Network Working Group
Internet-Draft
Intended status: Experimental
Expires: June 12, 2014

L. Howard
PADL
N. Williams
Cryptonector

December 9, 2013

A SASL and GSS-API Mechanism for the BrowserID Authentication Protocol
draft-howard-gss-browserid-07.txt

Abstract

This document defines protocols, procedures and conventions for a Generic Security Service Application Program Interface (GSS-API) security mechanism based on the BrowserID authentication mechanism. Through the GS2 family of mechanisms defined in RFC 5801, these protocols also define how Simple Authentication and Security Layer (SASL, RFC 4422) applications may use BrowserID.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

[BrowserID] is a web-based three-party security protocol by which user agents can present to a Relying Party (RP) a signed assertion of e-mail address ownership. BrowserID was intended to be used for web authentication. We find BrowserID to be useful in general, therefore we define herein how to use it in many more applications.

The Simple Authentication and Security Layer (SASL) [RFC4422] is a framework for providing authentication and message protection services via pluggable mechanisms. Protocols that support it include IMAP, SMTP, and XMPP.

The Generic Security Service Application Program Interface (GSS-API) [RFC2743] provides a framework for authentication and message protection services through a common programming interface. This document conforms to the SASL and GSS-API bridge specified in [RFC5801], so it defines both a SASL and GSS-API mechanism.

The BrowserID mechanism described in this document reuses the existing web-based BrowserID protocol, but profiles it for use in applications that support SASL or GSS-API, adding features such as key agreement, mutual authentication, and fast re-authentication.

The following diagram illustrates the interactions between the three parties in the GSS BrowserID protocol. Note that the terms client, initiator and user agent (UA) are used interchangeably in this document, as are server, acceptor and relying party (RP).

```
                       +------------+
                       | BrowserID  |
                       | identity   |
                       | provider   |
                       +------------+
                         //      \\
                        //        \\
                       //          \\
                      //            \\
      make signed    //              \\    fetch IdP public
      certificate   //                \\   key over HTTPS
      for user's   //                  \\  (RP may cache)
      public key  //                    \\
                 //                      \\
                //                        \\
               //                          \\
              |/                            \|
       +-------------+                     +-------------+
       | SASL or GSS |    GSS BrowserID    | SASL or GSS |
       | client/UA   |<------------------->| server/RP   |
       | (initiator) |                     | (acceptor)  |
       +-------------+                     +-------------+
```

Figure 1: Interworking Architecture

1.1. Discovery and Negotiation

The means of discovering GSS-API peers and their supported mechanisms is out of this specification's scope. They may use SASL [RFC4422] or the Simple and Protected Negotiation mechanism (SPNEGO) [RFC4178].

Discovery of a BrowserID identity provider (IdP) for a user is described in the BrowserID specification. A domain publishes a document containing their public key and URIs for authenticating and provisioning users, or pointer to an authority containing such a document.

1.2. Authentication

The GSS-API protocol involves a client, known as the initiator, sending an initial security context token of a chosen GSS-API security mechanism to a peer, known as the acceptor. The two peers subsequently exchange, synchronously, as many security context tokens as necessary to complete the authentication or fail. The specific number of context tokens exchanged varies by security mechanism: in the case of the BrowserID mechanism, it is typically two (i.e. a single round trip), however it can be more in some cases. Once

authentication is complete, the initiator and acceptor share a security context which identifies the peers and can optionally be used for integrity or confidentiality protecting subsequent application messages.

The original BrowserID protocol, as defined outside this document, specifies a bearer token authentication protocol for web applications. The user agent generates a short-term key pair, the public key of which is signed by the user's IdP. (The user must have already authenticated to the IdP; how this is done is not specified by BrowserID, but forms-based authentication is common.) The IdP returns a certificate for the user which may be cached by the user's browser. When authenticating to a Relying Party (RP), the browser generates an identity assertion containing the RP domain and an expiration time. The user agent signs this and presents both the assertion and certificate to the RP. (The combination of an assertion and zero or more certificates is termed a "backed assertion".) The RP fetches the public key for the IdP, validates the user's certificate (and those of any intermediate certifying parties) and then verifies the assertion.

The GSS BrowserID protocol extends this by having the RP always send back a response to the user agent, which at a minimum provides key confirmation (this is needed for some key agreement methods) and indicates the lifetime of the established security context. The key confirmation token is also required for mutual authentication, when the initiator application requests that feature.

1.3. Message protection services

GSS-API provides a number of a message protection services:

GSS_Wrap() integrity and optional confidentiality for a message

GSS_GetMIC() integrity for a message sent separately

GSS_Pseudo_random() shared key derivation (e.g., for keying external confidentiality+integrity layers)

These services may be used with security contexts that have a shared session key, to protect application-layer messages.

2. 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].

The reader is assumed to be familiar with the terms used in the BrowserID specification.

3. Naming

The GSS-API provides a rich security principal naming model. At its most basic the query forms of names consist of a user-entered/displayable string and a "name-type". Name-types are constants with names prefixed with "GSS_C_NT_" in the GSS-API. Names may also have attributes [RFC6680].

3.1. GSS name types

3.1.1. GSS_C_NT_BROWSERID_PRINCIPAL

This name may contain an e-mail address, or a service principal name identifying an acceptor. The encoding of service principal names is intended to be somewhat compatible with the Kerberos [RFC4120] security protocol (without the realm name).

The following ABNF defines the 'name' rule that names of this type must match.

[[anchor1: Should we reference RFC2822 here? The Mozilla BrowserID docs sure don't.]]

```
char-normal = %x00-2E/%x30-3F/%x41-5B/%x5D-FF
char-escaped = "\" %x2F / "\" %x40 / "\" %x5C
name-char = char-normal / char-escaped
name-string = 1*name-char
user = name-string
domain = name-string
email = user "@" domain
service-name = name-string
service-host = name-string
service-specific = name-string
service-specifics = service-specific 0*("/" service-specifics)
spn = service-name ["/" service-host [ "/" service-specifics]]
name = email / spn
```

3.1.2. GSS_C_NT_USER_NAME

This name is implicitly converted to a GSS_C_NT_BROWSERID_PRINCIPAL. A default domain may be appended when importing names of this type.

3.1.3. GSS_C_NT_HOSTBASED_SERVICE

This name is transformed by replacing the "@" symbol with a "/", and then implicitly converted to a GSS_C_NT_BROWSERID_PRINCIPAL.

3.1.4. GSS_C_NT_DOMAINBASED_SERVICE

[RFC5178] domain-based service names are transformed into a GSS_C_NT_BROWSERID_PRINCIPAL as follows:

- the <service> name becomes the first component of the BrowserID principal name (service-name in ABNF)
- the <hostname> becomes the second component (service-host)
- the <domain> name becomes the third component (service-specific)

3.1.5. GSS_C_NT_ANONYMOUS

If the initiator principal's leaf certificate does not contain a "principal" claim, then the initiator name has this name type.

3.2. Name canonicalization

The BrowserID GSS-API mechanism performs no name canonicalization. The mechanism's GSS_Canonicalize_name() returns an MN whose display form is the same as the query form. Of course, the principal named obtained from a CREDENTIAL HANDLE may be canonical in that the IdP might only issue credentials for canonical names, but credential acquisition is out of scope here.

3.3. Exported name token format

The exported name token format for the BrowserID GSS-API mechanism is the same as the query form, plus the standard exported name token format header mandated by the GSS-API [RFC2743].

[[anchor2: Do we wish to say anything about the exported composite name token format? It should be an encoding of the initiator's leaf certificate.]]

3.4. Naming extensions

The acceptor MAY surface attributes from the assertion and any certificates using GSS_Get_name_attribute() (see [RFC6680]). The URN prefix is "urn:<TBD>:params:gss:jwt". If a SAML assertion is present in the "saml" parameter of the leaf certificate, it may be surfaced using the URN prefix "urn:<TBD>:params:gss:federated-saml-attribute".

Attributes from the assertion MUST be marked as unauthenticated unless otherwise validated by the acceptor (e.g. the audience).

Attributes from certificates SHOULD be marked as authenticated.

4. Context tokens

All context tokens include a two-byte token identifier followed by a backed BrowserID assertion. This document defines the following token IDs:

Section	Token ID	ASCII	Description
4.1.1	0x632C	c,	Initiator context token
4.1.2	0x432C	C,	Acceptor context token
	0x442C	D,	Context deletion token
4.2.4	0x6D2C	m,	Initiator metadata token
4.2.4	0x4D2C	M,	Acceptor metadata token

The token ID has a human-readable ASCII encoding for the benefit of pure SASL implementations of this mechanism.

4.1. Base protocol

4.1.1. Initial context token

The initial context token is framed per Section 1 of [RFC2743]:

```
GSS-API DEFINITIONS ::=
    BEGIN

    MechType ::= OBJECT IDENTIFIER
    -- representing BrowserID mechanism
    GSSAPI-Token ::=
    [APPLICATION 0] IMPLICIT SEQUENCE {
        thisMech MechType,
        innerToken ANY DEFINED BY thisMech
            -- token ID and backed assertion
    }
    END
```

Unlike many other GSS-API mechanisms such as Kerberos, this token framing is not used by subsequent context or by [I-D.zhu-negoex] metadata tokens. As such, pure SASL implementations of this mechanism do not need to deal with DER encoding the mechanism object identifier.

GSS BrowserID is a family of mechanisms, where the last element in the OID arc indicates the [RFC4121] encryption type supported for message protection services. The OID prefix is 1.3.6.1.4.1.5322.24.1. The NULL encryption type is valid, in which case services that require a key are not available.

The innerToken consists of the initiator context token ID concatenated with a backed assertion for the audience corresponding to the target name passed into GSS_Init_sec_context(). In addition, the assertion MAY contain the additional claims, which are described later in this document:

- ECDH key agreement parameters (see Section 6.1.5)

- Channel binding information (see Section 6.1.6)

- A nonce for binding the request to a response signed with a private key for mutual authentication (see Section 6.1.7)

- A ticket identifier for fast re-authentication using an established session key rather than a BrowserID certificate (see Section 6.1.8)

The call to GSS_Init_sec_context() returns GSS_C_CONTINUE_NEEDED to indicate that a subsequent context token from the acceptor is expected.

4.1.2. Acceptor context token

Upon receiving a context token from the initiator, the acceptor validates that the token is well formed and contains a valid BrowserID mechanism OID and the initiator context token ID.

The acceptor then verifies the backed identity assertion per the BrowserID specification. This includes validating the expiry times, audience, certificate chain, and assertion signature. The acceptor then verifies the channel binding token, if present, and any other GSS-specific claims in the assertion. In case of failure, a response assertion containing GSS major and minor status codes SHOULD be returned.

If the [RFC3961] encryption type for the selected mechanism is not ENCTYPE_NULL, the acceptor generates a ECDH public key using the parameters received from the client (see Section 6.2.2), and from it derives the RP Response Key (RRK) (see Section 7.3). The acceptor then generates a response assertion containing its ECDH public key and context expiration time (note that the context expiration time is a purely informational quantity). The response assertion will be:

- signed in the acceptor's private key, if mutual authentication was requested, and the acceptor has a key (see Section 4.2);
- signed in the RRK, if the encryption type for the selected mechanism is not ENCTYPE_NULL;
- not signed in all other cases.

The response assertion is encoded as a backed assertion, prefixed with the acceptor context token ID. It SHALL have a certificate count of zero.

Finally, the Context Root Key (CRK) (see Section 7.4) is derived from the ECDH shared secret (if present) and GSS_S_COMPLETE is returned, along with the initiator name from the verified assertion. If the CRK is available, 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 set to TRUE.

Other assertion/certificate claims MAY be made available via GSS_Get_name_attribute().

4.1.3. Initiator context completion

Upon receiving the acceptor context token, the initiator unpacks the response assertion and, if applicable, computes the ECDH shared secret and RRK. The RRK is used to verify the response assertion unless mutual authentication is available, in which case the acceptor's public key will be used.

The initiator sets the context expiry time with that received in the response assertion, if present; otherwise, the context expires when the initiator principal's certificate expires.

The CRK is derived from the ECDH shared secret and GSS_S_COMPLETE is returned to indicate the initiator is authenticated and the context is ready for use. No output token is emitted. Security context flags are set as for the acceptor context.

4.2. Mutual authentication

Mutual authentication allows the acceptor to be authenticated to the initiator. The mechanism SHALL set the mutual_state security context flag (GSS_C_MUTUAL_FLAG) to TRUE if mutual authentication succeeded. Support for mutual authentication is OPTIONAL.

The base protocol is extended as follows to support this:

4.2.1. Initiator mutual authentication context token

If the initiator requested the mutual_state flag, it sends in its request assertion an "opts" claim (see Section 6.1.9) containing the "ma" value. It also includes a nonce (see Section 6.1.7) in order to bind the initiator and acceptor assertions.

4.2.2. Acceptor mutual authentication context token

If the acceptor has a private key and certificate available and the initiator indicated it desired mutual authentication by including the "ma" protocol option, the acceptor signs the response using a private key rather than the RP Response Key (RRK). The response includes the nonce from the initiator's assertion. The acceptor MUST reject requests for mutual authentication lacking a nonce.

While the response is a backed assertion, in order to take advantage of existing keying infrastructures BrowserID certificates MUST NOT be included in the backed assertion. Rather, an X.509 certificate SHALL be included as a value for the "x5c" header parameter in the assertion (see [I-D.ietf-jose-json-web-signature] 4.1.6). The certificate MUST be valid for signing.

[[anchor3: We don't want to burden the initiator with having to implement both methods of authenticating acceptors, and given that initiators and acceptors both will generally need a PKIX implementation, and given that acceptors will need a PKIX credential for TLS, and that there is as yet no standard protocol for automatic provisioning of BrowserID credentials for servers, using PKIX to authenticate the server seems to be the easiest way to go.]]

4.2.3. Initiator mutual authentication context completion

The initiator verifies the assertion signature and that the nonce matches, and validates the certificate chain according to [RFC5280].

Initiators MUST authenticate the service name using the matching rules below:

- o A service-name EKU from the registry defined by [I-D.zhu-pku2u]; id-kpServerAuth maps to the "http" service

- o A spn expressed as a KRB5PrincipalName in the id-pkinit-san otherName SAN (see [RFC4556] Section 3.2.2; the realm is ignored)

- o A service-name expressed as a SRVName SAN (see [RFC4985])

o Optionally, an out-of-band binding to the certificate

If there are no EKUs, or a single EKU containing id-kp-anyExtendedKeyUsage, and no SAN containing the service name is present, then all service names match. If a SAN containing the service name is present, then any EKUs are ignored.

If the the host component of the service name (service-host) is not expressed in a SAN as specified above, it MUST be present as a value for the dNSName SAN or as the least significant Common Name RDN.

Note only the id-pkinit-san or SRVName SANs provide the ability to authenticate the a service name containing a service-specific component.

4.2.4. Acceptor certificate advertisement

[I-D.zhu-negoex] may be used to advertise acceptor certificates.

If the acceptor supports mutual authentication, it MAY include its certificate and any additional certificates inside a backed assertion with an empty payload as output for GSS_Query_meta_data(). The "assertion" is prefixed with the two byte token identifier "M,".

Upon receiving this, the initiator MAY validate the certificate or fingerprint, or present either to the initiator before committing to authenticate.

The NegoEx signing key is the output of GSS_Pseudo_random() (see Section 7.7) with an input of GSS_C_PRF_KEY_FULL and "gss-browserid-negoex-initiator" or "gss-browserid-negoex-acceptor" (without quotes), depending on the party generating the signature.

The NegoEx authentication scheme is the binary encoding of the following hexadecimal string:

535538008647F5BC624BD8076949F0

where the third byte (zero above) is set to the [RFC3961] encryption type for the selected mechanism. The authentication scheme for encryption types greater than 255 is not specified here.

There is currently no initiator-sent metadata defined and acceptors should ignore any sent. The metadata is advisory and the initiator is free to ignore it.

[[anchor4: Delete this section as NegoEx will likely not be progressed.]]

4.3. Fast re-authentication

Fast re-authentication allows a security context to be established using a secret key derived from the initial certificate-signed ECDH key agreement.

The re-authentication assertion is signed with a HMAC using the Authenticator Root Key (ARK) (see Section 7.5), rather than a initiator principal's BrowserID certificate.

Support for fast re-authentication is OPTIONAL and is indicated by the acceptor returning a ticket in the response assertion.

4.3.1. Ticket generation

If the acceptor supports re-authentication, the following steps are added to Section 4.1.2:

1. A unique, opaque ticket identifier is generated.

2. The acceptor creates a JSON object containing the ticket identifier and expiry time and returns it in the response to the initiator (see Section 6.2.5).

The acceptor must be able to use the ticket identifier to securely retrieve the subject, issuer, audience, expiry time, ARK and any other relevant properties of the original security context. One implementation choice may be to use the ticket identifier as a key into a dictionary containing this information. Another would be to encrypt this information in a long-term secret only known to the acceptor and encode the resulting cipher-text in the opaque ticket identifier.

The ticket expiry time by default SHOULD match the initiator's certificate expiry, however it MAY be configurable so the ticket expires before or after the certificate.

The initiator MAY cache tickets, along with the ARK, received from the acceptor in order to re-authenticate to it at a future time.

4.3.2. Initiator re-authentication context token

The initiator looks in its ticket cache for an unexpired ticket for the desired acceptor. If none is found, the normal certificate-based authentication flow is performed, otherwise:

1. The initiator generates a re-authentication assertion containing: the name of the acceptor (see Section 6.1.1), an expiry time (see Section 6.1.2) and/or the current time (see Section 6.1.3), optional channel binding information (see Section 6.1.6), a random nonce (see Section 6.1.7), and the ticket identifier (see Section 6.1.8).

2. The initiator signs the re-authentication assertion with the ARK, using the hash algorithm associated with the original context key (see Section 10.1; HS256 is specified for the encryption types referenced in this document).

3. The re-authentication assertion is packed into a backed assertion. The certificate count is zero as the assertion is signed with an established symmetric key.

4. The initiator generates an Authenticator Session Key (ASK) (see Section 7.6) which is used to verify the response and derive the CRK.

[[anchor5: Question: do we want an option to do an ECDH session key exchange in the fast re-auth case? If we had a GSS req_flag for requesting perfect forward security (PFS) then we would want to have this option.]]

4.3.3. Acceptor re-authentication context token

1. The acceptor unpacks the re-authentication assertion and retrieves the ARK, ticket expiry time, mutual authentication state and any other properties (such as the initiator name) associated with the ticket identifier.

2. The acceptor validates that the ticket and re-authentication assertion have not expired.

3. The acceptor verifies the assertion using the ARK.

4. The acceptor generates the ASK (see Section 7.6) and derives the RRK and CRK from this (see Section 7.3 and Section 7.4, respectively).

5. The acceptor generates a response and signs and returns it. Note that, unlike the certificate-based mutual authentication case, the nonce need not be echoed back as the ASK (and thus the RRK) is cryptographically bound to the nonce.

If the ticket cannot be found, or the authentication fails, the acceptor SHOULD return a REAUTH_FAILED error, permitting the

initiator to recover and fallback to generating a BrowserID assertion. It MAY also include its local timestamp (see Section 6.2.1) so that the initiator can perform clock skew compensation.

4.3.4. Interaction with mutual authentication

The mutual authentication state of a re-authenticated context is transitive. The initiator and acceptor MUST NOT set the mutual_state flag for a re-authenticated context unless the original context was mutually authenticated.

As such, the mutual authentication state of the original context must be associated with the ticket.

4.3.5. Ticket renewal

Normally, re-authentication tickets are only issued when the initiator authenticated with a certificate-signed assertion. Acceptors MAY issue a new ticket with an expiry beyond the ticket lifetime when the initiator used a re-authentication assertion. The issuing of new tickets MUST be subject to a policy that prevents them from being renewed indefinitely.

4.4. Extra round-trip (XRT) option

The extra round-trip (XRT) option adds an additional round trip to the context token exchange. It allows the initiator to prove knowledge of the Context Master Key (CMK) (see Section 7.2) by sending an additional token signed in a key derived from the CMK and an acceptor-issued challenge. Support for the XRT option is OPTIONAL in the acceptor and REQUIRED in the initiator. The initiator is allowed to not request it, but MUST perform XRT if the acceptor requires it.

(Note that the term "extra round trip" is something of a misnomer; it only adds an additional token to the context token exchange. It is anticipated however that this mechanism will most commonly be used with pseudo-mechanisms or application protocols that require an even number of tokens.)

4.4.1. Initiator XRT advertisement

The initiator may advertise to the acceptor that it desires the XRT option by sending in its request assertion an "opts" claim (see Section 6.1.9) containing the "xrt" value. This option MUST be set if the caller requested GSS_C_DCE_STYLE (see [RFC4757]). Otherwise, the setting of this option is implementation dependent.

4.4.2. Acceptor XRT advertisement

If the initiator requested the XRT option and the acceptor supports it, or the acceptor requires it, the acceptor sends a "jti" claim (see Section 6.2.6) in the response assertion containing a random base 64 URL encoded value. This value MUST be at least 64 bits in length. The acceptor then returns GSS_C_CONTINUE_NEEDED to indicate that an additional context token is expected from the initiator.

4.4.3. Initiator XRT context token

If the acceptor indicated support for the XRT option by including a "jti" claim in its response, then the initiator sends an additional context token to the acceptor. This token contains the initiator context token ID concatenated with a backed assertion with zero certificates and an empty payload, signed using the XRTK (see Section 7.6.1).

4.4.4. Acceptor XRT context token validation

The acceptor MUST validate the XRT context token by first validating the context token ID, and then verifying the assertion signature with the XRTK. The acceptor SHOULD reject XRT context tokens with a certificate count greater than zero. Unknown claims in the assertion payload MUST be ignored. The acceptor then returns GSS_C_COMPLETE to the caller.

The acceptor MAY avoid using a replay cache when this option is in effect.

4.4.5. Interaction with message protection services

When the XRT option is in effect, the XRTK is used instead of the CMK to derive the Context Root Key (CRK) (see Section 7.4). Per-message tokens MUST have the AcceptorSubkey flag set (see [RFC4121] Section 4.2.2).

5. Validation

5.1. Expiry times

The expiry and, if present, issued-at and not-before times of all elements in a backed assertion, MUST be validated. This applies equally to re-authentication assertions, public key assertions, and the entire certificate chain. If the expiry time is absent, the issued-at time MUST be present, and the JWT implicitly expires a short, implementation-defined interval after the issued-at time. (A suggested interval is five minutes.)

The GSS context lifetime SHOULD NOT exceed the lifetime of the initiator principal's certificate.

The lifetime of a re-authentication ticket SHOULD NOT exceed the lifetime of the initiator principal's certificate. The acceptor MUST validate the ticket expiry time when performing re-authentication.

Message protections services such as GSS_Wrap() SHOULD be available beyond the GSS context lifetime for maximum application compatibility.

5.2. Audience

If the credential passed to GSS_Accept_sec_context() is not for GSS_C_NO_NAME, then its string representation as a BrowserID principal (see Section 3.1.1) MUST match the audience claim in the assertion.

5.3. Channel bindings

GSS-API channel binding is a protected facility for naming an enclosing channel between the initiator and acceptor. If the acceptor passed in channel bindings to GSS_Accept_sec_context(), the assertion MUST contain a matching channel binding claim. (Only the application_data component is validated.)

The acceptor SHOULD accept any channel binding provided by the initiator if NULL channel bindings are passed to GSS_Accept_sec_context().

5.4. Key agreement

The initiator MUST choose an ECDH curve with an equivalent strength to the negotiated [RFC4121] encryption type. Appropriate curves are given in Section 10.1.

The curve strength MUST be verified by the acceptor. A stronger than required curve MAY be selected by the initiator.

5.5. Signatures

Signature validation on assertions is the same as for the web usage of BrowserID, with the addition that response assertions may and re-authentication assertions must be signed with a symmetric key. In this case the HMAC algorithm associated with the mechanism OID is used, and there are no certificates in the backed assertion.

5.6. Replay detection

If the XRT option is not in effect, the acceptor MUST maintain a cache of received assertions in order to guard against replay attacks.

5.7. Return flags

The initiator and acceptor should set the returned flags as follows:

deleg_state never set

mutual_state set if the initiator requested mutual authentication and mutual authentication succeeded

replay_det_state set if message protection services are available

sequence_state set if message protection services are available

anon_state set if the initiator principal's leaf certificate lacks a "principal" claim

trans_state set if the implementation supports importing and exporting of security contexts

prot_ready_state may be set when or after the RP Response Token is produced or consumed

conf_avail set if message protection services are available

integ_avail set if message protection services are available

6. Assertion claims

6.1. Request (initiator/UA) assertion

These claims are included in the assertion sent to the acceptor and are authenticated by the initiator's private key and certificate chain (directly, or in the case of re-authentication assertions, transitively). Claims not specified here MUST be ignored by the acceptor.

Here is an example assertion containing Elliptic Curve Diffie-Hellman parameters, along with options and nonce claims indicating that mutual authentication is desired:


```
{
    "opts": [
        "ma"
    ],
    "exp": 1360158396188,
    "epk": {
        "kty": "EC",
        "crv": "P-256",
        "x": "JR5UPDgMLFPZwOGaKKSF24658tB1DccM1_oHPbCHeZg",
        "y": "S45Esx_6DfE5-xdB3X7sIIJ16MwO0Y_RiDc-i5ZTLQ8"
    },
    "nonce": "bbqT10Gyx3s",
    "aud": "imap/mail.example.com"
}
```


The following claims are permitted in the request assertion:

6.1.1. "aud" (Audience)

The audience is a StringOrURI (see [I-D.ietf-oauth-json-web-token] Section 2) containing the target service's principal name, formatted according to Section 3.1.1. This claim is REQUIRED. If the initiator specified a target name of GSS_C_NO_NAME, then the audience is the empty string.

[[anchor6: If the initiator wanted mutual authentication then we could find out the acceptor's name and provide it via GSS_Inquire_context(). This is only really useful and secure with mechanisms like this one where the initiator credential is based on a public/private key pair and either we use key agreement and per-message tokens or channel binding to a secure channel. This really should [have] be[en] explained in RFC2743.]]

6.1.2. "exp" (Expiry time)

This contains the time when the assertion expires, in milliseconds since January 1, 1970. At least one of "exp" or "iat" MUST be present.

6.1.3. "iat" (Issued at time)

This contains the time the assertion was issued (in milliseconds since January 1, 1970). If present, the acceptor MUST validate that the assertion was recently issued. At least one of "exp" or "iat" MUST be present.

6.1.4. "nbf" (Not before time)

This contains the time, in milliseconds since January 1, 1970, from which the assertion begins to be valid. This claim is OPTIONAL.

6.1.5. "epk" (Ephemeral Public Key)

These contain key parameters for deriving a shared session key with the relying party, represented as a JSON Web Key [I-D.ietf-jose-json-web-key] public key value. The key type MUST be EC and the parameters for Elliptic Curve Public Keys specified in [I-D.ietf-jose-json-web-algorithms] Section 6.2.1 MUST be present.

The "epk" claim is REQUIRED unless the associated encryption type is ENCTYPE_NULL, or there is already a prior session key (as is the case for re-authentication assertions).

6.1.6. "cb" (Channel binding)

This contains channel binding information for binding the GSS context to an outer channel (e.g. see [RFC5929]). Its value is the base64 URL encoding of the application-specific data component of the channel bindings passed to GSS_Init_sec_context() or GSS_Accept_sec_context(). This claim is OPTIONAL.

6.1.7. "nonce" (Mutual authentication nonce)

This is a random quantity of at least 64 bits, base 64 URL encoded, which is used to bind the request and response assertions in the case a freshly agreed key is not used to sign the response assertion. This claim is REQUIRED if mutual authentication is desired and the assertion is signed using a certificate, or if re-authentication is being performed.

6.1.8. "tkt" (Ticket)

When the assertion is being used for fast re-authentication, this contains a JSON object with a single parameter, "tid". The "tid" parameter matches the "tid" parameter from the initial response assertion ticket (see Section 6.2.5). This claim is REQUIRED for re-authentication assertions, otherwise it the assertion MUST be rejected. Other parameters SHOULD NOT be present in the "tkt" object.

6.1.9. "opts" (Options)

This contains a JSON array of string values indicating various protocol options that are supported by the initiator. Unknown options MUST be ignored by the acceptor. This document defines the following extensions:

Name	Description
ma	The initiator requested GSS_C_MUTUAL_FLAG
xrt	The initiator supports the extra round trip option (see Section 4.4)
dce	The initiator requested GSS_C_DCE_STYLE (see RFC4757 Section 7.1)
ify	The initiator requested GSS_C_IDENTIFY_FLAG (see RFC4757 Section 7.1)

6.2. Response (acceptor/RP) assertion

The response assertion is sent from the acceptor to the initiator to provide key agreement, and either key confirmation or mutual authentication. It is formatted as a backed assertion, however in the current specification it consists of a single assertion with zero certificates; that is, it is "unbacked". (It is encoded as a backed assertion in order to provide future support for mutual authentication using native BrowserID certificates. Such support is not specified here.)

In the case of a key successfully being negotiated, the response assertion is signed with the RP Response Key (RRK) (see Section 7.3). Alternatively, it may be signed with the acceptor's private RSA or DSA key. In this case, the acceptor's X.509 certificate is included in the "x5c" claim of the JWT header.

The HMAC-SHA256 (HS256) algorithm MUST be supported by implementors of this specification.

If the [RFC3961] encryption type for the mechanism is ENCTYPE_NULL, then the signature is absent and the value of the "alg" header parameter is "none". No signature verification is required in this case.

Claims not specified here MUST be ignored by the initiator.

Here is an example response assertion:


```
{
    "exp": 1362960258000,
    "nonce": "bbqT10Gyx3s",
    "epk": {
        "x": "bvNF6V1rpMeQyGOKCj0kBaOaSh3tlhUcbffaji4uCEI",
        "y": "Iuqs650FXzXFUD9kHknETfbqiB8XBbCHlJXoysx3rvw"
    },
    "tkt": {
        "tid": "Jgg7vKX2sEKlCWBfmLTg_n4qz3NVZxOU-a2B4qYMkXI",
        "exp": 1362992660000
    }
}
```


The following claims are permitted in the response assertion:

6.2.1. "iat" (Issued at time)

The current acceptor time, in milliseconds since January 1, 1970. This allows the initiator to compensate for clock differences when generating assertions. This claim is OPTIONAL.

6.2.2. "epk" (Ephemeral Public Key)

This contains a JSON object containing the x and y coordinates of the acceptor's ECDH public key (see [I-D.ietf-jose-json-web-algorithms] Section 6.2.1). This claim is REQUIRED unless the associated encryption type is ENCTYPE_NULL, or there is already an established session key, as is the case for re-authentication assertions.

The "crv" and "kty" properties SHOULD NOT be present; they are determined by the initiator.

6.2.3. "exp" (Expiry time)

This contains the time when the context expires, in milliseconds since January 1, 1970. This claim is OPTIONAL; the initiator should

use the certificate or ticket expiry time if absent.

6.2.4. "nonce" (Mutual authentication nonce)

The nonce as received from the initiator. This MUST NOT be present unless a nonce was received from the initiator, and the acceptor is signing the assertion with a private key.

6.2.5. "tkt" (Ticket)

This contains a JSON object that may be used for re-authenticating to the acceptor without acquiring an assertion. It has two parameters: "tid", an opaque identifier to be presented in a re-authentication assertion (this need not be a string); and "exp", the expiry time of the ticket. This claim is OPTIONAL.

6.2.6. "jti" (JWT ID)

This contains a base64 URL encoded random value of at least 64 bits that is used to uniquely identify the acceptor response, in the case that the extra round trip option is used. It SHOULD not be present unless the initiator requested the extra round trip option.

6.3. Error (acceptor/RP) assertion

Error assertions are backed assertions containing any or all of the following claims. In addition, they MUST have the "iat" claim, for initiator clock skew correction. All other response assertion claims are OPTIONAL or not applicable in error assertions. Conversely, the claims listed below MUST NOT be present in a non-error response assertion.

The error assertion MAY be signed if a key is available, otherwise the signature is absent and the value of the "alg" header parameter is "none".

6.3.1. "gss-maj" (GSS major status code)

This contains a GSS major status code represented as a number.

6.3.2. "gss-min" (GSS minor status code)

This contains a GSS minor status code represented as a number.

If REAUTH_FAILED is received, the initiator SHOULD attempt to send another initial context token containing a fresh assertion.

The following protocol minor status codes are defined. Note that the

API representation of these status codes is implementation dependent. Status codes with the high bit set are GSS BrowserID protocol errors; the remainder are BrowserID protocol errors.

Error	Protocol	Description
INVALID_JSON	8	Invalid JSON encoding
INVALID_BASE64	9	Invalid Base64 encoding
INVALID_ASSERTION	10	Invalid assertion
TOO_MANY_CERTS	13	Too many certificates
UNTRUSTED_ISSUER	14	Untrusted issuer
INVALID_ISSUER	15	Invalid issuer
MISSING_ISSUER	16	Missing issuer
MISSING_AUDIENCE	17	Missing audience
BAD_AUDIENCE	18	Bad audience
EXPIRED_ASSERTION	19	Assertion expired
ASSERTION_NOT_YET_VALID	20	Assertion not yet valid
EXPIRED_CERT	21	Certificate expired
CERT_NOT_YET_VALID	22	Certificate not yet valid
INVALID_SIGNATURE	23	Invalid signature
MISSING_ALGORITHM	24	Missing JWS algorithm
UNKNOWN_ALGORITHM	25	Unknown JWS algorithm
MISSING_PRINCIPAL	34	Missing principal attribute
UNKNOWN_PRINCIPAL_TYPE	35	Unknown principal type
MISSING_CERT	36	Missing certificate
MISSING_CHANNEL_BINDINGS	38	Missing channel bindings

CHANNEL_BINDINGS_MISMATCH	39	Channel bindings do not match
NOT_REAUTH_ASSERTION	70	Not a re-authentication assertion
BAD_SUBJECT	71	Bad subject name
MISMATCHED_RP_RESPONSE	72	Mismatched RP response token
REFLECTED_RP_RESPOSNE	73	Reflected RP response token
UNKNOWN_EC_CURVE	77	Unknown ECC curve
INVALID_EC_CURVE	78	Invalid ECC curve
MISSING_NONCE	79	Missing nonce
WRONG_SIZE	0x80000001	Buffer is incorrect size
WRONG_MECH	0x80000002	Mechanism OID is incorrect
BAD_TOK_HEADER	0x80000003	Token header is malformed or corrupt
TOK_TRUNC	0x80000004	Token is missing data
BAD_DIRECTION	0x80000005	Packet was replayed in wrong direction
WRONG_TOK_ID	0x80000006	Received token ID does not match expected
KEY_UNAVAILABLE	0x80000007	Key unavailable
KEY_TOO_SHORT	0x80000008	Key too weak
CONTEXT_ESTABLISHED	0x80000009	Context already established
CONTEXT_INCOMPLETE	0x8000000A	Context incomplete
BAD_CONTEXT_TOKEN	0x8000000B	Context token malformed or corrupt

BAD_ERROR_TOKEN	0x8000000C	Error token malformed or corrupt
BAD_CONTEXT_OPTION	0x8000000D	Bad context option
REAUTH_FAILED	0x8000000E	Re-authentication failure

6.4. XRT assertion

No claims are presently defined for the extra round trip assertion. Unknown claims MUST be ignored by the acceptor.

7. Key derivation

The following function is used as the base algorithm for deriving keys:

browserid-derive-key(K, usage) = HMAC(K, "BrowserID" || K || usage || 0x01)

The HMAC hash algorithm for all currently specified key lengths is SHA-256. Note that the inclusion of K in the HMAC input is for interoperability with some crypto implementations.

7.1. Diffie-Hellman Key (DHK)

This key is the shared secret resulting from the ECDH exchange. Its length corresponds to the selected EC curve. It is never used without derivation and thus may be used with implementations that do not expose the ECDH value directly.

7.2. Context Master Key (CMK)

This is the Diffie-Hellman Key (DHK) for all initially authenticated contexts and the Authenticator Session Key (ASK) for re-authenticated contexts.

7.3. RP Response Key (RRK)

If mutual authentication without a fast re-authentication ticket is performed then the response assertion will be signed with a public key signature using the private key for the acceptor's certificate.

Otherwise a symmetric RP Response Key (RRK) is derived as follows:

RRK = browserid-derive-key(CMK, "RRK")

7.4. Context Root Key (CRK)

The Context Root Key (CRK) is used for [RFC4121] message protection services, e.g. GSS_Wrap() and GSS_Get_MIC(). If the extra round-trip option is in effect, it is derived as follows:

CRK = random-to-key(browserid-derive-key(XRTK, "CRK"))

Otherwise, the CMK is used:

CRK = random-to-key(browserid-derive-key(CMK, "CRK"))

The random-to-key function is defined in [RFC3961].

7.5. Authenticator Root Key (ARK)

The Authenticator Root Key (ARK) is used to sign assertions used for fast re-authentication. (The term "authenticator" is equivalent to "re-authentication assertion" and exists for historical reasons.) It is derived as follows:

```
ARK = browserid-derive-key(CMK, "ARK")
```

7.6. Authenticator Session Key (ASK)

The Authenticator Session Key (ASK) is used instead of the DHK for re-authenticated contexts. It is derived as follows:

```
ASK = browserid-derive-key(ARK, nonce-binary)
```

The usage (nonce-binary) is the base64 URL decoding of the initiator "nonce" claim.

7.6.1. Extra Round Trip Key (XRTK)

The Extra Round Trip Key (XRTK) is used to sign the extra round trip token, and also as the master key for the CRK when the extra round trip option is used.

```
XRTK = browserid-derive-key(CMK, acceptor-jti-binary)
```

The usage (acceptor-jti-binary) is the base64 URL decoding of the acceptor "jti" claim.

7.7. GSS Pseudo-Random Function (PRF)

The BrowserID mechanism shares the same Pseudo-Random Function (PRF) as the Kerberos GSS mechanism, defined in [RFC4402]. GSS_C_PRF_KEY_FULL and GSS_C_PRF_KEY_PARTIAL are equivalent. The protocol key to be used for GSS_Pseudo_random() SHALL by the Context Root Key (CRK).

[[anchor7: Can we replace this with a function that imports less of RFC3962? We arguably should, because otherwise the only things we import from RFC3962 (and 3961) are random-to-key (the identity function in RFC3962) and the crypto bits needed for RFC4121 per-message tokens.]]

8. Example

Suppose a mail user agent for the principal lukeh@lukktone.com wishes to authenticate to an IMAP server rand.mit.de.padl.com. They do not have a re-authentication ticket. The mail user agent would display a dialog box in which the principal would sign in to their IdP and request a fresh assertion be generated.

```
C: <connects to IMAP port>
S: * OK
C: C1 CAPABILITY
S: * CAPABILITY IMAP4rev1 SASL-IR SORT [...] AUTH=BROWSERID-AES128
S: C1 OK Capability Completed
C: C2 AUTHENTICATE BROWSERID-AES128
```


biwsYyxleUpoYkdjaU9pSlNVekkxTmlKOS5leUp3ZFdKc2FXTXRhMlY
1SWpwN0ltRnNaMjl5YVhSb2JTSTZJa1JUSWl3aWVTSTZJak01TVRObE
9EZ3laRGhqTXpWa01qSm1ObVEwTURZNVkyVTJNREJrWW1OallqTTVOR
0ZqWVdGaFl6WTBPV1prTjJZNVptTmtObU0wTVRJME5tWTFOakk1TUdW
bU1HTmpNemMwTnpaaE1EUmhOREU0WXpGbE9ETXhPV0kxTkdJeFpXTml
ObVkyWTJWaE56VTBOR1kyWlRFMU5qTmxaR05sWkdNNU1EWmtOamcwTT
JRd01XSmpaVFJtTjJFMVpqY3dOMk5tWVRZd1lXTTVNVE0yWm1GbU5qS
m1aR0ZtTkRoa09HRTVPRGxoWVdGbE5EUXdOMlZrTmpjeU56ZGhNVGM0
TW1WallXRXhOVFppWkdOaFpXRXhOamRtTWpZek56STFaR1UyTTJWa09
HWXlPR0UyTUROaU5tWm1OVEV3WmpRNE1ESmtOelJrTjJWaFpUZGhZbU
15WldJaUxDSndJam9pWm1ZMk1EQTBPRE5rWWpaaFltWmpOV0kwTldWa
FlqYzROVGswWWpNMU16TmtOVFV3WkRsbU1XSm1NbUU1T1RKaE4yRTRa
R0ZoTm1Sak16Um1PREEwTldGa05HVTJaVEJqTkRJNVpETXpOR1ZsWld
GaFpXWmtOMlV5TTJRME9ERXdZbVV3TUdVMFkyTXhORGt5WTJKaE16ST
FZbUU0TVdabU1tUTFZVFZpTXpBMVlUaGtNVGRsWWpOaVpqUmhNRFpoT
XpRNVpETTVNbVV3TUdRek1qazNORFJoTlRFM09UTTRNRE0wTkdVNE1t
RXhPR00wTnprek16UXpPR1k0T1RGbE1qSmhaV1ZtT0RFeVpEWTVZemh
tTnpWbE16STJZMkkzTUdWaE1EQXdZek5tTnpjMlpHWmtZbVEyTURRMk
16aGpNbVZtTnpFM1ptTXlObVF3TW1VeE55SXNJbkVpT2lKbE1qRmxNR
FJtT1RFeFpERmxaRGM1T1RFd01EaGxZMkZoWkZoWWpOaVpqYzNOVGs0TkRN
d09XTXpJaXdpWnlJNkltTTFNbUZNbUwwWVRCbVpqTmlOMlUyTVdaa1pqRTR
OamRqWlRnME1ERTUUTTRNelk1WVRZeE5UUm1OR0ZtWVRreU9UWXhVWTJaVE5qT0
RJM1pUSTFZMlpoTm1ObU5UQTRZamt3WlRWa1pUUXhPV1V4TXpNM1pUQ
TNZVEpsT1dWeWlUTmpaRFZrWldGTM01EUmtNVkdMxWmpobFltWTJZV1l6
T1Rka05qbGxNVEV3V1dwrMllXWmlNVkdRqTjJGd016STFPVE15T1dVME9
ESTVZakJrTUROaVltTTNPRGsyWWpFMVlqUmhaR1UxTTJWeE16QTROVG
hqWXpNMFpEazJNalk1WVdFNE9UQTBNV1kwTURreE16WmpOekkwTW1Fe
k9EZzVOV001WkRWaVkyTmhaRFJtTXpnNVlXWXhaRGRoTkdKa01UTTVP
R0prTURjeVpHWm1ZVGc1TmpJek16TTVOMkVpZlN3aWNISnBibU5wY0d
Gc0lqcDdJbVZ0WVdsc0lqb2liSFZyWldoQWJIVnJHM1J2Ym1VdVkyOX
RJbjBzSW1saGRDSTZNVE0yTWprMk1UQTVOakV5TWl3aVpYaHdJam94T
XpZeU9UWTBOamsyTVRJeUxDSnBjM01pT2lKc2IyZHBiaTV3WlhKemIy
NWhMbTl5WnlKOS5mT3V5ZlZkNWFZZ285ckJncmdHVDJHYjkzUUoxVnp
LSE9rNjdFUXBEeU9pUENPdXFweUw5a2tVVDdxcGNZaWZsb0NTWjlPej

UtVWRrcldlcTZXUkRLcUdOeXg0OFdyVGduVkoyRlM3MU1Mbl9DeWhGM
Go1Y1ZsQ0E5WWh3YVlWTHhsbW9YU01uWTdyRzFWa0VSdjRtaWtCM3FD
cFB2NXJtSEswbkNiRlpiN1dXR3JkVEdkcmNHTkRkZHlDQkQ5a1dpUUd
VbkktenN3WXdiZXJUTmQ3Nmc1Z2N1c1MtbWxjVk5jbzNMTG4zMlNhbG
x0eDBCUHAtVTAyMXpvR00wWEhibm1Sa2VRdGVtblVXZGloYzRVbVpNR
EJJZ05nSFFCSmdXMGhBcTlHWVFmYzVObFNzZW5RX0p5MGR4anE1bHdE
Wll3SExsUXlmYnVYbGFtRTNDZ3ZkZUF+ZXlKaGJHY2lPaUpFVXpFeU9
DSjkuZXlKdWIyNWpaU0k2SW1nMVVEUkxja2M0ZVc1bklpd2laV05rYU
NJNmV5SjRJam9pWm1wYVRuQnpRbXBIYmw5WVFVTnRaMkpPZDBGemRuS
TRPR2MwUmxkNmRHOWljWEExVkUxaVgxbEdNQ0lzSW1OeWRpSTZJbEF0
TWpVMklpd2llU0k2SWxKTFJYWktlalU1WTNOaGRqaExZM2RsVlhZMVd
IRkdaM1E0UVZkRFFXdHlTa0o2TTFCUWNVeEtkSE1pZlN3aVkySjBJam
9pWW1sM2N5SXNJbVY0Y0NJNk1UTTJNamsyTVRJeE5qRTBPU3dpWVhWa
0lqb2lkWEp1T25ndFozTnpPbWx0WVhBdmNtRnVaQzV0YVhRdVpHVXVj
R0ZrYkM1amIyMGlmUS51ZHRvSTNVNUMtM3BwNHhJSloxbWstQ3o0Ymh
sQkxlSzAyNlVhbWRhMjhwTFk4c013Tk50Y0E=
S: + Qyx
+ZXlKaGJHY2lPaUpTVXpJMU5pSXNJbmcxWXlJNld5Sk5TVWxFZW1wRF
EwRnlZV2RCZDBsQ1FXZEpRa0o2UVU1Q1oydHhhR3RwUnpsM01FSkJVV
lZHUVVSQ1pFMVJjM2REVVZzRVZsRlJSMFYzU2tKV1ZFVmxUVUozUjBF
eFZVVkRaM2RXVlVWR1JWUkRRbFJpTWxvd1pESkdlVnBUUWxGa1NHdG5
WRWhTYTAxVE5IZE1RVmxFVmxGUUlJFUkRWbEZSVlZKTlNVWk9kbHB1VW
pOWldFcHNHNTVVZPYkdOdVVuQmFiV2hxV1ZoU1NHSXlOR2RSSV0ZZd1lVY
zVlV0ZZVWpWTlFqUllSlJGZWsxRVJYaE5WRUV4VFhwUmVVMUdiMWhF
VkVVeVRVUkZlRTFVUVRGTmVsRjVVVVp2ZDFSRVJVeE5RV3RIUUVRGVlJ
VSm9VVU5SVmxWNFNHcEJZMEpuTlZOVGxxQ1FXOU5SbFpDUWxKRmQYZFZNam
x0WkVoa2FHTnVSWGlUZEVWTlVkdFRtRlhaRWpWbVZrZEhRVm9wa0dSRlZ
2QxVmpiVVoxV2tNMWRHRllRbllUZFZLa0dkRXRkbGJVE5UJZN0NDRzFMaTh
MmRGYVVxQk1FZERVM0ZJVTBsZW0FVTBTkJuUUZZUmFhWk9yeVhZbVN1WT
kbG93VkxVRVYlKUWtGUlJJFSm9sa1p3Wmt3MmRRkRjM2d5UkhaUldsdQX
JSMUl3VlNc5dFJIQXZRMFZzSzA5SVRqQmFOR0l0NFRGV2VtWMVHT
lROdVVIRldWR0pkhGWVNEaEJJWWFZWmpkTm1WdE1WlpOamh2Vm14NHd1ZHN
UXdOWHB0TW1NdmRFVXpaMnRvVkhodFdGOVNaMUZ5WTNNWTVVWHozVnFUMWh
OUm1oSk5ITjJSVm9yUTJKSVFFGVm92VkhhbVJmRVRlRUVkcEFGSk1UbGhvVnQ2
QwNWFiakZJTlRkKQlRIRnRVMEZ2TlZ6RlRSnNZYTFtZUEZHRWRGWWFSWWRSNlFHNXXZR
kJTZWtSMFZXOFVFaV0N6TXpaWi0bFZhVlJYR3hvZDBXWmNnUmtoaGRVMzRXV1cx
aVZVSnFSaDZTZFQxV1dWOdE1FcZzVkJGMmNBVlNtRldFVlNhOBFTYkZDcnNlVkNZ0hN5Yk0oa
hMDltTjFwcVdqbDBjRjlKUlJyMhUxlEOXNlgTlNWUlVSVFYMWlVUnYWiMFhDJHNKVHNm1mMqJTh5aV
FjM05YR1lRlJSMUJCUmtrRVNmVNFRlhSVjZWjUpNQ2ZwDhNMUxJNHMVVkdGNYMJKVldZMUtOYVdrMWdj

```
Uk5RVEJIUTFOeFIxTkpZak5FVVVWQ1FsRlZRVUUwU1VKQlVVSkVNVUo
2VVZBcmNrNHhWVlY2TjBFMmVpdExSRkJoY1Roek1tbENSekJHZWxwNG
MxZ3lVVlZQZFhCQ1JVbGlkVnB3TUV0S1lYVnFWazFuTURGbVpHcHpkV
WRITUhWWVlrMW1aVkpIZVU1c1ZYTk5UaXRhUkhrNEwwMUpUMmd4WVZW
SGRqQlRWWGRMZEVOMFRIUlhja3AyTmpWMWQwaEhSM1ExZFVaTGVFMUZ
OakZXVkRRcmNYQkpNa0ZIY1hoNE5XUnljM2hGVEVKUFpIbFFibVYxUV
dsTVVIaEdkV0pTUm0xNmRXaFdVMGszUVZCTmJEYzVUMnN6TUc5WGRXU
kJORGxzVlZnNWQzb3paemx4T1haa2JEbDVhR2RsWlZWVFZYQk5hR3hh
TWpSVll6bFFkVXg2Y2pFMWFqWjJOak5ZZW5KVFpGZDBUbnAyTUVZeE1
HVkViRFI1VkZWT1YxTkthRGR4UW1obmNURkpiMWc1UVZCUFQzVk1Zaz
FPY25BMlltVkZaVVkzYURNMGNGWlhhbFJoVTNoSk4yNUxOVGRyU3pKN
GFGSlZORE5sZDFscU1ta3ZVM0o2T0VkelRWTTVNWFZ5TWpWSmRDSmRm
US5leUowYTNRaU9uc2lhblJwSWpvaVlXVmhlVEJIU21sNlJJZzNPVUZ
uTFMxWFRDMTJkelpaT1VKWWVGSjFRekZZYzFwNGNuazFNVk5WU1NJc0
ltVjRjQ0k2TVRNMk1qazVOekE1T0RBd01IMHNJbVZqWkdnaU9uc2llQ
0k2SWtveFNWZGlTREpCTlVNelkyaFBWVWx4YldaWWNGQmZVbEZGUlU5
dFpESkZlRmh2UzNKeFVWRllURTBpTENKNUlqb2lYekpGZEhoaWVsOTJ
TbVZsVlZWaWVUSnlabVJsYTFSVVVGVlNjR0pIU2tnM2EzbEpWM0Z0YT
BsRlp5SjlMQ0p1YjI1alpTSTZJbWcxVURSTGNrYzRlVzVuSWl3aVpYa
HdJam94TXpZeU9UWTBOamsyTURBd2ZRLnFaaFVxdXBWUHgzRTdNSTBH
dnNIZjZER3pzc3ByMkJsdUVUMFNwMERxdkpFS1F4S3BiOG9faVZsWHZ
Qa2p2SXp0Qm5JajNNb084UlZMUWJwdE9QZDFrN3FoTUVwRkhOVGI1WF
pKYWVJTlBpQUNSSzA5dUZpVE5ud1cxanMxQ3pPY2FMakxsSTN4bFdkL
Ul1em8zODhyTUxsSXVkbmkxak5uRS0yOXZfc1NUTnRxLUMwQmNoNUMw
T3drbDcxQk54eHgzaFVxeEcxT0w0UHQyZ0JKWUFQX3NOVk12aDFwWDl
hRzd0Vms0S2sxS2NjaXRqUFdGN0dXc3JGeld4ekRSMHU2REZ0RmFjaE
NPYmVmcmZnZkUxOXFlWnJLcnpJMFVkQ3JEUHpZazlYb1dKR2twRlNPd
1dhY192Q0N1dXY1VjNHZF9MTlNJM3JCaS1GYWVoWUhBRjFJUQ==
```


Unpacking the mail user agent's AUTHENTICATE message reveals the following:

```
n,,c,eyJhbGciOiJSUzI1NiJ9.eyJwdWJsaWMta2V5Ijp7ImFsZ29yaXRob
SI6IkRTIiwieSI6IjM5MTNlODgyZDhjMzVkMjJmNmQ0MDY5Y2U2MDBkYmNj
YjM5NGFjYWFhYzY0OWZkN2Y5ZmNkNmM0MTI0NmY1NjI5MGVmMGNjMzc0NzZ
hMDRhNDE4YzFlODMxOWI1NGIxZWNiNmY2Y2VhNzU0NGY2ZTE1NjNlZGNlZG
M5MDZkNjg0M2QwMWJjZTRmN2E1ZjcwN2NmYTYwYWM5MTM2ZmFmNjJmZGFmN
DhkOGE5ODlhYWFlNDQwN2VkNjcyNzdhMTc4MmVjYWExNTZiZGNhZWExNjdm
MjYzNzI1ZGU2M2VkOGYyOGE2MDNiNmZmNTEwZjQ4MDJkNzRkN2VhZTdhYmM
yZWIiLCJwIjoiZmY2MDA0ODNkYjZhYmZjNWI0NWVhYjc4NTk0YjM1MzNkNT
UwZDlmMWJmMmE5OTJhN2E4ZGFhNmRjMzRmODA0NWFkNGU2ZTBjNDI5ZDMzN
GVlZWFhZWZkN2UyM2Q0ODEwYmUwMGU0Y2MxNDkyY2JhMzI1YmE4MWZmMmQ1
YTViMzA1YThkMTdlYjNiZjRhMDZhMzQ5ZDM5MmUwMGQzMjk3NDRhNTE3OTM
4MDM0NGU4MmExOGM0NzkzMzQzOGY4OTFlMjJhZWVmODEyZDY5YzhmNzVlMz
I2Y2I3MGVhMDAwYzNmNzc2ZGZkYmQ2MDQ2MzhjMmVmNzE3ZmMyNmQwMmUxN
yIsInEiOiJlMjFlMDRmOTExZDFlZDc5OTEwMDhlY2FhYjNiZjc3NTk4NDMw
OWMzIiwiZyI6ImM1MmE0YTBmZjNiN2U2MWZkZjE4NjdjZTg0MTM4MzY5YTY
xNTRmNGFmYTkyOTY2ZTNjODI3ZTI1Y2ZhNmNmNTA4YjkwZTVkZTQxOWUxMz
M3ZTA3YTJlOWUyYTNjZDVkZWE3MDRkMTc1ZjhlYmY2YWYzOTdkNjllMTEwY
jk2YWZiMTdjN2EwMzI1OTMyOWU0ODI5YjBkMDNiYmM3ODk2YjE1YjRhZGU1
M2UxMzA4NThjYzM0ZDk2MjY5YWE4OTA0MWY0MDkxMzZjNzI0MmEzODg5NWM
5ZDViY2NhZDRmMzg5YWYxZDdhNGJkMTM5OGJkMDcyZGZmYTg5NjIzMzM5N2
EifSwicHJpbmNpcGFsIjp7ImVtYWlsIjoibHVrZWhAbHVra3RvbmUuY29tI
n0sImlhdCI6MTM2Mjk2MTA5NjEyMiwiZXhwIjoxMzYyOTY0Njk2MTIyLCJp
c3MiOiJsb2dpbi5wZXJzb25hLm9yZyJ9.fOuyfVd5aYgo9rBgrgGT2Gb93Q
J1VzKHOk67EQpDyOiPCOuqpyL9kkUT7qpcYifloCSZ9Oz5-UdkrWeq6WRDK
qGNyx48WrTgnVJ2FS71MLn_CyhF0j5cVlCA9YhwaYVLxlmoXSMnY7rG1VkE
Rv4mikB3qCpPv5rmHK0nCbFZb7WWGrdTGdrcGNDddyCBD9kWiQGUnI-zswY
wberTNd76g5gcusS-mlcVNco3LLn32Salltx0BPp-U021zoGM0XHbnmRkeQ
temnUWdihc4UmZMDBIgNgHQBJgW0hAq9GYQfc5NlSsenQ_Jy0dxjq5lwDZY
wHLlQyfbuXlamE3CgvdeA~eyJhbGciOiJEUzEyOCJ9.eyJub25jZSI6Img1
UDRLckc4eW5nIiwiZWNkaCI6eyJ4IjoiZmpaTnBzQmpHbl9YQUNtZ2JOd0F
zdnI4OGc0Rld6dG9icXA1VE1iX1lGMCIsImNydiI6IlAtMjU2IiwieSI6Il
JLRXZKejU5Y3NhdjhLY3dlVXY1WHFGZ3Q4QVdDQWtySkJ6M1BQcUxKdHMif
SwiY2J0IjoiYml3cyIsImV4cCI6MTM2Mjk2MTIxNjE0OSwiYXVkIjoidXJu
OngtZ3NzOmltYXAvcmFuZC5taXQuZGUucGFkbC5jb20ifQ.udtoI3U5C-3p
p4xIJZ1mk-Cz4bhlBLeK026Uamda28pLY8sMwNNtcA
```


The initial "n,," is the GS2 header (indicating that there are no channel bindings). The "c," denotes the token as being a BrowserID initial context token. The remaining base64 URL encoded data is a BrowserID backed assertion, containing the following certificate (for clarity, the payload has been reformatted and JWT header and signature omitted):

```
{
    "public-key": {
        "algorithm": "DS",
        "y": "3913e882d8c35d22f6d4069ce600dbccb394acaaac649
              fd7f9fcd6c41246f56290ef0cc37476a04a418c1e8319
              b54b1ecb6f6cea7544f6e1563edcedc906d6843d01bce
              4f7a5f707cfa60ac9136faf62fdaf48d8a989aaae4407
              ed67277a1782ecaa156bdcaea167f263725de63ed8f28
              a603b6ff510f4802d74d7eae7abc2eb",
        "p": "ff600483db6abfc5b45eab78594b3533d550d9f1bf2a9
              92a7a8daa6dc34f8045ad4e6e0c429d334eeeaaefd7e2
              3d4810be00e4cc1492cba325ba81ff2d5a5b305a8d17e
              b3bf4a06a349d392e00d329744a5179380344e82a18c4
              7933438f891e22aeef812d69c8f75e326cb70ea000c3f
              776dfdbd604638c2ef717fc26d02e17",
        "q": "e21e04f911d1ed7991008ecaab3bf775984309c3",
        "g": "c52a4a0ff3b7e61fdf1867ce84138369a6154f4afa929
              66e3c827e25cfa6cf508b90e5de419e1337e07a2e9e2a
              3cd5dea704d175f8ebf6af397d69e110b96afb17c7a03
              259329e4829b0d03bbc7896b15b4ade53e130858cc34d
              96269aa89041f409136c7242a38895c9d5bccad4f389a
              f1d7a4bd1398bd072dffa896233397a"
    },
    "principal": {
        "email": "lukeh@lukktone.com"
    },
    "iat": 1362961096122,
    "exp": 1362964696122,
    "iss": "login.persona.org"
}
```


and assertion:


```
{
    "nonce": "h5P4KrG8yng",
    "epk": {
        "x": "fjZNpsBjGn_XACmgbNwAsvr88g4FWztobqp5TMb_YF0",
        "crv": "P-256",
        "kty": "EC",
        "y": "RKEvJz59csav8KcweUv5XqFgt8AWCAkrJBz3PPqLJts"
    },
    "cb": "biws",
    "exp": 1362961216149,
    "aud": "imap/rand.mit.de.padl.com"
}
```


Note the channel binding token that protects the GS2 header.

[[anchor8: The encoded example needs to be regenerated to reflect that "cb" is now used for channel bindings.]]

Turning to the response backed assertion sent from the IMAP server to the mail user agent, we have the following after base64 decoding:


```
eyJhbGciOiJSUzI1NiIsIng1YyI6WyJNSUlEempDQ0FyYWdBd0lCQWdJQkJ
6QU5CZ2txaGtpRzl3MEJBUVVGQURCZE1Rc3dDUVlEVlFRR0V3SkJWVEVlTU
J3R0ExVUVDZ3dWVUVGRVRDQlRiMlowZDJGeVpTQlFkSGtnVEhSa01TNHdMQ
VlEVlFRRERDVlFRVVJNSUZOdlpuUjNZWEpsSUVObGNuUnBabWxqWVhScGIy
NGdRWFYwYUc5eWFYUjVNQjRYRFRFek1ERXhNVEExTXpReU1Gb1hEVEUyTUR
FeE1UQTFNelF5TUZvd1RERUxNQWtHQTFVRUJoTUNRVlV4SGpBY0JnTlZCQW
9NRlZCQlJFd2dVMjltZEhkaGNtVWdVSFI1SUV4MFpERWRNQnNHQTFVRUF3d
1VjbUZ1WkM1dGFYUXVaR1V1Y0dGa2JDNWpiMjB3Z2dFaU1BMEdDU3FHU0li
M0RRRUJBUVVBQTRJQkR3QXdnZ0VLQW9JQkFRREJoekZwZkw2dkh4c3gyRHZ
GWlArR1IwVW9tRHAvQ0VlK09ITjBaMGM2OTFZZzlnV1htVTNuUHFWVGJBSX
FYSDhBVXR2ZjdNemJZMjhvVmxwWlQwNXptMmMvdEUzZ2toVHhtWE9SZ1FyY
3V1Z3VqT1hNRmhJNHN2RVorQ2JIUGxaaVovVHprWExIUDI5RXo3d05abjFI
NTdBTHFtU0FvNVQ0cXhNRmdCWXVkdy9aeFBSekR0VW9JVjBzMjNZZzR4VDl
hd0pucjFHZ01VUmliVUJqRjd5YmNtMEs4c0pUK1VHZUI3cm1MbFB3K2ZBa0
9mN1pqWjl0cFRrRU1pOHVMRU1xY3hhR1NBSy8ra1c3NXFPeGRBRkk4ellaW
DUzZ3BnNG1pK1FXZkdZMVpOUUpNdUhHUVhnL3VmeE16YXhOTjRoMWFPbG1a
WllrQkhwNTJBOXlJTVViQWdNQkFBR2pnYWt3Z2FZd0NRWURWUjBUQkFJd0F
EQXNCZ2xnaGtnQmh2aENBUTBFSHhZZFQzQmxibE5UVENCSFpXNWxjbUYwWl
dRZ1EyVnlkR2xtYVdOaGRHVXdIUVlEVlIwT0JCWUVGS1NzdWJFRHViUklHS
ENCdHRBbFR2MkZXR2YrTUI4R0ExVWRJd1FZTUJhQUZMaXpabE1XbktLMVBZ
YWdkSmprVnVSaEVRSmpNQWtHQTFVZEVRUUNNQUF3Q3dZRFZSMFBCQVFEQWd
YZ01CTUdBMVVkSlFRTU1Bb0dDQ3NHQVFVRkJ3TURNQTBHQ1NxR1NJYjNEUU
VCQlFVQUE0SUJBUUJEMUJ6UVArck4xVVV6N0E2eitLRFBhcThzMmlCRzBGe
lp4c1gyUVVPdXBCRUlidVpwMEtKYXVqVk1nMDFmZGpzdUdHMHVYYk1mZVJH
eU5sVXNNTitaRHk4L01JT2gxYVVHdjBTVXdLdEN0THRXckp2NjV1d0hHR3Q
1dUZLeE1FNjFWVDQrcXBJMkFHcXh4NWRyc3hFTEJPZHlQbmV1QWlMUHhGdW
JSRm16dWhWU0k3QVBNbDc5T2szMG9XdWRBNDlsVVg5d3ozZzlxOXZkbDl5a
GdlZVVTVXBNaGxaMjRVYzlQdUx6cjE1ajZ2NjNYenJTZFd0Tnp2MEYxMGVE
bDR5VFVOV1NKaDdxQmhncTFJb1g5QVBPT3VMYk1OcnA2YmVFZW93aDM0cFZ
XZlRhU3hJN25LNTdrSzJ4aFJVNDNld1lqMmkvU3J6OEdzTVM5MXVyMjVJdC
JdfQ.eyJ0a3QiOnsianRpIjoiYWVheTBHSml6RHg3OUFnLS1XTC12dzZZOU
JYeFJ1QzFYc1p4cnk1MVNVSSIsImV4cCI6MTM2Mjk5NzA5ODAwMH0sImVjZ
GgiOnsieCI6IkoxSVdiSDJBNUMzY2hPVUlxbWZYcFBfUlFFRU9tZDJFeFhv
S3JxUVFYTE0iLCJ5IjoiXzJFdHhiel92SmVlVVVieTJyZmRla1RUUFVScGJ
HSkg3a3lJV3Fta0lFZyJ9LCJub25jZSI6Img1UDRLckc4eW5nIiwiZXhwIj
oxMzYyOTY0Njk2MDAwfQ.qZhUqupVPx3E7MI0GvsHf6DGzsspr2BluET0Sp
0DqvJEKQxKpb8o_iVlXvPkjvIztBnIj3MoO8RVLQbptOPd1k7qhMEpFHNTb
5XZJaeINPiACRK09uFiTNnwW1js1CzOcaLjLlI3xlWd-Iuzo388rMLlIudn
i1jNnE-29v_sSTNtq-C0Bch5C0Owkl71BNxxx3hUqxG1OL4Pt2gBJYAP_sN
VMvh1pX9aG7tVk4Kk1KccitjPWF7GWsrFzWxzDR0u6DFtFachCObefrfgfE
19qeZrKrzI0UdCrDPzYk9XoWJGkpFSOwWac_vCCuuv5V3Gd_LNSI3rBi-Fa
ehYHAF1IQ
```

Here we show the JWT header for the response assertion, as it includes an ASN.1 encoded X.509 certificate, which is used to mutually authenticate the IMAP server to the UA:


```
     {
         "alg": "RS256",
         "x5c": [
             "MIIDzjCCAragAwIBAgIBBzANBgkqhkiG9w0BAQUFADBdMQswCQ
              YDVQQGEwJBVTEeMBwGA1UECgwVUEFETCBTb2Z0d2FyZSBQdHkg
              THRkMS4wLAYDVQQDDCVQQURMIFNvZnR3YXJlIENlcnRpZmljYX
              Rpb24gQXV0aG9yaXR5MB4XDTEzMDExMTA1MzQyMFoXDTE2MDEx
              MTA1MzQyMFowTDELMAkGA1UEBhMCQVUxHjAcBgNVBAoMFVBBRE
              wgU29mdHdhcmUgUHR5IEx0ZDEdMBsGA1UEAwwUcmFuZC5taXQu
              ZGUucGFkbC5jb20wggEiMA0GCSqGSIb3DQEBAQUAA4IBDwAwgg
              EKAoIBAQDBhzFpfL6vHxsx2DvFZP+GR0UomDp/CEe+OHN0Z0c6
              91Yg9gWXmU3nPqVTbAIqXH8AUtvf7MzbY28oVlpZT05zm2c/tE
              3gkhTxmXORgQrcuugujOXMFhI4svEZ+CbHPlZiZ/TzkXLHP29E
              z7wNZn1H57ALqmSAo5T4qxMFgBYudw/ZxPRzDtUoIV0s23Yg4x
              T9awJnr1GgMURibUBjF7ybcm0K8sJT+UGeB7rmLlPw+fAkOf7Z
              jZ9tpTkEMi8uLEMqcxaGSAK/+kW75qOxdAFI8zYZX53gpg4mi+
              QWfGY1ZNQJMuHGQXg/ufxMzaxNN4h1aOlmZZYkBHp52A9yIMUb
              AgMBAAGjgakwgaYwCQYDVR0TBAIwADAsBglghkgBhvhCAQ0EHx
              YdT3BlblNTTCBHZW5lcmF0ZWQgQ2VydGlmaWNhdGUwHQYDVR0O
              BBYEFKSsubEDubRIGHCBttAlTv2FWGf+MB8GA1UdIwQYMBaAFL
              izZlMWnKK1PYagdJjkVuRhEQJjMAkGA1UdEQQCMAAwCwYDVR0P
              BAQDAgXgMBMGA1UdJQQMMAoGCCsGAQUFBwMDMA0GCSqGSIb3DQ
              EBBQUAA4IBAQBD1BzQP+rN1UUz7A6z+KDPaq8s2iBG0FzZxsX2
              QUOupBEIbuZp0KJaujVMg01fdjsuGG0uXbMfeRGyNlUsMN+ZDy
              8/MIOh1aUGv0SUwKtCtLtWrJv65uwHGGt5uFKxME61VT4+qpI2
              AGqxx5drsxELBOdyPneuAiLPxFubRFmzuhVSI7APMl79Ok30oW
              udA49lUX9wz3g9q9vdl9yhgeeUSUpMhlZ24Uc9PuLzr15j6v63
              XzrSdWtNzv0F10eDl4yTUNWSJh7qBhgq1IoX9APOOuLbMNrp6b
              eEeowh34pVWfTaSxI7nK57kK2xhRU43ewYj2i/Srz8GsMS91ur
              25It"]
     }
```


The assertion payload is below (again, for clarity the actual JWT signature has been omitted):

```
{
    "tkt": {
        "tid": "aeay0GJizDx79Ag--WL-vw6Y9BXxRuC1XsZxry51SUI",
        "exp": 1362997098000
    },
    "epk": {
        "x": "J1IWbH2A5C3chOUIqmfXpP_RQEEOmd2ExXoKrqQQXLM",
        "y": "_2Etxbz_vJeeUUby2rfdekTTPURpbGJH7kyIWqmkIEg"
    },
    "nonce": "h5P4KrG8yng",
    "exp": 1362964696000
}
```


Note the fast re-authentication ticket and the nonce echoed back from the initiator.

9. Security Considerations

This document defines a GSS-API security mechanism, and therefore deals in security and has security considerations text embedded throughout. This section only addresses security considerations associated with the BrowserID GSS mechanism described in this document. It does not address security considerations associated with the BrowserID protocol or the GSS-API themselves.

This mechanism provides for authentication of initiator principals using private keys to public key crypto-systems, using the BrowserID specification for user certificates (which are NOT PKIX [RFC5280] certificates). Authentication of the acceptor principal is optional. Fast re-authentication is supported via acceptor-issued fast re-authentication tickets.

All cryptography for per-message tokens is imported from the Kerberos GSS-API mechanism [RFC4121].

This mechanism actually has several mechanism OIDs, composed of a prefix identifying this family of mechanisms followed by an arc identifying the [RFC3961] encryption type for use with per-message tokens and the GSS_Pseudo_random() function. The NULL encryption type is supported, and when it is used then the GSS-API per-message tokens and GSS_Pseudo_random() function are not available, but channel binding and mutual authentication may be available. Also, when using the NULL encryption type the fast re-authentication feature is not available because key exchange is only performed the initiator application uses the variant of this mechanism that supports per-message tokens and the GSS_Pseudo_random() function.

Acceptor credentials are PKIX [RFC5280] certificates and their private keys.

9.1. Host certificates for mutual authentication

Allowing a match on only the DNS subjectAltName in an acceptor's X.509 certificate permits different services on the same host to impersonate each other. This should be subject to local policy.

9.2. Error statuses

Returning rich error information in the clear (see Section 6.3.2) may leak information. Implementations may squash status codes and/or avoid returning minor statuses entirely. Indeed, applications may even not send back error tokens at all, instead closing the connection or whatever might be appropriate for the application. (This is a generic GSS-API security consideration.)

10. IANA Considerations

This specification creates a number of IANA registries.

10.1. OID Registry

Prefix: iso.org.dod.internet.private.enterprise.padl.gssBrowserID (1.3.6.1.4.1.5322.24)

Decimal	Name	Description
0	Reserved	Reserved
1	mechanisms	A sub-arc containing BrowserID mechanisms
2	nametypes	A sub-arc containing BrowserID name types

Prefix: iso.org.dod.internet.private.enterprise.padl.gssBrowserID.mechanisms (1.3.6.1.4.1.5322.24.1)

Decimal	Name	Description	ECDH curve	Symmetric hash
0	gss-browserid-null	The NULL security mechanism	N/A	N/A
17	gss-browserid-aes128	The aes128-cts-hmac-sha1-96 mechanism	P-256	HS256
18	gss-browserid-aes256	The aes256-cts-hmac-sha1-96 mechanism	P-521	HS256

Prefix: iso.org.dod.internet.private.enterprise.padl.gssBrowserID.nametypes (1.3.6.1.4.1.5322.24.2)

Decimal	Name	Description
0	Reserved	Reserved
1	GSS_C_NT_BROWSERID_PRINCIPAL	3.1.1

10.2. SASL Registry

Subject: Registration of SASL mechanisms BROWSERID-AES128 and BROWSERID-AES128-PLUS

SASL mechanism names: BROWSERID-AES128 and BROWSERID-AES128-PLUS

Security considerations: See RFC 5801 and draft-howard-gss-browserid

Published specification (recommended): draft-howard-gss-browserid

Person & email address to contact for further information:

Luke Howard lukeh@padl.com

Intended usage: common

Owner/Change controller: iesg@ietf.org

Note: This mechanism describes the GSS BrowserID mechanism used with the aes128-cts-hmac-sha1-96 encryption type. The GSS-API OID for this mechanism is 1.3.6.1.4.1.5322.24.1.17. As described in RFC 5801 a PLUS variant of this mechanism is also required.

[[anchor9: We could use the NULL encryption type variant for SASL, as the GS2 bridge does not use message protection services. However, because that mechanisms is unkeyed, re-authentication would not be available. Defining a single AES128 mechanism is consistent with GSS EAP.]]

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.

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

[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.

[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.

[RFC4178] Zhu, L., Leach, P., Jaganathan, K., and W. Ingersoll, "The Simple and Protected Generic Security Service Application Program Interface (GSS-API) Negotiation Mechanism", RFC 4178, October 2005.

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

[RFC4556] Zhu, L. and B. Tung, "Public Key Cryptography for Initial Authentication in Kerberos (PKINIT)", RFC 4556, June 2006.

[RFC4757] Jaganathan, K., Zhu, L., and J. Brezak, "The RC4-HMAC Kerberos Encryption Types Used by Microsoft Windows", RFC 4757, December 2006.

[RFC4985] Santesson, S., "Internet X.509 Public Key Infrastructure Subject Alternative Name for Expression of Service Name", RFC 4985, August 2007.

[RFC5178] Williams, N. and A. Melnikov, "Generic Security Service Application Program Interface (GSS-API) Internationalization and Domain-Based Service Names and Name Type", RFC 5178, May 2008.

[RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key

Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

[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.

[RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, 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.

[I-D.ietf-jose-json-web-algorithms]
Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-18 (work in progress), November 2013.

[I-D.ietf-jose-json-web-key]
Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-18 (work in progress), November 2013.

[I-D.ietf-jose-json-web-signature]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Signature (JWS)", draft-ietf-jose-json-web-signature-18 (work in progress), November 2013.

[I-D.ietf-oauth-json-web-token]
Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", draft-ietf-oauth-json-web-token-13 (work in progress), November 2013.

[I-D.zhu-negoex]
Short, M., Zhu, L., Damour, K., and D. McPherson, "SPNEGO Extended Negotiation (NEGOEX) Security Mechanism", draft-zhu-negoex-04 (work in progress), January 2011.

[I-D.zhu-pku2u]
Zhu, L., Altman, J., and N. Williams, "Public Key Cryptography Based User-to-User Authentication - (PKU2U)", draft-zhu-pku2u-09 (work in progress), November 2008.

[BrowserID]
Adida, B., "BrowserID Specification", February 2013.

11.2. Informative References

[RFC4120] Neuman, C., Yu, T., Hartman, S., and K. Raeburn, "The Kerberos Network Authentication Service (V5)", RFC 4120, July 2005.

Authors' Addresses

Luke Howard
PADL Software
PO Box 59
Central Park, VIC 3145
Australia

Email: lukeh@padl.com

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com