Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 17, 2014

S. Cantor
Shibboleth Consortium
S. Josefsson
SJD AB

January 13, 2014

SAML Enhanced Client SASL and GSS-API Mechanisms
draft-ietf-kitten-sasl-saml-ec-11.txt

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 to facilitate an extensible authentication model. 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 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 17, 2014.

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 4
2. Terminology 6
3. Applicability for Non-HTTP Use Cases 7
4. SAML SASL Mechanism Specification 10
4.1. Advertisement 10
4.2. Initiation 10
4.3. Server Response 11
4.4. User Authentication with Identity Provider 11
4.5. Client Response 11
4.6. Outcome 11
4.7. Additional Notes 11
5. SAML EC GSS-API Mechanism Specification 13
5.1. GSS-API Credential Delegation 13
5.2. GSS-API Channel Binding 14
5.3. Session Key Derivation 15
5.3.1. Generated by Identity Provider 16
5.3.2. Alternate Key Derivation Mechanisms 16
5.4. Per-Message Tokens 17
5.5. Pseudo-Random Function (PRF) 17
5.6. GSS-API Principal Name Types for SAML EC 18
5.6.1. User Naming Considerations 18
5.6.2. Service Naming Considerations 19
6. Example 20
7. Security Considerations 28
7.1. Risks Left Unaddressed 28
7.2. User Privacy 28
7.3. Collusion between RPs 29
8. IANA Considerations 30
8.1. GSS-API and SASL Mechanism Registration 30
8.2. XML Namespace Name for SAML-EC 30
9. References 31
9.1. Normative References 31
9.2. Normative References for GSS-API Implementers 32
9.3. Informative References 33
Appendix A. XML Schema 34
Appendix B. Acknowledgments 36
Appendix C. Changes 37

Authors' Addresses 38

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). It includes a number of 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.

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, POP and XMPP [RFC3920]. The effect is to make authentication modular, so that newer authentication mechanisms can be added as needed.

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. This document defines a pure SASL mechanism for SAML, but it conforms to the bridge between SASL and the 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. 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], and therefore does not establish a separate authentication, integrity and confidentiality mechanism. It is anticipated that existing security layers, such as Transport Layer

Security (TLS) or Secure Shell (SSH), will continued to be used.

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.

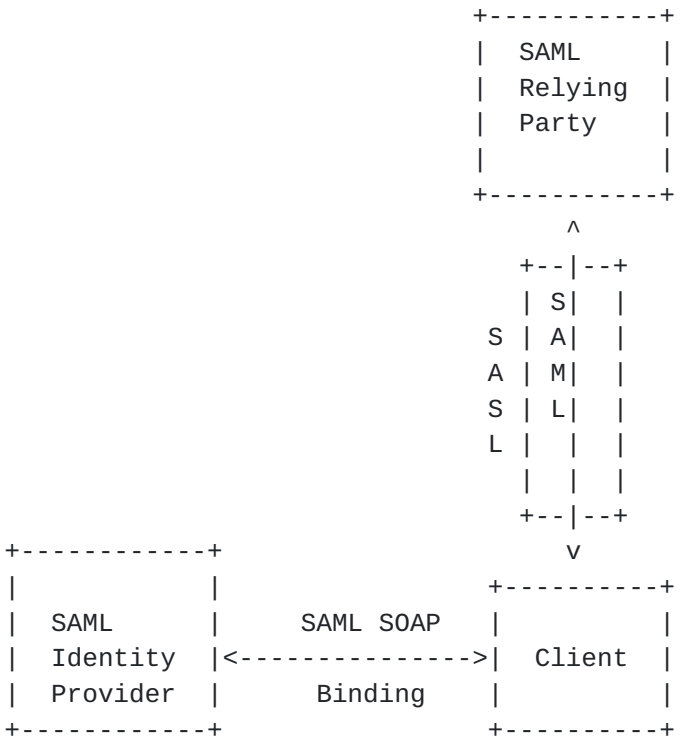


Figure 1: Interworking Architecture

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

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. To faciliate these two variants, the references has been split into two parts, one part that provides normative references for all readers, and one part that adds additional normative references required for implementers that wish to implement the GSS-API portion.

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 [RFC2616] applications. They are not, however, limited to browsers, because it was recognized that browsers suffer from a variety of functional and security deficiencies that would be useful to avoid where possible. 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 configuration ahead of time). 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 untouched. 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.

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

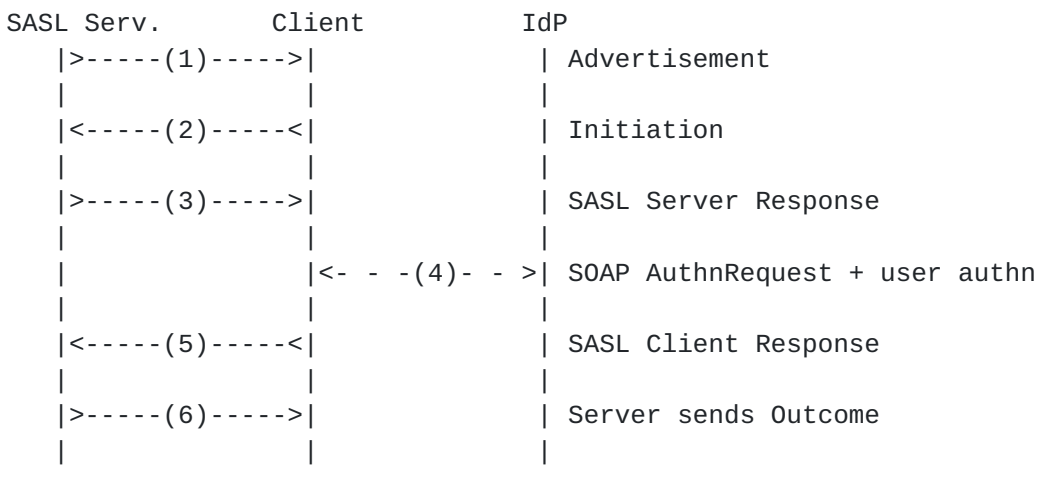


Figure 2: Authentication flow

4. SAML 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 (the latter indicating support for channel binding).

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 [RFC2617] and TLS client authentication as defined in [RFC5246].

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. 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. The parties then perform the steps described in [SAMLECP20] as usual.

Similarly, 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 SASL Enhanced Clients mechanism is also a GSS-API mechanism. The messages are the same, but a) the GS2 header on the client's first 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 prefixed to the client's first authentication message (context token).

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 "mutual-auth" 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 identity provider signals this to the client in an <ecp: RequestAuthenticated> SOAP header block.

If the mutual_state flag is not requested, or is not set, then the security layer managed by the application outside of the GSS-API mechanism is responsible for authenticating the acceptor. In this case, applications MUST match the server identity from the existing security layer with the target name. For TLS, this matching MUST be performed as discussed in [RFC6125]. For SSH, this matching MUST be performed as discussed in [RFC4462].

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 supports credential delegation through the issuance of SAML assertions that the issuing identity provider 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 identity provider 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 may have its deleg_state flag (GSS_C_DELEG_FLAG) set to TRUE.

The identity provider, 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 name 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 identity provider. 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 Identity Provider-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 name 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, 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 identity provider, 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 identity provider 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 SHOULD encrypt the assertion; if channel binding is not used, the assertion MUST be encrypted. 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 identity provider 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>aes128-cts-hmac-sha1-96</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 key.

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 encrytion 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 [RFC4402]. 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 identity provider 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 identity provider, and the information supplied by the identity provider 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 accordance with [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 identity provider, 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. Such a name is used directly by this mechanism as the effective AssertionConsumerService "location" associated with the service.

This value is used in the construction of the responseConsumerURL and AssertionConsumerServiceURL attributes, and for eventual comparison and validation by the client before completing the exchange. The value MUST be securely associated with the SAML entityID claimed by the server by the identity provider, such as through the use of SAML metadata [OASIS.saml-metadata-2.0-os].

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>:

<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUKICAgIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpT
QU1MOjIuMDphc3NlcnRpb24iCiAgICB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5h
bWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIgogICAgeG1sbnM6Uz0iaHR0cDovL3Nj
aGVtYXMueG1sc29hcC5vcmcvc29hcC9lbnZlbG9wZS8iPgogIDxTOkhlYWRlcj4K
ICAgIDxwYW9zOlJlcXVlc3QgeG1sbnM6cGFvcz0idXJuOmxpYmVydHk6cGFvczoy
MDAzLTA4IgogICAgICBtZXNzYWdlSUQ9ImMzYTRmOGI5YzJkIiBTOm11c3RVbmRl
cnN0YW5kPSIxIgogICAgICBTOmFjdG9yPSJodHRwOi8vc2NoZW1hcy54bWxzb2Fw
Lm9yZy9zb2FwL2FjdG9yL25leHQiCiAgICAgIHJlc3BvbnNlQ29uc3VtZXJVUkw9
InhtcHBAeG1wcC5leGFtcGxlLmNvbSIKICAgICAgc2VydmljZT0idXJuOm9hc2lz
Om5hbWVzOnRjOlNBTUw6Mi4wOnByb2ZpbGVzOlNTTzplY3AiLz4KICAgIDxlY3A6
UmVxdWVzdAogICAgICB4bWxuczplY3A9InVybjpvYXNpczpuYW1lczp0YzpTQU1M
OjIuMDpwcm9maWxlczpTU086ZWNwIgogICAgICBTOmFjdG9yPSJodHRwOi8vc2No
ZW1hcy54bWxzb2FwLm9yZy9zb2FwL2FjdG9yL25leHQiCiAgICAgIFM6bXVzdFVu
ZGVyc3RhbmQ9IjEiIFByb3ZpZGVyTmFtZT0iSmFiYmVyIGF0IGV4YW1wbGUuY29t
Ij4KICAgICAgPHNhbWw6SXNzdWVyPmh0dHBzOi8veG1wcC5leGFtcGxlLmNvbTwv
c2FtbDpJc3N1ZXI+CiAgICA8L2VjcDpSZXF1ZXN0PgogICAgPHNhbWxwOkV4dGVu
ZGVyc3RhbmQ9IjEiIFByb3ZpZGVyTmFtZT0iSmFiYmVyIGF0IGV4YW1wbGUuY29t
Ij4KICAgICAgPHNhbWw6SXNzdWVyPmh0dHBzOi8veG1wcC5leGFtcGxlLmNvbTwv
c2FtbDpJc3N1ZXI+CiAgICA8L2VjcDpSZXF1ZXN0PgogIDwvUzpIZWFkZXI+CiAg
ICAgIDxwYW9zOlJlcXVlc3QgeG1sbnM6cGFvcz0idXJuOmxpYmVydHk6cGFvczoy
MDAzLTA4IgogICAgICBtZXNzYWdlSUQ9ImMzYTRmOGI5YzJkIiBWZXJzaW9uPSIyLjAi
IElzc3VlSW5zdGFudD0iMjAwNy0xMi0xMFQxMTozOTozNFoiCiAgICAgIFByb3Rv
Y29sQmluZGluZz0idXJuOm9hc2lzOm5hbWVzOnRjOlNBTUw6Mi4wOmJpbmRpbmdz
OlBBT1MiCiAgICAgIEFzc2VydGlvbkNvbnN1bWVyU2VydmljZVVSTD0ieG1wcEB4
bXBwLmV4YW1wbGUuY29tIj4KICAgICAgPHNhbWw6SXNzdWVyIHhtbG5zOnNhbWw9
InVybjpvYXNpczpuYW1lczp0YzpTQU1MOjIuMDphc3NlcnRpb24iPgogICAgICAg
aHR0cHM6Ly94bXBwLmV4YW1wbGUuY29tCiAgICAgIDwvc2FtbDpJc3N1ZXI+CiAg
ICAgIDxzYW1scDpOYW1lSURQb2xpY3kgQWxsb3dDcmVhdGU9InRydWUiCiAgICAg
ICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgICAgIC

```
   Qm9keT4KPC9TOkVudmVsb3BlPg==
   </challenge>
```

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>aes128-cts-hmac-sha1-96</samlec:EncType>
      <samlec:EncType>aes256-cts-hmac-sha1-96</samlec:EncType>
    <samlec:SessionKey>
  </S:Header>
  <S:Body>
    <samlp:AuthnRequest
      ID="c3a4f8b9c2d" Version="2.0" IssueInstant="2007-12-10T11:39:34Z"
      ProtocolBinding="urn:oasis:names:tc:SAML:2.0:bindings:PAOS"
      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). HTTP portion not shown, 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="2007-12-10T11:42:34Z" 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:xml:ns:xmpp-sasl'>
PFM6RW52ZWxvcGUKICAgIHhtbG5zOnNhbWw9InVybjpvYXNpczpuYW1lczp0YzpT
QU1MOjIuMDphc3NlcnRpb24iCiAgICB4bWxuczpzYW1scD0idXJuOm9hc2lzOm5h
bWVzOnRjOlNBTUw6Mi4wOnByb3RvY29sIgogICAgeG1sbnM6Uz0iaHR0cDovL3Nj
aGVtYXMueG1sc29hcC5vcmcvc29hcC9lbnZlbG9wZS8iPgogIDxTOkhlYWRlcj4K
ICAgIDxwYW9zOlJlc3BvbnNlIHhtbG5zOnBhb3M9InVybjpsaWJlcnR5OnBhb3M6
MjAwMy0wOCIKICAgICAgUzphY3Rvcj0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5v
cmcvc29hcC9hY3Rvci9uZXh0IgogICAgICBTOm11c3RVbmRlcnN0YW5kPSIxIiBy
ZWZUb01lc3NhZ2VJRD0iNmMzYTRmOGI5YzJkIi8+CiAgICA8c2FtbGVjOlNlc3Np
b25LZXkgeG1sbnM6c2FtbGVjPSJ1cm46aWV0ZjpwYXJhbXM6eG1sOm5zOnNhbWxl
YyIKICAgICAgeG1sbnM6Uz0iaHR0cDovL3NjaGVtYXMueG1sc29hcC5vcmcvc29h
cC9lbnZlbG9wZS8iCiAgICAgIFM6bXVzdFVuZGVyc3RhbmQ9IjEiCiAgICAgIFM6
YWN0b3I9Imh0dHA6Ly9zY2hlbWFzLnhtbHNvYXAub3JnL3NvYXAvYWN0b3IvbmV4
dCI+CiAgICAgIDxzYW1sZWM6RW5jVHlwZT5hZXMxMjgtY3RzLWhtYWMtc2hhMS05
Njwvc2FtbGVjOkVuY1R5cGU+CiAgICA8c2FtbGVjOlNlc3Npb25LZXk+CiAgPC9T
OkhlYWRlcj4KICA8UzpCb2R5PgogICAgPHNhbWxwOlJlc3BvbnNlIElEPSJkNDNo
OTRyMzg5MzA5ciIgVmVyc2lvbj0iMi4wIgogICAgICAgIElzc3VlSW5zdGFudD0i
MjAwNy0xMi0xMFQxMTo0MjozNFoiIEluUmVzcG9uc2VUbz0iYzNhNGY4YjljMmQi
CiAgICAgICAgRGVzdGluYXRpb249InNhtcHBAeG1wcC5leGFtcGxlLmNvbSI+CiAg
ICAgIDxzYW1sOklzc3Vlcj5odHRwczovL3NhbWwuZXhhbXBsZS5vcmc8L3NhbWw6
SXNzdWVyPgogICAgICA8c2FtbHA6U3RhdHVzPgogICAgICAgIDxzYW1scDpTdGF0
dXNDb2RlCiAgICAgICAgICAgIFZhbHVlPSJ1cm46b2FzaXM6bmFtZXM6dGM6U0FN
TDoyLjA6c3RhdHVzOlN1Y2Nlc3MiLz4KICAgICAgPC9zYW1scDpTdGF0dXM+CiAg
ICAgIDxzYW1sOkVuY3J5cHRlZEFzc2VydGlvbj4KICAgICAgICA8IS0tIGNvbnRl
bnRzIGVsaWRlZCwgY29weSBvZiBzYW1sZWM6R2VuZXJhdGVkS2V5IGluIEFkdmlj
ZSAtLT4KICAgICAgPC9zYW1sOkVuY3J5cHRlZEFzc2VydGlvbj4KICAgICAgPC9z
bHA6UmVzcG9uc2U+CjvZHk+CjwvU0FtbFp2ZnZlbG9wZT4K
</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>aes128-cts-hmac-sha1-96</samlec:EncType>
     <samlec:SessionKey>
   </S:Header>
   <S:Body>
     <samlp:Response ID="d43h94r389309r" Version="2.0"
         IssueInstant="2007-12-10T11:42:34Z" 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 flawed commercial TLS infrastructure. 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 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, March 1997.

[RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4462] Hutzelman, J., Salowey, J., Galbraith, J., and V. Welch, "Generic Security Service Application Program Interface (GSS-API) Authentication and Key Exchange for the Secure Shell (SSH) Protocol", RFC 4462, May 2006.

[RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer

Security (TLS)", RFC 6125, March 2011.

[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. Normative References for GSS-API Implementers

[RFC2743] Linn, J., "Generic Security Service Application Program Interface Version 2, Update 1", RFC 2743, January 2000.

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

[RFC3962] Raeburn, K., "Advanced Encryption Standard (AES) Encryption for Kerberos 5", RFC 3962, February 2005.

[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.

[RFC4401] Williams, N., "A Pseudo-Random Function (PRF) API Extension for the Generic Security Service Application Program Interface (GSS-API)", RFC 4401, February 2006.

[RFC4402] Williams, N., "A Pseudo-Random Function (PRF) for the Kerberos V Generic Security Service Application Program Interface (GSS-API) Mechanism", RFC 4402, February 2006.

[RFC5554] Williams, N., "Clarifications and Extensions to the Generic Security Service Application Program Interface (GSS-API) for the Use of Channel Bindings", RFC 5554, May 2009.

[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.

[RFC6680] Williams, N., Johansson, L., Hartman, S., and S.

Josefsson, "Generic Security Service Application Programming Interface (GSS-API) Naming Extensions", RFC 6680, August 2012.

[RFC7056] Hartman, S. and J. Howlett, "Name Attributes for the GSS-API Extensible Authentication Protocol (EAP) Mechanism", RFC 7056, December 2013.

9.3. 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.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.

[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-SAMLTokenProfile, May 2012.

Appendix A. XML Schema

The following schema formally defines the "urn:ietf:params:xml:ns:samlec" 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="string"/>

     <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 like to thank Klaas Wierenga, Sam Hartman, Nico Williams, Jim Basney, and Venkat Yekkirala for their contributions.

Appendix C. Changes

This section to be removed prior to publication.

- 11, update EAP Naming reference to RFC
- 10, update SAML ECP reference to final CS
- 09, align delegation signaling to updated ECP draft
- 08, more corrections, added a delegation signaling header
- 07, corrections, revised section on delegation
- 06, simplified session key schema, moved responsibility for random-to-key to the endpoints, and defined advertisement of session key algorithm and enctypes by acceptor
- 05, revised session key material, added requirement for random-to-key, revised XML schema to capture enctype name, updated GSS naming reference
- 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
- 03, added TLS key export as a session key option, revised GSS naming material based on list discussion
- 02, major revision of GSS-API material and updated references
- 01, SSH language added, noted non-assumption of HTTP error handling, added guidance on life of security context.
- 00, Initial Revision, first WG-adopted draft. Removed support for unsolicited SAML responses.

Authors' Addresses

Scott Cantor
Shibboleth Consortium
2740 Airport Drive
Columbus, Ohio 43219
United States

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

Simon Josefsson
SJD AB
Hagagatan 24
Stockholm 113 47
SE

Email: simon@josefsson.org
URI: http://josefsson.org/