otp-counter       [9]  OCTET STRING                OPTIONAL,
    otp-format        [10] OTPFormat                   OPTIONAL,
    otp-tokenID       [11] OCTET STRING                OPTIONAL,
    otp-algID         [12] AnyURI                      OPTIONAL,
    otp-vendor        [13] UTF8String                  OPTIONAL,
    ...
}

KEY_USAGE_OTP_REQUEST  45

PA-OTP-ENC-REQUEST ::= SEQUENCE {
    nonce             [0] OCTET STRING,
    ...
}

```

flags

This field MUST be present.

If the "nextOTP" flag is set then the OTP was calculated based on the next token "state" rather than the current one. This flag MUST be set if and only if it was set in a corresponding PA-OTP-CHALLENGE.

If the "combine" flag is set then the OTP was calculated based on a challenge and the token state.

No other OTPFlag bits are applicable and MUST be ignored by the KDC.

nonce

This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the nonce value generated by the client and used in the generation of hashed OTP values as described in Section 3.6. This nonce string MUST be as long as the longest key length of the symmetric key types that the client supports and MUST be chosen randomly.

encData

This field MUST be present and MUST contain the pre-authentication data encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE then this MUST contain a PA-OTP-ENC-REQUEST with the nonce from the PA-OTP-CHALLENGE. If no challenge was received then this MUST contain a PA-ENC-TS-ENC.

hashAlg

This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the AlgorithmIdentifier of the hash algorithm used. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE then the AlgorithmIdentifier MUST be the first one supported by the client from the supportedHashAlg of the PA-OTP-CHALLENGE.

iterationCount

This field MUST be present if a hashed OTP value was used as input to string-to-key (see Section 3.6) and MUST NOT be present otherwise. If present, it MUST contain the iteration count used when hashing the OTP value. If the PA-OTP-REQUEST is sent as a result of a PA-OTP-CHALLENGE then the value MUST NOT be less than that specified in the PA-OTP-CHALLENGE.

otp-value

The generated OTP value. This value MUST NOT be present if the "must-encrypt-nonce" flag was set in the PA-OTP-CHALLENGE.

otp-pin

The OTP PIN value entered by the user. This value MUST NOT be present unless the "separate-pin-required" flag was set in the PA-OTP-CHALLENGE.

otp-challenge

Value used by the client in the OTP calculation. It MUST be sent to the KDC if and only if the value would otherwise be unknown to the KDC. For example, the token or client modified or generated challenge.

otp-time

This field MAY be included by the client to carry the time value as reported by the OTP token. Use of this element is OPTIONAL but it MAY be used by a client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-counter

This field MAY be included by the client to carry the token counter value, as reported by the OTP token. Use of this element is OPTIONAL but it MAY be used by a client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-format

This field MAY be used by the client to send the format of the generated OTP as reported by the OTP token. Use of this element is OPTIONAL but it MAY be used by the client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-tokenID

This field MAY be used by the client to send the identifier of the token key used, as reported by the OTP token. Use of this field is OPTIONAL but MAY be used by the client to simplify the authentication process by identifying a particular token key associated with the user. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

otp-algID

This field MAY be used by the client to send the identifier of the OTP algorithm used, as reported by the OTP token. Use of this element is OPTIONAL but it MAY be used by the client to simplify the OTP calculations carried out by the KDC. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value. The value of this field MUST be a URI [RFC3986] and SHOULD be obtained from the PSKC algorithm registry [RFC6030].

otp-vendor

If the PA-OTP-REQUEST is being sent in response to a PA-OTP-CHALLENGE that contained an otp-vendor field in the selected otp-tokenInfo then this field MUST be set to the same value, otherwise, this field SHOULD be set to the vendor identifier of the token if known to the client. It is RECOMMENDED that the KDC act upon this value if it is present in the request and it is capable of using it in the generation of the OTP value.

4.3. PA-OTP-PIN-CHANGE

The padata-type PA-OTP-PIN-CHANGE is returned by the KDC in the enc-fast-rep of a PA-FX-FAST-REPLY in the AS-REP if the user must change their PIN, if the user's PIN has been changed or to notify the user of the PIN's expiry time.

The corresponding padata-value field contains the DER encoding of a PA-OTP-PIN-CHANGE.

PA-OTP-PIN-CHANGE 144

```
PA-OTP-PIN-CHANGE ::= SEQUENCE {  
    flags      [0] PinFlags,  
    pin        [1] UTF8String OPTIONAL,  
    minLength  [2] INTEGER    OPTIONAL,  
    maxLength  [3] INTEGER    OPTIONAL,  
    last-req   [4] LastReq    OPTIONAL,  
    format     [5] OTPFormat  OPTIONAL,  
    ...  
}
```

```
PinFlags ::= KerberosFlags  
-- reserved(0),  
-- systemSetPin(1),  
-- mandatory(2)
```

flags

The "systemSetPin" flag is used to indicate the type of PIN change that is taking place. If the flag is set then the user's PIN has been changed for the user by the system. If the flag is not set then the user's PIN needs to be changed by the user.

If the "systemSetPin" flag is not set and the "mandatory" flag is set then user PIN change is required before the next authentication using the current OTP token. If the "mandatory" flag is not set then the user PIN change is optional. If the "systemSetPin" flag is set then the "mandatory" flag has no meaning and SHOULD be ignored by the client.

pin

The pin field is used to carry a new PIN value. If the "systemSetPin" flag is set then field is used to carry the new PIN value set for the user and MUST be present. If the "systemSetPin" flag is not set then use of this field OPTIONAL and MAY be used to carry a system generated PIN that MAY be used by the user when changing the PIN.

minLength and maxLength

Use of the minLength and maxLength fields is OPTIONAL. If the "systemSetPin" flag is not set then these fields MAY be included to pass restrictions on the size of the user selected PIN.

last-req

Use of the last-req field (as defined in section 5.4.2 of [RFC4120]) is OPTIONAL but MAY be included with an lr-type of 6 to carry PIN expiration information.

- * If the "systemSetPin" flag is set then the expiration time MUST be that of the new system-set PIN.
- * If the "systemSetPin" flag is not set then the expiration time MUST be that of the current PIN of the token used in the authentication.

The element MAY also be included with an lr-type of 7 to indicate when the OTP account will expire.

format

The format element MAY be included by the KDC to carry PIN format restrictions on the new PIN.

- * If the "systemSetPin" flag is set then the element MUST describe the format of the new system-generated PIN.

- * If the "systemSetPin" flag is not set then the element MUST describe restrictions on any new user generated PIN.

5. IANA Considerations

The OTP algorithm identifier URIs used as otp-algID values in the PA-OTP-CHALLENGE described in Section 4.1 and the PA-OTP-REQUEST described in Section 4.2 SHOULD be registered in the PSKC algorithm registry [RFC6030].

The following pre-authentication types are defined in this document:

PA-OTP-CHALLENGE	141
PA-OTP-REQUEST	142
PA-OTP-PIN-CHANGE	144

These values are currently registered in registry created by [RFC6113] but the entries will need to be updated to refer to this document.

The following error codes and key usage values are defined in this document:

KDC_ERR_INVALID_HASH_ALG	94
KDC_ERR_INVALID_ITERATION_COUNT	95
KDC_ERR_PIN_EXPIRED	96
KDC_ERR_PIN_REQUIRED	97
KEY_USAGE_OTP_REQUEST	45

These values are currently not managed by IANA and have been accounted for. There is currently work in progress [LHA10] to define IANA registries and a registration process for these values.

No other IANA actions are anticipated.

6. Security Considerations

6.1. Man-in-the-Middle

In the system described in this document, the OTP pre-authentication protocol is tunneled within the FAST Armor channel provided by the pre-authentication framework. As described in [AsNiNy02], tunneled protocols are potentially vulnerable to man-in-the-middle attacks if the outer tunnel is compromised and it is generally considered good practice in such cases to bind the inner encryption to the outer tunnel.

In order to mitigate against such attacks, the proposed system uses the outer Armor Key in the derivation of the inner Client and Reply keys and so achieve crypto-binding to the outer channel.

As described in section 5.4 of [RFC6113], FAST can use an anonymous TGT obtained using anonymous PKINIT [RFC6112] [RFC4556] as the Armor Key. However, the current anonymous PKINIT proposal is open to man-in-the-middle attacks since the attacker can choose a session key such that the session key between the MITM and the real KDC is the same as the session key between the client and the MITM.

As described in Section 3.6, if the OTP value is not being sent to the KDC then the Armor Key is used along with the OTP value in the generation of the Client Key and Reply Key. If the Armor Key is known then the only entropy remaining in the key generation is provided by the OTP value. If the OTP algorithm requires that the OTP value be sent to the KDC then it is sent encrypted within the tunnel provided by the FAST armor and so is exposed to the attacker if the attacker has the Armor Key.

Therefore, unless the identity of the KDC has been verified, anonymous PKINIT SHALL NOT be used with OTP algorithms that require the OTP value to be sent to the KDC. In addition, the security considerations should be carefully considered before anonymous PKINIT is used with other algorithms such as those with short OTP values.

Careful consideration should also be made if host key armor is used to provide the KDC-authentication facility with OTP algorithms where the OTP value is sent within the otp-value field of the PA-OTP-REQUEST since compromised host keys would allow an attacker to impersonate the KDC.

6.2. Reflection

The 4-pass system described above is a challenge-response protocol and such protocols are potentially vulnerable to reflection attacks. No such attacks are known at this point but to help mitigate against such attacks, the system uses different keys to encrypt the client and server nonces.

6.3. Denial of Service

The protocol supports the use of an iteration count in the generation of the Client and Reply keys and the client can send the number of iterations used as part of the PA-OTP-REQUEST. This could open the KDC up to a denial of service attack if a large value for the iteration count was specified by the attacker. It is therefore particularly important that, as described in Section 3.4, the KDC

reject any authentication requests where the iteration count is above a maximum value specified by local policy.

6.4. Replay

In the 4-pass version of this protocol, the client encrypts a KDC generated nonce and so replay can be detected by the KDC. The 2-pass version of the protocol does not involve a server nonce but the client instead encrypts a timestamp and so is not protected from replay in this way but will instead require some other mechanism such as an OTP-server based system or a timestamp-based replay cache on the KDC.

As described in section 5.2 of [RFC6113], a client can not be certain that it will use the same KDC for all messages in a conversation. Therefore, the client cannot assume that the PA-OTP-REQUEST will be sent to the same KDC that issued the PA-OTP-CHALLENGE. In order to support this, a KDC implementing this protocol requires a means of sharing session state. However, doing this does introduce the possibility of a replay attack where the same response is sent to multiple KDCs.

In the case of time or event-based tokens or server-generated challenges, protection against replay may be provided by the OTP server being used if that server is capable of keeping track of the last used value. This protection therefore relies upon the assumption that the OTP server being used in this protocol is either not redundant or involves a commit protocol to synchronize between replicas. If this does not hold for an OTP server being used then the system may be vulnerable to replay attack.

However, OTP servers may not be able to detect replay of OTPs generated using only a client generated challenge and since the KDC would not be able to detect replay in 2-pass mode, it is recommended that the use of OTPs generated from only a client-generated challenge (that is, not in combination with some other factor such as time) should not be supported in 2-pass mode.

6.5. Brute Force Attack

A compromised or hostile KDC may be able to obtain the OTP value used by the client via a brute force attack. If the OTP value is short then the KDC could iterate over the possible OTP values until a Client Key is generated that can decrypt the encData sent in the PA-OTP-REQUEST.

As described in Section 3.6, an iteration count can be used in the generation of the Client Key and the value of the iteration count can

be controlled by local client policy. Use of this iteration count can make it computationally infeasible/unattractive for an attacker to brute-force search for the given OTP within the lifetime of that OTP.

6.6. FAST Facilities

The secret used to generate the OTP is known only to the client and the KDC and so successful decryption of the encrypted nonce by the KDC authenticates the user. If the OTP value is used in the Reply Key generation then successful decryption of the encrypted nonce by the client proves that the expected KDC replied. The Reply Key is replaced by either a key generated from the OTP and Armor Key or by the Armor Key. This FAST factor therefore provides the following facilities: client-authentication, replacing-reply-key and, depending on the OTP algorithm, KDC-authentication.

7. Acknowledgments

Many significant contributions were made to this document by RSA employees but special thanks go to Magnus Nystrom, John Linn, Richard Zhang, Piers Bowness, Robert Philpott, Robert Polansky and Boris Khoutorski.

Many valuable suggestions were also made by members of the Kerberos Working Group but special thanks go to Larry Zhu, Jeffrey Hutzelman, Tim Alsop, Henry Hotz, Nicolas Williams, Sam Hartman, Frank Cusak, Simon Josefsson, Greg Hudson and Linus Nordberg.

I would also like to thank Tim Alsop and Srinivas Cheruku of CyberSafe for many valuable review comments.

8. References

8.1. Normative References

[ISOIEC7812]

ISO, "ISO/IEC 7812-1:2006 Identification cards -- Identification of issuers -- Part 1: Numbering system", October 2006, <http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=39698>.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, February 2005.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4013] Zeilenga, K., "SASLprep: Stringprep Profile for User Names and Passwords", RFC 4013, February 2005.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
- [RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.
- [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, May 2008.
- [RFC6112] Zhu, L., Leach, P., and S. Hartman, "Anonymity Support for Kerberos", RFC 6112, April 2011.
- [RFC6113] Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication", RFC 6113, April 2011.
- [X.680] ITU-T, "Recommendation X.680 (2002) | ISO/IEC 8824-1:2002, Information technology - Abstract Syntax Notation One (ASN.1): Specification of basic notation.", July 2002.
- [X.690] ITU-T, "Recommendation X.690 (2002) | ISO/IEC 8825-1:2002, Information technology - ASN.1 encoding Rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER).", December 2002.

8.2. Informative References

- [AsNiNy02] Asokan, N., Niemi, V., and K. Nyberg, "Man-in-the-Middle

in Tunneled Authentication Protocols", Cryptology ePrint
Archive Report 2002/163, November 2002.

[HoReNeZo04]

Horstein, K., Renard, K., Neuman, C., and G. Zorn,
"Integrating Single-use Authentication Mechanisms with
Kerberos", draft-ietf-krb-wg-kerberos-sam-03 (work in
progress), July 2004.

[LHA10]

Hornquist Astrand, L., "Kerberos number registry to IANA",
draft-lha-krb-wg-some-numbers-to-iana-00 (work in
progress), March 2010.

[RFC2289]

Haller, N., Metz, C., Nesser, P., and M. Straw, "A One-
Time Password System", RFC 2289, February 1998.

[RFC2808]

Nystrom, M., "The SecurID(r) SASL Mechanism", RFC 2808,
April 2000.

[RFC4226]

M'Raihi, D., Bellare, M., Hoornaert, F., Naccache, D., and
O. Ranen, "HOTP: An HMAC-Based One-Time Password
Algorithm", RFC 4226, December 2005.

[RFC6030]

Hoyer, P., Pei, M., and S. Machani, "Portable Symmetric
Key Container (PSKC)", RFC 6030, October 2010.

Appendix A. ASN.1 Module

OTPKerberos

DEFINITIONS IMPLICIT TAGS ::=

BEGIN

IMPORTS

```
KerberosTime, KerberosFlags, EncryptionKey, Int32,  
EncryptedData, LastReq, KerberosString  
FROM KerberosV5Spec2 {iso(1) identified-organization(3)  
    dod(6) internet(1) security(5)  
    kerberosV5(2) modules(4) krb5spec2(2)}  
-- as defined in RFC 4120.  
  
AlgorithmIdentifier  
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.
```

```
PA-OTP-CHALLENGE ::= SEQUENCE {
    nonce          [0] OCTET STRING,
    otp-service    [1] UTF8String OPTIONAL,
    otp-tokenInfo  [2] SEQUENCE (SIZE(1..MAX)) OF
                                     OTP-TOKENINFO,
    salt           [3] KerberosString OPTIONAL,
    s2kparams      [4] OCTET STRING OPTIONAL,
    ...
}

OTP-TOKENINFO ::= SEQUENCE {
    flags          [0] OTPFlags,
    otp-vendor     [1] UTF8String OPTIONAL,
    otp-challenge  [2] OCTET STRING (SIZE(1..MAX))
                                     OPTIONAL,
    otp-length     [3] Int32 OPTIONAL,
    otp-format     [4] OTPFormat OPTIONAL,
    otp-tokenID    [5] OCTET STRING OPTIONAL,
    otp-algID      [6] AnyURI OPTIONAL,
    supportedHashAlg [7] SEQUENCE OF AlgorithmIdentifier
                                     OPTIONAL,
    iterationCount [8] Int32 OPTIONAL,
    ...
}

OTPFormat ::= INTEGER {
    decimal(0),
    hexadecimal(1),
    alphanumeric(2),
    binary(3),
    base64(4)
}

OTPFlags ::= KerberosFlags
-- reserved(0),
-- nextOTP(1),
-- combine(2),
-- collect-pin(3),
-- do-not-collect-pin(4),
-- must-encrypt-nonce (5),
-- separate-pin-required (6),
-- check-digit (7)

PA-OTP-REQUEST ::= SEQUENCE {
    flags          [0] OTPFlags,
    nonce          [1] OCTET STRING OPTIONAL,
    encData        [2] EncryptedData,
    -- PA-OTP-ENC-REQUEST or PA-ENC-TS-ENC
}
```

```

-- Key usage of KEY_USAGE_OTP_REQUEST
hashAlg      [3]  AlgorithmIdentifier  OPTIONAL,
iterationCount [4]  Int32                OPTIONAL,
otp-value     [5]  OCTET STRING          OPTIONAL,
otp-pin       [6]  UTF8String            OPTIONAL,
otp-challenge [7]  OCTET STRING (SIZE(1..MAX)) OPTIONAL,
otp-time      [8]  KerberosTime          OPTIONAL,
otp-counter   [9]  OCTET STRING          OPTIONAL,
otp-format    [10] OTPFormat              OPTIONAL,
otp-tokenID   [11] OCTET STRING          OPTIONAL,
otp-algID     [12] AnyURI                OPTIONAL,
otp-vendor    [13] UTF8String            OPTIONAL,
...
}

PA-OTP-ENC-REQUEST ::= SEQUENCE {
    nonce      [0] OCTET STRING,
    ...
}

PA-OTP-PIN-CHANGE ::= SEQUENCE {
    flags      [0] PinFlags,
    pin        [1] UTF8String OPTIONAL,
    minLength  [2] INTEGER    OPTIONAL,
    maxLength  [3] INTEGER    OPTIONAL,
    last-req   [4] LastReq    OPTIONAL,
    format     [5] OTPFormat  OPTIONAL,
    ...
}

PinFlags ::= KerberosFlags
-- reserved(0),
-- systemSetPin(1),
-- mandatory(2)

AnyURI ::= UTF8String
(CONSTRAINED BY {
    -- MUST be a valid URI in accordance with IETF RFC 2396
})

```

END

Appendix B. Examples of OTP Pre-Authentication Exchanges

This section is non-normative.

B.1. Four Pass Authentication

In this mode, the client sends an initial AS-REQ to the KDC that does not contain a PA-OTP-REQUEST and the KDC responds with a KRB-ERROR containing a PA-OTP-CHALLENGE.

In this example, the user has been issued with a connected, time-based token and the KDC requires hashed OTP values in the key generation with SHA-384 as the preferred hash algorithm and a minimum of 1024 iterations. The local policy on the client supports SHA-256 and requires 100,000 iterations of the hash of the OTP value.

The basic sequence of steps involved is as follows:

1. The client obtains the user name of the user.
2. The client sends an initial AS-REQ to the KDC that does not contain a PA-OTP-REQUEST.
3. The KDC determines that the user identified by the AS-REQ requires OTP authentication.
4. The KDC constructs a PA-OTP-CHALLENGE as follows:

nonce

A randomly generated value.

otp-service

A string that can be used by the client to assist the user in locating the correct token.

otp-tokenInfo

Information about how the OTP should be generated from the token.

flags

must-encrypt-nonce and collect-pin bits set

supportedHashAlg

AlgorithmIdentifiers for SHA-384, SHA-256 and SHA-1

iterationCount

1024

5. The KDC returns a KRB-ERROR with an error code of KDC_ERR_PREAUTH_REQUIRED and the PA-OTP-CHALLENGE in the e-data.

6. The client displays the value of otp-service and prompts the user to connect the token.
7. The client collects a PIN from the user.
8. The client obtains the current OTP value from the token using the PIN and records the time as reported by the token.
9. The client generates the Client Key and Reply Key as described in Section 3.6 using SHA-256 from the list of algorithms sent by the KDC, the iteration count of 100,000 as required by local policy and a randomly generated nonce.
10. The client constructs a PA-OTP-REQUEST as follows:

flags

0

nonce

The randomly generated value.

encData

An EncryptedData containing a PA-OTP-ENC-REQUEST encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and the encryption type of the Armor Key. The PA-OTP-ENC-REQUEST contains the nonce from the PA-OTP-CHALLENGE.

hashAlg

SHA-256

iterationCount

100,000

otp-time

The time used in the OTP calculation as reported by the OTP token.

11. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.
12. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.
13. The KDC validates the pre-authentication data in the PA-OTP-REQUEST:

- * Generates the Client Key and Reply Key from the OTP value for the user identified in the AS-REQ, using an iteration count of 100,000, a hash algorithm of SHA-256 and the nonce as specified in the PA-OTP-REQUEST.
 - * Uses the generated Client Key to decrypt the PA-OTP-ENC-REQUEST in the encData of the PA-OTP-REQUEST.
 - * Authenticates the user by comparing the nonce value from the decrypted PA-OTP-ENC-REQUEST to that sent in the corresponding PA-OTP-CHALLENGE.
14. The KDC constructs a TGT for the user.
 15. The KDC returns an AS-REP to the client, encrypted using the Reply Key, containing the TGT and padata with the PA-FX-FAST-REPLY.
 16. The client authenticates the KDC and verifies the Reply Key change.
 - * Uses the generated Reply Key to decrypt the encrypted data in the AS-REP.

B.2. Two Pass Authentication

In this mode, the client includes a PA-OTP-REQUEST within a PA-FX-FAST-REQUEST pre-auth of the initial AS-REQ sent to the KDC.

In this example, the user has been issued with a hand-held token and so none of the OTP generation parameters (otp-length etc) are included in the PA-OTP-REQUEST. The KDC does not require hashed OTP values in the key generation.

It is assumed that the client has been configured with the following information or has obtained it from a previous PA-OTP-CHALLENGE.

- o The fact that the OTP value must not be carried in the otp-value
- o The fact that hashed OTP values are not required.

The basic sequence of steps involved is as follows:

1. The client obtains the user name and OTP value from the user.
2. The client generates the Client Key and Reply Key using unhashed OTP values as described in Section 3.6.

3. The client constructs a PA-OTP-REQUEST as follows:

flags
0

encData

An EncryptedData containing a PA-ENC-TS-ENC encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and an encryption type of the Armor Key. The PA-ENC-TS-ENC contains the current client time.

4. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.
5. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.
6. The KDC validates the pre-authentication data:
 - * Generates the Client Key and Reply Key from the unhashed OTP value for the user identified in the AS-REQ.
 - * Uses the generated Client Key to decrypt the PA-ENC-TS-ENC in the encData of the PA-OTP-REQUEST.
 - * Authenticates the user using the timestamp in the standard manner.
7. The KDC constructs a TGT for the user.
8. The KDC returns an AS-REP to the client, encrypted using the Reply Key, containing the TGT and padata with the PA-FX-FAST-REPLY.
9. The client authenticates the KDC and verifies the key change.
 - * Uses the generated Reply Key to decrypt the encrypted data in the AS-REP.

B.3. PIN Change

This exchange follows from the point where the KDC receives the PA-OTP-REQUEST from the client in the examples in Appendix B.1 and Appendix B.2. During the validation of the pre-authentication data (whether encrypted nonce or encrypted timestamp), the KDC determines that the user's PIN has expired and so must be changed. The KDC therefore includes a PA-OTP-PIN-CHANGE in the AS-REP.

In this example, the KDC does not generate PIN values for the user but requires that the user generate a new PIN that is between 4 and 8 characters in length. The actual PIN change is handled by a PIN change service.

The basic sequence of steps involved is as follows:

1. The client constructs and sends a PA-OTP-REQUEST to the KDC as described in the previous examples.
2. The KDC validates the pre-authentication data and authenticates the user as in the previous examples but determines that the user's PIN has expired.
3. KDC constructs a ticket for a PIN change service with a one minute lifetime.
4. KDC constructs a PA-OTP-PIN-CHANGE as follows:

 flags
 0

 minLength
 4

 maxLength
 8
5. KDC encrypts the PA-OTP-PIN-CHANGE within the enc-fast-rep of a PA-FX-FAST-REPLY.
6. KDC returns a KRB-ERROR to the client of type KDC_ERR_PIN_EXPIRED containing the ticket to the PIN change service and padata containing the PA-FX-FAST-REPLY.
7. The client uses the ticket requests a ticket for a PIN change service and changes the user's PIN.
8. The client sends a second AS-REQ to the KDC containing a PA-OTP-REQUEST constructed using the new PIN.
9. The KDC responds with an AS-REP containing a TGT.

B.4. Resynchronization

This exchange follows from the point where the KDC receives the PA-OTP-REQUEST from the client. During the validation of the pre-

authentication data (whether encrypted nonce or encrypted timestamp), the KDC determines that the local record of the token's state needs to be re-synchronized with the token. The KDC therefore includes a KRB-ERROR containing a PA-OTP-CHALLENGE with the "nextOTP" flag set.

The sequence of steps below follows is a variation of the four pass examples shown in Appendix B.1 but the same process would also work in the two-pass case.

1. The client constructs and sends a PA-OTP-REQUEST to the KDC as described in the previous examples.
2. The KDC validates the pre-authentication data and authenticates the user as in the previous examples but determines that user's token requires re-synchronizing.
3. KDC constructs a PA-OTP-CHALLENGE as follows:

nonce

A randomly generated value.

otp-service

Set to a string that can be used by the client to assist the user in locating the correct token.

otp-tokenInfo

Information about how the OTP should be generated from the token.

flags

must-encrypt-nonce, collect-pin and nextOTP bits set

supportedHashAlg

AlgorithmIdentifiers for SHA-256 and SHA-1

iterationCount

1024

4. KDC returns a KRB-ERROR with an error code of KDC_ERR_PREAUTH_REQUIRED and the PA-OTP-CHALLENGE in the e-data.
5. The client obtains the next OTP value from the token and records the time as reported by the token.
6. The client generates the Client Key and Reply Key as described in Section 3.6 using SHA-256 from the list of algorithms sent by the KDC, the iteration count of 100,000 as required by local policy and a randomly generated nonce.

7. The client constructs a PA-OTP-REQUEST as follows:
 - flags
 - nextOTP bit set
 - nonce
 - The randomly generated value.
 - encData
 - An EncryptedData containing a PA-OTP-ENC-REQUEST encrypted under the Client Key with a key usage of KEY_USAGE_OTP_REQUEST and the encryption type of the Armor Key. The PA-OTP-ENC-REQUEST contains the nonce from the PA-OTP-CHALLENGE.
 - hashAlg
 - SHA-256
 - iterationCount
 - 100,000
 - otp-time
 - The time used in the OTP calculation as reported by the OTP token.
8. The client encrypts the PA-OTP-REQUEST within the enc-fast-req of a PA-FX-FAST-REQUEST.
9. The client sends an AS-REQ to the KDC containing the PA-FX-FAST-REQUEST within the padata.
10. Authentication process now proceeds as with the classic sequence.

Author's Address

Gareth Richards
RSA, The Security Division of EMC
RSA House
Western Road
Bracknell, Berkshire RG12 1RT
UK

Email: gareth.richards@rsa.com

Kerberos Working Group
Internet-Draft
Intended status: Experimental
Expires: November 5, 2011

A. Perez-Mendez
R. Marin-Lopez
F. Pereniguez-Garcia
G. Lopez-Millan
University of Murcia
May 4, 2011

GSS-API pre-authentication for Kerberos
draft-perez-krb-wg-gss-preauth-00

Abstract

This document describes a pre-authentication mechanism for Kerberos based on the Generic Security Service Application Program Interface (GSS-API), which allows a Key Distribution Center (KDC) to authenticate clients by using any GSS mechanism.

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 <http://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 November 5, 2011.

Copyright Notice

Copyright (c) 2011 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 (<http://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.

Table of Contents

1. Introduction	3
1.1. Requirements Language	4
2. Definition of the Kerberos GSS padata	4
3. GSS Pre-authentication Operation	4
3.1. Generation of GSS preauth requests	4
3.2. Processing of GSS preauth requests	5
3.3. Generation of GSS preauth responses	5
3.4. Processing of GSS preauth responses	6
4. Data in the KDC_ERR_PREAUTH_REQUIRED	6
5. Supported pre-authentication facilities	7
6. Managing states for the KDC	7
7. Security Considerations	8
8. IANA Considerations	8
9. Normative References	8
Authors' Addresses	9

1. Introduction

The GSS-API (Generic Security Service Application Programming Interface) [RFC2743] provides a generic toolset of functions that allow applications to establish security contexts in order to protect their communications through security services such as authentication, confidentiality and integrity protection. Thanks to the GSS-API, applications remain independent from the specific underlying mechanism used to establish the context and provide security.

Kerberos [RFC4120] defines a process called pre-authentication. This feature is intended to avoid the security risk of providing tickets encrypted with the user's long-term key to attackers. The execution of a pre-authentication mechanism may require the exchange of several KRB_AS_REQ/KRB_ERROR messages before the KDC delivers the TGT requested by the client within a KRB_AS_REP. These messages transport authentication information by means of pre-authentication elements.

There exists a variety of pre-authentication mechanisms, like PKINIT [RFC4556] and encrypted time-stamp [RFC4120]. Furthermore, [I-D.ietf-krb-wg-preauth-framework] provides a generic framework for Kerberos pre-authentication, which aims to describe the features that a pre-authentication mechanism may provide (e.g. mutual authentication, replace reply key, etc.). Additionally, in order to simplify the definition of new pre-authentication mechanisms, it defines a mechanism called FAST (Flexible Authentication Secure Tunneling), which provides a generic and secure transport for pre-authentication elements. More specifically, FAST establishes a secure tunnel providing confidentiality and integrity protection between the client and the KDC prior to the exchange of any specific pre-authentication data. Within this tunnel, different pre-authentication methods can be executed. This inner mechanism is called a FAST factor. It is important to note that FAST factors cannot usually be used outside the FAST pre-authentication method since they assume the underlying security layer provided by FAST.

The aim of this draft is to define a new pre-authentication mechanism, following the recommendations of [I-D.ietf-krb-wg-preauth-framework], that relies on the GSS-API security services to pre-authenticate clients. This pre-authentication mechanism will allow the KDC to authenticate clients making use of any current or future GSS mechanism, as long as they satisfy the minimum security requirements described in this specification. The Kerberos client will play the role of the GSS initiator, while the Authentication Server (AS) in the KDC will play the role of the GSS acceptor.

1.1. Requirements Language

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 [RFC2119].

2. Definition of the Kerberos GSS padata

To establish the security context, the GSS-API defines the exchange of GSS tokens between the initiator and the acceptor. These tokens, which contain mechanism-specific information, are completely opaque to the application. However, how these tokens are transported between the initiator and the responder depends on the specific application. Since GSS-API is defined as independent of the underlying communications service, its use does not require to implement any specific security feature for the transport. For instance, tokens could just be sent by means of plain UDP datagrams. For this reason, security and ordered delivery of information must be implemented by each specific GSS mechanism (if required).

Therefore, GSS tokens are the atomic piece of information from the application point of view when using GSS-API, which require a proper transport between the initiator (Kerberos client) and the acceptor (AS). In particular, the proposed GSS-based pre-authentication mechanism defines a new pre-authentication element (hereafter padata) called PA-GSS-TOKEN to transport a generic GSS token from the Kerberos client to the AS and vice-versa. The ASN.1 specification for this padata is:

PA-GSS-TOKEN To be defined (TBD)

PA-GSS-TOKEN ::= OCTET STRING --value of the GSS token

3. GSS Pre-authentication Operation

3.1. Generation of GSS preauth requests

The Kerberos client (initiator) starts by calling to the GSS_Init_sec_context function. In the first call to this function, the client provides GSS_C_NO_CTX as the value of the context_handle and NULL as the input_token, given that no context has been initiated yet. When using multi round-trip GSS mechanisms, in subsequent calls to this routine the client will use both, the context_handle value obtained after the first call, and the input_token received from the KDC.

The `GSS_Init_sec_context` returns a `context_handle`, an `output_token` and a status value. If the obtained status is `GSS_S_COMPLETE`, the generated token contains the necessary information to establish the context and, therefore, no further tokens are expected from the KDC to complete the authentication process. On the contrary, if the obtained status is `GSS_S_CONTINUE_NEEDED`, the KDC is expected to provide a token in order to continue with the context establishment process. In both cases, the Kerberos client includes the obtained `output_token` into a new `PA-GSS-TOKEN` padata and sends it to the KDC through a `KRB_AS_REQ` message.

3.2. Processing of GSS preauth requests

When the KDC (GSS acceptor) receives a `KRB_AS_REQ` message containing a `PA-GSS-TOKEN`, but a `PA-FX-COOKIE` (see Section 6) is not included, the KDC assumes that this is the first message of a context establishment, and thus `GSS_C_NO_CTX` is used as `context_handle` to invoke the `GSS_Accept_sec_context` routine. Conversely, if a `PA-FX-COOKIE` is included, the KDC assumes that this message is part an ongoing authentication and the value of the `PA-FX-COOKIE` is used as the `context_handle` value. In both cases, after receiving the message, the KDC calls to the `GSS_Accept_sec_context` function, using the adequate `context_handle` value and using the received token in the `PA-GSS-TOKEN` padata as `input_token`.

Once the execution of the `GSS_Accept_sec_context` function is completed, the KDC obtains a `context_handle`, an `output_token` (optional) that MUST be sent to the initiator in order to continue with the authentication process, and a status value. If the obtained status is `GSS_S_COMPLETE`, the client is considered authenticated and the value of the `output_token` may be `NULL`. If the status is `GSS_S_CONTINUE_NEEDED`, further information is required to complete the process.

3.3. Generation of GSS preauth responses

Once the KDC has processed the `input_token` provided by the client (as described in Section 3.2, two main different situations may occur depending on the status value. If the client is successfully authenticated (`GSS_S_COMPLETE`), the KDC will reply to the client with a `KRB_AS_REP` message. This message will transport the final `output_token` (if generated) in a `PA-GSS-TOKEN` padata type. Additionally, there are three alternatives to encrypt the `enc-part` field of the `KRB_AS_REP` message. The first one is to make use of the client's password as described in the standard Kerberos. A second option is to strengthen this key by using keying material from the GSS context (more details are provide in Section 5). The final option is to employ a key cryptographically independent from the

user's password which could be generated by using the keying material from the GSS context. Section 5 provides further details regarding these two last options.

On the contrary, if further data is required to complete the establishment process (GSS_S_CONTINUE_NEEDED), the KDC will reply to the client with a KDC_ERR_MORE_PREAUTH_DATA_REQUIRED error message [I-D.ietf-krb-wg-preauth-framework]. In the e-data field of the message, the KDC will include two padata types: a PA-FX-COOKIE containing the context_handle of this context (Section 6), and a PA-GSS-TOKEN containing the obtained output_token.

3.4. Processing of GSS preauth responses

When the client receives a KDC_ERR_MORE_PREAUTH_DATA_REQUIRED error, it extracts the token from the PA-GSS-TOKEN element and invokes to the GSS_Init_sec_context function, as described in section Section 3.1. The received PA-FX-COOKIE is treated as an opaque element, which is simply copied and included into the following KRB_AS_REQ message without further processing.

On the other hand, when the client receives a KRB_AS_REP, the context establishment has finalized successfully. If the KRB_AS_REP message contains a PA-GSS-TOKEN padata type, the client invokes the GSS_Init_sec_context function using the transported input_token. Note that, to be consistent, this call MUST return GSS_S_COMPLETE and not generate any output_token, since the KDC does not expect further data from the client. Similarly, if the client does not expect any data from the KDC (it obtained a GSS_S_COMPLETE status value on the last call) and the KDC provides an input_token, an unexpected situation occurs and the context establishment must be aborted.

If the context establishment is completed correctly, the client must use the same process followed by the KDC (Section 3.3).

4. Data in the KDC_ERR_PREAUTH_REQUIRED

When the KDC sends a KDC_ERR_PREAUTH_REQUIRED error to the client, it includes a sequence of padata, each corresponding to an acceptable pre-authentication method. Optionally, these padatas contain data valuable for the client to configure the selected mechanism. The data to be included in the padata for this message is described in this section.

TBD. (For example, list of the OIDs of the GSS mechanisms supported by the KDC)

5. Supported pre-authentication facilities

The pre-authentication framework [I-D.ietf-krb-wg-preauth-framework] defines a set of facilities that the pre-authentication mechanisms may provide. Specifically, the GSS pre-authentication mechanism proposed in this draft may provide the following facilities:

- o Client-authentication facility. The GSS pre-authentication mechanism authenticates the client based on GSS-API calls. At the end of the GSS context establishment process, the client is authenticated against the KDC by means of the specific GSS mechanism credentials.
- o Strengthening-reply-key facility. After a successful authentication, client and KDC may strengthen the reply key (the key used to encrypt the enc-part field of the KRB_AS_REP message) by adding additional keying material to it. This additional keying material can be obtained by means of calls to the GSS_Pseudo_random [RFC4401] function, although the standard GSS_getMIC function could be used if the former is not available for the specific GSS mechanism.
- o Replacing-reply-key facility. Similarly to the strengthening facility, client and KDC may decide to completely replace the reply key used to encrypt the KRB_AS_REP by a new one that is cryptographically independent from the client's password stored in client password on the Kerberos users database. To generate this keying material, the same GSS-API functions used for the previous facility would be used.
- o KDC-authentication facility. This facility is also provided, as an optional feature, since the GSS-API allows the initiator of the security context to request mutual authentication during the establishment process. If the mutual_req_flag is indicated in the GSS_Init_sec_context call, the acceptor (KDC) must be authenticated by the initiator (client) before the context is established.

The selection of the facilities that the GSS pre-authentication mechanism will provide, and how will they be negotiated with the client is still under discussion.

6. Managing states for the KDC

The Kerberos standard [RFC4120] defines the KDC as a stateless entity. This means that, if the GSS mechanism requires more than one round-trip, the client must provide enough data to the KDC in the

following interactions to allow recovering the complete state of the ongoing authentication. This is specially relevant when the client switches from one KDC to different one (within the same realm) during a pre-authentication process. This second KDC must be able to continue with the process in a seamless way. In [I-D.ietf-krb-wg-preauth-framework], the PA-FX-COOKIE pre-authentication element is defined to transport opaque state information from the KDC to the client. This state information is included by the client in the following KRB_AS_REQ message as-is, without further processing. When the KDC receives the PA-FX-COOKIE padata, it tries to recover the state and, if successful, continue with the authentication process.

PA-FX-COOKIE

133

The GSS-API manages the so-called security contexts. They represent the whole context of an authentication, including all the state and relevant data of the ongoing security context. Every GSS-API function requires an input parameter (called `context_handle`) which identifies the specific context over which they are applied. The application obtains a value for this `context_handle` after the first call to the `GSS_Init_sec_context` function (when acting as GSS initiator) or `GSS_Accept_sec_context` function (when acting as GSS acceptor), which is used in subsequent calls regarding this security context. Hence, it seems reasonable that this is the value that must be transported in the PA-FX-COOKIE padata type as it allows the KDC to recover the complete state of an ongoing context establishment process.

7. Security Considerations

Protection of Request/Responses with FAST, restriction on GSS mechanism, etc. TBD.

8. IANA Considerations

This document has no actions for IANA.

9. Normative References

- [I-D.ietf-krb-wg-preauth-framework]
Hartman, S. and L. Zhu, "A Generalized Framework for Kerberos Pre-Authentication",
draft-ietf-krb-wg-preauth-framework-17 (work in progress),
June 2010.

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
- [RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.
- [RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.

Authors' Addresses

Alejandro Perez-Mendez (Ed.)
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 46 44
Email: alex@um.es

Rafa Marin-Lopez
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 01
Email: rafa@um.es

Fernando Pereniguez-Garcia
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 78 82
Email: pereniguez@um.es

Gabriel Lopez-Millan
University of Murcia
Campus de Espinardo S/N, Faculty of Computer Science
Murcia, 30100
Spain

Phone: +34 868 88 85 04
Email: gabilm@um.es

