

NETWORK WORKING GROUP  
Internet-Draft  
Intended status: Standards Track  
Expires: October 1, 2017

N. Williams  
Cryptonector LLC  
A. Melnikov  
Isode Ltd  
March 30, 2017

Namespace Considerations and Registries for GSS-API Extensions  
draft-ietf-kitten-gssapi-extensions-iana-11.txt

Abstract

This document describes the ways in which the GSS-API may be extended and directs the creation of an IANA registry for various GSS-API namespaces.

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 October 1, 2017.

Copyright Notice

Copyright (c) 2017 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. Conventions used in this document . . . . .	2
2. Introduction . . . . .	2
3. Extensions to the GSS-API . . . . .	2
4. Generic GSS-API Namespaces . . . . .	3
5. Language Binding-Specific GSS-API Namespaces . . . . .	3
6. Extension-Specific GSS-API Namespaces . . . . .	4
7. Registration Form . . . . .	4
8. IANA Considerations . . . . .	6
8.1. Initial Namespace Registrations . . . . .	7
8.1.1. Example registrations . . . . .	7
8.2. Registration Maintenance Guidelines . . . . .	9
8.2.1. Sub-Namespace Symbol Pattern Matching . . . . .	10
8.2.2. Expert Reviews of Individual Submissions . . . . .	10
8.2.3. Change Control . . . . .	11
9. Security Considerations . . . . .	12
10. References . . . . .	12
10.1. Normative References . . . . .	12
10.2. Informative References . . . . .	12
Authors' Addresses . . . . .	13

## 1. 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].

## 2. Introduction

There is a need for private-use and mechanism-specific extensions to the Generic Security Services Application Programming Interface (GSS-API). As such extensions are designed and standardized (or not), both at the IETF and elsewhere, there is a non-trivial risk of namespace pollution and conflicts. To avoid this we set out guidelines for extending the GSS-API and direct the creation of an IANA registry for GSS-API namespaces.

Registrations of individual items and sub-namespaces are allowed. Each sub-namespace may provide different rules for registration, e.g., for mechanism-specific and private-use extensions.

## 3. Extensions to the GSS-API

Extensions to the GSS-API can be categorized as follows:

- o Abstract API extensions

- o Implementation-specific
- o Mechanism-specific
- o Language binding-specific

Extensions to the GSS-API may be purely semantic, without effect on the GSS-API's namespaces. Or they may introduce new functions, constants, types, etc...; these clearly affect the GSS-API namespaces.

Extensions that affect the GSS-API namespaces should be registered with the IANA as described herein.

#### 4. Generic GSS-API Namespaces

The abstract API namespaces for the GSS-API are:

- o Type names
- o Function names
- o Constant names for various types
- o Constant values for various types
- o Name types (OID, type name and syntaxes)

Additionally we have namespaces associates with the OBJECT IDENTIFIER (OID) type. The IANA already maintains a registry of such OIDs:

- o Mechanism OIDs
- o Name Type OIDs

#### 5. Language Binding-Specific GSS-API Namespaces

Language binding specific namespaces include, among others:

- o Header/interface module names
- o Object classes and/or types
- o Methods and/or functions
- o Constant names
- o Constant values

## 6. Extension-Specific GSS-API Namespaces

Extensions to the GSS-API may create additional namespaces. See Section 8.2.

## 7. Registration Form

Registrations for GSS-API namespaces SHALL take the following form:

Registration Field	Possible Values	Description
Bindings	'Generic', 'C-bindings', 'Java', 'C#', <programming language name>	Indicates the name of the programming language that this registration involves, or, if 'Generic', that this is an entry for the generic abstract GSS-API (i.e., not specific to any programming language).
Registration type	'Instance', 'Sub-Namespace'	Indicates whether this entry reserves a given symbol name (and possibly, constant value), or whether it reserves an entire sub-namespace (the name is a pattern) or constant value range.
Object Type	<Symbol> defined by the binding language (for example 'Data-Type', 'Function', 'Method', 'Integer', 'String', 'OID', 'Context-Flag', 'Name-Type', 'Macro', 'Header-File-Name', 'Module-Name', 'Class')	Indicates the type of the object whose symbolic name or constant value this entry registers. The possible values of this field depend on the programming language in question, therefore they are not all specified here.
Symbol Name/Prefix	<Symbol name or name pattern>	The name of a symbol or symbol sub-namespace being

		registered. See Section 8.2.1
Binding of	<Name of abstract API element of which this object is a binding>	If the registration is for a specific language binding of the GSS-API, then this names the abstract API element of which it is a binding (OPTIONAL).
Constant Value/Range	<Constant value> or <constant value range>	The value of the constant named by the <Symbol Name/Prefix>. This field is present only for Instance and Sub-namespace registrations of Constant object types.
Description	<Text>	Description of the registration. Multiple instances of this field may result (see Section 8.2.3).
Registration Rules	<Reference> to an IANA registration Policy defined in [RFC5226] (or an RFC that updates it), for instance 'IESG Approval', 'Expert Review', 'First Come First Served', 'Private Use'.	Describes the rules for allocation of items that fall in this sub-namespace, for entries with Registration Type of Sub-namespace (OPTIONAL). For private use sub-namespaces the submitter MUST provide the e-mail address of a responsible contact. If this field is not specified for a sub-namespace, the default registration rules specified in Section 8.2 apply.
Reference	<Reference>	Reference to a document that describes the registration, if any (OPTIONAL). Multiple instances of this field are allowed, with one reference each.
Expert Reviewer	<Name of expert reviewers, possibly	OPTIONAL, see Section 8.2.2. Multiple instances of this

	WG names>	field are allowed, with one expert reviewer per-instance. Leave this field blank when requesting a registration. It will be filled in by the Expert who reviews the registration.
Expert Review Notes	<Notes from the expert review>	Expert reviewers may request that some comments be included with the registration, e.g., regarding security considerations of the registered extension.
Status	'Registered' or 'Obsoleted'	Status of the registration.
Obsoleting Reference	<Reference>	Reference to a document, if any, that obsoletes this registration. Multiple instances of this field are allowed, with one reference each. (OPTIONAL)

The IANA should create a single GSS-API namespace registry, or multiple registries, one for symbolic names and one for constant values, and/or it may create a registry per-programming language, at its convenience.

Entries in these registries should consist of all the fields from their corresponding registration entries.

Entries should be sorted by: programming language, registration type, object type, and symbol name/pattern.

## 8. IANA Considerations

This document deals with IANA considerations throughout. Specifically it creates a single registry of various kinds of things, though the IANA may instead create multiple registries, each for one of those kinds of things. Of particular interest may be that IANA will now be the registration authority for the GSS-API name type OID space.

## 8.1. Initial Namespace Registrations

Initial registry content corresponding to the items defined in [RFC2743], [RFC2744], [RFC2853], [RFC1964] and [RFC4121] and others will be supplied during the IANA review portion of the RFC publishing process. [[Note to RFC Editor: Delete the following sentence before publication:]] The KITTEN WG chairs MUST indicate that such content has been reviewed by the WG and that there is WG consensus that the entries are in agreement with those RFCs.

### 8.1.1. Example registrations

In order to sanity check recommended IANA registration templates, this section registers several entries.

Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Function
Symbol Name	gss_init_sec_context
Binding of	GSS_Init_sec_context
Constant Value/Range	N/A
Description	Create a security context by initiator
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A

Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Function
Symbol Name	gss_accept_sec_context
Binding of	GSS_Accept_sec_context
Constant Value/Range	N/A
Description	Accept a security context from initiator
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A



Registration Field	Possible Values
Bindings	C-bindings
Registration type	Instance
Object Type	Context-Flag
Symbol Name	GSS_C_DELEG_FLAG
Binding of	deleg_state or deleg_req_flag
Constant Value/Range	1
Description	On output (if set): Delegated credentials are available via the delegated_cred_handle parameter of GSS_Accept_sec_context. On input (if set): With the call to GSS_Init_sec_context, delegate credentials to the acceptor.
Registration Rules	N/A
Reference	RFC 2744
Expert Reviewer	Kitten WG
Expert Review Notes	
Status	Registered
Obsoleting Reference	N/A

## 8.2. Registration Maintenance Guidelines

Standards-Track RFCs can create new items with any non-conflicting Symbol Name/Prefix value for this registry by virtue of IESG approval to publish as a Standards-Track RFC -- that is, without additional expert review.

Standards-Track RFCs can mark existing entries as obsolete, and can even create conflicting entries if explicitly stated (the IESG, of course, should review conflicts carefully, and may reject them).

IANA shall also consider submissions from individuals, and via Informational and Experimental RFCs, subject to Expert Review. IANA SHALL allow such registrations if a) they are not conflicting, b) provided that the registration is for object types other than Context-Flags, and c) subject to expert review. Guidelines for expert reviews are given below.

#### 8.2.1. Sub-Namespace Symbol Pattern Matching

Sub-namespace registrations must provide a pattern for matching symbols for which the sub-namespace's registration rules apply. The pattern consists of a string with the following special tokens:

- o '\*', meaning "match any string."
- o "%m", meaning "match any mechanism family short-hand name."
- o "%i", meaning "match any implementor vanity short-hand name."

For example, "GSS\_%m\*" matches "GSS\_krb5\_foo" since "krb5" is a common short-hand for the Kerberos V GSS-API mechanism [RFC1964]. But "GSS\_%m\*" does not match "GSS\_foo\_bar" unless "foo" is asserted to be a short-hand for some mechanism.

#### 8.2.2. Expert Reviews of Individual Submissions

[[The following paragraph should be deleted from the document before publication, as it will not age well. It should be moved to the shepherding write-up.]]

Expert review selection SHALL be done as follows. If, at the time that the IANA receives an individual submission for registration in this registry, there are any IETF Working Groups chartered to produce GSS-API-related documents, then the IANA SHALL ask the chairs of such WGs to be expert reviewers or to name one. If there are no such WGs at that time, then the IANA SHALL ask past chairs of the KITTEN WG and the author/editor of this RFC to act as expert reviewers or name an alternate.

Expert reviewers of individual registration submissions with Registration Type == Sub-namespace should check that the registration request has a suitable description (which doesn't need to be sufficiently detailed for others to implement) and that the Symbol Name/Prefix is sufficiently descriptive of the purpose of the sub-namespace or reflective of the name of the submitter or associated company.

Expert reviewers of individual registration submissions with

Registration Type == Instance should check that the Symbol Name falls under a sub-namespace controlled by the submitter. Registration of such entries which do not fall under such a sub-namespace may be allowed provided that they correspond to long existing non-standard extensions to the GSS-API and this can be easily checked or demonstrated, otherwise IESG Protocol Action is REQUIRED (see previous section). Also, reviewers should check that any registration of constant values have a detailed description that is suitable for other implementors to reproduce, and that they don't conflict with other usages or are otherwise dangerous in the reviewers estimation.

Expert reviewers should review impact on mechanisms, security and interoperability, and may reject or annotate registrations which can have mechanism impact that requires IESG protocol action. Consider, for example, new versions of `GSS_Init_sec_context()` and/or `GSS_Accept_sec_context` which have new input and/or output parameters which imply changes on the wire or in behaviour that may result in interoperability issues. A reviewer could choose to add notes to the registration describing such issues, or the reviewer might conclude that the danger to Internet interoperability is sufficient to warrant rejecting the registration.

#### 8.2.3. Change Control

Registered entries may be marked obsoleted using the same expert review process as for registering new entries. Obsoleted entries are not, however, to be deleted, but merely marked having Obsoleted Status. Note that entries may be created as obsoleted to record the fact that the given symbol(s) have been used before, even though continued use of them is discouraged.

Registered entries may also be updated in two other ways: additional references, obsoleting references, and descriptions may be added.

All changes are subject to expert review, except for changes to registrations in a sub-namespace which are subject to the rules of the relevant sub-namespace. The submitter of a change request need not be the same as the original submitter.

Registrations may be modified by addition, but under no circumstance may any fields be modified except for the Status field or Contact Address, or to correct for transcription errors in filing or processing registration requests.

The IANA SHALL add a field describing the date that a an addition or modification was made, and a description of the change.

## 9. Security Considerations

General security considerations relating to IANA registration services apply; see [RFC5226].

Also, expert reviewers should look for and may document security related issues with submitters' GSS-API extensions, to the best of the reviewers' ability given the information furnished by the submitter. Reviewers may add comments regarding their limited ability to review a submission for security problems if the submitter is unwilling to provide sufficient documentation.

## 10. References

### 10.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <<http://www.rfc-editor.org/info/rfc2743>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

### 10.2. Informative References

- [RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, DOI 10.17487/RFC1964, June 1996, <<http://www.rfc-editor.org/info/rfc1964>>.
- [RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, DOI 10.17487/RFC2744, January 2000, <<http://www.rfc-editor.org/info/rfc2744>>.
- [RFC2853] Kabat, J. and M. Upadhyay, "Generic Security Service API Version 2 : Java Bindings", RFC 2853, DOI 10.17487/RFC2853, June 2000, <<http://www.rfc-editor.org/info/rfc2853>>.

[RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, DOI 10.17487/RFC4121, July 2005, <<http://www.rfc-editor.org/info/rfc4121>>.

#### Authors' Addresses

Nicolas Williams  
Cryptonector LLC

Email: [nico@cryptonector.com](mailto:nico@cryptonector.com)

Alexey Melnikov  
Isode Ltd  
5 Castle Business Village  
36 Station Road  
Hampton, Middlesex TW12 2BX  
UK

Email: [Alexey.Melnikov@isode.com](mailto:Alexey.Melnikov@isode.com)

Network Working Group  
Internet-Draft  
Updates: rfc4120 (if approved)  
Intended status: Standards Track  
Expires: October 1, 2017

T. Yu  
MIT Kerberos Consortium  
March 30, 2017

Move Kerberos protocol parameter registries to IANA  
draft-ietf-kitten-kerberos-iana-registries-04

Abstract

The Keberos 5 network authentication protocol has several numeric protocol parameters. Most of these parameters are not currently under IANA maintenance. This document requests that IANA take over the maintenance of the remainder of these Kerberos parameters.

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 October 1, 2017.

Copyright Notice

Copyright (c) 2017 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.

## 1. Requirements Notation

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

## 2. Introduction

The Kerberos 5 network authentication protocol[RFC4120][RFC1510] has several numeric protocol parameters. This document requests that IANA take over the maintenance of the Kerberos protocol parameters that are not currently under IANA maintenance. Several instances of number conflicts in Kerberos implementations could have been prevented by having IANA registries for those numbers. This document updates [RFC4120].

## 3. General registry format

Unless otherwise specified, each Kerberos protocol number registry will have the following fields: "number", "name", "reference", and "comments".

The name must begin with a lowercase letter, and must consist of ASCII letters, digits, and hyphens. Two or more hyphens must not appear directly adjacent to each other. A hyphen must not appear at the end of a name. It is preferred that words in a name be separated by hyphens, and that all of the letters be lowercase.

(These rules are consistent with the lexical rules for an ASN.1 valuereference or identifier. Where the constraints are stricter than the ASN.1 lexical rules, they make it easier to systematically transform the names for use in implementation languages.)

Names for numeric parameter values have no inherent meaning in the Kerberos protocol, but they can guide choices for internal implementation symbol names and for user-visible non-numeric representations. When written in English prose in specifications, or when used as symbolic constants in implementation languages (e.g., C preprocessor macros), it is common to transform the name into all uppercase letters, and possibly to replace hyphens with underscores.

## 4. General registration procedure

This document requests that the IESG establish a pool of Kerberos experts who will manage the Kerberos registries using these guidelines. The IESG may wish to consider including the set of designated IANA experts for existing Kerberos IANA registries as candidates for this pool.

IANA will select an expert from this pool for each registration request. The expert will review the registration request and may approve the registration, decline the registration with comments, or recommend that the registration request should follow a specific alternative process. The alternative processes that the expert may recommend are the IETF review process and the standards action process.

Initially, the expert reviewers will use a permissive process, generally approving registrations that are architecturally consistent with Kerberos and the protocol parameter in question. Over time, with input from the community, the experts may refine the requirements that registrations are expected to meet. The experts will maintain a current version of these guidelines in a manner that is generally accessible to the entire community. As the guidelines evolve, experts may consider the technical quality of specifications, security impacts of the registrations, architectural consistency, and interoperability impact. Experts may require a publicly available specification in order to make certain registrations.

[ For the individual registries, include "Registrations in this registry are managed by the expert review process [RFC5226] or in exceptional cases by IESG approval. See section x for guidelines for the experts to be used with this registry." ]

## 5. Integer assignments

Names for integer assignments must be unique across all Kerberos integer parameter registries. This is normally accomplished by including a name prefix that identifies the registry.

Assignments for integers parameters will follow the general registration procedure outlined above, except as otherwise noted in the section that contains the description of the parameter. Kerberos integer parameters take on signed 32-bit values (-2147483648 to 2147483647). Negative values are for private or local use.

### 5.1. Address types

Registry name: Address types

Assignment policy: General registration procedure

Valid values: Signed 32-bit integers

Address types historically align with numeric constants used in the Berkeley sockets API. Future address type assignments should conform



to this historical practice when possible. The name prefix for address types is "addrtype-".

#### 5.2. Authorization data types

Registry name: Authorization data types

Assignment policy: General registration procedure

Valid values: Signed 32-bit integers

The name prefix for authorization data types is "ad-".

#### 5.3. Error codes

Registry name: Error codes

Assignment policy: Standards action

Valid values: Signed 32-bit integers

Assignments for error codes require standards action due to their scarcity: assigning error codes greater than 127 could require significant changes to certain implementations. The name prefixes for error codes are "kdc-err-", "krb-err-", and "krb-ap-err-".

#### 5.4. Key usages

Registry name: Key usages

Assignment policy: General registration procedure

Valid values: Unsigned 32-bit integers

Key usages are unsigned 32-bit integers (0 to 4294967295). Zero is reserved and may not be assigned.

The name prefix for key usages is "ku-".

#### 5.5. Name types

Registry name: Name types

Assignment policy: General registration procedure

Valid values: Signed 32-bit integers

The name prefix for name types is "nt-".

number	name	reference	comment
0	nt-unknown	RFC4120	Name type not known
1	nt-principal	RFC4120	Just the name of the principal as in DCE, or for users
2	nt-srv-inst	RFC4120	Service and other unique instance (krbtgt)
3	nt-srv-hst	RFC4120	Service with host name as instance (telnet, rcommands)
4	nt-srv-xhst	RFC4120	Service with host as remaining components
5	nt-uid	RFC4120	Unique ID
6	nt-x500-principal	RFC4120	Encoded X.509 Distinguished name [RFC2253]
7	nt-smtp-name	RFC4120	Name in form of SMTP email name (e.g., user@example.com)
10	nt-enterprise	RFC4120	Enterprise name - may be mapped to principal name
11	nt-wellknown	RFC6111	Well-known principal name
12	nt-srv-hst-domain	RFC5179	Domain-based names

#### 5.6. Pre-authentication and typed data

Registry name: Pre-authentication and typed data

Assignment policy: General registration procedure

Valid values: Signed 32-bit integers

This document requests that IANA modify the existing Kerberos Pre-authentication and typed data registry to be consistent with the procedures in this document.

The name prefix for pre-authentication type numbers is "pa-". The name prefix for typed data numbers is "td-". Pre-authentication and typed data numbers are in the same registry, but a pre-authentication number may be also be assigned to a related typed data number.

## 6. Named bit assignments

Assignments for named bits require standards action, due to their scarcity: assigning bit numbers greater than 31 could require significant changes to implementations. Names for named bit assignments must be unique within a given named bit registry, and typically do not have name prefixes that identify which registry they belong to.

### 6.1. AP-REQ options

Registry name: AP-REQ options  
Assignment policy: Standards action  
Valid values: ASN.1 bit numbers 0 through 31

### 6.2. KDC-REQ options

Registry name: KDC-REQ options  
Assignment policy: Standards action  
Valid values: ASN.1 bit numbers 0 through 31

### 6.3. Ticket flags

Registry name: Ticket flags  
Assignment policy: Standards action  
Valid values: ASN.1 bit numbers 0 through 31

## 7. Numbers that will not be registered

ASN.1 application tag numbers (which are always equal to the "msg-type" field in Kerberos messages where they appear) will not be registered. Any Kerberos protocol change that requires a new application tag number will be a sufficiently major change that the specification of the change MUST define a new ASN.1 module and MUST be Standards Track.

Transited encoding values will not be registered. There is only one transited encoding type for the Kerberos protocol. The interoperability concerns inherent to the cross-realm operation of Kerberos mean that specifications of new transited encoding types are very unlikely. Any specification of new transited encoding types MUST be Standards Action.

Protocol version number (pvno) values will not be registered. The location of the "pvno" value in Kerberos messages is not in a place that implementations can meaningfully use to distinguish among different variants of the Kerberos protocol.

## 8. Contributors

Sam Hartman proposed the text of the expert review guidelines. Love Hornquist Astrand wrote a previous document (draft-lha-krb-wg-some-numbers-to-iana-00) with the same goals as this document.

## 9. Acknowledgments

Thanks to Tom Petch for providing useful feedback on previous versions of this document.

## 10. Security Considerations

Assignments of new Kerberos protocol parameter values can have security implications. In cases where the assignment policy calls for expert review, the reviewer is responsible for evaluating whether adequate documentation exists concerning the security considerations for the requested assignment. For assignments that require IETF review or standards action, the normal IETF processes ensure adequate treatment of security considerations.

## 11. IANA Considerations

This document requests that IANA create several registries for Kerberos protocol parameters:

- o Address types
- o Authorization data types
- o Error codes
- o Key usages
- o Name types
- o AP-REQ options
- o KDC-REQ options
- o Ticket flags

This document requests that IANA modify the existing "Pre-authentication data and typed data" registry to contain an additional reference to this document, and to transform existing names in that registry to the lowercase-and-hyphens style.

## 12. Open issues

Do we make a registry for application tag numbers (equal to message type numbers)? We've said that we would replace the entire ASN.1 module in that case, but Nico's recent proposal doesn't do that, and if we want to accommodate that sort of proposal, it would probably be best to establish a registry. (It should require standards action for registrations.)

Do transited encodings need a registry? They would probably require standards action, even if there were a registry.

## 13. References

### 13.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<http://www.rfc-editor.org/info/rfc2119>>.
- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <<http://www.rfc-editor.org/info/rfc3961>>.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, DOI 10.17487/RFC4120, July 2005, <<http://www.rfc-editor.org/info/rfc4120>>.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <<http://www.rfc-editor.org/info/rfc5226>>.

### 13.2. Informative References

- [RFC1510] Kohl, J. and C. Neuman, "The Kerberos Network Authentication Service (V5)", RFC 1510, DOI 10.17487/RFC1510, September 1993, <<http://www.rfc-editor.org/info/rfc1510>>.

Author's Address

Tom Yu  
MIT Kerberos Consortium  
77 Massachusetts Ave  
Cambridge, Massachusetts  
USA

Email: [tlyu@mit.edu](mailto:tlyu@mit.edu)

KITTEN  
Internet-Draft  
Intended status: Standards Track  
Expires: November 30, 2015

W. Mills  
Microsoft  
T. Showalter  
  
H. Tschofenig  
ARM Ltd.  
May 29, 2015

A set of SASL Mechanisms for OAuth  
draft-ietf-kitten-sasl-oauth-23.txt

Abstract

OAuth enables a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf.

This document defines how an application client uses credentials obtained via OAuth over the Simple Authentication and Security Layer (SASL) to access a protected resource at a resource server. Thereby, it enables schemes defined within the OAuth framework for non-HTTP-based application protocols.

Clients typically store the user's long-term credential. This does, however, lead to significant security vulnerabilities, for example, when such a credential leaks. A significant benefit of OAuth for usage in those clients is that the password is replaced by a shared secret with higher entropy, i.e., the token. Tokens typically provide limited access rights and can be managed and revoked separately from the user's long-term password.

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 30, 2015.

#### Copyright Notice

Copyright (c) 2015 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. Terminology . . . . .	5
3. OAuth SASL Mechanism Specifications . . . . .	6
3.1. Initial Client Response . . . . .	7
3.1.1. Reserved Key/Values . . . . .	8
3.2. Server's Response . . . . .	8
3.2.1. OAuth Identifiers in the SASL Context . . . . .	9
3.2.2. Server Response to Failed Authentication . . . . .	9
3.2.3. Completing an Error Message Sequence . . . . .	11
3.3. OAuth Access Token Types using Keyed Message Digests . . . . .	11
4. Examples . . . . .	12
4.1. Successful Bearer Token Exchange . . . . .	12
4.2. Successful OAuth 1.0a Token Exchange . . . . .	13
4.3. Failed Exchange . . . . .	14
4.4. SMTP Example of a Failed Negotiation . . . . .	15
5. Security Considerations . . . . .	16
6. Internationalization Considerations . . . . .	17
7. IANA Considerations . . . . .	17
7.1. SASL Registration . . . . .	18
8. References . . . . .	18
8.1. Normative References . . . . .	18
8.2. Informative References . . . . .	19
Appendix A. Acknowledgements . . . . .	20
Appendix B. Document History . . . . .	20
Authors' Addresses . . . . .	24



## 1. Introduction

OAuth 1.0a [RFC5849] and OAuth 2.0 [RFC6749] are protocol frameworks that enable a third-party application to obtain limited access to a protected resource, either on behalf of a resource owner by orchestrating an approval interaction, or by allowing the third-party application to obtain access on its own behalf.

The core OAuth 2.0 specification [RFC6749] specifies the interaction between the OAuth client and the authorization server; it does not define the interaction between the OAuth client and the resource server for the access to a protected resource using an Access Token. Instead, the OAuth client to resource server interaction is described in separate specifications, such as the bearer token specification [RFC6750]. OAuth 1.0a included the protocol specification for the communication between the OAuth client and the resource server in [RFC5849].

The main use cases for OAuth 2.0 and OAuth 1.0a have so far focused on an HTTP-based [RFC7230] environment only. This document integrates OAuth 1.0a and OAuth 2.0 into non-HTTP-based applications using the integration into SASL. Hence, this document takes advantage of the OAuth protocol and its deployment base to provide a way to use the Simple Authentication and Security Layer (SASL) [RFC4422] to gain access to resources when using non-HTTP-based protocols, such as the Internet Message Access Protocol (IMAP) [RFC3501] and the Simple Mail Transfer Protocol (SMTP) [RFC5321]. This document gives examples of use in IMAP and SMTP.

To illustrate the impact of integrating this specification into an OAuth-enabled application environment, Figure 1 shows the abstract message flow of OAuth 2.0 [RFC6749]. As indicated in the figure, this document impacts the exchange of messages (E) and (F) since SASL is used for interaction between the client and the resource server instead of HTTP.

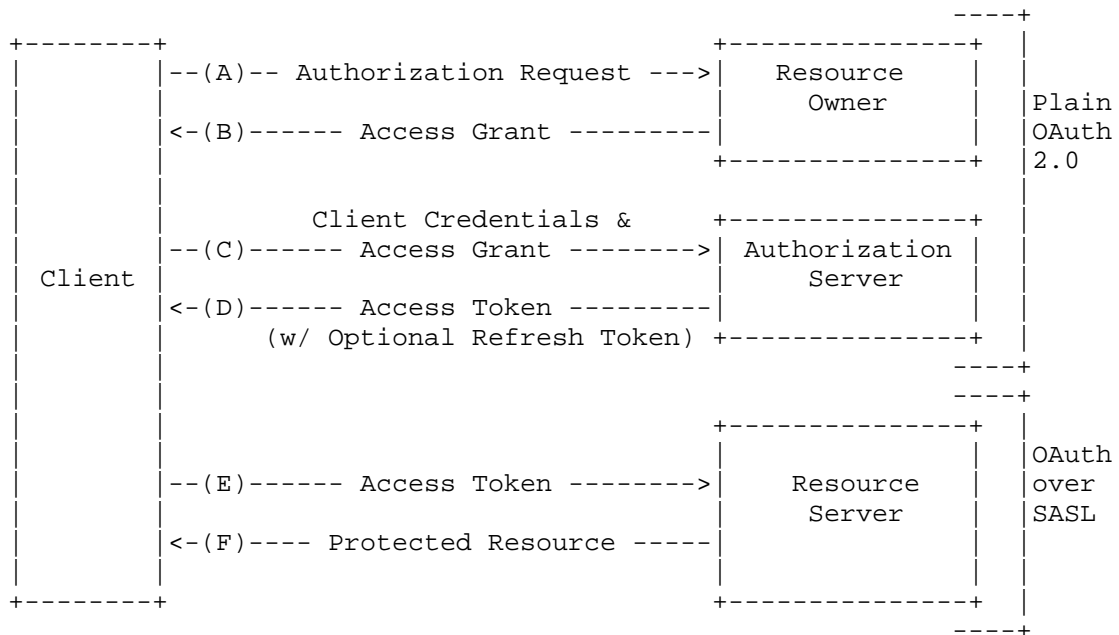


Figure 1: OAuth 2.0 Protocol Flow

The Simple Authentication and Security Layer (SASL) is a framework for providing authentication and data security services in connection-oriented protocols via replaceable authentication mechanisms. It provides a structured interface between protocols and mechanisms. The resulting framework allows new protocols to reuse existing authentication mechanisms and allows old protocols to make use of new authentication mechanisms. The framework also provides a protocol for securing subsequent exchanges within a data security layer.

When OAuth is integrated into SASL the high-level steps are as follows:

(A) The client requests authorization from the resource owner. The authorization request can be made directly to the resource owner (as shown), or indirectly via the authorization server as an intermediary.

(B) The client receives an authorization grant which is a credential representing the resource owner's authorization, expressed using one of the grant types defined in [RFC6749] or [RFC5849] or using an extension grant type. The authorization

grant type depends on the method used by the client to request authorization and the types supported by the authorization server.

(C) The client requests an access token by authenticating with the authorization server and presenting the authorization grant.

(D) The authorization server authenticates the client and validates the authorization grant, and if valid issues an access token.

(E) The client requests the protected resource from the resource server and authenticates by presenting the access token.

(F) The resource server validates the access token, and if valid, indicates a successful authentication.

Again, steps (E) and (F) are not defined in [RFC6749] (but are described in, for example, [RFC6750] for the OAuth Bearer Token instead) and are the main functionality specified within this document. Consequently, the message exchange shown in Figure 1 is the result of this specification. The client will generally need to determine the authentication endpoints (and perhaps the service endpoints) before the OAuth 2.0 protocol exchange messages in steps (A)-(D) are executed. The discovery of the resource owner, authorization server endpoints, and client registration are outside the scope of this specification. The client must discover the authorization endpoints using a discovery mechanism such as OpenID Connect Discovery [OpenID.Discovery] or Webfinger using host-meta [RFC7033]. Once credentials are obtained the client proceeds to steps (E) and (F) defined in this specification. Authorization endpoints MAY require client registration and generic clients SHOULD support the Dynamic Client Registration protocol [I-D.ietf-oauth-dyn-reg].

OAuth 1.0 follows a similar model but uses a different terminology and does not separate the resource server from the authorization server.

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

The reader is assumed to be familiar with the terms used in the OAuth 2.0 specification [RFC6749] and SASL [RFC4422].

In examples, "C:" and "S:" indicate lines sent by the client and server respectively. Line breaks have been inserted for readability.

Note that the IMAP SASL specification requires base64 encoding, as specified in Section 4 of [RFC4648].

### 3. OAuth SASL Mechanism Specifications

SASL is used as an authentication framework in a variety of application layer protocols. This document defines the following SASL mechanisms for usage with OAuth:

    OAUTHBEARER: OAuth 2.0 bearer tokens, as described in [RFC6750]. RFC 6750 uses Transport Layer Security (TLS) [RFC5246] to secure the protocol interaction between the client and the resource server.

    OAUTH10A: OAuth 1.0a MAC tokens (using the HMAC-SHA1 keyed message digest), as described in Section 3.4.2 of [RFC5849].

New extensions may be defined to add additional OAuth Access Token Types. Such a new SASL OAuth mechanism can be added by registering the new name(s) with IANA in the SASL Mechanisms registry and citing this specification for the further definition.

SASL mechanisms using this document as their definition do not provide a data security layer; that is, they cannot provide integrity or confidentiality protection for application messages after the initial authentication. If such protection is needed, TLS or some similar solution should be used. Additionally, for the two mechanisms specified in this document, TLS **MUST** be used for OAUTHBEARER to protect the bearer token; for OAUTH10A the use of TLS is **RECOMMENDED**.

These mechanisms are client initiated and lock-step, the server always replying to a client message. In the case where the client has and correctly uses a valid token the flow is:

1. Client sends a valid and correct initial client response.
2. Server responds with a successful authentication.

In the case where authentication fails the server sends an error result, then client **MUST** then send an additional message to the server in order to allow the server to finish the exchange. Some protocols and common SASL implementations do not support both sending

a SASL message and finalizing a SASL negotiation. The additional client message in the error case deals with this problem. This exchange is:

1. Client sends an invalid initial client response.
2. Server responds with an error message.
3. Client sends a dummy client response.
4. Server fails the authentication.

### 3.1. Initial Client Response

Client responses are a GS2 [RFC5801] header followed by zero or more key/value pairs, or may be empty. The gs2-header is defined here for compatibility with GS2 if a GS2 mechanism is formally defined, but this document does not define one. The key/value pairs take the place of the corresponding HTTP headers and values to convey the information necessary to complete an OAuth style HTTP authorization. Unknown key/value pairs MUST be ignored by the server. The ABNF [RFC5234] syntax is:

```
kvsep      = %x01
key        = 1*(ALPHA)
value      = *(VCHAR / SP / HTAB / CR / LF )
kvpair     = key "=" value kvsep
;;gs2-header = See RFC 5801
client_resp = (gs2-header kvsep *kvpair kvsep) / kvsep
```

The GS2 header MAY include the user name associated with the resource being accessed, the "authzid". It is worth noting that application protocols are allowed to require an authzid, as are specific server implementations.

The client response consisting of only a single kvsep is used only when authentication fails, and is only valid in that context. If sent as the first message from the client the server MAY simply fail the authentication without returning discovery information since there is no user or server name indication.

The following keys and corresponding values are defined in the client response:

auth (REQUIRED): The payload that would be in the HTTP Authorization header if this OAuth exchange was being carried out over HTTP.

host: Contains the host name to which the client connected. In an HTTP context this is the value of the HTTP Host header.

port: Contains the destination port that the client connected to, represented as a decimal positive integer string without leading zeros.

For OAuth token types such as OAuth 1.0a that use keyed message digests the client MUST send host and port number key/values, and the server MUST fail an authorization request requiring keyed message digests that are not accompanied by host and port values. In OAuth 1.0a for example, the so-called "signature base string calculation" includes the reconstructed HTTP URL.

### 3.1.1. Reserved Key/Values

In these mechanisms values for path, query string and post body are assigned default values. OAuth authorization schemes MAY define usage of these in the SASL context and extend this specification. For OAuth Access Token Types that include a keyed message digest of the request the default values MUST be used unless explicit values are provided in the client response. The following key values are reserved for future use:

mthd (RESERVED): HTTP method, the default value is "POST".

path (RESERVED): HTTP path data, the default value is "/".

post (RESERVED): HTTP post data, the default value is the empty string ("").

qs (RESERVED): The HTTP query string, the default value is the empty string ("").

### 3.2. Server's Response

The server validates the response according to the specification for the OAuth Access Token Types used. If the OAuth Access Token Type utilizes a keyed message digest of the request parameters then the client must provide a client response that satisfies the data requirements for the scheme in use.

The server fully validates the client response before generating a server response; this will necessarily include the validation steps listed in the specification for the OAuth Access Token Type used. However, additional validation steps may be needed, depending on the particular application protocol making use of SASL. In particular, values included as kvpairs in the client response (such as host and port) which correspond to values known to the application server by some other mechanism (such as an application protocol data unit or pre-configured values) MUST be validated to match between the initial client response and the the other source(s) of such information. As a concrete example, when SASL is used over IMAP to an IMAP server for a single domain the hostname can be available via configuration; this hostname must be validated to match the value sent in the 'host' kvpair.

The server responds to a successfully verified client message by completing the SASL negotiation. The authenticated identity reported by the SASL mechanism is the identity securely established for the client with the OAuth credential. The application, not the SASL mechanism, based on local access policy determines whether the identity reported by the mechanism is allowed access to the requested resource. Note that the semantics of the authzid is specified by the SASL framework [RFC4422].

#### 3.2.1. OAuth Identifiers in the SASL Context

In the OAuth framework the client may be authenticated by the authorization server and the resource owner is authenticated to the authorization server. OAuth access tokens may contain information about the authentication of the resource owner and about the client and may therefore make this information accessible to the resource server.

If both identifiers are needed by an application the developer will need to provide a way to communicate that from the SASL mechanism back to the application.

#### 3.2.2. Server Response to Failed Authentication

For a failed authentication the server returns a JSON [RFC7159] formatted error result, and fails the authentication. The error result consists of the following values:

status (REQUIRED): The authorization error code. Valid error codes are defined in the IANA "OAuth Extensions Error Registry" specified in the OAuth 2 core specification.

scope (OPTIONAL): An OAuth scope which is valid to access the service. This may be omitted which implies that unscoped tokens are required. If a scope is specified then a single scope is preferred. At the time this document was written there are several implementations that do not properly support space separated lists of scopes, so the use of a space separated list of scopes is NOT RECOMMENDED.

openid-configuration (OPTIONAL): The URL for a document following the OpenID Provider Configuration Information schema as described in OpenID Connect Discovery (OIDCD) [OpenID.Discovery] section 3 that is appropriate for the user. As specified in OIDCD this will have the "https" URL scheme. This document MUST have all OAuth related data elements populated. The server MAY return different URLs for users in different domains and the client SHOULD NOT cache a single returned value and assume it applies for all users/domains that the server supports. The returned discovery document SHOULD have all data elements required by the OpenID Connect Discovery specification populated. In addition, the discovery document SHOULD contain the 'registration\_endpoint' element to identify the endpoint to be used with the Dynamic Client Registration protocol [I-D.ietf-oauth-dyn-reg] to obtain the minimum number of parameters necessary for the OAuth protocol exchange to function. Another comparable discovery or client registration mechanism MAY be used if available.

The use of the 'offline\_access' scope, as defined in [OpenID.Core] is RECOMMENDED to give clients the capability to explicitly request a refresh token.

If the resource server provides a scope then the client MUST always request scoped tokens from the token endpoint. If the resource server does not return a scope the client SHOULD presume an unscoped token is required to access the resource.

Since clients may interact with a number of application servers, such as email servers and XMPP [RFC6120] servers, they need to have a way to determine whether dynamic client registration has been performed already and whether an already available refresh token can be re-used to obtain an access token for the desired resource server. This specification RECOMMENDS that a client uses the information in the 'iss' element defined in OpenID Connect Core [OpenID.Core] to make this determination.



### 3.2.3. Completing an Error Message Sequence

Section 3.6 of SASL [RFC4422] explicitly prohibits additional information in an unsuccessful authentication outcome. Therefore, the error message is sent in a normal message. The client MUST then send either an additional client response consisting of a single %x01 (control A) character to the server in order to allow the server to finish the exchange or send a SASL cancellation token as generally defined in section 3.5 of SASL [RFC4422]. A specific example of a cancellation token can be found in IMAP [RFC3501] section 6.2.2.

### 3.3. OAuth Access Token Types using Keyed Message Digests

OAuth Access Token Types may use keyed message digests and the client and the resource server may need to perform a cryptographic computation for integrity protection and data origin authentication.

OAuth is designed for access to resources identified by URIs. SASL is designed for user authentication, and has no facility for more fine-grained access control. In this specification we require or define default values for the data elements from an HTTP request which allow the signature base string to be constructed properly. The default HTTP path is "/" and the default post body is empty. These atoms are defined as extension points so that no changes are needed if there is a revision of SASL which supports more specific resource authorization, e.g., IMAP access to a specific folder or FTP access limited to a specific directory.

Using the example in the OAuth 1.0a specification as a starting point, on an IMAP server running on port 143 and given the OAuth 1.0a style authorization request (with %x01 shown as ^A and line breaks added for readability) below:

```
n,a=user@example.com,^A
host=example.com^A
port=143^A
auth=OAuth realm="Example",
      oauth_consumer_key="9djdj82h48djs9d2",
      oauth_token="kkk9d7dh3k39sjv7",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137131201",
      oauth_nonce="7d8f3e4a",
      oauth_signature="Tm90IGEgcmVhbCBzaWduYXR1cmU"^A^A
```

The signature base string would be constructed per the OAuth 1.0 specification [RFC5849] with the following things noted:

- o The method value is defaulted to POST.

- o The scheme defaults to be "http", and any port number other than 80 is included.
- o The path defaults to "/".
- o The query string defaults to "".

In this example the signature base string with line breaks added for readability would be:

```
POST&http%3A%2F%2Fexample.com:143%2F&oauth_consumer_key%3D9djdj82h4
8djs9d2%26oauth_nonce%3D7d8f3e4a%26oauth_signature_method%3DHMAC-SH
A1%26oauth_timestamp%3D137131201%26oauth_token%3Dkkk9d7dh3k39sjv7
```

#### 4. Examples

These examples illustrate exchanges between IMAP and SMTP clients and servers. All IMAP examples use SASL-IR [RFC4959] and send payload in the initial client response. The Bearer Token examples assume encrypted transport; if the underlying connection is not already TLS then STARTTLS MUST be used as TLS is required in the Bearer Token specification.

Note to implementers: The SASL OAuth method names are case insensitive. One example uses "Bearer" but that could as easily be "bearer", "BEARER", or "BeArEr".

##### 4.1. Successful Bearer Token Exchange

This example shows a successful OAuth 2.0 bearer token exchange in IMAP. Note that line breaks are inserted for readability.

```
[Initial connection and TLS establishment...]
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR
S: t0 OK Completed
C: t1 AUTH OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9c2Vy
    dmVYLnV4YW1wbGUuY29tAXBvcnQ9MTQzAWFldGg9QmVhcmVYIHZGOWRmd
    DRxbVRjMk52YjNSbGNrQmhiSFJoZG1semRHRXVZMj10Q2c9PQEB
S: t1 OK SASL authentication succeeded
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response (with %x01 represented as ^A and long lines wrapped for readability) is:

```
n,a=user@example.com,^Ahost=server.example.com^Aport=143^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhbHRhdmVzZGEuY29tCg==^A^A
```

The same credential used in an SMTP exchange is shown below. Again this example assumes that TLS is already established per the Bearer Token specification requirements.

```
[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250-STARTTLS
S: 250 PIPELINING
[Negotiate TLS...]
C: t1 AUTH OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb2sAwHvc3Q9c2Vy
dmVyLmV4YWlwbGUuY29tAXBvcnQ9NTg3AWFlldGg9QmVhcmVYIHZGOWRmd
DRxbVRjMk52YjNSbGNrQmhiSFJoZGZlsemRHRXVZMj10Q2c9PQEB
S: 235 Authentication successful.
[connection continues...]
```

The decoded initial client response is:

```
n,a=user@example.com,^Ahost=server.example.com^Aport=587^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhbHRhdmlzdgEuY29tCg==^A^A
```

## 4.2. Successful OAuth 1.0a Token Exchange

This IMAP example shows a successful OAuth 1.0a token exchange. Note that line breaks are inserted for readability. This example assumes that TLS is already established. Signature computation is discussed in Section 3.3.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER OAUTH10A SASL-IR
S: t0 OK Completed
C: t1 AUTH OAUTH10A bixhPXVzZXJAZXhhbXBsZS5jb20sAWhvc3Q9ZXhhb
XBsZS5jb20BcG9ydD0xNDMBYXV0aD1PQXV0aCByZWZfb20iRXhhbXBsZSIsb2F1
dGhfY29uc3VtZXJfa2V5PSI5ZGpkaJgyaDQ4ZGpzOWQyIixvYXV0aF90b2t1bj0
ia2trOWQ3ZGgzazM5c2p2NyIsb2F1dGhfY29uc3VtZXJfa2V5PSI5ZGpkaJgyaDQ4
ZGpzOWQyIixvYXV0aF90aW1lc3Rhbm9uY2U5Ij0kOGYyZTRhIixvYXV0aF9zaWduYXR1cmU9IlRtOTBJR0VnY21WaGJDQnphV2R1
WVhSMWNtVSUzRCIBAQ==
S: t1 OK SASL authentication succeeded
```

As required by IMAP [RFC3501], the payloads are base64-encoded. The decoded initial client response (with %x01 represented as ^A and lines wrapped for readability) is:

```
n,a=user@example.com,^A
host=example.com^A
port=143^A
auth=OAuth realm="Example",
      oauth_consumer_key="9djdj82h48djs9d2",
      oauth_token="kkk9d7dh3k39sjv7",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137131201",
      oauth_nonce="7d8f3e4a",
      oauth_signature="SSdtIGEgbGl0dGx1IHRLYSBwb3Qu"^A^A
```

#### 4.3. Failed Exchange

This IMAP example shows a failed exchange because of the empty Authorization header, which is how a client can query for the needed scope. Note that line breaks are inserted for readability.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR
S: t0 OK Completed
C: t1 AUTH OAUTHBEARER bixhPXVzZXJAZXhhbXBsZS5jb20sAW
      hvc3Q9c2VydmVyLmV4YW1wbGUuY29tAXBvcnQ9MTQzAWFldGg9AQE=
S: + eyJzdGF0dXMiOiJpbnZhbkx3Rva2VuIiwic2NvcGUiOiJleGFtcGxl
      X3Njb3BlIiwib3BlbmlkLWNvbmZpZ3VyYXRpb24iOiJodHRwczovL2V4
      YW1wbGUuY29tLy53ZWxsLWtub3duL29wZW5pZC1jb25maWcifQ==
C: AQ==
S: t1 NO SASL authentication failed
```

The decoded initial client response is:

```
n,a=user@example.com,^A
host=server.example.com^A
port=143^A
auth=^A^A
```

The decoded server error response is:

```
{
  "status": "invalid_token",
  "scope": "example_scope",
  "openid-configuration": "https://example.com/.well-known/openid-config"
}
```

The client responds with the required dummy response, "AQ==" is the base64 encoding of the ASCII value 0x01. The same exchange using the

IMAP specific method of cancelling an AUTHENTICATE command sends "\*" and is shown below.

```
S: * OK IMAP4rev1 Server Ready
C: t0 CAPABILITY
S: * CAPABILITY IMAP4rev1 AUTH=OAUTHBEARER SASL-IR IMAP4rev1
S: t0 OK Completed
C: t1 AUTH OAUTHBEARER bixhPXVzZXJAZXhxbXBsZS5jb20sAW
    hvc3Q9c2VydmVyLmV4YWlwbGUuY29tAXBvcnQ9MTQzAWFldGg9AQE=
S: + eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NvcGUiOiJleGFtcGxl
    X3Njb3BlIiwib3BlbmlkLWNvbmZpZ3VyYXRpb24iOiJodHRwciovL2V4
    YWlwbGUuY29tLy53ZWxsLWtub3duL29wZW5pZC1jb25maWdlcmF0aW9u
    In0=
C: *
S: t1 NO SASL authentication failed
```

#### 4.4. SMTP Example of a Failed Negotiation

This example shows an authorization failure in an SMTP exchange. TLS negotiation is not shown but as noted above it is required for the use of Bearer Tokens.

```
[connection begins]
S: 220 mx.example.com ESMTP 12sm2095603fks.9
C: EHLO sender.example.com
S: 250-mx.example.com at your service,[172.31.135.47]
S: 250-SIZE 35651584
S: 250-8BITMIME
S: 250-AUTH LOGIN PLAIN OAUTHBEARER
S: 250-ENHANCEDSTATUSCODES
S: 250 PIPELINING
[Negotiate TLS...]
C: AUTH OAUTHBEARER bixlc2VyPXNvbWVlc2VyQGV4YWlwbGUuY29tLAFhdXRoPUJlYXJl
    ciB2RjlkZnQ0cWlUYzJOdmIzUmxja0JoZEhSaGRtbHpkR0VlWTI5dENnPT0BAQ==
S: 334 eyJzdGF0dXMiOiJpbnZhbGlkX3Rva2VuIiwic2NoZWl1cyI6ImJlYXJlciBtYWMiL
    CJzY29wZSI6Imh0dHBzOi8vbWFpbC5leGFtcGxlLmNvbS8ifQ==
C: AQ==
S: 535-5.7.1 Username and Password not accepted. Learn more at
S: 535 5.7.1 http://support.example.com/mail/oauth
[connection continues...]
```

The initial client response is:

```
n,user=someuser@example.com,^A
auth=Bearer vF9dft4qmTc2Nvb3RlckBhdHRhdmlzdGEuY29tCg==^A^A
```

The server returned an error message in the 334 SASL message, the client responds with the required dummy response, and the server finalizes the negotiation.

```
{
  "status":"invalid_token",
  "schemes":"bearer mac",
  "scope":"https://mail.example.com/"
}
```

## 5. Security Considerations

OAuth 1.0a and OAuth 2 allow for a variety of deployment scenarios, and the security properties of these profiles vary. As shown in Figure 1 this specification is aimed to be integrated into a larger OAuth deployment. Application developers therefore need to understand their security requirements based on a threat assessment before selecting a specific SASL OAuth mechanism. For OAuth 2.0 a detailed security document [RFC6819] provides guidance to select those OAuth 2.0 components that help to mitigate threats for a given deployment. For OAuth 1.0a Section 4 of RFC 5849 [RFC5849] provides guidance specific to OAuth 1.0.

This document specifies two SASL Mechanisms for OAuth and each comes with different security properties.

**OAuthBEARER:** This mechanism borrows from OAuth 2.0 bearer tokens [RFC6750]. It relies on the application using TLS to protect the OAuth 2.0 Bearer Token exchange; without TLS usage at the application layer this method is completely insecure. Consequently, TLS **MUST** be provided by the application when choosing this authentication mechanism.

**OAuth10A:** This mechanism re-uses OAuth 1.0a MAC tokens (using the HMAC-SHA1 keyed message digest), as described in Section 3.4.2 of [RFC5849]. To compute the keyed message digest in the same way as in RFC 5839 this specification conveys additional parameters between the client and the server. This SASL mechanism only supports client authentication. If server-side authentication is desirable then it must be provided by the application underneath the SASL layer. The use of TLS is strongly **RECOMMENDED**.

Additionally, the following aspects are worth pointing out:

An access token is not equivalent to the user's long term password.

Care has to be taken when these OAuth credentials are used for actions like changing passwords (as it is possible with some

protocols, e.g., XMPP [RFC6120]). The resource server should ensure that actions taken in the authenticated channel are appropriate to the strength of the presented credential.

Lifetime of the application sessions.

It is possible that SASL will be used to authenticate a connection and the life of that connection may outlast the life of the access token used to establish it. This is a common problem in application protocols where connections are long-lived, and not a problem with this mechanism per se. Resource servers may unilaterally disconnect clients in accordance with the application protocol.

Access tokens have a lifetime.

Reducing the lifetime of an access token provides security benefits and OAuth 2.0 introduces refresh tokens to obtain new access token on the fly without any need for a human interaction. Additionally, a previously obtained access token might be revoked or rendered invalid at any time. The client MAY request a new access token for each connection to a resource server, but it SHOULD cache and re-use valid credentials.

## 6. Internationalization Considerations

The identifier asserted by the OAuth authorization server about the resource owner inside the access token may be displayed to a human. For example, when SASL is used in the context of IMAP the client may assert the resource owner's email address to the IMAP server for usage in an email-based application. The identifier may therefore contain internationalized characters and an application needs to ensure that the mapping between the identifier provided by OAuth is suitable for use with the application layer protocol SASL is incorporated into.

At the time of writing the standardization of the various claims in the access token (in JSON format) is still ongoing, see [I-D.ietf-oauth-json-web-token]. Once completed it will provide a standardized format for exchanging identity information between the authorization server and the resource server.

## 7. IANA Considerations

### 7.1. SASL Registration

The IANA is requested to register the following entry in the SASL Mechanisms registry:

SASL mechanism name: OAUTHBEARER

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Intended usage: common

Owner/Change controller: the IESG

Note: None

The IANA is requested to register the following entry in the SASL Mechanisms registry:

SASL mechanism name: OAUTH10A

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Intended usage: common

Owner/Change controller: the IESG

Note: None

## 8. References

### 8.1. Normative References

[I-D.ietf-oauth-dyn-reg]

Richer, J., Jones, M., Bradley, J., Machulak, M., and P. Hunt, "OAuth 2.0 Dynamic Client Registration Protocol", draft-ietf-oauth-dyn-reg-27 (work in progress), March 2015.



- [OpenID.Core]  
Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0", February 2014.
- [OpenID.Discovery]  
Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", July 2011.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, July 2010.
- [RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.
- [RFC6749] Hardt, D., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012.
- [RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012.
- [RFC7159] Bray, T., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014.

## 8.2. Informative References

- [I-D.ietf-oauth-json-web-token]  
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-32 (work in progress), December 2014.

- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, March 2003.
- [RFC4959] Siemborski, R. and A. Gulbrandsen, "IMAP Extension for Simple Authentication and Security Layer (SASL) Initial Client Response", RFC 4959, September 2007.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6819] Lodderstedt, T., McGloin, M., and P. Hunt, "OAuth 2.0 Threat Model and Security Considerations", RFC 6819, January 2013.
- [RFC7033] Jones, P., Salgueiro, G., Jones, M., and J. Smarr, "WebFinger", RFC 7033, September 2013.
- [RFC7230] Fielding, R. and J. Reschke, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, June 2014.

#### Appendix A. Acknowledgements

The authors would like to thank the members of the Kitten working group, and in addition and specifically: Simon Josefson, Torsten Lodderstadt, Ryan Troll, Alexey Melnikov, Jeffrey Hutzelman, Nico Williams, Matt Miller, and Benjamin Kaduk.

This document was produced under the chairmanship of Alexey Melnikov, Tom Yu, Shawn Emery, Josh Howlett, Sam Hartman. The supervising area director was Stephen Farrell.

#### Appendix B. Document History

[[ to be removed by RFC editor before publication as an RFC ]]

-23

- o AD feedback from IESG review and comments.
- o Fixed port number in SMTP examples.
- o Minor editorial changes.
- o Dyn-Reg draft becomes normative.

- o Added explicit TLS start indicator in all examples, removed text that said we assume that.

-19

- o Last call feedback again.
- o Clarified usage of TLS in examples and fixed them some more. Adding reference to RFC4422 and cancellation token and an example for that.

-18

- o Last call feedback round #5. Fixed -17 change log.
- o Corrected "issue" to "iss", other minor changes.

-17

- o Last call feedback again (WGLC #4). eradicated comma splicing. Removed extra server message in example 4.3.
- o Added recommendations for discovery and dynamic client registration support.

-16

- o Last call feedback again. Primarily editorial changes. Corrected examples.

-15

- o Last call feedback on the GS2 stuff being ripped out completely.
- o Removed the "user" parameter and put stuff back into the gs2-header. Call out that the authzid goes in the gs2-header with some prose about when it might be required. Very comparable to -10.
- o Added an OAuth 1.0A example explicitly.

-14

- o Last call feedback on RFC citations needed, small editorial.
- o Added the "user" parameter back, which was pulled when we started down the GS2 path. Same language as -03.

- o Defined a stub GS2 header to make sure that when the GS2 bride is defined for this that nothing will break when it actually starts to get populated.

-13

- o Changed affiliation.

-12

- o Removed -PLUS components from the specification.

-11

- o Removed GSS-API components from the specification.

- o Updated security consideration section.

-10

- o Clarifications throughout the document in response to the feedback from Jeffrey Hutzelman.

-09

- o Incorporated review by Alexey and Hannes.

- o Clarified the three OAuth SASL mechanisms.

- o Updated references

- o Extended acknowledgements

-08

- o Fixed the channel binding examples for p=\$cbtype

- o More tuning of the authcid language and edited and renamed 3.2.1.

-07

- o Struck the MUST language from authzid.

o

-06

- o Removed the user field. Fixed the examples again.

- o Added canonicalization language.

o

-05

- o Fixed the GS2 header language again.
- o Separated out different OAuth schemes into different SASL mechanisms. Took out the scheme in the error return. Tuned up the IANA registrations.
- o Added the user field back into the SASL message.
- o Fixed the examples (again).

o

-04

- o Changed user field to be carried in the gs2-header, and made gs2 header explicit in all cases.
- o Converted MAC examples to OAuth 1.0a. Moved MAC to an informative reference.
- o Changed to sending an empty client response (single control-A) as the second message of a failed sequence.
- o Fixed channel binding prose to refer to the normative specs and removed the hashing of large channel binding data, which brought more problems than it solved.
- o Added a SMTP examples for Bearer use case.

-03

- o Added user field into examples and fixed egregious errors there as well.
- o Added text reminding developers that Authorization scheme names are case insensitive.

-02

- o Added the user data element back in.
- o Minor editorial changes.

-01

- o Ripping out discovery. Changed to refer to I-D.jones-appsawg-webfinger instead of WF and SWD older drafts.
- o Replacing HTTP as the message format and adjusted all examples.

-00

- o Renamed draft into proper IETF naming format now that it's adopted.
- o Minor fixes.

#### Authors' Addresses

William Mills  
Microsoft

Email: [wimills@microsoft.com](mailto:wimills@microsoft.com)

Tim Showalter

Email: [tjs@psaux.com](mailto:tjs@psaux.com)

Hannes Tschofenig  
ARM Ltd.  
110 Fulbourn Rd  
Cambridge CB1 9NJ  
Great Britain

Email: [Hannes.tschofenig@gmx.net](mailto:Hannes.tschofenig@gmx.net)  
URI: <http://www.tschofenig.priv.at>

Network Working Group  
Internet-Draft  
Intended status: Standards Track  
Expires: November 11, 2021

S. Cantor  
Shibboleth Consortium  
M. Cullen  
Painless Security  
S. Josefsson  
SJD AB  
May 10, 2021

SAML Enhanced Client SASL and GSS-API Mechanisms  
draft-ietf-kitten-sasl-saml-ec-20

Abstract

Security Assertion Markup Language (SAML) 2.0 is a generalized framework for the exchange of security-related information between asserting and relying parties. Simple Authentication and Security Layer (SASL) and the Generic Security Service Application Program Interface (GSS-API) are application frameworks that facilitate an extensible authentication model, among other things. This document specifies a SASL and GSS-API mechanism for SAML 2.0 that leverages the capabilities of a SAML-aware "enhanced client" to address significant barriers to federated authentication in a manner that encourages reuse of existing SAML bindings and profiles designed for non-browser scenarios.

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 <https://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 11, 2021.

Copyright Notice

Copyright (c) 2021 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 (<https://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. Terminology . . . . .	5
3. Applicability for Non-HTTP Use Cases . . . . .	6
4. SAML Enhanced Client SASL Mechanism Specification . . . . .	8
4.1. Advertisement . . . . .	8
4.2. Initiation . . . . .	9
4.3. Server Response . . . . .	9
4.4. User Authentication with Identity Provider . . . . .	10
4.5. Client Response . . . . .	10
4.6. Outcome . . . . .	10
4.7. Additional Notes . . . . .	10
5. SAML EC GSS-API Mechanism Specification . . . . .	11
5.1. GSS-API Credential Delegation . . . . .	12
5.2. GSS-API Channel Binding . . . . .	13
5.3. Session Key Derivation . . . . .	13
5.3.1. Generated by Identity Provider . . . . .	14
5.3.2. Alternate Key Derivation Mechanisms . . . . .	15
5.4. Per-Message Tokens . . . . .	15
5.5. Pseudo-Random Function (PRF) . . . . .	15
5.6. GSS-API Principal Name Types for SAML EC . . . . .	16
5.6.1. User Naming Considerations . . . . .	16
5.6.2. Service Naming Considerations . . . . .	17
6. Example . . . . .	17
7. Security Considerations . . . . .	25
7.1. Risks Left Unaddressed . . . . .	26
7.2. User Privacy . . . . .	26
7.3. Collusion between RPs . . . . .	27
8. IANA Considerations . . . . .	27
8.1. GSS-API and SASL Mechanism Registration . . . . .	27
8.2. XML Namespace Name for SAML-EC . . . . .	27
9. References . . . . .	28
9.1. Normative References . . . . .	28
9.2. Informative References . . . . .	30
Appendix A. XML Schema . . . . .	31
Appendix B. Acknowledgments . . . . .	33
Appendix C. Changes . . . . .	33



Authors' Addresses	34
--------------------	----

## 1. Introduction

### Security Assertion Markup Language (SAML) 2.0

[OASIS.saml-core-2.0-os] is a modular specification that provides various means for a user to be identified to a relying party (RP) through the exchange of (typically signed) assertions issued by an identity provider (IdP).

Simple Authentication and Security Layer (SASL) [RFC4422] is a generalized mechanism for identifying and authenticating a user and for optionally negotiating a security layer for subsequent protocol interactions. SASL is used by application protocols like IMAP [RFC3501], the Post Office Protocol (POP [RFC1939]) and XMPP [RFC6120]. The effect of SASL is to make authentication modular, so that newer authentication mechanisms can be added as needed.

There are related protocols, protocol bindings [OASIS.saml-bindings-2.0-os], and interoperability profiles [OASIS.saml-profiles-2.0-os] designed for different use cases. Additional profiles and extensions are also routinely developed and published.

The Generic Security Service Application Program Interface (GSS-API) [RFC2743] provides a framework for applications to support multiple authentication mechanisms through a unified programming interface, as well as additional optional cryptographic functionality. This document defines a pure SASL mechanism for SAML, but it conforms to the bridge between SASL and GSS-API called GS2 [RFC5801]. This means that this document defines both a SASL mechanism and a GSS-API mechanism. The GSS-API interface is optional for SASL implementers, and the GSS-API considerations can be avoided in environments that use SASL directly without GSS-API.

The mechanisms specified in this document allow a SASL- or GSS-API-enabled server to act as a SAML relying party, or service provider (SP), by advertising this mechanism as an option for SASL or GSS-API clients that support the use of SAML to communicate identity and attribute information. Clients supporting this mechanism are termed "enhanced clients" in SAML terminology because they understand the federated authentication model and have specific knowledge of the IdP(s) associated with the user. This knowledge, and the ability to act on it, addresses a significant problem with browser-based SAML profiles known as the "discovery", or "where are you from?" (WAYF) problem. In a "dumb" client such as a web browser, various intrusive user interface techniques are used to determine the appropriate IdP to use because the request to the IdP is generated as an HTTP

redirect by the RP, which does not generally have prior knowledge of the IdP to use. Obviating the need for the RP to interact with the client to determine the right IdP (and its network location) is both a user interface and security improvement.

The SAML mechanism described in this document is an adaptation of an existing SAML profile, the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20].

Figure 1 describes the interworking between SAML and SASL: this document requires enhancements to the RP and to the client (as the two SASL communication endpoints) but no changes to the SAML IdP are assumed apart from its support for the applicable SAML profile. To accomplish this, a SAML protocol exchange between the RP and the IdP, brokered by the client, is tunneled within SASL. There is no assumed communication between the RP and the IdP, but such communication may occur in conjunction with additional SAML-related profiles not in scope for this document.

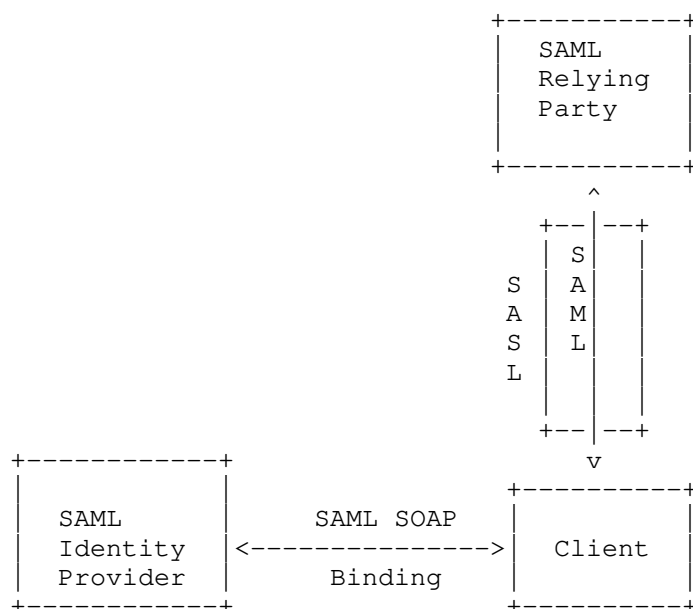


Figure 1: Interworking Architecture

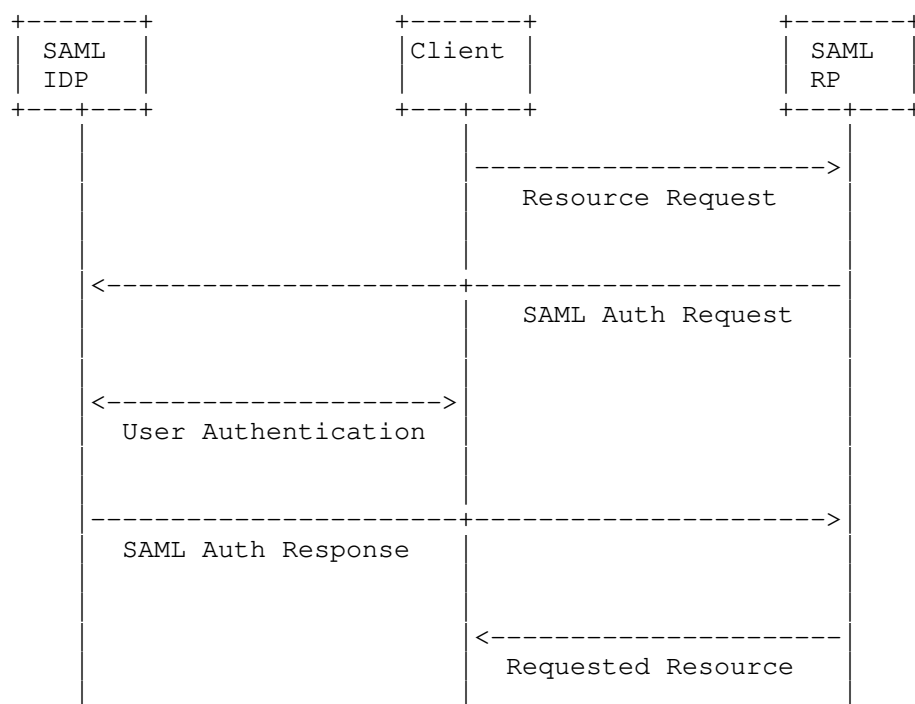


Figure 2: Communication Flow

## 2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119][RFC8174] when, and only when, they appear in all capitals, as shown here.

The reader is also assumed to be familiar with the terms used in the SAML 2.0 specification, and an understanding of the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20] is necessary, as part of this mechanism explicitly reuses and references it.

This document can be implemented without knowledge of GSS-API since the normative aspects of the GS2 protocol syntax have been duplicated in this document. The document may also be implemented to provide a GSS-API mechanism, and then knowledge of GSS-API is essential.

### 3. Applicability for Non-HTTP Use Cases

While SAML is designed to support a variety of application scenarios, the profiles for authentication defined in the original standard are designed around HTTP [RFC7230] applications. They are not, however, limited to browsers, because browsers do not always meet the needs of more security-sensitive applications. Specifically, the notion of an "Enhanced Client" (or a proxy acting as one on behalf of a browser, thus the term "ECP") was specified for a software component that acts somewhat like a browser from an application perspective, but includes limited, but sufficient, awareness of SAML to play a more conscious role in the authentication exchange between the RP and the IdP. What follows is an outline of the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20], as applied to the web/HTTP service use case:

1. The Enhanced Client requests a resource of a Relying Party (RP) (via an HTTP request). In doing so, it advertises its "enhanced" capability using HTTP headers.
2. The RP, desiring SAML authentication and noting the client's capabilities, responds not with an HTTP redirect or form, but with a SOAP [W3C.soap11] envelope containing a SAML <AuthnRequest> along with some supporting headers. This request identifies the RP (and may be signed), and may provide hints to the client as to what IdPs the RP finds acceptable, but the choice of IdP is generally left to the client.
3. The client is then responsible for delivering the body of the SOAP message to the IdP it is instructed to use (often via out-of-band configuration). The user authenticates to the IdP ahead of, during, or after the delivery of this message, and perhaps explicitly authorizes the response to the RP.
4. Whether authentication succeeds or fails, the IdP responds with its own SOAP envelope, generally containing a SAML <Response> message for delivery to the RP. In a successful case, the message will include one or more SAML <Assertion> elements containing authentication, and possibly attribute, statements about the subject. Either the response or each assertion is signed, and the assertion(s) may be encrypted to a key negotiated with or known to belong to the RP.
5. The client then delivers the SOAP envelope containing the <Response> to the RP at a location the IdP directs (which acts as an additional, though limited, defense against MITM attacks). This completes the SAML exchange.

6. The RP now has sufficient identity information to approve the original HTTP request or not, and acts accordingly. Everything between the original request and this response can be thought of as an "interruption" of the original HTTP exchange.

When considering this flow in the context of an arbitrary application protocol and SASL, the RP and the client both must change their code to implement this SASL mechanism, but the IdP can remain unmodified. The existing RP/client exchange that is tunneled through HTTP maps well to the tunneling of that same exchange in SASL. In the parlance of SASL [RFC4422], this mechanism is "client-first" for consistency with GS2. The steps are shown below:

1. The server MAY advertise the SAML20EC and/or SAML20EC-PLUS mechanisms.
2. The client initiates a SASL authentication with SAML20EC or SAML20EC-PLUS.
3. The server sends the client a challenge consisting of a SOAP envelope containing its SAML <AuthnRequest>.
4. The SASL client unpacks the SOAP message and communicates with its chosen IdP to relay the SAML <AuthnRequest> to it. This communication, and the authentication with the IdP, proceeds separately from the SASL process.
5. Upon completion of the exchange with the IdP, the client responds to the SASL server with a SOAP envelope containing the SAML <Response> it obtained, or a SOAP fault, as warranted.
6. The SASL Server indicates success or failure.

Note: The details of the SAML processing, which are consistent with the Enhanced Client or Proxy (ECP) Profile (V2.0) [SAMLECP20], are such that the client MUST interact with the IdP in order to complete any SASL exchange with the RP. The assertions issued by the IdP for the purposes of the profile, and by extension this SASL mechanism, are short lived, and therefore cannot be cached by the client for later use.

Encompassed in step four is the client-driven selection of the IdP, authentication to it, and the acquisition of a response to provide to the SASL server. These processes are all external to SASL.

Note also that unlike an HTTP-based profile, the IdP cannot participate in the selection of, or evaluation of, the location to which the SASL Client Response will be delivered by the client. The

use of GSS-API Channel Binding is an important mitigation of the risk of a "Man in the Middle" attack between the client and RP, as is the use of a negotiated or derived session key in whatever protocol is secured by this mechanism.

With all of this in mind, the typical flow appears as follows:

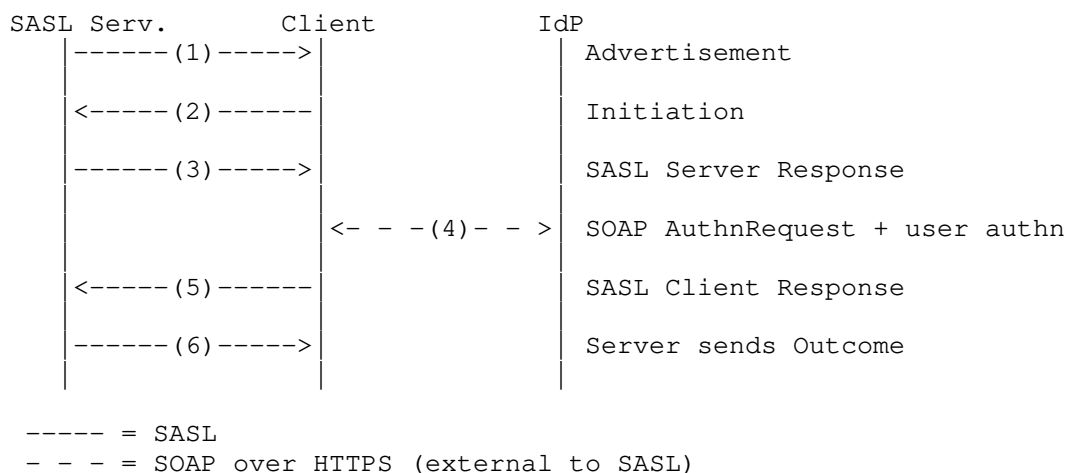


Figure 3: Authentication flow

#### 4. SAML Enhanced Client SASL Mechanism Specification

Based on the previous figures, the following operations are defined by the SAML SASL mechanism:

##### 4.1. Advertisement

To advertise that a server supports this mechanism, during application session initiation, it displays the name "SAML20EC" and/or "SAML20EC-PLUS" in the list of supported SASL mechanisms.

In accordance with [RFC5801] the "-PLUS" variant indicates that the server supports channel binding and would be selected by a client with that capability.

#### 4.2. Initiation

A client initiates "SAML20EC" or "SAML20EC-PLUS" authentication. If supported by the application protocol, the client MAY include an initial response, otherwise it waits until the server has issued an empty challenge (because the mechanism is client-first).

The format of the initial client response ("initresp") is as follows:

```
hok = "urn:oasis:names:tc:SAML:2.0:cm:holder-of-key"
```

```
mut = "urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp:2.0:" \
      "WantAuthnRequestsSigned"
```

```
del = "urn:oasis:names:tc:SAML:2.0:conditions:delegation"
```

```
initresp = gs2-cb-flag "," [gs2-authzid] "," [hok] "," [mut] "," [del]
```

The gs2-cb-flag flag MUST be set as defined in [RFC5801] to indicate whether the client supports channel binding. This takes the place of the PAOS HTTP header extension used in [SAMLECP20] to indicate channel binding support.

The optional "gs2-authzid" field holds the authorization identity, as requested by the client.

The optional "hok" field is a constant that signals the client's support for stronger security by means of a locally held key. This takes the place of the PAOS HTTP header extension used in [SAMLECP20] to indicate "holder of key" support.

The optional "mut" field is a constant that signals the client's desire for mutual authentication. If set, the SASL server MUST digitally sign its SAML <AuthnRequest> message. The URN constant above is a single string; the linefeed is shown for RFC formatting reasons.

The optional "del" field is a constant that signals the client's desire for the acceptor to request an assertion usable for delegation of the client's identity to the acceptor.

#### 4.3. Server Response

The SASL server responds with a SOAP envelope constructed in accordance with section 2.3.2 of [SAMLECP20]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing

profile. Various SOAP headers are also consumed by the client in exactly the same manner prescribed by that section.

#### 4.4. User Authentication with Identity Provider

Upon receipt of the Server Response (Section 4.3), the steps described in sections 2.3.3 through 2.3.6 of [SAMLECP20] are performed between the client and the chosen IdP. The means by which the client determines the IdP to use, and where it is located, are out of scope of this mechanism.

The exact means of authentication to the IdP are also out of scope, but clients supporting this mechanism MUST support HTTP Basic Authentication as defined in [RFC7617] and TLS 1.3 client authentication as defined in [RFC8446].

#### 4.5. Client Response

Assuming a response is obtained from the IdP, the client responds to the SASL server with a SOAP envelope constructed in accordance with section 2.3.7 of [SAMLECP20]. This includes adhering to the SOAP header requirements of the SAML PAOS Binding [OASIS.saml-bindings-2.0-os], for compatibility with the existing profile. If the client is unable to obtain a response from the IdP, or must otherwise signal failure, it responds to the SASL server with a SOAP envelope containing a SOAP fault.

#### 4.6. Outcome

The SAML protocol exchange having completed, the SASL server will transmit the outcome to the client depending on local validation of the client responses (including the assertion conveyed from the chosen IDP). This outcome is transmitted in accordance with the application protocol in use.

#### 4.7. Additional Notes

Because this mechanism is an adaptation of an HTTP-based profile, there are a few requirements outlined in [SAMLECP20] that make reference to a response URL that is normally used to regulate where the client returns information to the RP. There are also security-related checks built into the profile that involve this location.

For compatibility with existing IdP and profile behavior, and to provide for mutual authentication, the SASL server MUST populate the responseConsumerURL and AssertionConsumerServiceURL attributes with its service name. As discussed in Section 5.6.2, most SASL profiles rely on a service name format of "service@host", but regardless of



the form, the service name is used directly rather than transformed into an absolute URI if it is not already one, and MUST be percent-encoded per [RFC3986].

The IdP MUST securely associate the service name with the SAML entityID claimed by the SASL server, such as through the use of SAML metadata [OASIS.saml-metadata-2.0-os]. If metadata is used, a SASL service's <SPSSODescriptor> role MUST contain a corresponding <AssertionConsumerService> whose Location attribute contains the appropriate service name, as described above. The Binding attribute MUST be one of "urn:ietf:params:xml:ns:saml" (RECOMMENDED) or "urn:oasis:names:tc:SAML:2.0:bindings:PAOS" (for compatibility with older implementations of the ECP profile in existing IdP software).

Finally, note that the use of HTTP status signaling between the RP and client mandated by [SAMLECP20] may not be applicable.

## 5. SAML EC GSS-API Mechanism Specification

This section and its sub-sections and all normative references of it not referenced elsewhere in this document are INFORMATIONAL for SASL implementors, but they are NORMATIVE for GSS-API implementors.

The SAML Enhanced Client SASL mechanism is also a GSS-API mechanism. The messages are the same, but a) the GS2 [RFC5801] header on the client's first authentication message is excluded when SAML EC is used as a GSS-API mechanism, and b) the [RFC2743] section 3.1 initial context token header is used for the client's first authentication message (context token) instead, with the body of the message being the same as for the SASL mechanism case.

The GSS-API mechanism OID for SAML EC is OID-TBD (IANA to assign: see IANA considerations). The DER encoding of the OID is TBD.

The mutual\_state request flag (GSS\_C\_MUTUAL\_FLAG) MAY be set to TRUE, resulting in the "mut" option set in the initial client response. The security context mutual\_state flag is set to TRUE only if the server digitally signs its SAML <AuthnRequest> message and the signature and signing credential are appropriately verified by the IdP. The IdP signals this to the client in an <ecp:RequestAuthenticated> SOAP header block.

The lifetime of a security context established with this mechanism SHOULD be limited by the value of a SessionNotOnOrAfter attribute, if any, in the <AuthnStatement> element(s) of the SAML assertion(s) received by the RP. By convention, in the rare case that multiple valid/confirmed assertions containing <AuthnStatement> elements are

received, the most restrictive SessionNotOnOrAfter is generally applied.

#### 5.1. GSS-API Credential Delegation

This mechanism can support credential delegation through the issuance of SAML assertions that an IdP will accept as proof of authentication by a service on behalf of a subject. An initiator may request delegation of its credentials by setting the "del" option field in the initial client response to "urn:oasis:names:tc:SAML:2.0:conditions:delegation".

An acceptor, upon receipt of this constant, requests a delegated assertion by including in its <AuthnRequest> message a <Conditions> element containing an <AudienceRestriction> identifying the IdP as a desired audience for the assertion(s) to be issued. In the event that the specific IdP to be used is unknown, the constant "urn:oasis:names:tc:SAML:2.0:conditions:delegation" may be used as a stand-in, per Section 2.3.2 of [SAMLECP20].

Upon receipt of an assertion satisfying this property, and containing a <SubjectConfirmation> element that the acceptor can satisfy, the security context will have its deleg\_state flag (GSS\_C\_DELEG\_FLAG) set to TRUE.

The IdP, if it issues a delegated assertion to the acceptor, MUST include in the SOAP response to the initiator a <samlec:Delegated> SOAP header block, indicating that delegation was enabled. It has no content, other than mandatory SOAP attributes (an example follows):

```
<samlec:Delegated xmlns:samlec="urn:ietf:params:xml:ns:samlec"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  S:mustUnderstand="1"
  S:actor="http://schemas.xmlsoap.org/soap/actor/next" />
```

Upon receipt of such a header block, the initiator MUST fail the establishment of the security context if it did not request delegation in its initial client response to the acceptor. It SHOULD signal this failure to the acceptor with a SOAP fault message in its final client response.

As noted previously, the exact means of client authentication to the IdP is formally out of scope of this mechanism. This extends to the use of a delegation assertion as a means of authentication by an acceptor acting as an initiator. In practice, some profile of

[WSS-SAML] is used to attach the assertion and a confirmation proof to the SOAP message from the client to the IdP.

## 5.2. GSS-API Channel Binding

GSS-API channel binding [RFC5554] is a protected facility for exchanging a cryptographic identifier for an enclosing channel between the initiator and acceptor. The initiator sends channel binding data and the acceptor confirms that channel binding data has been checked.

The acceptor SHOULD accept any channel binding provided by the initiator if null channel bindings are passed into `gss_accept_sec_context`. Protocols such as HTTP Negotiate [RFC4559] depend on this behavior of some Kerberos implementations.

The exchange and verification of channel binding information is described by [SAMLECP20].

## 5.3. Session Key Derivation

Some GSS-API features (discussed in the following sections) require a session key be established as a result of security context establishment. In the common case of a "bearer" assertion in SAML, a mechanism is defined to communicate a key to both parties via the IdP. In other cases such as assertions based on "holder of key" confirmation bound to a client-controlled key, there may be additional methods defined in the future, and extension points are provided for this purpose.

Information defining or describing the session key, or a process for deriving one, is communicated between the initiator and acceptor using a `<samlec:SessionKey>` element, defined by the XML schema in Appendix A. This element is a SOAP header block. The content of the element further depends on the specific use in the mechanism. The Algorithm XML attribute identifies a mechanism for key derivation. It is omitted to identify the use of an IdP-generated key (see following section) or will contain a URI value identifying a derivation mechanism defined outside this specification. Each header block's `mustUnderstand` and `actor` attributes MUST be set to "1" and "`http://schemas.xmlsoap.org/soap/actor/next`" respectively.

In the acceptor's first response message containing its SAML request, one or more `<samlec:SessionKey>` SOAP header blocks MUST be included. The element MUST contain one or more `<EncType>` elements containing the number of a supported encryption type defined in accordance with [RFC3961]. Encryption types should be provided in order of preference by the acceptor.

In the final client response message, a single <samlec:SessionKey> SOAP header block MUST be included. A single <EncType> element MUST be included to identify the chosen encryption type used by the initiator.

All parties MUST support the "aes128-cts-hmac-sha1-96" encryption type, number 17, defined by [RFC3962].

Further details depend on the mechanism used, one of which is described in the following section.

#### 5.3.1. Generated by Identity Provider

The IdP, if issuing a bearer assertion for use with this mechanism, SHOULD provide a generated key for use by the initiator and acceptor. This key is used as pseudorandom input to the "random-to-key" function for a specific encryption type defined in accordance with [RFC3961]. The key is base64-encoded and placed inside a <samlec:GeneratedKey> element. The IdP does not participate in the selection of the encryption type and simply generates enough pseudorandom bits to supply key material to the other parties.

The resulting <samlec:GeneratedKey> element is placed within the <saml:Advice> element of the assertion issued. The identity provider MUST encrypt the assertion (implying that it MUST have the means to do so, typically knowledge of a key associated with the RP). If multiple assertions are issued (allowed, but not typical), the element need only be included in one of the assertions issued for use by the relying party.

A copy of the element is also added as a SOAP header block in the response from the IdP to the client (and then removed when constructing the response to the acceptor).

If this mechanism is used by the initiator, then the <samlec:SessionKey> SOAP header block attached to the final client response message will identify this via the omission of the Algorithm attribute and will identify the chosen encryption type using the <samlec:EncType> element:

```
<samlec:SessionKey xmlns:samlec="urn:ietf:params:xml:ns:samlec"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  S:mustUnderstand="1"
  S:actor="http://schemas.xmlsoap.org/soap/actor/next">
  <samlec:EncType>17</samlec:EncType>
</samlec:SessionKey>
```

Both the initiator and acceptor MUST execute the chosen encryption type's random-to-key function over the pseudorandom value provided by the <samlec:GeneratedKey> element. The result of that function is used as the protocol and session key. Support for subkeys from the initiator or acceptor is not specified.

#### 5.3.2. Alternate Key Derivation Mechanisms

In the event that a client is proving possession of a secret or private key, a formal key agreement algorithm might be supported. This specification does not define such a mechanism, but the <samlec:SessionKey> element is extensible to allow for future work in this space by means of the Algorithm attribute and an optional <ds:KeyInfo> child element to carry extensible content related to key establishment.

However a key is derived, the <samlec:EncType> element will identify the chosen encryption type, and both the initiator and acceptor MUST execute the encryption type's random-to-key function over the result of the key agreement or derivation process. The result of that function is used as the protocol key.

#### 5.4. Per-Message Tokens

The per-message tokens SHALL be the same as those for the Kerberos V5 GSS-API mechanism [RFC4121] (see Section 4.2 and sub-sections).

The replay\_det\_state (GSS\_C\_REPLAY\_FLAG), sequence\_state (GSS\_C\_SEQUENCE\_FLAG), conf\_avail (GSS\_C\_CONF\_FLAG) and integ\_avail (GSS\_C\_INTEG\_FLAG) security context flags are always set to TRUE.

The "protocol key" SHALL be a key established in a manner described in the previous section. "Specific keys" are then derived as usual as described in Section 2 of [RFC4121], [RFC3961], and [RFC3962].

The terms "protocol key" and "specific key" are Kerberos V5 terms [RFC3961].

SAML20EC is PROT\_READY as soon as the SAML response message has been seen.

#### 5.5. Pseudo-Random Function (PRF)

The GSS-API has been extended with a Pseudo-Random Function (PRF) interface in [RFC4401]. The purpose is to enable applications to derive a cryptographic key from an established GSS-API security context. This section defines a GSS\_Pseudo\_random that is applicable for the SAML20EC GSS-API mechanism.

The `GSS_Pseudo_random()` [RFC4401] SHALL be the same as for the Kerberos V5 GSS-API mechanism [RFC7802]. There is no acceptor-asserted sub-session key, thus `GSS_C_PRF_KEY_FULL` and `GSS_C_PRF_KEY_PARTIAL` are equivalent. The protocol key to be used for the `GSS_Pseudo_random()` SHALL be the same as the key defined in the previous section.

#### 5.6. GSS-API Principal Name Types for SAML EC

Services that act as SAML relying parties are typically identified by means of a URI called an "entityID". Clients that are named in the <Subject> element of a SAML assertion are typically identified by means of a <NameID> element, which is an extensible XML structure containing, at minimum, an element value that names the subject and a Format attribute.

In practice, a GSS-API client and server are unlikely to know in advance the name of the initiator as it will be expressed by the SAML IdP upon completion of authentication. It is also generally incorrect to assume that a particular acceptor name will directly map into a particular RP entityID, because there is often a layer of naming indirection between particular services on hosts and the identity of a relying party in SAML terms.

To avoid complexity, and avoid unnecessary use of XML within the naming layer, the SAML EC mechanism relies on the common/expected name types used for acceptors and initiators, `GSS_C_NT_HOSTBASED_SERVICE` and `GSS_C_NT_USER_NAME`. The mechanism provides for validation of the host-based service name in conjunction with the SAML exchange. It does not attempt to solve the problem of mapping between an initiator "username", the user's identity while authenticating to the IdP, and the information supplied by the IdP to the acceptor. These relationships must be managed through local policy at the initiator and acceptor.

SAML-based information associated with the initiator SHOULD be expressed to the acceptor using GSS-API naming extensions [RFC6680], in a similar manner to [RFC7056].

##### 5.6.1. User Naming Considerations

The `GSS_C_NT_USER_NAME` form represents the name of an individual user. Clients often rely on this value to determine the appropriate credentials to use in authenticating to the IdP, and supply it to the server for use by the acceptor.

Upon successful completion of this mechanism, the server MUST construct the authenticated initiator name based on the <saml:NameID>

element in the assertion it successfully validated. The name is constructed as a UTF-8 string in the following form:

```
name = element-value "!" Format "!" NameQualifier  
      "!" SPNameQualifier "!" SPProvidedID
```

The "element-value" token refers to the content of the <saml:NameID> element. The other tokens refer to the identically named XML attributes defined for use with the element. If an attribute is not present, which is common, it is omitted (i.e., replaced with the empty string). The Format value is never omitted; if not present, the SAML-equivalent value of "urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified" is used.

Not all SAML assertions contain a <saml:NameID> element. In the event that no such element is present, including the exceptional cases of a <saml:BaseID> element or a <saml:EncryptedID> element that cannot be decrypted, the GSS\_C\_NT\_ANONYMOUS name type MUST be used for the initiator name.

As noted in the previous section, it is expected that most applications able to rely on SAML authentication would make use of naming extensions to obtain additional information about the user based on the assertion. This is particularly true in the anonymous case, or in cases in which the SAML name is pseudonymous or transient in nature. The ability to express the SAML name in GSS\_C\_NT\_USER\_NAME form is intended for compatibility with applications that cannot make use of additional information.

#### 5.6.2. Service Naming Considerations

The GSS\_C\_NT\_HOSTBASED\_SERVICE name form represents a service running on a host; it is textually represented as "service@host". This name form is required by most SASL profiles and is used by many existing applications that use the Kerberos GSS-API mechanism. As described in the SASL mechanism's Section 4.7, such a name is used directly by this mechanism as the effective AssertionConsumerService "location" associated with the service and applied in IdP verification of the request against the claimed SAML entityID.

#### 6. Example

Suppose the user has an identity at the SAML IdP saml.example.org and a Jabber Identifier (jid) "somenode@example.com", and wishes to authenticate his XMPP connection to xmpp.example.com (and example.com and example.org have established a SAML-capable trust relationship). The authentication on the wire would then look something like the following:

Step 1: Client initiates stream to server:

```
<stream:stream xmlns='jabber:client'
xmlns:stream='http://etherx.jabber.org/streams'
to='example.com' version='1.0'>
```

Step 2: Server responds with a stream tag sent to client:

```
<stream:stream
xmlns='jabber:client' xmlns:stream='http://etherx.jabber.org/streams'
id='some_id' from='example.com' version='1.0'>
```

Step 3: Server informs client of available authentication mechanisms:

```
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <mechanism>SAML20EC</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Client selects an authentication mechanism and sends the initial client response (it is base64 encoded as specified by the XMPP SASL protocol profile):

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl' mechanism='SAML20EC'>
biwsLCw=
</auth>
```

The initial response is "n,,," which signals that channel binding is not used, there is no authorization identity, and the client does not support key-based confirmation, or want mutual authentication or delegation.

Step 5: Server sends a challenge to client in the form of a SOAP envelope containing its SAML <AuthnRequest>:



[illegible]

The Base64 [RFC4648] decoded envelope:

```

<S:Envelope
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  <S:Header>
    <paos:Request xmlns:paos="urn:liberty:paos:2003-08"
      messageID="c3a4f8b9c2d" S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      responseConsumerURL="xmpp@xmpp.example.com"
      service="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"/>
    <ecp:Request
      xmlns:ecp="urn:oasis:names:tc:SAML:2.0:profiles:SSO:ecp"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      S:mustUnderstand="1" ProviderName="Jabber at example.com">
      <saml:Issuer>https://xmpp.example.com</saml:Issuer>
    </ecp:Request>
    <samlec:SessionKey xmlns:samlec="urn:ietf:params:xml:ns:samlec"
      xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
      S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next">
      <samlec:EncType>17</samlec:EncType>
      <samlec:EncType>18</samlec:EncType>
    </samlec:SessionKey>
  </S:Header>
  <S:Body>
    <samlp:AuthnRequest
      ID="c3a4f8b9c2d" Version="2.0" IssueInstant="2020-12-10T11:39:34Z"
      AssertionConsumerServiceURL="xmpp@xmpp.example.com">
      <saml:Issuer xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion">
        https://xmpp.example.com
      </saml:Issuer>
      <samlp:NameIDPolicy AllowCreate="true"
        Format="urn:oasis:names:tc:SAML:2.0:nameid-format:persistent"/>
      <samlp:RequestedAuthnContext Comparison="exact">
        <saml:AuthnContextClassRef>
          urn:oasis:names:tc:SAML:2.0:ac:classes:PasswordProtectedTransport
        </saml:AuthnContextClassRef>
      </samlp:RequestedAuthnContext>
    </samlp:AuthnRequest>
  </S:Body>
</S:Envelope>

```

Step 5 (alt): Server returns error to client:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Step 6: Client relays the request to IdP in a SOAP message transmitted over HTTP (over TLS). The HTTP portion is not shown, so the use of Basic Authentication is assumed. The body of the SOAP envelope is exactly the same as received in the previous step.

```
<S:Envelope
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Body>
    <samlp:AuthnRequest>
      <!-- same as above -->
    </samlp:AuthnRequest>
  </S:Body>
</S:Envelope>
```

Step 7: IdP responds to client with a SOAP response containing a SAML <Response> containing a short-lived SSO assertion (shown as an encrypted variant in the example). A generated key is included in the assertion and in a header for the client.

```
<S:Envelope
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <ecp:Response S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      AssertionConsumerServiceURL="xmpp@xmpp.example.com"/>
    <samlec:GeneratedKey xmlns:samlec="urn:ietf:params:xml:ns:samlec">
      3w1wSBKUosRLsU69xGK7dg==
    </samlec:GeneratedKey>
  </S:Header>
  <S:Body>
    <samlp:Response ID="d43h94r389309r" Version="2.0"
      IssueInstant="2020-12-10T11:42:34Z" InResponseTo="c3a4f8b9c2d"
      Destination="xmpp@xmpp.example.com">
      <saml:Issuer>https://saml.example.org</saml:Issuer>
      <samlp:Status>
        <samlp:StatusCode
          Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
        </samlp:Status>
      <saml:EncryptedAssertion>
        <!-- contents elided, copy of samlec:GeneratedKey in Advice -->
      </saml:EncryptedAssertion>
    </samlp:Response>
  </S:Body>
</S:Envelope>
```

Step 8: Client sends SOAP envelope containing the SAML <Response> as a response to the SASL server's challenge:

```
<response xmlns='urn:ietf:params:xmml:sasasl'>  
PFM6RW52ZWxvcGUKICAgIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lc3p0Y2pzP  
QUlMOjIuMDphc3NlcnRpb24iCiAgICB4bWxuczpwYW1scD0idXJuOm9hc2lwZm5h  
bWVzOnRjOlNBtUw6Mi4wOnByb3RvY29sIgogICAgeGlSbnM6Uz0iaHR0cDovL3Nj  
agVtYXMueGlSc29hcC5vcmcvc29hcC9lbZlbg9wZS8iPgogIDxtOkhlYWRLcj4K  
ICAgIDxwYW9zOllJlc3BvbnlIHhtbG5zOnBhb3M9InVybjpsawJlcnR5OnBhb3M6  
MjAwMy0wOCIKICAgICAgUzphY29hcC9laHR0cDovL3NjaGVtYXMueGlSc29hcC5v  
cmcv29hcC9hY3Rvc1uZkY0IGogICAgICBTOM1lc3RVbmRlcnN0YW5kPSixLiBiY  
zcWUb01lc3NhZ2VJRd0iNmMZYTRMOGI5YzJkIi8+CiAgICAgICA8c2FtbGVjOlNlc3Np  
b25LZXkgeGlSbnM6c2FtbGVjPSJlcm46aWV0ZjpwaXhjbXBM6eGlSOm5zOnNhbWxl  
YyIKICAgICAgeGlSbnM6Uz0iaHR0cDovL3NjaGVtYXMueGlSc29hcC5vcmcvc29h  
cC9lbZlbg9wZS8iCiAgICAgIFM6bXVzdFVuZGVyc3RhbmQ9IjeiEiCiAgICAgIFM6  
YWNOb3I9Imh0dHA6Ly9zY2h1bWFzfLnhtbHNvYXAub3JnL3NvYXAvYWNOb3IvbmV4  
dCI+CiAgICAgIDxzYWlsZWMM6RW5jVHlwZT5hZXMXmjgtY3RzLWhtYWMTc2hhMS05  
Njwvc2FtbGVjOkVuY1R5cGU+CIAgICA8c2FtbGVjOlNlc3Npb25LZXk+CIAgPC9T  
OkhlYWRLcj4KICAgUzpCb2R5PgogICAgPHNhbWxwOlJlc3BvbnlIELEPSJkdNDNO  
OTryMzg5MZA5ciIgVmVyc2lvbj0iMCA4IGogICAgICAgICAgICAgICAgICAgICAgIC  
MjAwNy0xmI0i0XFQgcXMTOMj0mgzNFoiIEluUmVzcG9uc2VUBz0iYzNhNGY4YjljMmQi  
CiAgICAgICAgRGVzdGluYXRpb249InhtCHBAeGlwcC5leGFtcGxlLmNvbSI+CIAg  
ICAgIDxzYWlsOkIzc3Vlcj5odHRwczovL3NhbWwuZXhhbXBsZS5vcmc8L3NhbWw6  
SXNZdWVyPgogICAgICA8c2FtbHA6U3RhdHVzPgogICAgICAgIDxzYWlsCdPtTdGF0  
dXNDb2RlCiAgICAgICAgICAgIFZhbnVlPSJlcm46b2FzaXM6bmFtZXMM6dGM6U0FN  
TDoyLjA6c3RhdHVzOlN1Y2Nlc3MiLz4KICAgICAgPC9zYW1scDptTdGF0dXM+CIAg  
ICAgIDxzYWlsOkVuY3J5cHRlZEZFzc2VydGlwbj4KICAgICAgICA8IS0tIGNvbnlRl  
bnRzIGVsawRlZCWgY29weSBvZiBzYWlsZWMM6R2VuZXJhdGVkS2V5IGluIEFkdmlj  
ZSATLT4KICAgICAgPC9zYWlsOkVuY3J5cHRlZEZFzc2VydGlwbj4KICAgIDwvc2Ft  
bHA6UmVzcG9uc2U+CIAgPC9TOkJvZHk+CjwwUzpfbnZlbg9wZT4K  
</response>
```

The Base64 [RFC4648] decoded envelope:

```

<S:Envelope
  xmlns:saml="urn:oasis:names:tc:SAML:2.0:assertion"
  xmlns:samlp="urn:oasis:names:tc:SAML:2.0:protocol"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/">
  <S:Header>
    <paos:Response xmlns:paos="urn:liberty:paos:2003-08"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next"
      S:mustUnderstand="1" refToMessageID="6c3a4f8b9c2d"/>
    <samlec:SessionKey xmlns:samlec="urn:ietf:params:xml:ns:samlec"
      xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
      S:mustUnderstand="1"
      S:actor="http://schemas.xmlsoap.org/soap/actor/next">
      <samlec:EncType>17</samlec:EncType>
    <samlec:SessionKey>
  </S:Header>
  <S:Body>
    <samlp:Response ID="d43h94r389309r" Version="2.0"
      IssueInstant="2020-12-10T11:42:34Z" InResponseTo="c3a4f8b9c2d"
      Destination="xmpp@xmpp.example.com">
      <saml:Issuer>https://saml.example.org</saml:Issuer>
      <samlp:Status>
        <samlp:StatusCode
          Value="urn:oasis:names:tc:SAML:2.0:status:Success"/>
      </samlp:Status>
      <saml:EncryptedAssertion>
        <!-- contents elided, copy of samlec:GeneratedKey in Advice -->
      </saml:EncryptedAssertion>
    </samlp:Response>
  </S:Body>
</S:Envelope>

```

Step 9: Server informs client of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl' />
```

Step 9 (alt): Server informs client of failed authentication:

```

<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</failure>
</stream:stream>

```

Step 10: Client initiates a new stream to server:

```
<stream:stream xmlns='jabber:client'  
xmlns:stream='http://etherx.jabber.org/streams'  
to='example.com' version='1.0'>
```

Step 11: Server responds by sending a stream header to client along with any additional features (or an empty features element):

```
<stream:stream xmlns='jabber:client'  
xmlns:stream='http://etherx.jabber.org/streams'  
id='c2s_345' from='example.com' version='1.0'>  
<stream:features>  
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind' />  
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session' />  
</stream:features>
```

Step 12: Client binds a resource:

```
<iq type='set' id='bind_1'>  
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>  
    <resource>someresource</resource>  
  </bind>  
</iq>
```

Step 13: Server informs client of successful resource binding:

```
<iq type='result' id='bind_1'>  
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>  
    <jid>somenode@example.com/someresource</jid>  
  </bind>  
</iq>
```

Please note: line breaks were added to the base64 for clarity.

## 7. Security Considerations

This section will address only security considerations associated with the use of SAML with SASL applications. For considerations relating to SAML in general, the reader is referred to the SAML specification and to other literature. Similarly, for general SASL Security Considerations, the reader is referred to that specification.

Version 2.0 of the Enhanced Client or Proxy Profile [SAMLECP20] adds optional support for channel binding and use of "Holder of Key" subject confirmation. The former is strongly recommended for use with this mechanism to detect "Man in the Middle" attacks between the client and the RP without relying on the commercial TLS infrastructure that does not provide the level of assurance desired by sensitive SAML applications. The latter may be impractical in many cases, but is a valuable way of strengthening client authentication, protecting against phishing, and improving the overall mechanism.

#### 7.1. Risks Left Unaddressed

The adaptation of a web-based profile that is largely designed around security-oblivious clients and a bearer model for security token validation results in a number of basic security exposures that should be weighed against the compatibility and client simplification benefits of this mechanism.

When channel binding is not used, protection against "Man in the Middle" attacks is left solely to lower layer protocols such as TLS, and the development of user interfaces able to implement that has not been effectively demonstrated. Failure to detect a MITM can result in phishing of the user's credentials if the attacker is between the client and IdP, or the theft and misuse of a short-lived credential (the SAML assertion) if the attacker is able to impersonate a RP. SAML allows for source address checking as a minor mitigation to the latter threat, but this is often impractical. IdPs can mitigate to some extent the exposure of personal information to RP attackers by encrypting assertions with authenticated keys.

#### 7.2. User Privacy

The IdP is aware of each RP that a user logs into. There is nothing in the protocol to hide this information from the IdP. It is not a requirement to track the activity, but there is nothing technically that prohibits the collection of this information. Servers should be aware that SAML IdPs will track - to some extent - user access to their services. This exposure extends to the use of session keys generated by the IdP to secure messages between the parties, but note that when bearer assertions are involved, the IdP can freely impersonate the user to any relying party in any case.

It is also out of scope of the mechanism to determine under what conditions an IdP will release particular information to a relying party, and it is generally unclear in what fashion user consent could be established in real time for the release of particular



information. The SOAP exchange with the IdP does not preclude such interaction, but neither does it define that interoperably.

### 7.3. Collusion between RPs

Depending on the information supplied by the IdP, it may be possible for RPs to correlate data that they have collected. By using the same identifier to log into every RP, collusion between RPs is possible. SAML supports the notion of pairwise, or targeted/directed, identity. This allows the IdP to manage opaque, pairwise identifiers for each user that are specific to each RP. However, correlation is often possible based on other attributes supplied, and is generally a topic that is beyond the scope of this mechanism. It is sufficient to say that this mechanism does not introduce new correlation opportunities over and above the use of SAML in web-based use cases.

## 8. IANA Considerations

### 8.1. GSS-API and SASL Mechanism Registration

The IANA is requested to assign a new entry for this GSS mechanism in the sub-registry for SMI Security for Mechanism Codes, whose prefix is iso.org.dod.internet.security.mechanisms (1.3.6.1.5.5) and to reference this specification in the registry.

The IANA is requested to register the following SASL profile:

SASL mechanism profiles: SAML20EC and SAML20EC-PLUS

Security Considerations: See this document

Published Specification: See this document

For further information: Contact the authors of this document.

Owner/Change controller: the IETF

Note: None

### 8.2. XML Namespace Name for SAML-EC

A URN sub-namespace for XML constructs introduced by this mechanism is defined as follows:

URI: urn:ietf:params:xml:ns:samlec

Specification: See Appendix A of this document.

Description: This is the XML namespace name for XML constructs introduced by the SAML Enhanced Client SASL and GSS-API Mechanisms.

Registrant Contact: the IESG

## 9. References

### 9.1. Normative References

- [OASIS.saml-bindings-2.0-os]  
Cantor, S., Hirsch, F., Kemp, J., Philpott, R., and E. Maler, "Bindings for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-bindings-2.0-os, March 2005.
- [OASIS.saml-core-2.0-os]  
Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-core-2.0-os, March 2005.
- [OASIS.saml-profiles-2.0-os]  
Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., and E. Maler, "Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0", OASIS Standard OASIS.saml-profiles-2.0-os, March 2005.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <<https://www.rfc-editor.org/info/rfc2119>>.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, DOI 10.17487/RFC2743, January 2000, <<https://www.rfc-editor.org/info/rfc2743>>.
- [RFC3961] Raeburn, K., "Encryption and Checksum Specifications for Kerberos 5", RFC 3961, DOI 10.17487/RFC3961, February 2005, <<https://www.rfc-editor.org/info/rfc3961>>.
- [RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, DOI 10.17487/RFC3962, February 2005, <<https://www.rfc-editor.org/info/rfc3962>>.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <<https://www.rfc-editor.org/info/rfc3986>>.
- [RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, DOI 10.17487/RFC4121, July 2005, <<https://www.rfc-editor.org/info/rfc4121>>.
- [RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, DOI 10.17487/RFC4401, February 2006, <<https://www.rfc-editor.org/info/rfc4401>>.
- [RFC4422] Melnikov, A., Ed. and K. Zeilenga, Ed., "Simple Authentication and Security Layer (SASL)", RFC 4422, DOI 10.17487/RFC4422, June 2006, <<https://www.rfc-editor.org/info/rfc4422>>.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, DOI 10.17487/RFC4648, October 2006, <<https://www.rfc-editor.org/info/rfc4648>>.
- [RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, DOI 10.17487/RFC5554, May 2009, <<https://www.rfc-editor.org/info/rfc5554>>.
- [RFC5801] Josefsson, S. and N. Williams, "Using Generic Security Service Application Program Interface (GSS-API) Mechanisms in Simple Authentication and Security Layer (SASL): The GS2 Mechanism Family", RFC 5801, DOI 10.17487/RFC5801, July 2010, <<https://www.rfc-editor.org/info/rfc5801>>.
- [RFC6680] Williams, N., Johansson, L., Hartman, S., and S. Josefsson, "Generic Security Service Application Programming Interface (GSS-API) Naming Extensions", RFC 6680, DOI 10.17487/RFC6680, August 2012, <<https://www.rfc-editor.org/info/rfc6680>>.
- [RFC7056] Hartman, S. and J. Howlett, "Name Attributes for the GSS-API Extensible Authentication Protocol (EAP) Mechanism", RFC 7056, DOI 10.17487/RFC7056, December 2013, <<https://www.rfc-editor.org/info/rfc7056>>.

- [RFC7617] Reschke, J., "The 'Basic' HTTP Authentication Scheme", RFC 7617, DOI 10.17487/RFC7617, September 2015, <<https://www.rfc-editor.org/info/rfc7617>>.
- [RFC7802] Emery, S. and N. Williams, "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 7802, DOI 10.17487/RFC7802, March 2016, <<https://www.rfc-editor.org/info/rfc7802>>.
- [RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <<https://www.rfc-editor.org/info/rfc8174>>.
- [RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <<https://www.rfc-editor.org/info/rfc8446>>.
- [SAMLECP20] Cantor, S., "SAML V2.0 Enhanced Client or Proxy Profile Version 2.0", OASIS Committee Specification OASIS.sstc-saml-ecp-v2.0-cs01, August 2013.
- [W3C.soap11] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H., Thatte, S., and D. Winer, "Simple Object Access Protocol (SOAP) 1.1", W3C Note soap11, May 2000, <<http://www.w3.org/TR/SOAP/>>.

## 9.2. Informative References

- [OASIS.saml-metadata-2.0-os] Cantor, S., Moreh, J., Philpott, R., and E. Maler, "Metadata for the Security Assertion Markup Language (SAML) V2.0", OASIS Standard saml-metadata-2.0-os, March 2005.
- [RFC1939] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, DOI 10.17487/RFC1939, May 1996, <<https://www.rfc-editor.org/info/rfc1939>>.
- [RFC3501] Crispin, M., "INTERNET MESSAGE ACCESS PROTOCOL - VERSION 4rev1", RFC 3501, DOI 10.17487/RFC3501, March 2003, <<https://www.rfc-editor.org/info/rfc3501>>.

- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, DOI 10.17487/RFC4559, June 2006, <<https://www.rfc-editor.org/info/rfc4559>>.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, DOI 10.17487/RFC6120, March 2011, <<https://www.rfc-editor.org/info/rfc6120>>.
- [RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <<https://www.rfc-editor.org/info/rfc7230>>.
- [W3C.REC-xmlschema-1]  
Thompson, H., Beech, D., Maloney, M., and N. Mendelsohn, "XML Schema Part 1: Structures", W3C REC-xmlschema-1, May 2001, <<http://www.w3.org/TR/xmlschema-1/>>.
- [WSS-SAML]  
Monzillo, R., "Web Services Security SAML Token Profile Version 1.1.1", OASIS Standard OASIS.wss-SAMLSecurityTokenProfile, May 2012.

## Appendix A. XML Schema

The following schema formally defines the "urn:ietf:params:xml:ns:saml" namespace used in this document, in conformance with [W3C.REC-xmlschema-1] While XML validation is optional, the schema that follows is the normative definition of the constructs it defines. Where the schema differs from any prose in this specification, the schema takes precedence.

```
<schema
  targetNamespace="urn:ietf:params:xml:ns:samlec"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ds="http://www.w3.org/2000/09/xmldsig#"
  xmlns:S="http://schemas.xmlsoap.org/soap/envelope/"
  xmlns:samlec="urn:ietf:params:xml:ns:samlec"
  elementFormDefault="unqualified"
  attributeFormDefault="unqualified"
  blockDefault="substitution"
  version="1.0">

  <import namespace="http://www.w3.org/2000/09/xmldsig#" />
  <import namespace="http://schemas.xmlsoap.org/soap/envelope/" />

  <element name="SessionKey" type="samlec:SessionKeyType" />
  <complexType name="SessionKeyType">
    <sequence>
      <element ref="samlec:EncType" maxOccurs="unbounded" />
      <element ref="ds:KeyInfo" minOccurs="0" />
    </sequence>
    <attribute ref="S:mustUnderstand" use="required" />
    <attribute ref="S:actor" use="required" />
    <attribute name="Algorithm" />
  </complexType>

  <element name="EncType" type="integer" />

  <element name="GeneratedKey" type="samlec:GeneratedKeyType" />
  <complexType name="GeneratedKeyType">
    <simpleContent>
      <extension base="base64Binary">
        <attribute ref="S:mustUnderstand" />
        <attribute ref="S:actor" />
      </extension>
    </simpleContent>
  </complexType>

  <element name="Delegated" type="samlec:DelegatedType" />
  <complexType name="DelegatedType">
    <sequence />
    <attribute ref="S:mustUnderstand" use="required" />
    <attribute ref="S:actor" use="required" />
  </complexType>

</schema>
```

## Appendix B. Acknowledgments

The authors would also like to thank Klaas Wierenga, Sam Hartman, Nico Williams, Jim Basney, Venkat Yekkirala, and Ben Kaduk for their contributions.

## Appendix C. Changes

This section to be removed prior to publication.

- o 20, address nits and easy fixes from Ben Kaduk's AD review
- o 19, update obsoleted references
- o 15,16,17,18 avoid expiration
- o 14, address some minor comments
- o 13, clarify SAML metadata usage, adding a recommended Binding value alongside the backward-compatibility usage of PAOS
- o 12, clarifying comments based on WG feedback, with a normative change to use enctype numbers instead of names
- o 11, update EAP Naming reference to RFC
- o 10, update SAML ECP reference to final CS
- o 09, align delegation signaling to updated ECP draft
- o 08, more corrections, added a delegation signaling header
- o 07, corrections, revised section on delegation
- o 06, simplified session key schema, moved responsibility for random-to-key to the endpoints, and defined advertisement of session key algorithm and encypes by acceptor
- o 05, revised session key material, added requirement for random-to-key, revised XML schema to capture enctype name, updated GSS naming reference
- o 04, stripped down the session key material to simplify it, and define an IdP-brokered keying approach, moved session key XML constructs from OASIS draft into this one
- o 03, added TLS key export as a session key option, revised GSS naming material based on list discussion

- o 02, major revision of GSS-API material and updated references
- o 01, SSH language added, noted non-assumption of HTTP error handling, added guidance on life of security context.
- o 00, Initial Revision, first WG-adopted draft. Removed support for unsolicited SAML responses.

#### Authors' Addresses

Scott Cantor  
Shibboleth Consortium  
1050 Carmack Rd  
Columbus, Ohio 43210  
United States

Phone: +1 614 247 6147  
Email: cantor.2@osu.edu

Margaret Cullen  
Painless Security  
4 High St, Suite 134  
North Andover, Massachusetts 01845  
United States

Phone: +1 781 405 7464  
Email: mrcullen42@painless-security.com

Simon Josefsson  
SJD AB  
Hagagatan 24  
Stockholm 113 47  
SE

Email: [simon@josefsson.org](mailto:simon@josefsson.org)  
URI: <http://josefsson.org/>



KERBEROS WORKING GROUP  
Internet-Draft  
Intended status: Standards Track  
Expires: July 18, 2013

Johansson  
SUNET  
January 14, 2013

An information model for Kerberos version 5  
draft-ietf-krb-wg-kdc-model-16

Abstract

This document describes an information model for Kerberos version 5 from the point of view of an administrative service. There is no standard for administrating a kerberos 5 KDC. This document describes the services exposed by an administrative interface to a KDC.

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 July 18, 2013.

Copyright Notice

Copyright (c) 2013 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 . . . . .	3
2. Requirements notation . . . . .	4
3. Information model demarcation . . . . .	6
4. Information model specification . . . . .	7
4.1. Principal . . . . .	7
4.1.1. Principal: Attributes . . . . .	7
4.1.2. Principal: Associations . . . . .	8
4.2. KeySet . . . . .	9
4.2.1. KeySet: Attributes . . . . .	9
4.2.2. KeySet: Associations . . . . .	9
4.3. Key . . . . .	9
4.3.1. Key: Attributes . . . . .	10
4.3.2. Key: Associations . . . . .	10
4.3.3. Key: Remarks . . . . .	11
4.4. Policy . . . . .	11
4.4.1. Policy: Attributes . . . . .	11
4.4.2. Mandatory-to-implement Policy . . . . .	12
5. Implementation Scenarios . . . . .	13
5.1. LDAP backend to KDC . . . . .	13
5.2. LDAP frontend to KDC . . . . .	13
5.3. SOAP . . . . .	13
5.4. Netconf . . . . .	13
6. Security Considerations . . . . .	14
7. IANA Considerations . . . . .	15
8. Acknowledgments . . . . .	16
9. References . . . . .	17
9.1. Normative References . . . . .	17
9.2. Informative References . . . . .	17
Author's Address . . . . .	18

## 1. Introduction

The Kerberos version 5 authentication service described in [RFC4120] describes how a Key Distribution Center (KDC) provides authentication to clients. The standard does not stipulate how a KDC is managed and several "kadmin" servers have evolved. This document describes the services required to administer a KDC and the underlying information model assumed by a kadmin-type service.

The information model is written in terms of "attributes" and "services" or "interfaces" but the use of these particular words must not be taken to imply any particular modeling paradigm. Neither an object oriented model nor an LDAP [RFC4510] schema is intended. The author has attempted to describe in natural language the intended semantics and syntax of the components of the model. An LDAP schema (for instance) based on this model will be more precise in the expression of the syntax while preserving the semantics of this model.

Implementations of this document MAY decide to change the names used (e.g. principalName). If so an implementation MUST provide a name to name mapping to this document. In particular schema languages may have different conventions for caseing, eg camelCase vs use of '\_' and '-' to separate 'words' in a name. Implementations MUST call out such conventions explicitly.

Implementations of this document MUST be able to support default values for attributes as well as the ability to specify syntax for attribute values.

## 2. Requirements notation

This document uses the standard key words ("MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL") that are defined in [RFC2119] but with modifications to those definitions as described below. The reason for this (which was discussed extensively in the kerberos WG) is as follows:

This document describes an information model for kerberos 5 but does not directly describe any mapping onto a particular schema- or modelling language. Hence an implementation of this model consists of a mapping to such a language - e.g. an LDAP or SQL schema. The standard normative key word therefore require precise definition:

The terms MUST or REQUIRED means that schema implementing this model must have a way to represent a feature (i.e that it is mandatory to implement in schema) but that unless otherwise specified the feature may represent an optional element in the chosen schema definition language.

However MUST also means that a KDC or administrative interface implementing this information model MUST provide the feature and associated behavior consistent with schema.

For instance, `principalLastFailedAuthentication` (cf below) represents the last time an authentication failed for a principal. In an LDAP schema (for instance) this may be represented as an optional attribute even though all KDCs implementing this specification must support this attribute.

The terms MAY or OPTIONAL means that the feature is optional to implement by a KDC or administrative interface implementing this information model. It also means that the feature is optional to implement in schema.

Implementors of schema should be aware that unless there is a way to represent critical but optional elements in the schema definition language confusion may arise when optional elements are used but not understood by all implementations in a particular deployment.

The expression "MUST NOT be OPTIONAL" means that a feature is mandatory to implement ("MUST" cf above) and that additionally it must not be marked optional in the schema language. In particular this means that the feature is both mandatory to implement and must be present in all representations of the object to which it applies.

The term SHOULD or RECOMMENDED means that the consequences of not

implementing the feature as if it was described with the "MUST" keyword must be carefully weighed before choosing a different course. In particular this implies that interoperability concerns may arise from not following the recommended practice in schema that implements this model.

The context will determine if the "SHOULD" key word applies to schema, or to underlying behaviour of the KDC or both. For instance, `principalIsDisabled` (cf below) SHOULD default to FALSE implies both a recommendation for the behaviour of KDCs aswell as a recommendation for the representation of that behaviour in schema.

### 3. Information model demarcation

The information model specified in the next chapter describes objects, properties of those objects and relations between those objects. These elements comprise an abstract view of the data represented in a KDC. It is important to understand that the information model is not a schema. In particular the way objects are compared for equality beyond that which is implied by the specification of a syntax is not part of this specification. Nor is ordering specified between elements of a particular syntax.

Further work on Kerberos will undoubtedly prompt updates to this information model to reflect changes in the functions performed by the KDC. Such extensions to the information model should always use a normative reference to the relevant RFCs detailing the change in KDC function.

This model describes a number of elements related to password policy management. Not all of the elements in this model are unique to Kerberos; an LDAP implementation of this model should incorporate existing LDAP schema where functional overlap exists, rather than defining additional Kerberos-specific elements.

#### 4. Information model specification

##### 4.1. Principal

The fundamental entity stored in a KDC is the principal. The Principal is associated to keys and generalizes the "user" concept. The Principal **MUST** be implemented in full and **MUST NOT** be **OPTIONAL** in an implementation

##### 4.1.1. Principal: Attributes

###### 4.1.1.1. principalName

The principalName **MUST** uniquely identify the Principal within the administrative context of the KDC. The principalName **MUST** be equivalent to the string representation of the Principal name (section 2.1.1 of [RFC1964]) including, if applicable for the name type, the realm.

The attribute **MAY** be multi-valued if the implementation supports aliases and/or enterprise names. In that case exactly one of the principalName values **MAY** be designated the canonical principalName and if the implementation supports encetypes which require salt then exactly one of the values of principalName **MAY** be designated as the canonical salting principalName.

Implementations (i.e. schema) that support enterprise names and/or aliases **SHOULD** provide for efficient lookup of Principal objects based on alias/enterprise name.

###### 4.1.1.2. principalNotUsedBefore

The Principal **MUST NOT** be used before this date. The syntax of the attribute **MUST** be Internet Date/Time Format from [RFC3339]. The attribute **MUST** be single-valued.

###### 4.1.1.3. principalNotUsedAfter

The Principal **MUST NOT** be used after this date. The syntax of the attribute **MUST** be Internet Date/Time Format from [RFC3339]. The attribute **MUST** be single-valued.

###### 4.1.1.4. principalIsDisabled

A boolean attribute used to disable a Principal. The attribute **SHOULD** default to boolean **FALSE**.

#### 4.1.1.5. `principalLastCredentialChangeTime`

This single-valued attribute contains the time of the last successful change of credential (e.g. password or private key) associated with this Principal. The syntax of the attribute MUST be Internet Date/Time Format from [RFC3339].

#### 4.1.1.6. `principalCreateTime`

This single-valued attribute contains the time and date when this Principal was created. The syntax of the attribute MUST be Internet Date/Time Format from [RFC3339].

#### 4.1.1.7. `principalModifyTime`

This single-valued attribute contains the time and date when this Principal was last modified excluding credentials change. The syntax of the attribute MUST be Internet Date/Time Format from [RFC3339].

#### 4.1.1.8. `principalMaximumTicketLifetime`

This single-valued attribute contains the time in seconds representing the maximum lifetime for tickets issued for this Principal.

#### 4.1.1.9. `principalMaximumRenewableTicketLifetime`

This single-valued attribute contains the delta time in seconds representing the maximum amount of time a ticket may be renewed for this Principal.

#### 4.1.1.10. `principalAllowedEncatype`

This OPTIONAL multi-valued attribute lists the encatypes allowed for this principal. If empty or absent any encatype supported by the implementation is allowed for this Principal.

This attribute is intended as a policy attribute and restricts all uses of encatypes including server, client, and session keys. Data models MAY choose to use policy objects in order to represent more complex decision mechanisms.

#### 4.1.2. Principal: Associations

Each Principal MAY be associated with 0 or more KeySet and MAY be associated with 0 or more Policies. The KeySet is represented as an object in this model since it has attributes associated with it (the key version number). In typical situations the Principal is



associated with exactly 1 KeySet but implementations MUST NOT assume this case, i.e. an implementation of this standard MUST be able to handle the general case of multiple KeySet associated with each principal. Multiple KeySets may for instance be useful when performing a key rollover for a principal.

#### 4.2. KeySet

In Kerberos principals are associated with zero or more symmetric secret keys, and each key has a key version number (kvno) and enctype. In this model we group keys by kvno into KeySet objects. A Principal can have zero or more KeySet objects associated with it, each of which MUST have one or more keys. Each KeySet is associated with exactly one principal. Schemas derived from this model MAY lack a direct analogue of KeySet as described in this document.

It is expected that most Kerberos implementations will use a special-purpose interface for setting and changing Principal passwords and keys.

If a server supports an enctype for a Principal that enctype must be present in at least one key for the Principal in question. For any given enctype a KeySet MUST NOT contain more than one Key with that enctype.

The security of Kerberos 5 depends absolutely on the confidentiality and integrity of the Key objects stored in the KDC. Implementations of this standard MUST facilitate, to the extent possible, an administrator's ability to place more restrictive access controls on KeySets than on other Principal data, and to arrange for more secure backup for KeySets.

##### 4.2.1. KeySet: Attributes

###### 4.2.1.1. kvno

Also known as the key version number. This is a single-valued attribute containing a non-negative integer. This number is incremented by one each time a key in the KeySet is changed.

##### 4.2.2. KeySet: Associations

To each KeySet MUST be associated a set of 1 or more Keys.

#### 4.3. Key

Implementations of this model MUST NOT REQUIRE keys to be represented.

#### 4.3.1. Key: Attributes

##### 4.3.1.1. keyEncryptionType

The enctype SHOULD be represented as an enumeration of the encetypes supported by the KDC using the string name ("encryption type") of the enctype from the IANA registry of Kerberos Encryption Type Numbers. One example is 'aes128-cts-hmac-sha1-96'.

##### 4.3.1.2. keyValue

The binary representation of the key data. This MUST be a single-valued octet string.

##### 4.3.1.3. keySaltValue

The binary representation of the key salt. This MUST be a single-valued octet string.

##### 4.3.1.4. keyStringToKeyParameter

This MUST be a single-valued octet string representing an opaque parameter associated with the enctype. This parameter is specified in the "string-to-key" method in section 3 of [RFC3961].

##### 4.3.1.5. keyNotUsedBefore

This key MUST NOT be used before this date. The syntax of the attribute MUST be semantically equivalent with the standard ISO date format. This MUST be a single-valued attribute.

##### 4.3.1.6. keyNotUsedAfter

This key MUST NOT be used after this date. The syntax of the attribute MUST be semantically equivalent with the standard ISO date format. This MUST be a single-valued attribute.

##### 4.3.1.7. keyIsDisabled

This is a boolean attribute which SHOULD be set to false by default. If this attribute is true the key MUST NOT be used. This is used to temporarily disable a key.

#### 4.3.2. Key: Associations

None

#### 4.3.3. Key: Remarks

The security of the keys is an absolute requirement for the operation of Kerberos 5. If keys are implemented adequate protection from unauthorized modification and disclosure **MUST** be available and **REQUIRED** by the implementation.

#### 4.4. Policy

Implementations **SHOULD** implement policy but **MAY** allow them to be **OPTIONAL**. The Policy should be thought of as a 'typed hole'. i.e. an opaque binary value paired with an identifier of type of data contained in the binary value. Both attributes (type and value) must be present.

##### 4.4.1. Policy: Attributes

###### 4.4.1.1. policyIdentifier

The policyIdentifier **MUST** be globally unique. Possible types of identifiers include:

An Object Identifier (OID) [RFC4517]

A URI [RFC3986]

A UUID [RFC4122]

Implementations of this specification are expected to assign globally unique identifiers to the list of standard policy below in accordance with best-practice for identifier-management for the schema-language used.

###### 4.4.1.2. policyIsCritical

This boolean attribute indicates that the KDC **MUST** be able to correctly interpret and apply this policy for the Principal to be used.

###### 4.4.1.3. policyContent

This is an optional single opaque binary value used to store a representation of the policy. In general a policy cannot be fully expressed using attribute-value pairs. The policyContent is **OPTIONAL** in the sense that an implementation **MAY** use it to store an opaque value for those policy-types which are not directly representable in that implementation.

#### 4.4.1.4. policyUse

This is an optional single enumerated string value used to describe the use of the policy. Implementations SHOULD provide this attribute and MUST (if the attribute is implemented) describe the enumerated set of possible values. The intent is that this attribute be useful in providing an initial context-based filtering.

#### 4.4.2. Mandatory-to-implement Policy

All implementations that represent Policy objects MUST be able to represent the policies listed in this section. Implementations are not required to use the same underlying data-representation for the policyContent binary value but SHOULD use the same OIDs as the policyIdentifier. In general the expression of policy may require a Turing-complete language. This specification does not attempt to model policy expression language.

##### 4.4.2.1. Password Quality Policy

Password quality policy controls the requirements placed by the KDC on new passwords.

##### 4.4.2.2. Password Management Policy

Password management policy controls how passwords are changed.

##### 4.4.2.3. Keying Policy

A keying policy specifies the association of enctypes with new principals, e.g. when a Principal is created one of the applicable keying policies is used to determine the set of keys to associate with the principal.

##### 4.4.2.4. Ticket Flag Policy

A ticket flag policy specifies the ticket flags allowed for tickets issued for a principal.

## 5. Implementation Scenarios

There are several ways to implement an administrative service for Kerberos 5 based on this information model. In this section we list a few of them.

### 5.1. LDAP backend to KDC

Given an LDAP schema implementation of this information model it would be possible to build an administrative service by back-ending the KDC to a directory server where principals and keys are stored. Using the security mechanisms available on the directory server keys are protected from access by anyone apart from the KDC. Administration of the principals, policy, and other non-key data is done through the directory server while the keys are modified using the set/change password protocol [I-D.ietf-krb-wg-kerberos-set-passwd].

### 5.2. LDAP frontend to KDC

An alternative way to provide a directory interface to the KDC is to implement an LDAP-frontend to the KDC which exposes all non-key objects as entries and attributes. As in the example above all keys are modified using the set/change password protocol [I-D.ietf-krb-wg-kerberos-set-passwd]. In this scenario the implementation would typically not use a traditional LDAP implementation but treat LDAP as an access protocol to data in the native KDC database.

### 5.3. SOAP

Given an XML schema implementation of this information model it would be possible to build a SOAP interface to the KDC. This demonstrates the value of creating an abstract information model which is mappable to multiple schema representations.

### 5.4. Netconf

Given a YAML implementation of this information model it would be possible to create a Netconf-based interface to the KDC, enabling management of the KDC from standard network management applications.

## 6. Security Considerations

This document describes an abstract information model for Kerberos 5. The Kerberos 5 protocol depends on the security of the keys stored in the KDC. The model described here assumes that keys **MUST NOT** be transported in the clear over the network and furthermore that keys are treated as write-only attributes that **SHALL** only be modified (using the administrative interface) by the change-password protocol [I-D.ietf-krb-wg-kerberos-set-passwd].

Exposing the object model of a KDC typically implies that objects can be modified and/or deleted. In a KDC not all principals are created equal, so that for instance deleting `krbtgt/EXAMPLE.COM@EXAMPLE.COM` effectively disables the `EXAMPLE.COM` realm. Hence access control is paramount to the security of any implementation. This document does not mandate access control. This only implies that access control is beyond the scope of the standard information model, i.e. that access control may not be accessible via any protocol based on this model. If access control objects are exposed via an extension to this model the presence of access control may in itself provide points of attack by giving away information about principals with elevated rights etc.

## 7. IANA Considerations

This document has no IANA actions.

## 8. Acknowledgments

The author wishes to extend his thanks to Love Hoernquist-Aestrand and Sam Hartman for their important contributions to this document.



## 9. References

### 9.1. Normative References

- [RFC1964] Linn, J., "The Kerberos Version 5 GSS-API Mechanism", RFC 1964, June 1996.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.
- [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.
- [RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.
- [RFC4517] Legg, S., "Lightweight Directory Access Protocol (LDAP): Syntaxes and Matching Rules", RFC 4517, June 2006.

### 9.2. Informative References

- [I-D.ietf-krb-wg-kerberos-set-passwd]  
Williams, N., "Kerberos Set/Change Key/Password Protocol Version 2", draft-ietf-krb-wg-kerberos-set-passwd-08 (work in progress), November 2008.
- [RFC4510] Zeilenga, K., "Lightweight Directory Access Protocol (LDAP): Technical Specification Road Map", RFC 4510, June 2006.

Author's Address

Leif Johansson  
Swedish University Network  
Thulegatan 11  
Stockholm

Email: [leifj@sunset.se](mailto:leifj@sunset.se)  
URI: <http://www.sunet.se>



Network Working Group  
Internet-Draft  
Intended status: Informational  
Expires: June 17, 2013

N. Williams  
Cryptonector  
December 14, 2012

Simplified and Minimized Profiles of the Generic Security Services  
Application Programming Interface  
draft-williams-gss-profiles-00

## Abstract

The Generic Security Service Application Programming Interface (GSS-API) is often mistaken for a bloated framework. The GSS-API is really several things: a basis for formal descriptions of application authentication protocols (what happens when), a pattern for actual programming APIs, a set of constraints and requirements for generic security mechanisms, and concrete programming APIs. Only the first of these is relevant to Internet application protocols.

This document describes simplified and minimized profiles of the GSS-API for two purposes: explaining its use, whether in standards specifications or actual applications, and specifying sub-sets of the API for a) newcomers to the API, b) application developers that wish to use it but not have to link the whole thing into their applications.

## 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 June 17, 2013.

## Copyright Notice

Copyright (c) 2012 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.	A Brief Primer on the GSS-API . . . . .	3
1.2.	Conventions used in this document . . . . .	4
2.	Simple Profiles of the GSS-API . . . . .	5
2.1.	Trivial Initiator (client) Application . . . . .	5
2.2.	Simple Initiator Application . . . . .	6
2.3.	Trivial Acceptor (server) Application . . . . .	6
2.4.	Simple Acceptor Application . . . . .	7
3.	Transport Security Layer Profiles . . . . .	8
3.1.	Null Transport Security . . . . .	8
3.2.	TLS with Server Name Binding . . . . .	8
3.3.	TLS with Channel Binding . . . . .	8
3.4.	Per-Message Tokens . . . . .	9
3.5.	GSS-Keyed Application Transport Security Layer . . . . .	9
4.	Complex Profiles . . . . .	10
4.1.	Advanced Initiator Application . . . . .	10
4.2.	Advanced Acceptor Application . . . . .	10
5.	GSS-API Profile and Subset Identifiers . . . . .	11
5.1.	C-Bindings . . . . .	12
6.	Mechanism Profiles . . . . .	13
6.1.	Intranet Profile: SPNEGO, Kerberos . . . . .	13
6.2.	Internet Profile: SPNEGO, <TBD> . . . . .	13
7.	Minimized GSS-API Implementations . . . . .	14
8.	IANA Considerations . . . . .	15
9.	Security Considerations . . . . .	16
10.	Informative References . . . . .	17
	Author's Address . . . . .	18

## 1. Introduction

There has been some confusion in the Internet community as to the nature of the GSS-API. One problem that has been identified by some is that the GSS-API is quite large, that simpler subsets of it may be desirable. This document proposes some subsets and "profiles" of the GSS-API. We hope this will make the API more approachable, acceptable, and easier to understand.

The GSS-API is first and foremost an `_abstract_` API whose purpose is to describe with high precision how Internet application protocols handle user/service authentication and transport security.

The GSS-API is also a set of constraints on its security mechanisms.

The GSS-API is also a pattern for programming language bindings of the abstract GSS-API. In the IETF we have standardized C and Java bindings for the abstract API.

Note that there is `_absolutely no requirement_` that implementors use the C or Java, or any other bindings of the GSS-API to implement applications whose specifications use the `_abstract_` GSS-API. For example, one major implementor uses a different API (though in broad strokes quite similar to the GSS-API) to access generic security mechanisms, and applications using that API interoperate perfectly with applications that use bonafide GSS-API implementations.

By providing minimized profiles/subsets of the GSS-API we hope to make the GSS-API easier to use.

### 1.1. A Brief Primer on the GSS-API

In GSS terminology we have:

Principals    Entities to be authenticated;

Credentials    Credentials are used to authenticate principals;

Security context tokens    Messages exchanged by two principals to authenticate one to the other (and possibly the other to the one);

Security contexts    The shared state (e.g., session keys) that results from a successful exchange of security context tokens;

Initiator    The party (typically the client) that initiates a security context token exchange;

Acceptor The party (typically the server) that receives an initial security context token.

The abstract API has a few data types:

Buffers Octet strings;

OIDs Constants identifying GSS mechanisms and name types;

OID sets Sets of OIDs;

NAME Opaque object representing the name of a principal;

CREDENTIAL HANDLE Opaque object representing a credential for authenticating the entity that has access to it;

SECURITY CONTEXT HANDLE Opaque object representing a security context.

## 1.2. 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].

## 2. Simple Profiles of the GSS-API

We define profiles of the GSS-API for a variety of applications. These can be used both, for application security protocol specifications, and for implementation (but the latter is never required).

- o Trivial client ("initiator") applications;
- o Simple initiator applications;
- o Various security layer options;
- o Trivial server ("acceptor") applications;
- o Simple acceptor applications.

We also define C bindings for these profiles, as well as for specific additional subsets of GSS-API functionality. These will take the form of C pre-processor macros indicating what profiles and functionality subsets are available.

### 2.1. Trivial Initiator (client) Application

This profile requires just six GSS-API functions, three of which are destructors, and one of which is a function for formatting error strings:

GSS\_Import\_name() Produces a NAME object given a string and a name type;

GSS\_Init\_sec\_context() Drives the exchange of security context tokens (authentication messages) on the initiator side;

GSS\_Display\_status() Produces a human-readable representation of a status code (error code);

destructor functions:

- o GSS\_Release\_name()
- o GSS\_Release\_buffer()
- o GSS\_Delete\_sec\_context()

The trivial initiator application profile REQUIRES two name-types: GSS\_C\_NT\_HOSTBASED\_SERVICE and GSS\_C\_NO\_OID (denoting a mechanism-specific default name syntax).



A trivial initiator application begins by using `GSS_Import_name()` to get a `NAME` object for the target acceptor (server). This will be a host-based service name like `"HTTP@hostname"`. The application then loops over `GSS_Init_sec_context()` to drive the exchange security context tokens (authentication messages) with the target.

Most input arguments to `GSS_Init_sec_context()` will be defaulted.

If a trivial initiator application has multiple initiator credentials to choose from then the choice of credential will be left to the GSS-API framework or security mechanism. GSS-API implementations **SHOULD** provide some facility for configuring reasonable initiator identity selection.

## 2.2. Simple Initiator Application

This profile adds two functions to the trivial initiator application profile, one of which is a destructor:

`GSS_Acquire_cred()` Produces a `CREDENTIAL HANDLE` for a given desired `NAME`.

destructor function:

- o `GSS_Release_cred()`

The initiator application may have multiple credentials and want to choose a specific one to authenticate as to a target. Such an application does so by calling `GSS_Import_name()` for the desired name, then `GSS_Acquire_cred()` to acquire a `CREDENTIAL HANDLE` to pass to `GSS_Init_sec_context()` as the initiator credential.

## 2.3. Trivial Acceptor (server) Application

This profile requires just eight GSS-API functions, three of which are destructors, and one of which is a function for formatting error strings:

`GSS_Accept_sec_context()` Driver of security context token exchange;

`GSS_Display_name_ext()` Produces human-readable representation of a `NAME` object in the syntax of selected name type, if possible;

`GSS_Display_name()` Produces a human-readable representation of a `NAME` object in a mechanism-specific syntax;

GSS\_Export\_name() Produces a machine-readable representation of a NAME object in a mechanism-specific format;

GSS\_Display\_status() Produces a human-readable representation of a status code (error code);

destructor functions:

- o GSS\_Release\_name()
- o GSS\_Release\_buffer()
- o GSS\_Delete\_sec\_context()

This profile REQUIRES one name-type: GSS\_C\_NT\_USERNAME.

A trivial acceptor application begins by using GSS\_Accept\_sec\_context() to drive an exchange of security context tokens by which to authenticate an initiator. A successful authentication will yield a NAME representing the identity of the initiator - the acceptor will perform authorization based on the name of the initiator. The acceptor application MUST use one of GSS\_Display\_name\_ext(), GSS\_Display\_name(), or GSS\_Export\_name() to obtain a textual or binary representation of the initiator name suitable for authorization.

Most input arguments to GSS\_Accept\_sec\_context() will be defaulted.

If a trivial acceptor application has multiple acceptor credentials to choose from then the choice will be the initiator's.

## 2.4. Simple Acceptor Application

This profile adds three functions to the trivial initiator application profile:

GSS\_Inquire\_context() A simple acceptor application may be interested in knowing what name it was called by the initiator, and may do so by calling GSS\_Inquire\_context().

GSS\_Store\_cred() A simple acceptor application may have a use for delegated credentials. This function makes it possible to make delegated credentials available to other applications.

GSS\_Release\_cred() This is the destructor for the delegated CREDENTIAL HANDLE.

### 3. Transport Security Layer Profiles

Internet applications often require transport security - integrity protection or integrity and confidentiality protection. In simpler terms: some applications require encryption.

The GSS-API was originally designed to provide facilities that the application could use to protect its in-flight data. Since then a number of options have been added to make it easier to use the GSS-API in existing applications that already have transport security facilities. The GSS-API now supports applications that use TLS for transport security, applications that have their own cryptographic transport protection facilities (like AFS' RX transport), as well as applications that can just use the original GSS-API transport security facilities.

It is now possible to retrofit the GSS-API for authentication (and, where required, key exchange) into the widest possible range of applications.

#### 3.1. Null Transport Security

In some environments strong authentication of users and services is sufficient, with no need to protect in-flight data from passive nor active attacks. The trivial and simple profiles of the GSS-API suffice for these applications.

#### 3.2. TLS with Server Name Binding

This profile adds not functions to any of the trivial and simple acceptor application profiles.

Applications using this profile use TLS [RFC5246] for transport security and the GSS-API for authentication. This profile **REQUIRES** that applications use the 'tls-server-end-point' channel binding type [RFC5929] a TLS server certificate to authenticate the server to the client and that the same server name used to authenticate the server in TLS also be used to authenticate the server in the GSS-API.

#### 3.3. TLS with Channel Binding

This profile adds no functions to any of the initiator or acceptor application profiles. Instead this profile makes support for channel binding [RFC5056] **REQUIRED**.

Applications using this profile use TLS [RFC5246] for transport security, possibly using anonymous Diffie-Hellman (DH) cipher suites. Channel binding ensures that the end-points of the TLS connection are

logically the same as the GSS-API end-points, thus ensuring that there are no men in the middle (MITMs) even if anonymous DH TLS cipher suites are used.

### 3.4. Per-Message Tokens

This profile adds four functions to the simple initiator application profile:

GSS\_GetMIC() Get an authentication code for a message (something like a MAC);

GSS\_VerifyMIC() Verify an authentication code sent by the peer;

GSS\_Wrap() Wrap an application message, typically to provide confidentiality and integrity protection (i.e., encrypt with authentication);

GSS\_Unwrap() Unwrap a wrapped application message (i.e., decrypt with authentication).

An application of this type will rely on the GSS-API security mechanism to provide confidentiality and/or integrity protection to the application's messages. This kind of application typically does not use TLS or any other framework for transport security.

Applications may use these functions to implement either of octet streams and unsequenced datagram security layers.

### 3.5. GSS-Keyed Application Transport Security Layer

This profile adds one function to the simple initiator application profile:

GSS\_Pseudo\_random() Get a pseudo-random output octet string based on session keys exchanged by the GSS-API security mechanism and an input octet string provided by the application.

An application of this type typically has its own transport security layer (i.e., does its own encryption) and only requires a session key, which it can get from the GSS-API security context by using the GSS\_Pseudo\_random() [RFC4401] function to obtain an octet string of suitable length.

## 4. Complex Profiles

### 4.1. Advanced Initiator Application

A complex initiator application requires most of the functions defined in [RFC5587] in addition to those of the simple initiator application profile:

GSS\_Indicate\_mechs\_by\_attrs() Select mechanisms based on desired attributes;

GSS\_Inquire\_attrs\_for\_mech() Get the set of attributes of a given mechanism.

These functions are useful for mechanism negotiation.

### 4.2. Advanced Acceptor Application

A complex acceptor application typically requires fine-grained access to the initiator names so as to implement more advanced authorization schemes. This means all of the naming extensions APIs:

GSS\_Display\_name\_ext() Already covered in Section 2.3.

GSS\_Inquire\_name() Needed to list attributes available in the initiator name.

GSS\_Get\_name\_attribute() Needed to access individual attributes of the initiator name.

GSS\_Export\_name\_composite() Needed to be able to save a binary version of the initiator name with all its attributes.

Complex acceptor applications may also need at least one extended mechanism inquiry function [RFC5587]:

GSS\_Inquire\_attrs\_for\_mech() Get the set of attributes of a given mechanism.

## 5. GSS-API Profile and Subset Identifiers

We define identifiers for GSS-API profiles and subsets. These should be used as follows:

- o Internet application specifications using the GSS-API must list the profiles/subsets of the API required to implement the specification;
- o GSS-API implementations SHOULD use these identifiers to advertise which profiles/subsets of the API are available
- o GSS-API application implementations using the GSS-API in the implementation SHOULD use these identifiers to request profiles/subsets of the API to use.

The identifiers are:

GSS\_PR\_V2U1 This refers to the base GSS-API version 2 update 1.

GSS\_PR\_TRIVIAL\_INITIATOR See Section 2.1.

GSS\_PR\_SIMPLE\_INITIATOR See Section 2.2.

GSS\_PR\_TRIVIAL\_ACCEPTOR See Section 2.3.

GSS\_PR\_SIMPLE\_ACCEPTOR See Section 2.4.

GSS\_PR\_ADVANCED\_INITIATOR See Section 4.1.

GSS\_PR\_ADVANCED\_ACCEPTOR See Section 4.2.

GSS\_PR\_CB See Section 3.3.

GSS\_PR\_MSG\_TOKENS See Section 3.4.

GSS\_PR\_PRF See Section 3.5 [RFC4401].

GSS\_PR\_STORE\_CRED See Section 2.4 and [RFC5588]

GSS\_PR\_EXTENDED\_MECH\_INQUIRY Functions from [RFC5587]

GSS\_PR\_EXTENDED\_NAMING Functions from [RFC6680]

### 5.1. C-Bindings

C pre-processor macros MUST be defined by `<gssapi.h>` or `<gssapi/gssapi.h>` for each of the supported profiles and subsets of the GSS-API supported by the GSS-API implementation. The C pre-processor macro names MUST be as given in the preceding section, and the value should be 1 if the macro is defined.

## 6. Mechanism Profiles

A mechanism profile is a set of GSS-API mechanisms that MUST be provided.

### 6.1. Intranet Profile: SPNEGO, Kerberos

...

### 6.2. Internet Profile: SPNEGO, <TBD>

...



## 7. Minimized GSS-API Implementations

Typically a GSS-API implementation includes all of the basic version 2, update 1 API [RFC2743]. For embedded and mobile applications it may be desirable to have minimized GSS-API implementations, both for the framework itself and the mechanisms. A minimized implementation should provide at least the functionality required by the application and, preferably, no more than is required by the application. In practice this may mean that a complete implementation may have to provide tools for reducing the functionality provided.

One possibility may be to implement a system whereby all the sources for a C implementation are combined into a single file and then C pre-processing can be used to eliminate all functionality that is not desired, then the resulting object file is linked into the application. This is known by some as an "amalgamation", and is much like static linking, but can be significantly more selective. This approach is used by SQLite3, for example.

Another possibility is to structure the framework library and the mechanism plugins such that subsets are in distinct static linking archives or shared objects. This approach may result in a multiplicity of libraries and/or packages that reflect the profiles of the GSS-API.

For minimization we recommend either the amalgamation approach or a scheme whereby static linking archives are used. For general purpose use we recommend dynamic linking and no minimization.

## 8. IANA Considerations

We ask the IANA to create a registry of GSS-API profiles and subsets, with the identifiers list from of Section 5 as its contents.

## 9. Security Considerations

The security considerations of the GSS-API version 2, update 1, and its various extensions apply. Using too trivial a profile for an application can have security impact. For example, applications using the trivial initiator application profile (GSS\_PR\_TRIVIAL\_INITIATOR) cannot select an initiator identity and depend entirely on the framework and/or selected mechanism for this.

Internet applications that use GSS mechanisms should specify one of the transport security layers from Section 3 other than the null layer as required to implement. The simplest transport security layer is, of course, none at all (see Section 3.1), but this is not secure in the Internet threat model; all the other options listed in Section 3 offer good security in the Internet threat model. The next simplest may well be to use TLS with server name binding (see Section 3.2).

## 10. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.
- [RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.
- [RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.
- [RFC6680] Williams, N., Johansson, L., Hartman, S., and S. Josefsson, "Generic Security Service Application Programming Interface (GSS-API) Naming Extensions", RFC 6680, August 2012.
- [RFC5588] Williams, N., "Generic Security Service Application Program Interface (GSS-API) Extension for Storing Delegated Credentials", RFC 5588, July 2009.
- [RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.

Author's Address

Nicolas Williams  
Cryptonector, LLC

Email: [nico@cryptonector.com](mailto:nico@cryptonector.com)



Network Working Group  
Internet-Draft  
Updates: RFC2743 RFC2744  
(if approved)  
Intended status: Standards Track  
Expires: August 26, 2013

N. Williams  
Cryptonector  
February 22, 2013

Channel Binding Signalling for the Generic Security Services Application  
Programming Interface  
draft-williams-kitten-channel-bound-flag-02

Abstract

This Internet-Draft proposes the addition of a "channel bound" return flag for the GSS\_Init\_sec\_context() and GSS\_Accept\_sec\_context() functions. Two behaviors are specified: a default, safe behavior, and a behavior that is only safe when the application specifically tells the Generic Security Services Application Programming Interface (GSS-API) that it (the applicaiton) supports the new behavior.

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 August 26, 2013.

Copyright Notice

Copyright (c) 2013 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.	Error in RFC2743 . . . . .	3
1.2.	Design . . . . .	3
1.3.	Alternative Design . . . . .	4
1.4.	Future Directions . . . . .	4
1.5.	Conventions used in this document . . . . .	4
2.	Channel Binding State Extension . . . . .	5
2.1.	GSS_Create_sec_context() . . . . .	5
2.1.1.	C-Bindings . . . . .	5
2.2.	GSS_Set_context_flags() . . . . .	6
2.2.1.	C-Bindings . . . . .	6
2.3.	Return Flag for Channel Binding State Signalling . . . . .	7
2.3.1.	C-Bindings . . . . .	7
3.	Modified Channel Binding Semantics . . . . .	8
4.	Security Considerations . . . . .	9
5.	IANA Considerations . . . . .	10
6.	References . . . . .	11
6.1.	Normative References . . . . .	11
6.2.	Informative References . . . . .	11
	Author's Address . . . . .	12



## 1. Introduction

The GSS-API [RFC2743] supports "channel binding" [RFC5056], a technique for detection of man-in-the-middle (MITM) attacks in secure channels at lower network layers. This facility is meant to be all-or-nothing: either both the initiator and acceptor use it and it succeeds, or both must not use it. This has created a negotiation problem when retrofitting the use of channel binding into existing application protocols.

Many implementations of the Kerberos V5 GSS-API mechanism [RFC4121] cause the acceptor to succeed when the initiator used channel binding but the acceptor application did not. This has helped deployment of channel binding in existing applications: first fix all the initiators, then fix all the acceptors. But even this is insufficient when there are many clients to fix, such that fixing them all will take a long time.

This document proposes a new method for deployment of channel binding that allows the feature to be enabled on the acceptor side before fixing all initiators. If the GSS-API had always had a return flag by which to indicate channel binding state then we could have had a simpler method of deploying channel binding: applications check that return flag and act accordingly (e.g., fail when channel binding is required). We cannot safely introduce this behavior now without an indication of support by the application.

It is worth noting that at least one implementor of GSS-API mechanisms (but not of the GSS-API itself) has similar semantics in its API to those proposed herein. [XXX add references to the relevant SSPI docs? -Nico]

### 1.1. Error in RFC2743

The GSS-APIv2ul [RFC2743] seems to indicate that mechanisms must ignore channel bindings when one party provided none. In practice some mechanisms ignore channel bindings when the acceptor provides none, but not when the initiator provides none. Note that it would be useless to allow security context establishment to succeed when the initiator does not provide channel bindings but the acceptor does, at least as long as there's no outward indication of whether channel binding was used! And indeed, the GSS-APIv2ul does not provide any such indication. We correct this flaw in this document.

### 1.2. Design

After some discussion on the mailing list of various designs for signalling application support for the new flag we've settled on

copying an aspect of the Java Bindings of the GSS-API [RFC5653], specifically the notion of creating an "empty" SECURITY CONTEXT handle that can then be passed to GSS\_Init\_sec\_context() and GSS\_Accept\_sec\_context() where they normally expect a NULL handle. This empty security context handle can then be used to set options relating to security context token establishment.

### 1.3. Alternative Design

The previous design was based on an existing, non-standard extension for carrying security context establishment options in CREDENTIAL HANDLES. Note that a notion of CREDENTIAL HANDLE options might still be useful for options that are really specific to credentials rather than security context tokens (for example: setting an acceptable cryptographic security profile on a CREDENTIAL HANDLE and receiving a new handle with possibly fewer elements, reflecting that some credentials cannot meet the requirement).

### 1.4. Future Directions

We're likely to introduce additional mutator functions of empty contexts, with mutators corresponding to many of the existing input arguments of GSS\_Init\_sec\_context() and GSS\_Accept\_sec\_context(), as well as a few additional security context inquiry functions. We're also likely to then introduce new variants of GSS\_Init\_sec\_context() and GSS\_Accept\_sec\_context() with all of those input and output parameters removed that could be set or retrieved with the other new functions. The only inputs that the new GSS\_Init/Accept\_sec\_context() must have are: a security context handle (never NULL), and an input context token, and the only outputs should be the status indicators and an output token -- in fact, we may want to have just one new function called, perhaps, GSS\_Step\_sec\_context(), with the role of initiator or acceptor set as a context option.

### 1.5. 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].

## 2. Channel Binding State Extension

We propose a new return flag for `GSS_Init_sec_context()` and `GSS_Accept_sec_context()`, as well as a pair of functions for setting a) creating "empty" security context handles, b) setting `req_flags` and indicating which `ret_flags` the application understands.

C bindings of these extensions are provided along the lines of [RFC2744] and [RFC5587].

In the future we might move more of the many input (and output) arguments to `GSS_Init_sec_context()` and `GSS_Accept_sec_context()` into mutators on empty security context handles.

### 2.1. `GSS_Create_sec_context()`

Inputs:

- o <none>

Outputs:

- o `major_status` INTEGER
- o `minor_status` INTEGER -- note: mostly useless, but we should keep it
- o `context` SECURITY CONTEXT

Return major status codes:

- o `GSS_S_COMPLETE` indicates success.
- o `GSS_S_UNAVAILABLE` indicates that memory is not available, for example.
- o `GSS_S_FAILURE` indicates a general failure.

This function creates an "empty" security context handle that can be passed to `GSS_Init_sec_context()` or `GSS_Accept_sec_context()` where they expect a NULL context.

#### 2.1.1. C-Bindings

```
OM_uint32
gss_create_sec_context(OM_uint32 *minor_status,
                      gss_ctx_id_t *context);
```

Figure 1: C-Bindings of GSS\_Create\_sec\_context()

## 2.2. GSS\_Set\_context\_flags()

### Inputs:

context CONTEXT HANDLE

req\_flags FLAGS Requested flags. Applicable to acceptors and initiators.

ret\_flags\_understood FLAGS Return flags understood by the caller.

### Outputs:

- o major\_status INTEGER

- o minor\_status INTEGER

### Return major status codes:

- o GSS\_S\_COMPLETE indicates success.

- o GSS\_S\_FAILURE indicates a general failure.

This function tells the mechanism (when one is eventually chosen and invoked) that the application requests the given req\_flags and understands the given ret\_flags. Initiators can override the req\_flags in their GSS\_Init\_sec\_context() call, but if no flags are requested there then the req\_flags set on the empty context will be used.

NOTE: The abstract GSS-API [RFC2743] uses individual elements -one per-flag- instead of a "FLAGS" type. This is unwieldy, therefore we introduce an abstract type named "FLAGS" to act as a set of all the request/return flags defined for the abstract GSS-API.

### 2.2.1. C-Bindings

```
OM_uint32
gss_set_context_flags(OM_uint32 *minor_status,
                      gss_ctx_id_t context,
                      uint64_t req_flags,
                      uint64_t ret_flags);
```

Figure 2: C-Bindings of GSS\_Set\_context\_flags()

### 2.3. Return Flag for Channel Binding State Signalling

Whenever both the initiator and the acceptor provide matching channel bindings to GSS\_Init\_sec\_context() and GSS\_Accept\_sec\_context(), respectively, then the mechanism SHALL indicate that the context is channel bound via an output flag, ret\_channel\_bound\_flag, for the established context.

#### 2.3.1. C-Bindings

```
#define GSS_C_CHANNEL_BOUND_FLAG 2048 /* 0x00000800 */
```

Figure 3: C-Bindings of channel\_bound\_flag

### 3. Modified Channel Binding Semantics

The channel binding semantics of the base GSS-API are modified as follows:

- o Whenever both, the initiator and acceptor shall have provided `input_channel_bindings` to `GSS_Init/Accept_sec_context()` and the channel bindings do not match, then the mechanism **MUST** fail to establish a security context token. This is a restatement of an existing requirement in the base specification, restated for convenience.
- o Whenever the acceptor application shall have a) provided channel bindings to `GSS_Accept_sec_context()`, and b) not indicated support for the `ret_channel_bound_flag` flag, then the mechanism **MUST** fail to establish a security context if the initiator did not provide channel bindings data. This requirement is for security purposes, to make applications predating this document secure, and this requirement reflects actual implementations as deployed.
- o Whenever the initiator application shall have a) provided channel bindings to `GSS_Init_sec_context()`, and b) not indicated support for the `ret_channel_bound_flag` flag, then the mechanism **SHOULD NOT** fail to establish a security context just because the acceptor failed to provide channel bindings data. This recommendation is for interoperability purposes, and reflects actual implementations that have been deployed. It is possible that not all security mechanism protocols can implement this requirement easily.
- o Whenever the application shall have a) provided channel bindings to `GSS_Init_sec_context()` or `GSS_Accept_sec_context()`, and b) indicated support for the `ret_channel_bound_flag` flag, then the mechanism **MUST NOT** fail to establish a security context just because the peer did not provide channel bindings data. The mechanism **MUST** output the `ret_channel_bound_flag` if both peers provided the same `input_channel_bindings` to `GSS_Init_sec_context()` and `GSS_Accept_sec_context`. The mechanism **MUST NOT** output the `ret_channel_bound_flag` if either (or both) peer did not provide `input_channel_bindings` to `GSS_Init/Accept_sec_context()`. This requirement restores the original base GSS-API specified behavior, with the addition of the `ret_channel_bound_flag` flag

#### 4. Security Considerations

This document deals with security. There are no security considerations that should be documented separately in this section. To recap, this document fixes a significant flaw in the base GSS-API [RFC2743] specification that fortunately has not been implemented, and it adds a feature (that should have been in the base specification) for improved negotiation of use of channel binding [RFC5056].

## 5. IANA Considerations

This document has no IANA considerations.



## 6. References

### 6.1. Normative References

- [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.
- [RFC2744] Wray, J., "Generic Security Service API Version 2 : C-bindings", RFC 2744, January 2000.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5587] Williams, N., "Extended Generic Security Service Mechanism Inquiry APIs", RFC 5587, July 2009.

### 6.2. Informative References

- [RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.
- [RFC5653] Upadhyay, M. and S. Malkani, "Generic Security Service API Version 2: Java Bindings Update", RFC 5653, August 2009.

Author's Address

Nicolas Williams  
Cryptonector, LLC

Email: nico@cryptonector.com



Network Working Group  
Internet-Draft  
Updates: 4121 (if approved)  
Intended status: Standards Track  
Expires: May 25, 2015

N. Williams  
Cryptonector  
R. Dowdeswell  
Dowdeswell Security Architecture  
November 21, 2014

Negotiation of Extra Security Context Tokens for Kerberos V5 Generic  
Security Services Mechanism  
draft-williams-kitten-krb5-extra-rt-04

Abstract

This Internet-Draft proposes an extension to the Kerberos V5 security mechanism for the Generic Security Services Application Programming Interface (GSS-API) for using extra security context tokens in order to recover from certain errors. Other benefits include: user-to-user authentication, authenticated errors, replay cache avoidance, and others.

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 25, 2015.

Copyright Notice

Copyright (c) 2014 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.	Conventions used in this document . . . . .	3
2.	New Protocol Elements . . . . .	4
2.1.	Fields of KRB-ERROR2 . . . . .	4
2.2.	Distinction between KRB-ERROR2 and AP-REP2 PDUs . . . . .	5
3.	Negotiation and Use of Extra Context Tokens . . . . .	7
3.1.	Number of Security Context Tokens . . . . .	8
3.2.	Possible Context Token Sequences . . . . .	9
3.3.	Per-Message Token Sequence Numbers . . . . .	10
3.4.	Early PROT_READY State . . . . .	10
3.5.	Other Requirements, Recommendations, and Non-Requirements . . . . .	12
4.	ASN.1 Module for New Protocol Elements . . . . .	13
5.	Recoverable Errors and Error Recovery . . . . .	15
5.1.	Authenticated Errors . . . . .	16
6.	Replay Cache Avoidance . . . . .	17
6.1.	Replay Cache Avoidance without Extensions . . . . .	17
7.	User-to-User Authentication . . . . .	18
8.	Acceptor Clock Skew Correction . . . . .	19
9.	Security Considerations . . . . .	20
10.	IANA Considerations . . . . .	21
11.	References . . . . .	22
11.1.	Normative References . . . . .	22
11.2.	Informative References . . . . .	22
	Authors' Addresses . . . . .	23

## 1. Introduction

The Kerberos V5 [RFC4120] AP protocol, and therefore the Kerberos V5 GSS-API [RFC2743] mechanism [RFC4121] security context token exchange, is a one-round trip protocol. Occasionally there are errors that the protocol could recover from by using an additional round trip, but until now there was no way to execute such an additional round trip. For many application protocols the failure of the Kerberos AP protocol is fatal, requiring closing TCP connections and starting over; often there is no automatic recovery.

This document proposes a negotiation of additional security context tokens for automatic recovery from certain errors. This is done in a backwards-compatible way, thus retaining the existing mechanism OID for the Kerberos V5 GSS mechanism. This also enables other new features.

New features enabled by this extension include:

- o error recovery (see Section 5)
- o user-to-user authentication (see Section 7)
- o some authenticated errors (see Section 5.1)
- o replay cache avoidance (see Section 6)
- o acceptor clock skew correction (see Section 8)
- o symmetric authorization data flows

No new interfaces are needed for GSS-API applications to use the features added in this document.

### 1.1. 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].

## 2. New Protocol Elements

We introduce the following new protocol elements. A partial ASN.1 [CCITT.X680.2002] module (for inclusion in the base Kerberos ASN.1 module) is given in Section 4, and references to its contents are made below.

- o a new ap-options flag for use in the clear-text part of AP-REQs to indicate the desire for an extra round trip if need be;
- o a new authorization data (AD) element for integrity protection of ap-options;
- o a new AD element for use in Authenticators for quoting back a challenge from the acceptor;
- o a new PDU: KRB-ERROR2, also known as AP-REP2, with additional fields and support for integrity- (and confidentiality-)protected errors and optional `_key confirmation_` :
  - \* a flag is used to indicate which key is used to encrypt the KRB-ERROR2's private part, as in some cases there can be two keys to choose from;
  - \* when no key available for encrypting the private part of a KRB-ERROR2, the null enctype is used.

These elements are used to construct security context token exchanges with potentially more than two context tokens.

All context tokens are to be prefixed with the InitialContextToken pseudo-ASN.1/DER header from RFC2743, section 3.1, just as RFCs 1964 and 4121 require of the first two context tokens.

### 2.1. Fields of KRB-ERROR2

The new KRB-ERROR2 PDU is defined in Section 4. The fields of the KRB-ERROR2 encrypted part have the following purpose/semantics:

`continue-challenge` A challenge to be quoted back in any subsequent context tokens.

`stime` The acceptor's current time.

`susec` Microsecond portion of the acceptor's current time.

**subkey** The acceptor's sub-session key. This MUST be absent when the KRB-ERROR2 enc-part is "encrypted" in the null enctype and key or when the acceptor failed to decrypt the initiator's Authenticator (but, obviously, succeeded at decrypting the Ticket); otherwise it MUST be present.

**seq-number** The acceptor's initial per-message token sequence number. This MUST be absent when the subkey is absent; otherwise it MUST be present.

**error-code** When zero-valued, the KRB-ERROR2 is not an error token, but a key-confirmation that requires continuation with an additional AP-REQ.

**e-flags** Indicates whether the KRB-ERROR2 is final (error token) or not.

**e-text** A human-readable string (in any language and script) description of the error, if any.

**e-data** Currently unused but specified for extensibility reasons. SHOULD be absent and MUST be ignored.

**e-typed-data** TYPED-DATA; see [RFC4120]. Currently unused but specified for extensibility reasons. SHOULD be absent and MUST be ignored.

**your-addresses** The initiator's network address(es) as seen on the acceptor side. Currently unused due to insufficient GSS-API interfaces, but specified for extensibility reasons. SHOULD be absent, MUST be ignored.

**ad-data** Authorization-data. This is intended for symmetry, so that acceptors can assert authorization data to the initiator just as the initiator can assert authorization data to the acceptor. (For example, this might be useful in user-to-user authentication.) When present this has the same semantics as in the AP-REQ's Authenticator, but in the opposite direction.

**tgt** A TGT for use in user-to-user authentication.

## 2.2. Distinction between KRB-ERROR2 and AP-REP2 PDUs

The ASN.1 does not distinguish between KRB-ERROR2 and AP-REP2 PDUs. A KRB-ERROR2 can serve either or both, the purpose of conveying error information, as well as the purpose of completing the acceptor's side of the context token exchange and providing key confirmation. We could have used three distinct PDUs instead of one.



It is true that a KRB-ERROR2 that only serves the purpose of final key confirmation without continuation could have a different ASN.1 type for its encrypted part, and a different application tag, however, there seems to be little value in this. Distinguishing between errors with and without key confirmation is even less valuable. Therefore we do not distinguish these three possible PDUs.

### 3. Negotiation and Use of Extra Context Tokens

In the following text "initiator" refers to the mechanism's initiator functionality (invoked via `GSS_Init_sec_context()`), and "acceptor" refers to the mechanism's acceptor functionality (invoked via `GSS_Accept_sec_context()`).

To use this feature, the Kerberos GSS mechanism MUST act as follows:

- o To request this feature, initiators SHALL add the new ap-options flag to their AP-REQs.
  - \* And the initiators SHALL repeat the ap-options in the new AD-AP-OPTIONS AD type in the Authenticator.
- o Acceptors that wish to request an additional security context token can only do so when initiators indicate support for it, and MUST do so by returning a KRB-ERROR2. The encrypted part of the KRB-ERROR2 SHALL be encrypted in a key derived (with key usage <TBD>) from one of the following keys: the sub-session key from the AP-REQ's Authenticator (use-initiator-subkey) if it could be decrypted, else the session key from the Ticket (use-ticket-session-key), if it could be decrypted, else the null enc-type/key (use-null-etype).
- o Any KRB-ERROR2 emitted by the acceptor SHALL have the continue-needed e-flag set when the `GSS_Accept_sec_context()` returns `GSS_S_CONTINUE_NEEDED` to the application, and in this case the token ID SHALL be 02 00 (KRB\_AP\_REP, even though the token isn't actually an AP-REP) (see [RFC4121] section 4.1).
- o When it consumes a KRB-ERROR2, `GSS_Init_sec_context()` can return an error (`GSS_S_FAILURE`) and optionally output an error token, or it can attempt recovery (see Section 5) and output a new AP-REQ security context token.
  - \* Any error token output by `GSS_Init_sec_context()` MUST be a KRB-ERROR2, and `GSS_Init_sec_context()` MUST return `GSS_S_FAILURE`.
  - \* The initiator MUST quote the challenge from the KRB-ERROR2 using an AD-CONTINUE-CHALLENGE (see below) authorization data element in any AP-REQ or KRB-ERROR2 response to the acceptor's KRB-ERROR2.
  - \* When `GSS_Init_sec_context()` outputs a new AP-REQ security context token, it SHALL return `GSS_S_CONTINUE_NEEDED` if the application requested mutual authentication and the previous acceptor security context token was a recoverable error (rather

than a request for one more AP-REQ), else it SHALL return GSS\_S\_COMPLETE.

- \* When GSS\_Init\_sec\_context() returns an error and the acceptor is awaiting a security context token, GSS\_Init\_sec\_context() MAY generate a KRB-ERROR2 or KRB-ERROR to send to the acceptor.
- o Acceptors MUST reject additional AP-REQs which do not have a challenge response nonce matching the one sent by the acceptor in the previous KRB-ERROR2.
- o Acceptors MUST reject initial security context tokens that contain a challenge response nonce.
- o When GSS\_Accept\_sec\_context() returns an error and outputs an error token, the token MUST be either a KRB-ERROR or a KRB-ERROR2, with the latter having the continue-needed flag cleared.

All non-recoverable KRB-ERROR2 tokens SHALL use the token ID 03 00.

Additional AP-REQs produced by the authenticator MUST have the mutual-required ap-options flag set when a) the application requested mutual authentication, and b) the acceptor's KRB-ERROR2 did not supply the required key confirmation. The acceptor MUST respond to the client's last AP-REQ with an AP-REP when the mutual-required ap-options flag is set or when the GSS\_C\_MUTUAL\_FLAG is set in the "checksum 0x8003", otherwise GSS\_Accept\_sec\_context() MUST NOT produce a response token when it returns GSS\_S\_COMPLETE.

### 3.1. Number of Security Context Tokens

The first AP-REQ may well result in an error; the second generally should not. Therefore acceptors SHOULD return a fatal error when a second error results in one security context establishment attempt, except when the first error is that the initiator should use user-to-user authentication. This limits the maximum number of round trips to two (not user-to-user) or three (user-to-user).

The mechanism SHOULD impose some limit on the maximum number of security context tokens. For the time being that limit is six.

Note that in the user-to-user cases (see Section 7) it's possible to have up to three round trips under normal conditions if, for example, the acceptor wishes to avoid the use of replay caches (see Section 6), or if the initiator's clock is too skewed, for example.

### 3.2. Possible Context Token Sequences

The following successful security context token exchange sequences are possible:

- o One token (per-RFC4121; mutual authentication not requested): AP-REQ.
  - \* In principle this can yield an error token in the case of errors, per-RFC2743.
- o Two tokens (per-RFC4121; mutual authentication requested): AP-REQ and AP-REP.
- o Two tokens (per-RFC4121; mutual authentication requested): AP-REQ and KRB-ERROR.
- o Two tokens (per-RFC4121; mutual authentication requested): AP-REQ and KRB-ERROR2 (non-recoverable error, or recoverable error but the acceptor mechanism is configured to not continue).
- o Two tokens (per-RFC4121; mutual authentication requested): AP-REQ and KRB-ERROR2 (recoverable error for the acceptor, but not for the initiator, or the initiator application abandons the partially-established security context).
- o Three tokens: AP-REQ, KRB-ERROR2 (recoverable error), AP-REQ.
  - \* The initiator indicates it supports multiple round trips, and a recoverable error results on the acceptor side.
  - \* Either the initiator did not request mutual authentication, or the KRB-ERROR2 supplied the necessary key confirmation.
- o Three tokens: AP-REQ, KRB-ERROR2 (no error, continue needed), AP-REQ.
  - \* The initiator indicates it supports multiple round trips, and its Authenticator and Ticket decrypt correctly on the acceptor side, but the acceptor wants to continue, e.g., to avoid the need for a replay cache (see Section 6).
  - \* This can happen in any recoverable error case where the initiator's Authenticator (and Ticket) decrypt successfully on the acceptor side.

- o Four tokens: AP-REQ, KRB-ERROR2 (recoverable error), AP-REQ, AP-REP.
  - \* The initiator wanted mutual authentication and a recoverable error occurred where the KRB-ERROR2 could not provide key confirmation, leading to the second round trip.
  - \* This can happen in any recoverable error case where the initiator's Authenticator did not decrypt successfully.
  - \* This can also happen in the user-to-user case.
  - \* This case provides replay cache avoidance without a fifth token because the acceptor provides a challenge in its first (KRB-ERROR2) token and the initiator completes the challenges in its second token.
- o Five tokens: AP-REQ, KRB-ERROR2 (with user-to-user TGT), AP-REQ, KRB-ERROR2 (recoverable error), AP-REQ.
  - \* The initiator does not want mutual authentication, the acceptor wants user-to-user authentication, and the initiator's second AP-REQ elicits a recoverable error.
- o Six tokens: AP-REQ, KRB-ERROR2 (with user-to-user TGT), AP-REQ, KRB-ERROR2 (recoverable error), AP-REQ, AP-REP.
  - \* The initiator wants mutual authentication, the acceptor wants user-to-user authentication, and the initiator's second AP-REQ elicits a recoverable error; none of the KRB-ERROR2 tokens was a key-confirmation token.

Other context token sequences might be possible in the future.

In the above sequences the AP-REP tokens can be AP-REP2 tokens as well.

### 3.3. Per-Message Token Sequence Numbers

It is REQUIRED that each real AP-REQ in a single security token exchange specify the same start sequence number as preceding AP-REQs in the same security context token exchange.

### 3.4. Early PROT\_READY State

The GSS-API allows security mechanisms to support the use of per-message tokens prior to full security context establishment. In this section we'll call this "early PROT\_READY". Early PROT\_READY is

optional for the GSS-API and for implementations of mechanisms that support it.

The Kerberos V GSS mechanism supports this in the two-token exchange, with the initiator being `PROT_READY` before consuming the AP-REP. This extension also supports early `PROT_READY`, which works as follows:

1. The initiator asserts a sub-session key in each AP-REQ that does not follow a key-confirmation `KRB-ERROR2`, and `GSS_Init_sec_context()` sets the `prot_ready_state` return flag on the first call.
  1. If there are multiple such AP-REQs in a security context token exchange, then each such AP-REQ must assert the same sub-session key.
  2. Subsequent AP-REQs need not carry a sub-session key; acceptors **MUST** ignore sub-session keys from subsequent AP-REQs.
2. `GSS_Accept_sec_context()` **MUST NOT** set the `prot_ready_state` return flag until it has successfully decrypted an AP-REQ's Ticket and Authenticator from the initiator. If the acceptor requests additional context tokens and signals `PROT_READY` at that point, then it too will be `PROT_READY`.

Replay protection for early `prot_ready` per-message tokens depends on the initiator always generating a fresh sub-session key for every security context's initial context token, on the acceptor always generating a fresh sub-session key for its key confirmation token, and on either a replay cache or the challenge/response token provided for in this document:

- o An attacker cannot replay an early per-message token without also replaying the corresponding initial security context token (as otherwise the initiator-asserted sub-session keys won't match), and replay protection for the initial security context token provides replay protection for any subsequent early per-message tokens.
- o Per-message tokens made after full security context establishment are protected against replay by the use of the acceptor's sub-session key hierarchy (since the initiator must then use that key).
- o AP-REPs and key-confirmation `KRB-ERROR2`s are protected against replays to initiators by the use of the initiator's sub-session

key.

- o Initial security context tokens (and error-recovery AP-REQs) are protected against replay either by a replay cache on the acceptor side, or by the use of additional context tokens for challenge/response replay cache avoidance (see Section 6).

### 3.5. Other Requirements, Recommendations, and Non-Requirements

All error PDUs in an AP exchange where the AP-REQ has the continue-needed-ok ap-options flag MUST be KRB-ERROR2 PDUs.

Whenever an acceptor is able to decrypt the Ticket from an AP-REQ and yet wishes or has to output a KRB-ERROR2, then the enc-part of the KRB-ERROR2 MUST be encrypted in either the initiator's sub-session key (from the Authenticator) or the Ticket's session key (if the acceptor could not decrypt the Authenticator).

## 4. ASN.1 Module for New Protocol Elements

A partial ASN.1 module appears below. This ASN.1 is to be used as if it were part of the base Kerberos ASN.1 module (see RFC4120), therefore the encoding rules to be used are the Distinguished Encoding Rules (DER) [CCITT.X690.2002], and the environment is one of explicit tagging.

```
KerberosExtraContextTokens DEFINITIONS ::=
BEGIN
EXPORTS ad-continue-challenge,
        AD-CONTINUE-CHALLENGE,
        KrbErrorEncPartFlags,
        KRB-ERROR2,
        ErrorFlags;
IMPORTS UInt32, Int32, KerberosTime,
        Microseconds, KerberosFlags,
        Checksum, EncryptedData,
        EncryptionKey, KerberosString,
        AuthorizationData, TYPED-DATA,
        HostAddresses, Ticket FROM KERBEROS5;

APOptions          ::= KerberosFlags
    -- reserved(0),
    -- use-session-key(1),
    -- mutual-required(2),
    -- continue-needed-ok(TBD)

-- Challenge (for use in Authenticator)
ad-continue-challenge      Int32 ::= -5 -- <TBD>
AD-CONTINUE-CHALLENGE ::= OCTET STRING

-- AP options, integrity-protected
ad-ap-options              Int32 ::= -6 -- <TBD>
AD-AP-OPTIONS              ::= KerberosFlags

KrbErrorEncPartFlags ::= ENUMERATED {
    use-null-entype(0),
    use-initiator-subkey(1),
    use-ticket-session-key(2),
    ...
}

-- Application tag TBD
KRB-ERROR2                ::= [APPLICATION 55] SEQUENCE {
    pvno                    [0] INTEGER (5),
    msg-type                [1] INTEGER (55), -- TBD
    enc-part-key            [2] KrbErrorEncPartFlags,
```



```

        enc-part          [3] EncryptedData -- EncKRBErrorPart
    }

    -- Alias type name
    AP-REP2                ::= KRB-ERROR2

    ErrorFlags ::= ENUMERATED {
        final(0),
        continue-needed(1),
        ...
    }

    -- Application tag TBD
    EncKRBErrorPart ::= [APPLICATION 56] SEQUENCE {
        continue-challenge [0] AD-CHALLENGE-RESPONSE,
        stime               [1] KerberosTime,
        susec               [2] Microseconds,
        subkey              [3] EncryptionKey OPTIONAL,
        seq-number          [4] UInt32 OPTIONAL,
        error-code          [5] Int32,
        e-flags             [6] ErrorFlags,
        e-text              [7] UTF8String OPTIONAL,
        e-data              [8] OCTET STRING OPTIONAL,
        e-typed-data        [9] TYPED-DATA OPTIONAL,
        -- For recovery from KRB_AP_ERR_BADADDR:
        your-addresses      [10] HostAddresses OPTIONAL,
        ad-data             [11] AuthorizationData OPTIONAL,
        tgt                 [12] Ticket OPTIONAL, -- for user2user
        ...
    }

END
```

Figure 1: ASN.1 module (with explicit tagging)

## 5. Recoverable Errors and Error Recovery

The following Kerberos errors can be recovered from automatically using this protocol:

- o KRB\_AP\_ERR\_TKT\_EXPIRED: the initiator should get a new service ticket;
- o KRB\_AP\_ERR\_TKT\_NYV: the initiator should get a new service ticket;
- o KRB\_AP\_ERR\_REPEAT: the initiator should build a new AP-REQ;
- o KRB\_AP\_ERR\_SKEW: see Section 8;
- o KRB\_AP\_ERR\_BADKEYVER: the initiator should get a new service ticket;
- o KRB\_AP\_PATH\_NOT\_ACCEPTED: the initiator should get a new service ticket using a different transit path;
- o KRB\_AP\_ERR\_INAPP\_CKSUM: the initiator should try again with a different checksum type.

Error codes that denote PDU corruption (and/or an active attack) can also be recovered from by attempting a new AP-REQ, though subsequent AP-REQs may fail for the same reason:

- o KRB\_AP\_ERR\_BAD\_INTEGRITY
- o KRB\_AP\_ERR\_BADVERSION
- o KRB\_AP\_ERR\_BADMATCH
- o KRB\_AP\_ERR\_MSG\_TYPE
- o KRB\_AP\_ERR\_MODIFIED

Other error codes that may be recovered from:

- o KRB\_AP\_ERR\_BADADDR: the acceptor SHOULD include a list of one or more client network addresses as reported by the operating system, but if the acceptor does not then the continue-needed e-flag MUST NOT be included and the error must be final.

### 5.1. Authenticated Errors

The following errors, at least, can be authenticated in AP exchanges:

- o KRB\_AP\_ERR\_TKT\_EXPIRED
- o KRB\_AP\_ERR\_TKT\_NYV
- o KRB\_AP\_ERR\_REPEAT
- o KRB\_AP\_ERR\_SKEW
- o KRB\_AP\_PATH\_NOT\_ACCEPTED
- o KRB\_AP\_ERR\_INAPP\_CKSUM
- o KRB\_AP\_ERR\_BADADDR

## 6. Replay Cache Avoidance

By using an additional AP-REQ and a challenge/response nonce, this protocol is immune to replays of AP-REQ PDUs and does not need a replay cache. Acceptor implementations **MUST** not insert Authenticators from extra round trips into a replay cache when there are no other old implementations on the same host (and with access to the same acceptor credentials) that ignore critical authorization data or which don't know to reject initial AP-REQs that contain a challenge response nonce.

In the replay cache avoidance case where there's no actual error (e.g., time skew) the acceptor's KRB-ERROR2 will have KDC\_ERR\_NONE as the error code, with the continue-needed e-flag.

### 6.1. Replay Cache Avoidance without Extensions

Many Kerberos services can avoid the use of a replay cache altogether, but it's tricky to know when it's safe to do so. For Kerberos it's safe to not use a replay cache for AP-REQs/ Authenticators when either:

- o the application doesn't require replay detection at all and
  - \* no other acceptor/service application shares the same long-term service keys for its service principal

or

- o the application protocol always has the initiator/client send the first per-message token (or KRB-SAFE/PRIV PDU) which can then function as a challenge response, and
  - \* no other acceptor/service application shares the same long-term service keys for its service principal

It is difficult to establish the second part of the above conjunctions programmatically. In practice this is best left as a local configuration matted on a per-service name basis.

For example, it's generally safe for NFSv4 [RFC3530] to not use a replay cache for the Kerberos GSS mechanism, but it is possible for multiple Kerberos host-based service principals on the same host to share the same keys, therefore in practice, the analysis for NFSv4 requires more analysis. The same is true for SSHv2 [RFC4251] (SSHv2 implementations share the same service principal as other non-GSS Kerberos applications that do sometimes need a replay cache).

## 7. User-to-User Authentication

There are two user2user authentication cases:

1. the KDC only allows a service principal to use user2user authentication,
2. the service principal does not know its long-term keys or otherwise wants to use user2user authentication even though the KDC vended a service ticket.

In the first case the initiator knows this because the KDC returns `KDC_ERR_MUST_USE_USER2USER`. The initiator cannot make a valid `AP-REQ` in this case, yet it must send some sort of initial security context token! For this case we propose that the initiator make an `AP-REQ` with a Ticket with zero-length enc-part (and null enctype) and a zero-length authenticator (and null enctype). The acceptor will fail to process the `AP-REQ`, of course, and SHOULD respond with a continue-needed `KRB-ERROR2` (using the null enc-type for the enc-part) that includes a TGT for the acceptor.

In the second case the initiator does manage to get a real service ticket for the acceptor but the acceptor nonetheless wishes to use user2user authentication.

In both cases the acceptor responds with a `KRB-ERROR2` with the `KRB_AP_ERR_USER_TO_USER_REQUIRED` error code and including a TGT for itself.

In both cases the initiator then does a TGS request with a second ticket to get a new, user2user Ticket. Then the initiator makes a new `AP-REQ` using the new Ticket, and proceeds.

## 8. Acceptor Clock Skew Correction

An initiator in possession of a (short-lived) valid service ticket for a given service principal... must have had little clock skew relative to the service principal's realm's KDC(s), or the initiator must have been able to correct its local clock skew. But the acceptor's clock might be skewed, yielding a KRB\_AP\_ERR\_SKEW error with a challenge. The client could recover from this by requesting a new service ticket with this challenge as an authorization data element. The acceptor should be able to verify this in the subsequent AP-REQ, and then it should be able to detect that its clock is skewed and to estimate by how much.

## 9. Security Considerations

This document deals with security.

The new KRB-ERROR2 PDU is cryptographically distinguished from the original mechanism's acceptor success security context token (AP-REQ).

Not every KRB-ERROR2 can be integrity protected. This is unavoidable.

Because in the base Kerberos V5 GSS-API security mechanism all errors are unauthenticated, and because even with this specification some elements are unauthenticated, it is possible for an attacker to cause one peer to think that the security context token exchange has failed while the other thinks it will continue. This can cause an acceptor to waste resources while waiting for additional security context tokens from the initiator. This is not really a new problem, however: acceptor applications should already have suitable timeouts on security context establishment.

There is a binding of preceding security context tokens in each additional AP-REQ, via the challenge-response nonce. This binding is weak, and does not detect all modifications of unauthenticated plaintext in preceding security context tokens.

[[anchor1: We could use the GSS\_EXTS\_FINISHED extension from draft-ietf-kitten-iakerb to implement a strong binding of all context tokens.]]

Early prot\_ready per-message tokens have security considerations that are beyond the scope of this document and which are not exhaustively described elsewhere yet. Use only with care.

## 10. IANA Considerations

[[anchor2: Various allocations are required...]]



## 11. References

### 11.1. Normative References

- [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.
- [RFC4121] Zhu, L., Jaganathan, K., and S. Hartman, "The Kerberos Version 5 Generic Security Service Application Program Interface (GSS-API) Mechanism: Version 2", RFC 4121, July 2005.
- [CCITT.X680.2002] International Telephone and Telegraph Consultative Committee, "Abstract Syntax Notation One (ASN.1): Specification of basic notation", CCITT Recommendation X.680, July 2002.
- [CCITT.X690.2002] International Telephone and Telegraph Consultative Committee, "ASN.1 encoding rules: Specification of basic encoding Rules (BER), Canonical encoding rules (CER) and Distinguished encoding rules (DER)", CCITT Recommendation X.690, July 2002.

### 11.2. Informative References

- [RFC3530] Shepler, S., Callaghan, B., Robinson, D., Thurlow, R., Beame, C., Eisler, M., and D. Noveck, "Network File System (NFS) version 4 Protocol", RFC 3530, April 2003.
- [RFC4251] Ylonen, T. and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture", RFC 4251, January 2006.
- [I-D.swift-win2k-krb-user2user] Swift, M., Brezak, J., and P. Moore, "User to User Kerberos Authentication using GSS-API", draft-swift-win2k-krb-user2user-03 (work in progress), February 2011.

Authors' Addresses

Nicolas Williams  
Cryptonector, LLC

Email: nico@cryptonector.com

Roland Charles Dowdeswell  
Dowdeswell Security Architecture

Email: elric@imrryr.org

