SAS-ESC Protocol

Key Bridge LLC
1750 Tysons Blvd., Suite 1500
McLean, VA 22102
Phone: +1 (703) 542-4140
http://keybridgewireless.com

Document Information
Document Status Public
Version 1.0.0
Date Printed November 6, 2017
Copyright © 2017 Key Bridge LLC. All Rights Reserved

Table of Contents

Annexes

Glossary

SAS-ESC Peering Protocol

This document specifies the Application Programming Interface (API) for Interface between Spectrum Access System (SAS) and Environmental Sensing Capability (ESC).

Scope

This document is a Technical Specification of a communication protocol and procedures for the SAS-ESC interface. The key words "required", "shall", "shall not", "should", "should not", "recommended", "may", and "optional" in this document are to be interpreted as described in RFC-2119. In addition, the key word “conditional” shall be interpreted to mean that the definition is an absolute requirement of this specification only if the stated condition is met.

SAS-ESC Communication Concept

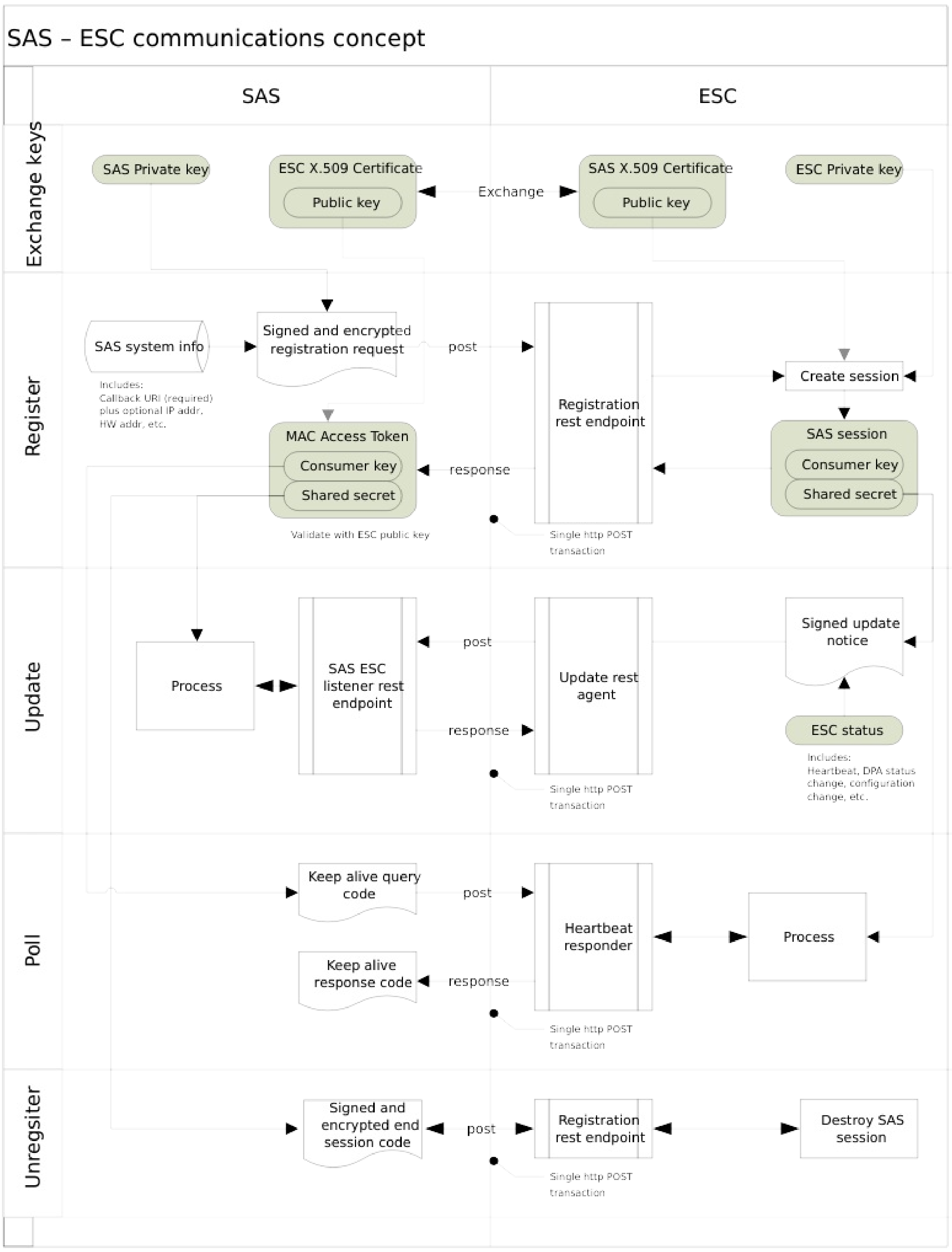
SAS – ESC communications concept
SAS
ESC
Exchange keys
Register
Update
Poll
Unregsiter
SAS Private key
ESC X.509 Certificate
Public key
Exchange
SAS X.509 Certificate
Public key
ESC Private key
SAS system info
Includes: Callback URI (required) plus optional IP addr, HW addr, etc.
Signed and encrypted registration request
post
Registration rest endpoint
Create session
SAS session
Consumer key
Shared secret
MAC Access Token
Consumer key
Shared secret
response
Validate with ESC public key
Single http POST transaction
Process
SAS ESC listener rest endpoint
post
response
Update rest agent
Signed update notice
ESC status
Includes: Heartbeat, DPA status change, configuration change, etc.
Single http POST transaction
Keep alive query code
post
Keep alive response code
response
Heartbeat responder
Process
Single http POST transaction
Signed and encrypted end session code
post
Registration rest endpoint
Destroy SAS session
Single http POST transaction

Figure 1: *SAS-ESC Communication Protocol Diagram*

SAS-ESC Communication Lifecycle

At a high level, the lifecycle of the SAS-ESC communication protocol consists of three parts (figure 2):

1. **Initialion/Authentication**. In this stage human SAS Administrator and ESC Operator exchange the Public Key

Infrastructure (PKI). This ensures full knowledge of identities of systems that are communicating.

2. **Normal Operation**. The SAS registers with the ESC and begins the normal mode of operation:
   - Heartbeat polling to ensure connection and system liveness
   - Accepting and processing event notifications from the ESC
   - Updating the ESC with the most recent information about the SAS, should it change
3. **End of Service**. Here the SAS ceases co-operating with the ESC by sending a de-registration message. The ESC Operator may remove the SAS's PKI information. This step may never occur

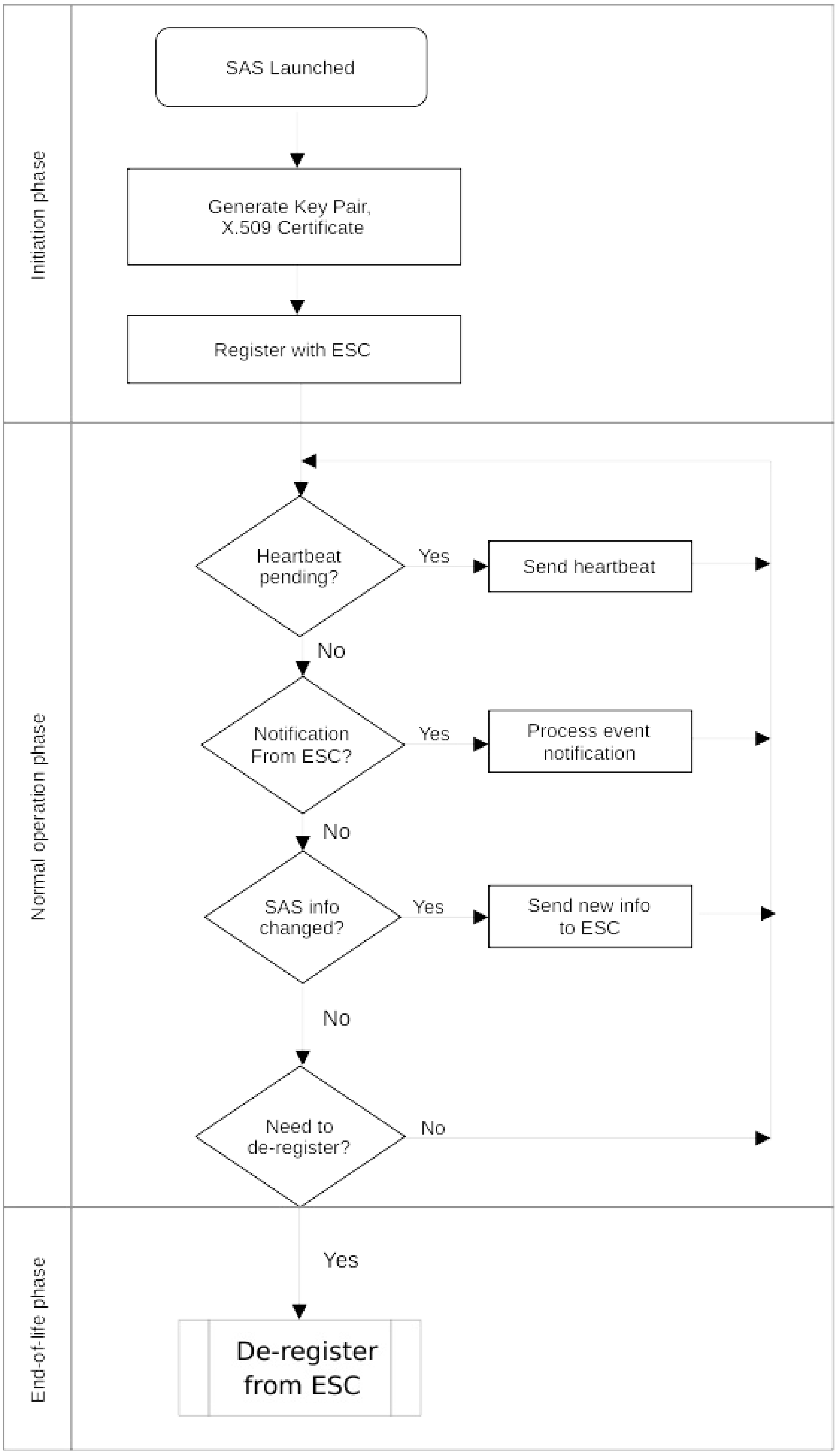
Initiation phase
Normal operation phase
End-of-life phase
SAS Launched
Generate Key Pair, X.509 Certificate
Register with ESC
Heartbeat pending?
Yes
Send heartbeat
No
Notification From ESC?
Yes
Process event notification
No
SAS info changed?
Yes
Send new info to ESC
No
Need to de-register?
No
Yes
De-register from ESC

Figure 1: *High-Level Communication Lifecycle from a SAS Perspective*

Data flows in the Communication Lifecycle

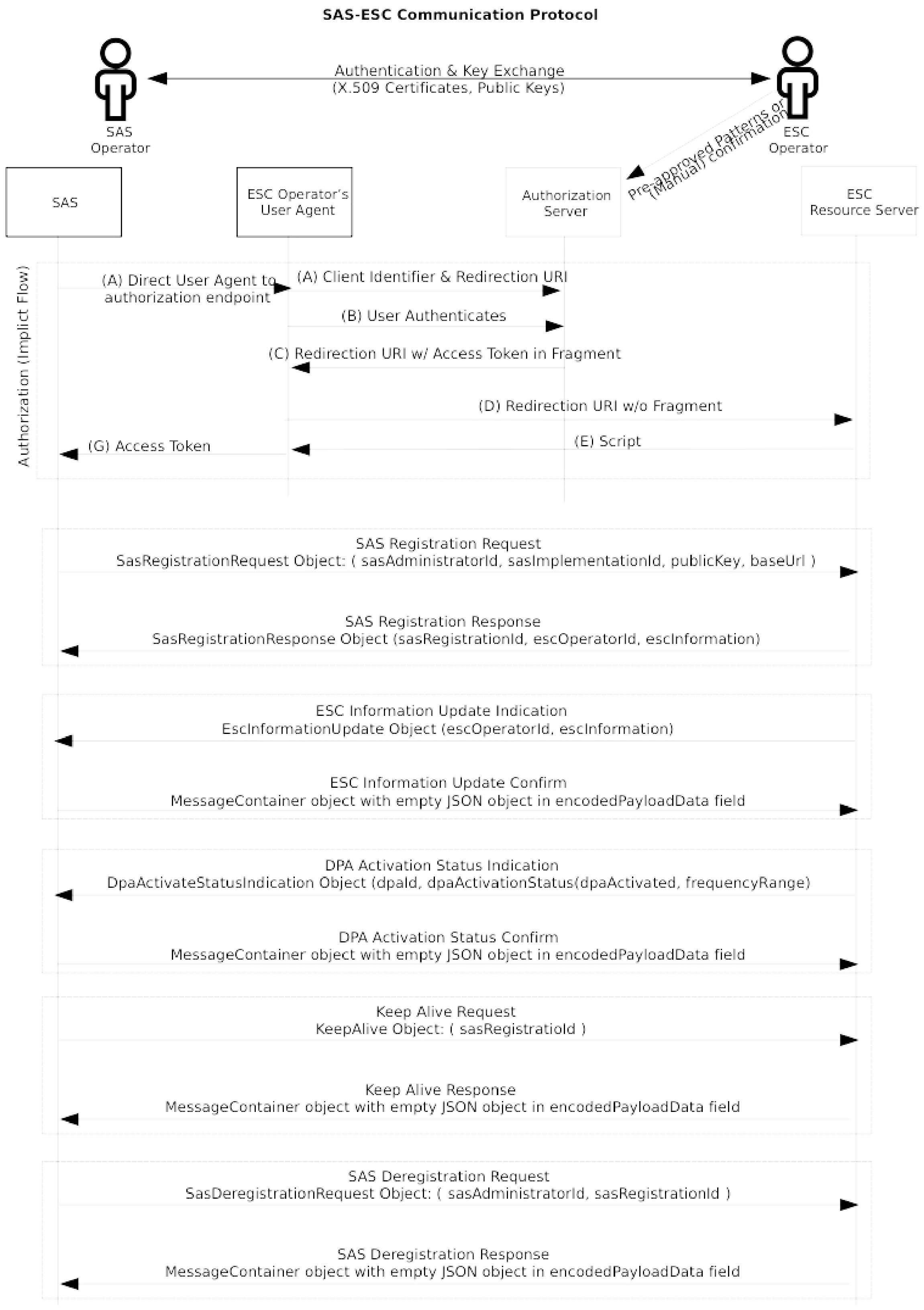
SAS-ESC Communication Protocol
Authentication & Key Exchange
(X.509 Certificates, Public Keys)
SAS Operator
ESC Operator
Pre-approved Patterns or (Manual) confirmation
SAS
ESC Operator's User Agent
Authorization Server
ESC Resource Server
Authorization (Implict Flow)
(A) Direct User Agent to authorization endpoint
(A) Client Identifier & Redirection URI
(B) User Authenticates
(C) Redirection URI w/ Access Token in Fragment
(D) Redirection URI w/o Fragment
(E) Script
(G) Access Token
SAS Registration Request
SasRegistrationRequest Object: (sasAdministratorId, sasImplementationId, publicKey, baseUrl)
SAS Registration Response
SasRegistrationResponse Object (sasRegistrationId, escOperatorId, escInformation)
ESC Information Update Indication
EscInformationUpdate Object (escOperatorId, escInformation)
ESC Information Update Confirm
MessageContainer object with empty JSON object in encodedPayloadData field
DPA Activation Status Indication
DpaActivateStatusIndication Object (dpaId, dpaActivationStatus(dpaActivated, frequencyRange)
DPA Activation Status Confirm
MessageContainer object with empty JSON object in encodedPayloadData field
Keep Alive Request
KeepAlive Object: (sasRegistratioId)
Keep Alive Response
MessageContainer object with empty JSON object in encodedPayloadData field
SAS Deregistration Request
SasDeregistrationRequest Object: (sasAdministratorId, sasRegistrationId)
SAS Deregistration Response
MessageContainer object with empty JSON object in encodedPayloadData field

Figure 3: *SAS-ESC Communication Lifecycle*

SAS-ESC Independence

Key Bridge expects the FCC to authorize more than one party to operate a ESC, and that the internal architecture and configuration of each ESC may differ. Key Bridge further expects that each ESC, or at least the ESC in this proposal, will be independent and autonomous to the one or more SASs that depend upon their respective ESC for incumbent detection. The same principle applies in reverse as well: a SAS's internal architecture and configuration is also independent and autonomous to its ESC, and each ESC may pursue different strategies and implement different technologies for incumbent detection.

Regardless of their respective internal architecture, technology and detection strategies, a ESC and SAS must interface to share information about spectrum availability or un-availability. This essential concept is called peering and is shown in the illustration below. Through peering a SAS and ESC may exchange data in a neutral manner irrespective of their internal architecture and configuration.

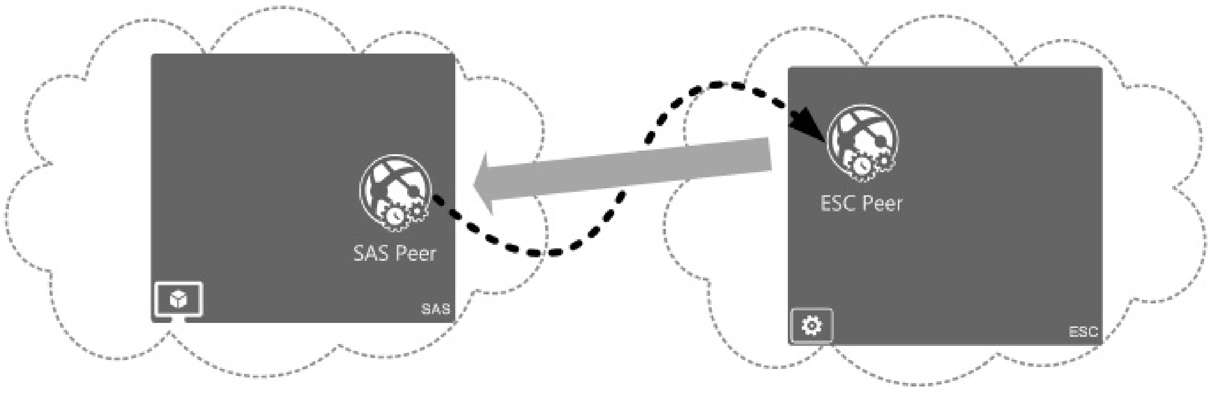

Illustration 1: *SAS to ESC Peering masks internal architecture*

SAS-ESC Peering Protocol

A SAS exterior peering protocol standard will place no constraint on the internal architecture and operation of the various entities that use the protocol. Referring again to Illustration 1, a generic peering process enables SAS of unknown constitution on the left peers with a Key Bridge ESC on the right.

> **This paragraph needs to be reviewed** SAS to ESC peering operational security policies and configurations are under development within a multi-stakeholder group in which Key Bridge is a participant (The Wireless Innovation Forum, Spectrum Sharing Committee, Working Group 2: Security). The actual protocol and data structures for SAS to ESC peering is not the subject of multi-stakeholder group consideration. Key Bridge has therefore invented proprietary and also proposed non-proprietary methods that a SAS may use when coordinating with a ESC to dynamically protect a non-informing incumbent spectrum user. The Key Bridge SAS Gateway Protocol is, at present, a proprietary application peering protocol that includes mechanisms to support cryptographically secure message exchange for both SAS to SAS and SAS to ESC peering, including all security prescriptions identified by the multi-stakeholder group.

A SAS and ESC Peering relationship using the SAS Gateway Protocol is secure and neutral. A SAS Infrastructure may register with and receive information from any inter-operable ESC Infrastructure supporting a peering service. When using the Key Bridge SAS Gateway Protocol a compatible SAS may register with and receive information from a compatible ESC via a designated ESC Peering API.

In the Key Bridge architecture ESC Service Nodes are responsible for Non-Informing Incumbent User (NIIU) detection services and peer directly with a subscribing SAS via the SAS Gateway Protocol. This strategy is designed to limit the geographic extent of NIIU information conveyed to a SAS to that necessary for the SAS to administer CBSDs in a specific geographic region. However, the ESC places no constraint on SAS internal architecture or operation, and a SAS may peer with multiple ESC Service Nodes.

Generic SAS Peering Workflow

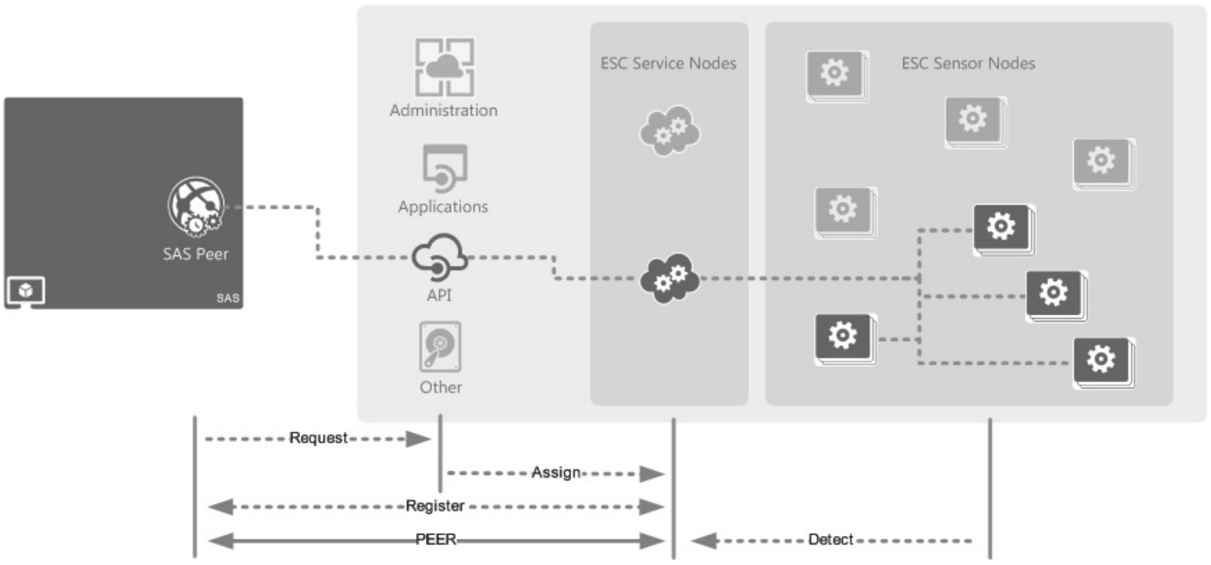

Illustration 2: *ESC Service Nodes provide Non-Informing Incumbent User (NIIU) detection services to a SAS*

The process of a generic SAS peering with the Key Bridge ESC Infrastructure is detailed in Illustration 2, and proceeds according to the following general strategy.

- **Request**. A SAS first inquires about service availability to the ESC Infrastructure via a general peering API interface. The SAS service request includes the geographic area or areas where the SAS wishes to provide spectrum administration. If ESC services are not available in the requested geographic area the request is rejected (i.e. fails), otherwise the request is assigned to a matching ESC Service Node.
- **Assignment**. If NIIU detection capability is available in the SAS requested geographic area the ESC Infrastructure will assign the SAS registration request to a matching ESC Service Node.
- **Register**. After a geographic match has been established the respective ESC Service Node and the SAS exchange detailed information necessary to establish a direct and persistent peering relationship. The registration process includes conveying to the ESC Peer the specific SAS geographic regions of responsibility plus instructions describing how the ESC may query the SAS for additional information and also how to notify the SAS of ESC spectrum availability information, updates and instructions.
- **Peer**. Once established the ESC Service Node will respond to spectrum availability requests within its geographic area of service and also inform the SAS about any changes in spectrum availability.

An artifact of this CONOPS is that the Key Bridge ESC Service Nodes are assigned responsibility to determine the quantity and quality of protection for the incumbent user within their ESC Service Area, while responsibility to effect that protection is retained within a SAS. That is to say: An ESC Service Node is the service providing component of the ESC Infrastructure, and it is the ESC Service Node that authorizes or de-authorizes CBSD operations within their responsible service area according to a *geographic partitioning strategy* within their respective ESC Service Area.

SAS Client Authentication

Prior to SAS-ESC communications, the (human) ESC operator and SAS administrators must exchange the Public Key Infrastructure (PKI). In particular, at the end of this step, the ESC must posses the SAS's public key and the SAS must posses the public key of the ESC.

If the client type is confidential, the client and authorization server establish a client authentication method suitable for the security requirements of the authorization server. The authorization server MAY accept any form of client authentication meeting its security requirements.

Confidential clients are typically issued (or establish) a set of client credentials used for authenticating with the authorization server (e.g., password, public/private key pair).

The client MUST NOT use more than one authentication method in each request.

Authorization

Authorization is based on OAuth 2.0 Implicit Flow as described in RFC 6749 "The OAuth 2.0 Authorization Framework".

OAuth 2.0 enables a third-party application to obtain limited access to an HTTP service, either on behalf of a resource owner by orchestrating an approval interaction between the resource owner and the HTTP service, or by allowing the third-party application to obtain access on its own behalf.

In OAuth, the client requests access to resources controlled by the resource owner and hosted by the resource server, and is issued a different set of credentials than those of the resource owner. Instead of using the resource owner's credentials to access protected resources, the client obtains an access token -- a string denoting a specific scope, lifetime, and other access attributes. Access tokens are issued to third-party clients by an authorization server with the approval of the resource owner. The client uses the access token to access the protected resources hosted by the resource server.

For example, an end-user (resource owner) can grant a printing service (client) access to her protected photos stored at a photo-sharing service (resource server), without sharing her username and password with the printing service. Instead, she authenticates directly with a server trusted by the photo-sharing service (authorization server), which issues the printing service delegation-specific credentials (access token).

To request an access token, the client obtains authorization from the resource owner. The authorization is expressed in the form of an authorization grant, which the client uses to request the access token.

OAuth defines four grant types:

- authorization code
- implicit
- resource owner password credentials, and
- client credentials.

It also provides an extension mechanism for defining additional grant types.

In this application, implicit grant is used.

Roles

OAuth defines four roles:

1. Resource Owner (ESC Operator):

> An entity capable of granting access to a protected resource. When the resource owner is a person, it is referred to as an end-user.

2. Resource Server (ESC Server):

> The server hosting the protected resources, capable of accepting and responding to protected resource requests using access tokens.

3. Client (SAS):

> An application making protected resource requests on behalf of the resource owner and with its authorization. The term "client" does not imply any particular implementation characteristics (e.g., whether the application executes on a server, a desktop, or other devices).

4. Authorization Server:

> The server issuing access tokens to the client after successfully authenticating the resource owner and obtaining authorization. The authorization server may be the same server as the resource server or a separate entity.

Additionally, in the context of Implicit flow, user agent is also involved:

5. User Agent:

> The agent used by the Resource Owner to interact with the Client, for example a browser or a native application.

Abstract OAuth 2.0 Protocol Flow

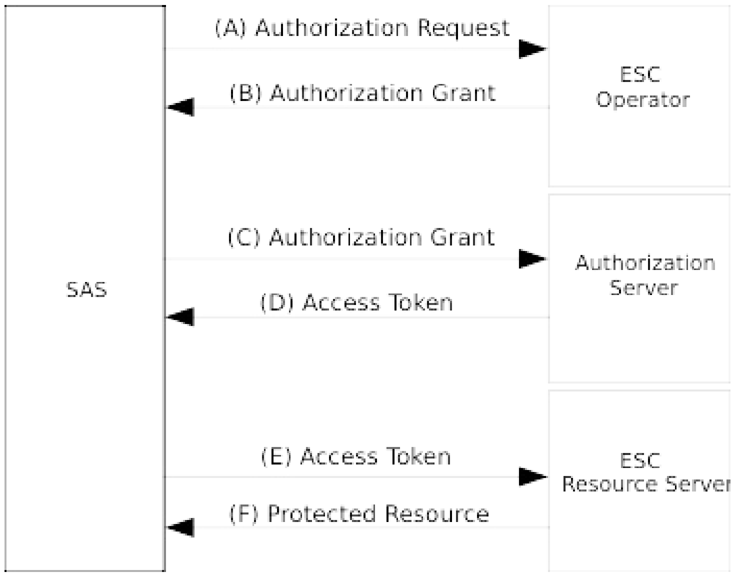

Figure 1: *Abstract Protocol Flow*

Based on RFC 6749, the abstract OAuth 2.0 flow illustrated in Figure 1 describes the interaction between the four roles and includes the following steps:

- (A) The SAS requests authorization from the ESC Operator. The authorization request can be made directly to the ESC Operator (as shown), or preferably indirectly via the authorization server as an intermediary.
- (B) The SAS receives an authorization grant, which is a credential representing the ESC Operator's authorization, expressed using one of four grant types defined in this specification or using an extension grant type. The authorization grant type depends on the method used by the SAS to request authorization and the types supported by the authorization server.
- (C) The SAS requests an access token by authenticating with the authorization server and presenting the authorization grant.
- (D) The authorization server authenticates the SAS and validates the authorization grant, and if valid, issues an access token.
- (E) The SAS requests the protected resource from the resource server and authenticates by presenting the access token.
- (F) The ESC server validates the access token, and if valid, serves the request.

The preferred method for the SAS to obtain an authorization grant from the ESC Operator (depicted in steps (A) and (B)) is to use the authorization server as an intermediary, which is illustrated in the figure below.

Obtaining Authorization

To request an access token, the client obtains authorization from the resource owner. The authorization is expressed in the form of an authorization grant, which the client uses to request the access token. OAuth defines four grant types:

- authorization code
- implicit
- resource owner password credentials, and
- client credentials
  It also provides an extension mechanism for defining additional grant types.

Key Bridge ESC implements Implicit grant.

Authorization Grant

An authorization grant is a credential representing the resource owner's authorization (to access its protected resources) used by the client to obtain an access token. This specification defines four grant types -- authorization code, implicit, resource owner password credentials, and client credentials -- as well as an extensibility mechanism for defining additional types.

Implicit Authorization Grant

The implicit grant is a simplified authorization code flow optimized for clients implemented in a browser using a scripting language such as JavaScript. In the implicit flow, instead of issuing the client an authorization code, the client is issued an access token directly (as the result of the resource owner authorization). The grant type is implicit, as no intermediate credentials (such as an authorization code) are issued (and later used to obtain an access token).

When issuing an access token during the implicit grant flow, the authorization server does not authenticate the client. In some cases, the client identity can be verified via the redirection URI used to deliver the access token to the client. The access token may be exposed to the resource owner or other applications with access to the resource owner's user-agent.

Implicit grants improve the responsiveness and efficiency of some clients (such as a client implemented as an in-browser application), since it reduces the number of round trips required to obtain an access token.

Implicit Flow - Authorization

The implicit grant type is used to obtain access tokens (it does not support the issuance of refresh tokens) and is optimized for public clients known to operate a particular redirection URI. These clients are typically implemented in a browser using a scripting language such as JavaScript.

Since this is a redirection-based flow, the client must be capable of interacting with the resource owner's user-agent (typically a web browser) and capable of receiving incoming requests (via redirection) from the authorization server.

Unlike the authorization code grant type, in which the client makes separate requests for authorization and for an access token, the client receives the access token as the result of the authorization request.

The implicit grant type does not include client authentication, and relies on the presence of the resource owner and the registration of the redirection URI. Because the access token is encoded into the redirection URI, it may be exposed to the resource owner and other applications residing on the same device.

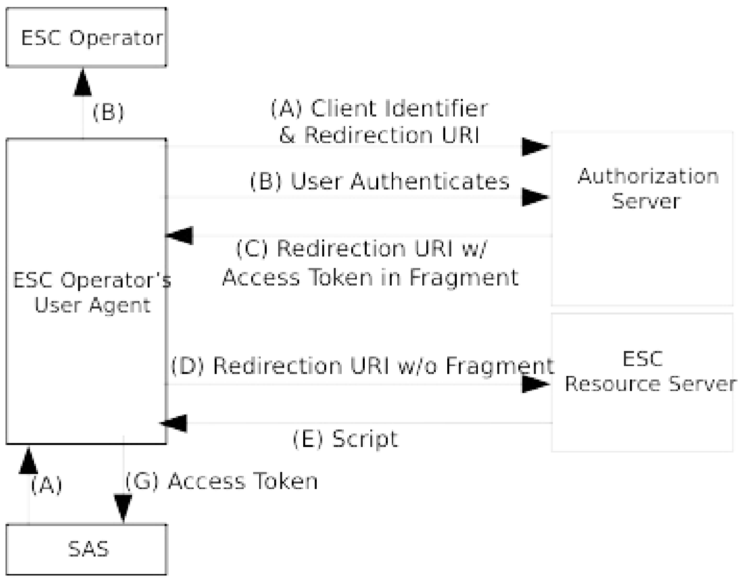
ESC Operator
(B)
(A) Client Identifier & Redirection URI
(B) User Authenticates
Authorization Server
(C) Redirection URI w/ Access Token in Fragment
ESC Operator's User Agent
(D) Redirection URI w/o Fragment
ESC Resource Server
(E) Script
(A)
(G) Access Token
SAS

Figure 4: *Implicit Grant Flow*

Note: The lines illustrating steps (A) and (B) are broken into two parts as they pass through the user-agent.

Based on RFC 6749, the flow illustrated in Figure 4 includes the following steps:

(A) The SAS client initiates the flow by directing the ESC Operator's user-agent to the authorization endpoint. The SAS includes its SAS client identifier, requested scope, local state, and a redirection URI to which the authorization server will send the user-agent back once access is granted (or denied).

(B) The authorization server authenticates the ESC Operator (via the user-agent) and establishes whether the ESC Operator grants or denies the client's access request.

(C) Assuming the ESC Operator grants access, the authorization server redirects the user-agent back to the SAS client using the redirection URI provided earlier. The redirection URI includes the access token in the URI fragment.

(D) The user-agent follows the redirection instructions by making a request to the web-hosted client resource (which does not include the fragment per [RFC2616]). The user-agent retains the fragment information locally.

(E) The web-hosted client resource returns a web page (typically an HTML document with an embedded script) capable of accessing the full redirection URI including the fragment retained by the user-agent, and extracting the access token (and other parameters) contained in the fragment.

(F) The user-agent executes the script provided by the web-hosted client resource locally, which extracts the access token.

(G) The user-agent passes the access token to the SAS client.

Implicit Flow - Authorization Request

The client constructs the request URI by adding the following parameters to the query component of the authorization endpoint URI using the "application/x-www-form-urlencoded" format:

FIELD NAME	R/O/C	DESCRIPTION
response_type	REQUIRED	Value MUST be set to "token".
client_id	REQUIRED	The client identifier
redirect_uri	OPTIONAL	As described in Section Redirection Endpoint
scope	OPTIONAL	The scope of the access request as described by Section Access Token Scope
state	RECOMMENDED	An opaque value used by the client to maintain state between the request and callback. The authorization server includes this value when redirecting the user-agent back to the client. The parameter SHOULD be used for preventing cross-site request forgery as described in Security Considerations

The client directs the resource owner to the constructed URI using an HTTP redirection response, or by other means available to it via the user-agent.

For example, the client directs the user-agent to make the following HTTP request using TLS (with extra line breaks for display purposes only):

```
GET /authorize?response_type=token&client_id=s6BhdRkqt3&state=xyz
    &redirect_uri=https%3A%2F%2Fclient%2Eexample%2Ecom%2Fcb HTTP/1.1
Host: server.example.com
```

The authorization server validates the request to ensure that all required parameters are present and valid. The authorization server MUST verify that the redirection URI to which it will redirect the access token matches a redirection URI registered by the client as described in Redirection Endpoint.

If the request is valid, the authorization server authenticates the resource owner and obtains an authorization decision (by asking the resource owner or by establishing approval via other means).

When a decision is established, the authorization server directs the user-agent to the provided client redirection URI using an HTTP redirection response, or by other means available to it via the user-agent.

Implicit Flow - Access Token Response

If the resource owner grants the access request, the authorization server issues an access token and delivers it to the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format, per Appendix B:

FIELD NAME	R/O/C	DESCRIPTION
access_token	REQUIRED	The access token issued by the authorization server.
token_type	REQUIRED	The type of the token issued as described below. Value is case insensitive.
expires_in	RECOMMENDED	The lifetime in seconds of the access token. For example, the value "3600" denotes that the access token will expire in one hour from the time the response was generated. If omitted, the authorization server SHOULD provide the expiration time via other means or document the default value.
scope	OPTIONAL if identical to the scope requested by the client; otherwise, REQUIRED	The scope of the access token as described by Section "Action Token Scope"
state	REQUIRED if the "state" parameter was present in the client authorization request.	The exact value received from the client.

The authorization server MUST NOT issue a refresh token.

For example, the authorization server redirects the user-agent by sending the following HTTP response (with extra line breaks for display purposes only):

```
HTTP/1.1 302 Found
Location: http://example.com/cb#access_token=2YotnFZFEjr1zCsicMWpAA
          &state=xyz&token_type=example&expires_in=3600
```

Developers should note that some user-agents do not support the inclusion of a fragment component in the HTTP "Location" response header field. Such clients will require using other methods for redirecting the client than a 3xx redirection response -- for example, returning an HTML page that includes a 'continue' button with an action linked to the redirection URI.

The client MUST ignore unrecognized response parameters. The access token string size is left undefined by this specification. The client should avoid making assumptions about value sizes. The authorization server SHOULD document the size of any value it issues.

Access Token Types

The access token type provides the client with the information required to successfully utilize the access token to make a protected resource request (along with type-specific attributes). The client MUST NOT use an access token if it does not understand the token type.

For example, the "bearer" token type defined in [RFC6750] is utilized by simply including the access token string in the request:

```
GET /resource/1 HTTP/1.1
```

```
     Host: example.com
     Authorization: Bearer mF_9.B5f-4.1JqM
```

while the "mac" token type defined in [OAuth-HTTP-MAC] is utilized by issuing a Message Authentication Code (MAC) key together with the access token that is used to sign certain components of the HTTP requests:

```
     GET /resource/1 HTTP/1.1
     Host: example.com
     Authorization: MAC id="h480djs93hd8",
                        nonce="274312:dj83hs9s",
                        mac="kDZvddkndxvhGRXZhvuDjEWhGeE="
```

The above examples are provided for illustration purposes only. Developers are advised to consult the [RFC6750] and [OAuth-HTTP-MAC] specifications before use.

Each access token type definition specifies the additional attributes (if any) sent to the client together with the "access_token" response parameter. It also defines the HTTP authentication method used to include the access token when making a protected resource request.

Implicit Flow - Error Response

If the request fails due to a missing, invalid, or mismatching redirection URI, or if the client identifier is missing or invalid, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.

If the resource owner denies the access request or if the request fails for reasons other than a missing or invalid redirection URI, the authorization server informs the client by adding the following parameters to the fragment component of the redirection URI using the "application/x-www-form-urlencoded" format, per Appendix B:

FIELD NAME	R/O/C	DESCRIPTION
error	REQUIRED	A single ASCII [USASCII] error code from the Error Codes list below. Values for the "error" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.
error_description	OPTIONAL	Human-readable ASCII [USASCII] text providing additional information, used to assist the client developer in understanding the error that occurred. Values for the "error_description" parameter MUST NOT include characters outside the set %x20-21 / %x23-5B / %x5D-7E.
error_uri	OPTIONAL	A URI identifying a human-readable web page with information about the error, used to provide the client developer with additional information about the error. Values for the "error_uri" parameter MUST conform to the URI-reference syntax and thus MUST NOT include characters outside the set %x21 / %x23-5B / %x5D-7E.
state	REQUIRED	if a "state" parameter was present in the client authorization request. The exact value received from the client.

List Of Error Codes

```
invalid_request
      The request is missing a required parameter, includes an
      invalid parameter value, includes a parameter more than
      once, or is otherwise malformed.

unauthorized_client
      The client is not authorized to request an access token
      using this method.

access_denied
      The resource owner or authorization server denied the
      request.

unsupported_response_type
      The authorization server does not support obtaining an
      access token using this method.

invalid_scope
      The requested scope is invalid, unknown, or malformed.

server_error
      The authorization server encountered an unexpected
      condition that prevented it from fulfilling the request.
      (This error code is needed because a 500 Internal Server
      Error HTTP status code cannot be returned to the client
      via an HTTP redirect.)

temporarily_unavailable
      The authorization server is currently unable to handle
      the request due to a temporary overloading or maintenance
```

```
        of the server.  (This error code is needed because a 503
        Service Unavailable HTTP status code cannot be returned
        to the client via an HTTP redirect.)
```

For example, the authorization server redirects the user-agent by sending the following HTTP response:

```
     HTTP/1.1 302 Found
     Location: https://client.example.com/cb#error=access_denied&state=xyz
```

Protocol Endpoints

The authorization process utilizes two authorization server endpoints (HTTP resources):

1. **Authorization endpoint** - used by the client to obtain authorization from the resource owner via user-agent redirection.
2. **Token endpoint** - used by the client to exchange an authorization grant for an access token, typically with client authentication. Token endpoint is not used for the implicit grant type (since an access token is issued directly).

As well as one client endpoint:

1. **Redirection endpoint** - used by the authorization server to return responses containing authorization credentials to the client via the resource owner user-agent.

Not every authorization grant type utilizes both endpoints.
Extension grant types MAY define additional endpoints as needed.

Authorization Endpoint

The authorization endpoint is used to interact with the resource owner and obtain an authorization grant. The authorization server MUST first verify the identity of the resource owner.

```
 TO DO:
The client obtains the location of the authorization endpoint by  ...
```

The endpoint URI MAY include an "application/x-www-form-urlencoded" formatted query component ([RFC3986] Section 3.4), which MUST be retained when adding additional query parameters. The endpoint URI MUST NOT include a fragment component.

Since requests to the authorization endpoint result in user authentication and the transmission of clear-text credentials (in the HTTP response), the authorization server MUST require the use of TLS when sending requests to the authorization endpoint.

The authorization server MUST support the use of the HTTP "GET" method [RFC2616] for the authorization endpoint and MAY support the use of the "POST" method as well.

Parameters sent without a value MUST be treated as if they were omitted from the request. The authorization server MUST ignore unrecognized request parameters. Request and response parameters MUST NOT be included more than once.

Response Type

The authorization endpoint is used by the authorization code grant type and implicit grant type flows. Key Bridge ESC uses the Implicit grant type flow.

The client informs the authorization server of the desired grant type using the following parameter:

```
response_type  REQUIRED.
```

The value MUST be "token" for requesting an access token (implicit grant) as described by Implicit Flow Authorization Request.

Extension response types MAY contain a space-delimited (%x20) list of values, where the order of values does not matter (e.g., response type `a b` is the same as `b a`). The meaning of such composite response types is defined by their respective specifications.

If an authorization request is missing the "response_type" parameter, or if the response type is not understood, the authorization server MUST return an error response

Redirection Endpoint

After completing its interaction with the resource owner, the authorization server directs the resource owner's user-agent back to the client. The authorization server redirects the user-agent to the client's redirection endpoint previously established with the authorization server during the client registration process or when making the authorization request.

The redirection endpoint URI MUST be an absolute URI as defined by [RFC3986] Section 4.3. The endpoint URI MAY include an "application/x-www-form-urlencoded" query component ([RFC3986] Section 3.4), which MUST be retained when adding additional query parameters. The endpoint URI MUST NOT include a fragment component.

3.1.2.1. Endpoint Request Confidentiality

The redirection endpoint SHOULD require the use of TLS as described in Section 1.6 when the requested response type is "code" or "token", or when the redirection request will result in the transmission of sensitive credentials over an open network. This specification does not mandate the use of TLS because at the time of this writing, requiring clients to deploy TLS is a significant hurdle for many client developers. If TLS is not available, the authorization server SHOULD warn the resource owner about the insecure endpoint prior to redirection (e.g., display a message during the authorization request).

Lack of transport-layer security can have a severe impact on the security of the client and the protected resources it is authorized to access. The use of transport-layer security is particularly critical when the authorization process is used as a form of delegated end-user authentication by the client (e.g., third-party sign-in service).

Registration Requirements

The authorization server MUST require the following clients to register their redirection endpoint:

- Public clients.
- Confidential clients utilizing the implicit grant type.

  The authorization server SHOULD require all clients to register their redirection endpoint prior to utilizing the authorization endpoint.

  The authorization server SHOULD require the client to provide the complete redirection URI (the client MAY use the "state" request parameter to achieve per-request customization). If requiring the registration of the complete redirection URI is not possible, the authorization server SHOULD require the registration of the URI scheme, authority, and path (allowing the client to dynamically vary only the query component of the redirection URI when requesting authorization).

  The authorization server MAY allow the client to register multiple redirection endpoints.

  Lack of a redirection URI registration requirement can enable an attacker to use the authorization endpoint as an open redirector.

Open Redirectors (Security Consideration)

The authorization server, authorization endpoint, and client redirection endpoint can be improperly configured and operate as open redirectors. An open redirector is an endpoint using a parameter to automatically redirect a user-agent to the location specified by the parameter value without any validation.

Open redirectors can be used in phishing attacks, or by an attacker to get end-users to visit malicious sites by using the URI authority component of a familiar and trusted destination. In addition, if the authorization server allows the client to register only part of the redirection URI, an attacker can use an open redirector operated by the client to construct a redirection URI that will pass the authorization server validation but will send the authorization code or access token to an endpoint under the control of the attacker.

Dynamic Configuration

If multiple redirection URIs have been registered, if only part of the redirection URI has been registered, or if no redirection URI has been registered, the client MUST include a redirection URI with the authorization request using the "redirect_uri" request parameter.

When a redirection URI is included in an authorization request, the authorization server MUST compare and match the value received against at least one of the registered redirection URIs (or URI components) as defined in [RFC3986] Section 6, if any redirection URIs were registered. If the client registration included the full redirection URI, the authorization server MUST compare the two URIs using simple string comparison as defined in [RFC3986] Section 6.2.1.

Invalid Endpoint

If an authorization request fails validation due to a missing, invalid, or mismatching redirection URI, the authorization server SHOULD inform the resource owner of the error and MUST NOT automatically redirect the user-agent to the invalid redirection URI.

Endpoint Content

The redirection request to the client's endpoint typically results in an HTML document response, processed by the user-agent. If the HTML response is served directly as the result of the redirection request, any script included in the HTML document will execute with full access to the redirection URI and the credentials it contains.

The client SHOULD NOT include any third-party scripts (e.g., third- party analytics, social plug-ins, ad networks) in the redirection endpoint response. Instead, it SHOULD extract the credentials from the URI and redirect the user-agent again to another endpoint without exposing the credentials (in the URI or elsewhere). If third-party scripts are included, the client MUST ensure that its own scripts (used to extract and remove the credentials from the URI) will execute first.

Access Token Scope

The authorization and token endpoints allow the client to specify the scope of the access request using the "scope" request parameter. In turn, the authorization server uses the "scope" response parameter to inform the client of the scope of the access token issued.

The value of the scope parameter is expressed as a list of space- delimited, case-sensitive strings. The strings are defined by the authorization server. If the value contains multiple space-delimited strings, their order does not matter, and each string adds an additional access range to the requested scope.

```
     scope       = scope-token *( SP scope-token )
     scope-token = 1*( %x21 / %x23-5B / %x5D-7E )
```

The authorization server MAY fully or partially ignore the scope requested by the client, based on the authorization server policy or the resource owner's instructions. If the issued access token scope is different from the one requested by the client, the authorization server MUST include the "scope" response parameter to inform the client of the actual scope granted.

If the client omits the scope parameter when requesting authorization, the authorization server MUST either process the request using a pre-defined default value or fail the request indicating an invalid scope. The authorization server SHOULD document its scope requirements and default value (if defined).

Message exchange flow and message types

Message exchanges between SAS and ESC are shown in Figures 1 and 2 below.

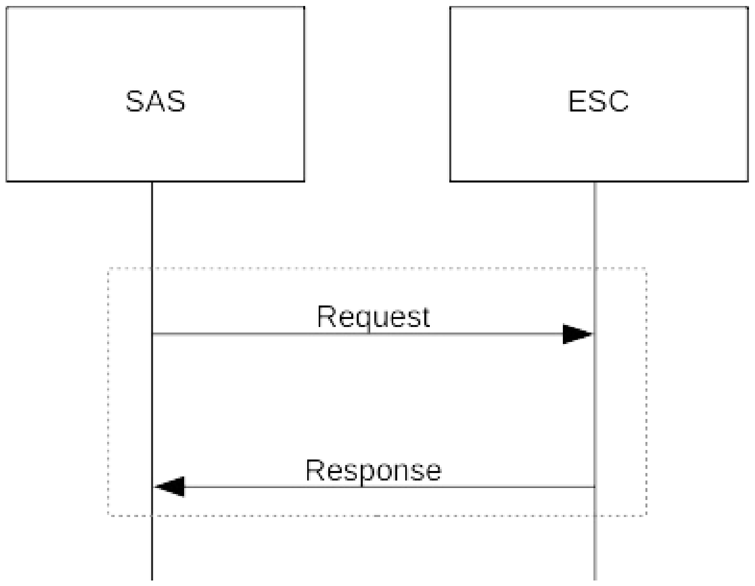

Figure 1: *Request-Response flow*

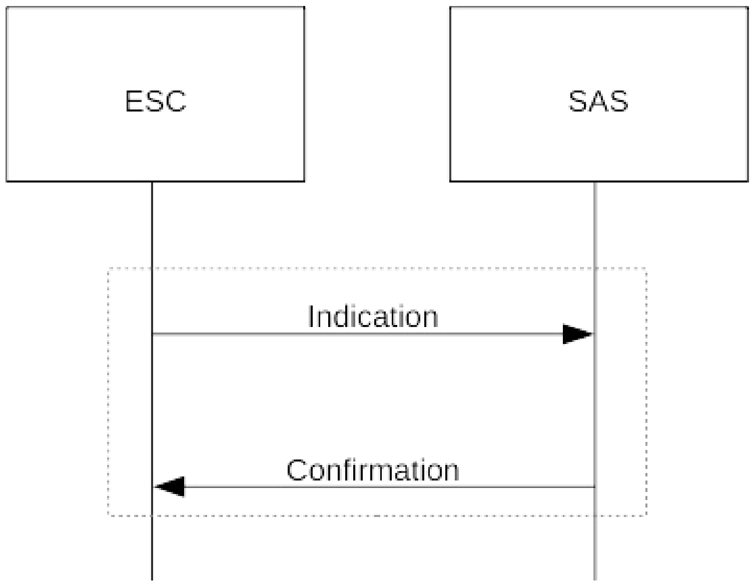

Figure 2: *Indication-Confirmation flow*

As per this message flow, the following messages shall be exchanged between SAS and ESC.

- **SAS Registration Message**: The message for the SAS to register with the ESC. Request-Response flow in Figure 1 is used.
- **ESC Information Update Message**: The message for the ESC to indicate the update of the ESC information to the SAS. Indication-Confirmation flow in Figure 2 is used.
- **DPA State Message**: The message for the ESC to indicate the DPA activation status to the SAS. Indication-Confirmation flow in Figure 2 is used.
- **Keep Alive Message**: The message for the SAS to detect a failure with ESC. Request-Response flow in Figure 1 is used.

- **SAS Deregistration Message**: The message for the SAS to deregister from the ESC. Request-Response flow in Figure 1 is used.

Message encoding and transport

Message encoding

The contents of SAS-ESC messages shall be generated by encoding the MessageContainer object using JSON (JavaScript Object Notation) as defined in RFC 7159 [n.2]. Unicode characters shall be used and its default encoding shall be UTF-8.

Message transport

HTTPS shall be used as the transport protocols for SAS-ESC message exchanges, in particular, HTTP version 1.1 as specified in [n.4].

The TLS protocol as specified in [n.3] and shall be used:

> At a minimum TLS version 1.2 is required in SAS TLS implementations to mitigate different attacks on earlier versions. In addition, SAS should be configured to support TLS version 1.3 when that standard is finalized. SAS shall not support TLS versions 1.1, 1.0, or SSL version 3.0 or any other earlier versions of these standards.

The HTTP **POST** method shall be used for all requests from the ESC to the SAS and from the SAS to the ESC.

POST requests shall be sent to the URL provided by the SAS or the ESC. The URL shall be configured in accordance with format `$BASE_URL/$RELEASE_NUMBER/$METHOD_NAME` , where

- `$BASE_URL` represents the base URL of the SAS or the ESC.
- `$RELEASE_NUMBER` represents the CBRS release number corresponding to the ESC Requirements developed by WinnForum [n.1]. In this specification, `$RELEASE_NUMBER` shall be “v1.3”.
- `$METHOD_NAME` represents method name of the API specified in this document and corresponding to the specific SAS-ESC message. The methods in the following table shall be used in the API.

Table 1: List of methods in API

SAS-ESC message name	$METHOD_NAME
SAS Registration Message	sasRegistration
ESC Information Update Message	escUpdate
DPA Activation Status Message	dpaStatusMessage
Keep Alive Message	keepAlive
SAS Deregistration Message	sasDeregistration

Success or failure of any request shall be indicated by using appropriate HTTP status codes.

Table 2: Response status codes

HTTP Status	Meaning	Condition
200	SUCCESS	Request successfully processed
400	BAD REQUEST	Request contains invalid or malformed parameters
404	NOT FOUND	Invalid or malformatted URL

SAS-ESC Messages

Message Container

All the SAS-ESC Messages shall be generated by encoding the MessageContainer object using JSON. The MessageContainer object is a Flattened JSON Web Encryption (JWE) object as per RFC 7516 § 7.2.2, using a JSON Web Signature-signed data as its payload. This container structure ensures:

- integrity-protection (JWS): the receiver will be able to validate the signature using the counterparty's public key
- confidentiality (JWE): only the intended receiver will be able to decrypt the message using their private key
- assurance of the sender identity (JWS): only the holder of the sender private key (the sender) can sign the message
- a single message recipient (due to the Flattened JWE representation)
- a single message signer (the to the Flattened JWS representation)

Table 1: MessageContainer object

Field	Type	R/O/C	Descriptions
protected	base64url-encoded JOSEHeader object	Required	The value of this parameter is base64url-encoded JOSE header that provides sufficient information to enable the recipient of the message to decrypt the message
header	object JOSEHeader	Optional	JOSEHeader that may be used for transfering additional metadata that do not require integrity protection or confidentiality
encrypted_key	string	Required	The base64url-encoded bytes of the encrypted symemtric key
aad	string	Optional	Base64url-encoded bytes of the Additional Authenticated Data
iv	string	Required	Base64url-encoded Initialization Vector (IV) bytes. A random IV MUST be generated for each message
ciphertext	string	Required	Base64url-encoded SignedMessage object
tag	string	Required	Base64url-encoded thentication tag. The Authentication Tag is an additional output provided by Authenticated Encryption methods or Authenticated Encryption with Authenticated Data

Table 2: SignedMessage object

Field	Type	R/O/C	Descriptions
protected	string	Required	The value of this parameter is base64url-encoded JOSE header that provides sufficient information to enable the recipient of the message to validate the signature
header	object JOSEHeader	Optional	An optional JOSEHeader that may be used for transfering additional metadata that do not require integrity protection
payload	string	Required	The value of this field is the base64url-encoded JSON representation of the payload objects listed in section Payload Data
signature	string	Required	This parameter contains the digital signature encoded in Base64URL encoding. This parameter is calculated by taking the BASE64 encoding of the digital signature applied to the Payload Data and Protected header, prepared according to the Example JWK procedures in Section 3 of RFC 7515 which is reproduced below, using the algorithm as declared in the protected header "alg" field.

JSON representation of the payload objects listed in section Payload Data

Base64 and Base64URL encodings are described in RFC 4648 [n.8].

Samples

JSON Web Encryption object in Flattened JWE Serialization

```
{
    "protected":"<integrity-protected header contents>",
    "unprotected":<non-integrity-protected header contents>,
    "header":<more non-integrity-protected header contents>,
    "encrypted_key":"<encrypted key contents>",
    "aad":"<additional authenticated data contents>",
    "iv":"<initialization vector contents>",
    "ciphertext":"<ciphertext contents>",
    "tag":"<authentication tag contents>"
}
```

JSON Web Signature object in Flattened JWS Serialization

```
{
    "payload":"<payload contents>",
    "protected":"<integrity-protected header contents>",
    "header":<non-integrity-protected header contents>,
    "signature":"<signature contents>"
}
```

Payload Data

Payload Data for SAS Registration Message

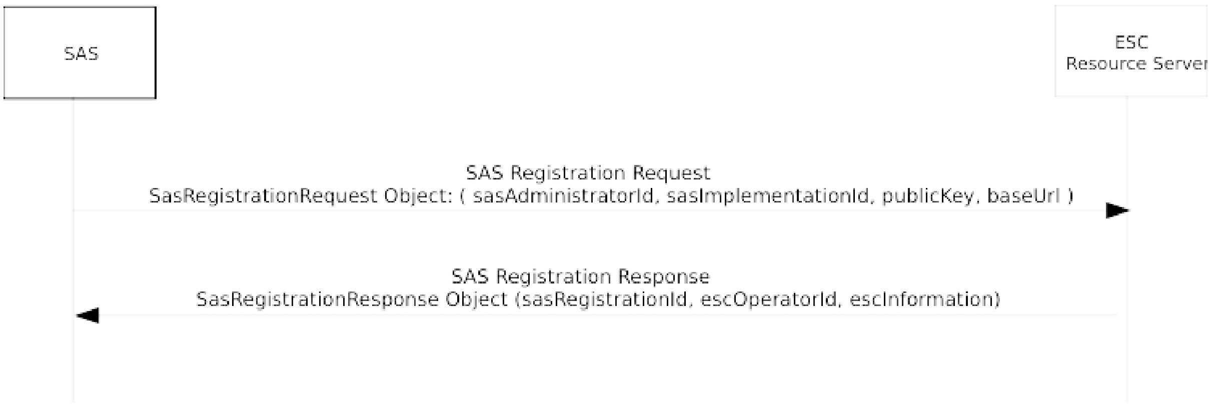
SAS
ESC
Resource Server
SAS Registration Request
SasRegistrationRequest Object: (sasAdministratorId, sasImplementationId, publicKey, baseUrl)
SAS Registration Response
SasRegistrationResponse Object (sasRegistrationId, escOperatorId, escInformation)

Figure: *SAS Registration Request & Response*

SAS Registration Request

SAS Registration Request shall be sent by the SAS to register with the ESC. The SAS Registration Request may be sent also when the SAS information in the *SasRegistrationRequest* object in the following table is updated. The SAS Registration Request shall be generated by encoding the *MessageContainer* object in which JSON-encoded *SasRegistrationRequest* object is used to generate *encodedPayloadData* field using JSON.

Table 3: SasRegistrationRequest object

Field	Type	R/O/C	Descriptions
sasAdministratorId	string	Required	This field shall be included to indicate which SAS Administrator manages the SAS. The format of this field shall be same as $ADMINISTRATOR_ID used in the SAS-SAS Protocol.
sasImplementationId	string	Required	This field shall be included to indicate the identification of the SAS. The format of this field shall be same as $SAS_IMPLEMENTATION used in the SAS-SAS Protocol.
publicKey	string	Required	This field shall be included to indicate the public key of SAS.
baseUrl	string	Required	This field shall be included to indicate the base URL ($BASE_URL) of the SAS identified by the *sasImplementationId* field in this object.

SAS Registration Response

SAS Registration Response shall be sent by the ESC to the SAS for the response to the SAS Registration Request. The SAS Registration Response shall be generated by encoding the *MessageContainer* object in which JSON-encoded *SasRegistrationResponse* object is used to generate *encodedPayloadData* field. in the following table using JSON.

Table 4: SasRegistrationResponse object

Field	Type	R/O/C	Descriptions
sasRegistrationId	string	Required	This field shall be generated by the ESC and included to indicate the registration identifier for the SAS.
escOperatorId	string	Required	This field shall be included to indicate the identification of ESC Operator managing ESC. The format of this field shall be FFS.

escInformation	object EscInformation	Required	This field shall be included to indicate the ESC information.

Table 5: EscInformation object

Field	**Type**	**R/O/C**	**Descriptions**
escImplementationId	string	Required	This field shall be included to indicate the identification of the ESC implementation. The format of this field shall be FFS.
escSensors	array of object EscSensorData[5]	Required	This field shall be included to indicate the information of ESC sensors deployed by the ESC Operator and managed by the ESC identified by the *escOperatorId* and *escImplementationId* fields in this object, respectively. See details in 7.1.2.1.

```
 Editor’s Note: Should the format of each ID be “ESC-CA certified unique ESC Operator identifier” similar to both
SAS Administrator and Implementation ID?
```

```
 Editor’s Note: “escImplementationId” is included here for the similar purpose of “SAS Implementation” in SAS-SAS.
In other words, one or more ESC instances might be operated. Decision to remove or keep this field strongly depends
on Key Bridge for now.
```

Enhancements to EscSensorData object for SAS-ESC Interface

In this specification, the EscSensorData object specified in the SAS-SAS Protocol [n.5] shall be reused with enhancements. Enhanced definition of the EscSensorData object is described in the following table.

Table 6: Enhanced definition of EscSensorData object

Field	Type	R/O/C	Descriptions
id	string	N/A	This field shall not be included.
sensorId	string	Required	This field shall be included to indicate a unique identifier of the ESC Sensor.
installationParam	object InstallationParam	N/A	This field shall not be included.
escInstallationParam	object EscInstallationParam	Required	This field shall be included to indicate the installation parameters of the ESC Sensor identified by thesensorIdfield in this object.
protectionLevel	number	Required	This field shall be included to indicate the protection level to be applied to the ESC Sensor identified by thesensorIdfield in this object. The value of this field shall be in units of dBm/MHz with decimal point.

Table 7: EscInstallationParam object

Field	Type	R/O/C	Descriptions
latitude	number	Required	Latitude of the ESC Sensor location in degrees relative to the WGS 84 datum. The allowed range is from -90.000000 to +90.000000. Positive values represent latitudes north of the equator; negative values south of the equator. Values are specified using 6 digits to the right of the decimal point.
longitude	number	Required	Longitude of the ESC Sensor location in degrees relative to the WGS84 datum. The allowed range is from -180.000000 to +180.000000. Positive values represent longitudes east of the prime meridian; negative values west of the prime meridian. Values are specified using 6 digits to the right of the decimal point.
height	number	Required	The antenna height of ESC Sensor in meters. When the heightType parameter value is “AGL”, the antenna height shall be given relative to ground level. When the heightType parameter value is “AMSL”, it is given with respect to WGS84 datum.
heightType	string	Required	The value shall be “AGL” or “AMSL”. AGL height is measured relative to the ground level. AMSL height is measured relative to the mean sea level.
horizontalAccuracy	number	Required	A positive number in meters to indicate accuracy of the ESC Sensor antenna horizontal location.
verticalAccuracy	number	Required	A positive number in meters to indicate accuracy of the ESC Sensor vertical location.
antennaAzimuth	number	Required	Boresight direction of the horizontal plane of the antenna in degrees with respect to true north. The value of this parameter is an integer with a value between 0 and 359 inclusive. A value of 0 degrees means true north; a value of 90 degrees means east.
			Antenna down tilt in degrees and is an integer with a value

antennaDowntilt	number	Required	between -90 and +90 inclusive; a negative value means the antenna is tilted up (above horizontal).
azimuthAntennaPattern	array of object AntennaPattern	Required	This parameter specifies anantenna pattern in any direction for the ESC Sensor antenna in the azimuthal plane.

Table 8: AntennaPattern object

Field	**Type**	**R/O/C**	**Description**
angle	number	Required	This is the angle **in the azimuth plane:**the value is given in degrees relative to the boresight of the antenna. The value of this parameter is an integer between 0 and 360 inclusive. **In the elevation plane**: the angle is given in degrees relative to the horizon. The value of this parameter is an integer between -180 and 180 inclusive.
gain	number	Required	The gain in dBi includes both antenna gain and beamforming gain. This parameter is an integer with a value between -127 and +128 (dBi). The gain provided is the gain in the direction of ‘angle’.

Payload Data for ESC Information Update Message

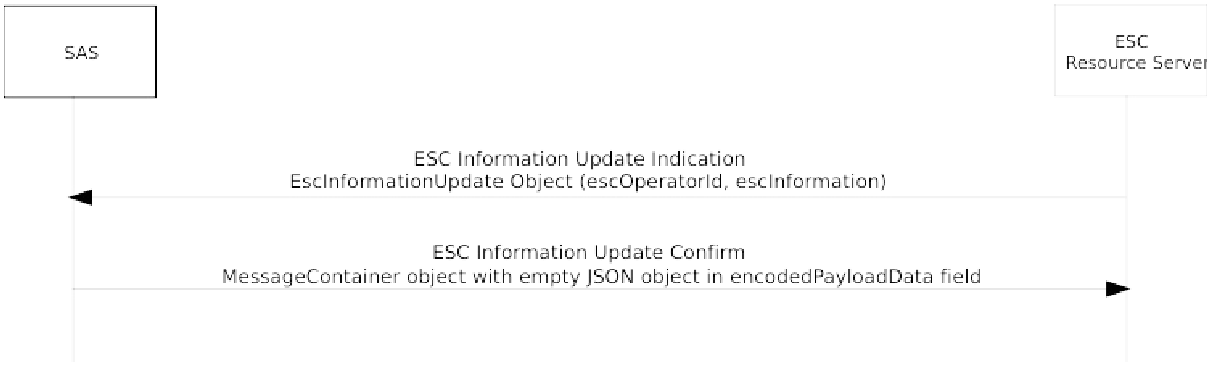
SAS
ESC
Resource Server
ESC Information Update Indication
EscInformationUpdate Object (escOperatorId, escInformation)
ESC Information Update Confirm
MessageContainer object with empty JSON object in encodedPayloadData field

Figure: *ESC Information Update Indication & Confirm*

ESC Information Update Indication

ESC Information Update Indication shall be sent by the ESC to the SAS when the ESC information is updated. The ESC Information Update Indication shall be generated by encoding the MessageContainer object in which JSON-encoded EscInformationUpdate object is used to generate encodedPayloadData field in the following table using JSON.

Table 9: EscInformationUpdate object

Field	Type	R/O/C	Description
escOperatorId	string	Required	This field shall be included to indicate the identification of ESC Operator managing ESC. The format of this field shall be FFS.
escInformation	object EscInformation	Required	This field shall be included to indicate the ESC information.

ESC Information Update Confirm

ESC Information Update Confirm shall be sent by the ESC to the SAS for the response to the ESC Information Update Indication. The ESC Information Update Confirm shall be generated by encoding the *MessageContainer* object in which empty JSON object (i.e. `{}`) is used to generate *encodedPayloadData* field.

Payload Data for DPA Activation Status Message

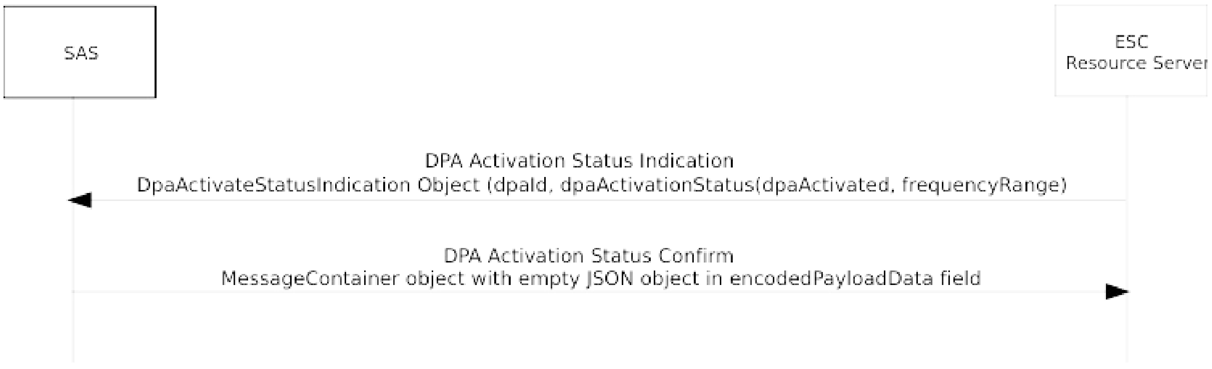
SAS
ESC
Resource Server
DPA Activation Status Indication
DpaActivateStatusIndication Object (dpaId, dpaActivationStatus(dpaActivated, frequencyRange)
DPA Activation Status Confirm
MessageContainer object with empty JSON object in encodedPayloadData field

Figure: *DPA Activation Status Indication & Confirm*

DPA Activation Status Indication

DPA Activation Status Indication shall be sent by the ESC to the SAS. The DPA Activation Status Indication shall be generated by encoding the *MessageContainer* object in which the *DpaActivateStatusIndication* object is used to generate *encodedPayloadData* field in the following table using JSON.

Table 10: DpaActivateStatusIndication object

Field	Type	R/O/C	Descriptions
dpaId	string	Required	This field shall be included to indicate which a unique identifier of the DPA.
dpaActivationStatus	object DpaActivationStatus	Required	This field shall be included to indicate the DPA activation status.

Table 11: DpaActivationStatus object

Field	Type	R/O/C	Descriptions
dpaActivated	boolean	Required	This field shall be included to indicate the DPA activation status of the frequency range indicated by the *frequencyRange* field in this object "true": DPA is (has been) activated "false": DPA is (has been) deactivated
frequencyRange	object FrequencyRange[6]	Required	This field shall be included to indicate the frequency range.

```
Editor’s Note: Atomic Transaction = 1 DPA + 1 Channel

Editor’s Note: Open Issues.

1: ID Assignment,

2: Partial coverages \(geometry and channel block\).
```

DPA Activation Status Confirm

DPA Activation Status Confirm shall be sent by the ESC to the SAS for the response to the DPA Activation Status Indication. The DPA Activation Status Confirm shall be generated by encoding the *MessageContainer* object in which empty JSON object (i.e. `{}`) is used to generate *encodedPayloadData* field.

Payload Data for Keep Alive Message

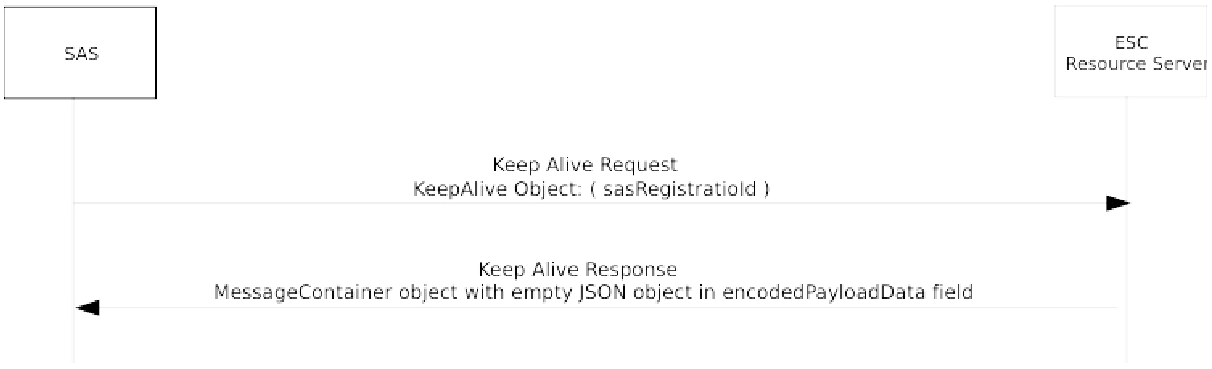
SAS
ESC
Resource Server
Keep Alive Request
KeepAlive Object: (sasRegistratioId)
Keep Alive Response
MessageContainer object with empty JSON object in encodedPayloadData field

Figure: *Keep Alive Indication & Confirm*

Keep Alive Indication

Keep Alive Indication shall be sent from the SAS to ESC. The Keep Alive Request shall be generated by encoding the MessageContainer object in which the *KeepAlive* object is used to generate encodedPayloadData field.in the following table using JSON.

Table 12: KeepAlive object

Field	**Type**	**R/O/C**	**Descriptions**
sasRegistrationId	string	Required	This field shall be included to indicate the registration identifier of the SAS.

Keep Alive Confirm

Keep Alive Confirm shall be sent from the ESC to the SAS for the response to the Keep Alive Request. The Keep Alive Response shall be generated by encoding the MessageContainer object in which empty JSON object (i.e. `{}`) is used to generate encodedPayloadData field.

Payload Data for SAS Deregistration Message

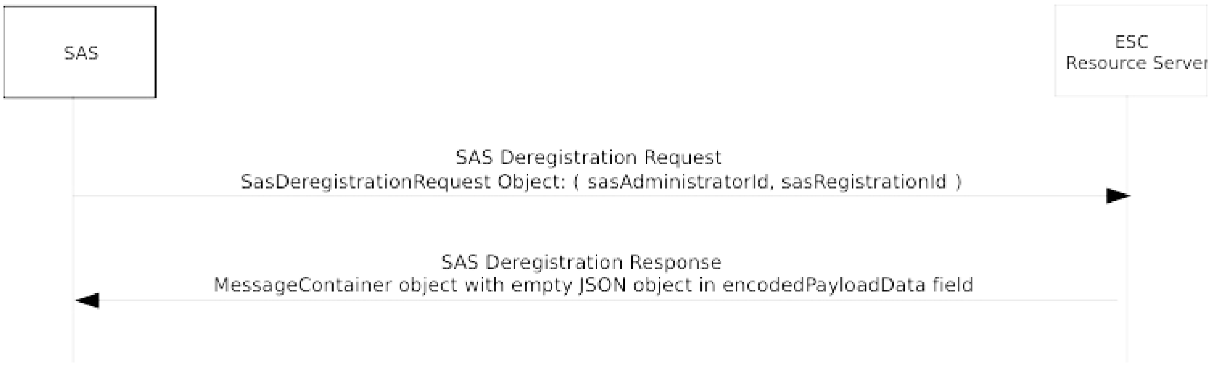
SAS
ESC
Resource Server
SAS Deregistration Request
SasDeregistrationRequest Object: (sasAdministratorId, sasRegistrationId)
SAS Deregistration Response
MessageContainer object with empty JSON object in encodedPayloadData field

Figure: *SAS Deregistration Request & Response*

SAS Deregistration Request

SAS Deregistration Request may be sent by the SAS to deregister from the ESC when the SAS wants to stop receiving indications from the ESC. The SAS Deregistration Request shall be generated by encoding the MessageContainer object in which the SasDeregistrationRequest object is used to generate encodedPayloadData field using JSON.

Table 13: SasDeregistrationRequest object

Field	Type	R/O/C	Descriptions
sasAdministratorId	string	Required	This field shall be included to indicate which SAS Administrator manages the SAS. The format of this field shall be same as $ADMINISTRATOR_ID used in the SAS-SAS Protocol.
sasRegistrationId	string	Required	This field shall be generated by the ESC and included to indicate the registration identifier for the SAS.

SAS Deregistration Response

SAS Deregistration Response shall be sent by the ESC to the SAS for the response to the SAS Deregistration Request. The SAS Deregistration Response shall be generated by encoding the MessageContainer object in which empty JSON object (i.e. `{}`) is used to generate encoded PayloadData field.

Annex A (Normative) DPA State Machine

SAS and ESC employing the SAS-ESC Protocol specified in this present document shall consider the DPA State Machine. This DPA State Machine shall be considered in each channel per DPA. The figure below shows the DPA State Machine in a channel per DPA.

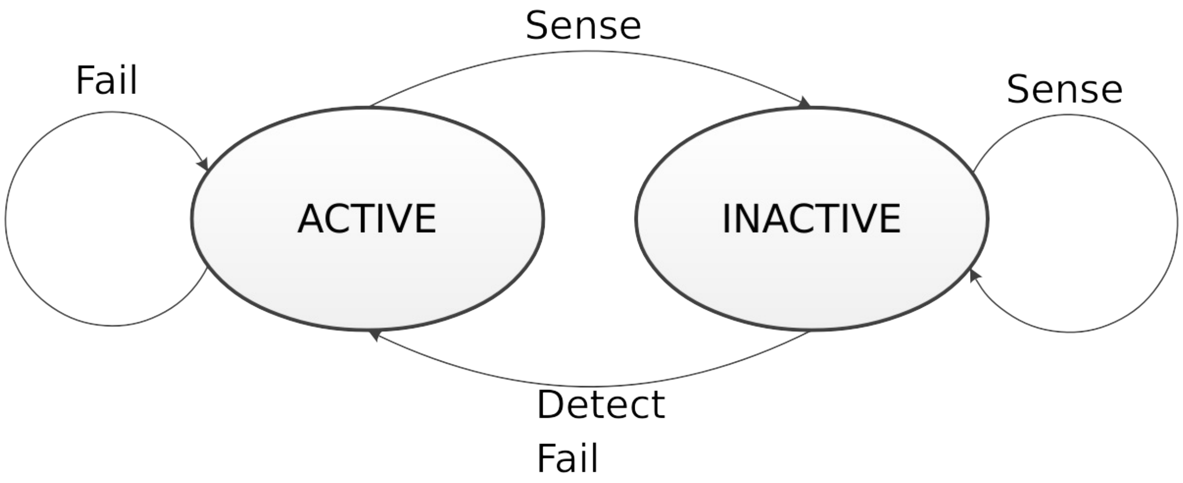

Figure 2: DPA State Machine in a channel per DPA

The DPA State shall be defined as follows:

- **ACTIVE**: “ACTIVE” refers to the state of a DPA in which the ESC detects the presence of the incumbent, in which two hours haves not passed after the ESC senses disappearance of the presence of any incumbents, or in which CBSDs are forbidden to use .a channel in its Neighborhood Area regardless of the presence of the incumbents. Both SAS and ESC shall consider “ACTIVE” the initial state of a DPA.
- **INACTIVE**: “INACTIVE” refers to the state of a DPA in which CBSDs are allowed to use. a channel in its Neighborhood Area, in which two hours have passed after the ESC sensed disappearance of the presence of any incumbents when the DPA was ACTIVE State, or in which the ESC senses no presence of the incumbents.

The following trigger events shall be applied to the DPA State Transitions:

- **Sense**: “Sense” refers to an event in which the ESC senses no presence of the incumbents or in which two hours has passed after the ESC sensed disappearance of the presence of the incumbent. All the DPA States shall transition to “INACTIVE” after this trigger event.
- **Detect**: “Detect” refers to an event in which the ESC detects the presence of the incumbent in the DPA. All the DPA States shall transition to “ACTIVE” after this trigger event.
- **Fail**: “Fail” refers to an event in which the SAS detects any failures in the Keep Alive Message exchange with the ESC. In particular, if the SAS doesn’t receive any Keep Alive Response for more than x seconds after sending a Keep Alive Message, it shall move to the “Fail” state. All the DPA States shall transition to “ACTIVE” after this trigger event.

Security Concerns

Implicit Grant

Implicit grants improve the responsiveness and efficiency of some clients (such as a client implemented as an in-browser application), since it reduces the number of round trips required to obtain an access token. However, this convenience should be weighed against the security implications of using implicit grants, such as those described in Sections *Access Tokens* and *Misuse of Access Token to Impersonate Resource Owner in Implicit Flow*, especially when the authorization code grant type is available.

TLS Version

Whenever Transport Layer Security (TLS) is used by this specification, the appropriate version (or versions) of TLS will vary over time, based on the widespread deployment and known security vulnerabilities. At the time of this writing, TLS version 1.2 [RFC5246] is the most recent version, but has a very limited deployment base and might not be readily available for implementation. TLS version 1.0 [RFC2246] is the most widely deployed version and will provide the broadest interoperability.

Implementations MAY also support additional transport-layer security mechanisms that meet the requirements of this protocol, as defined in Message Encoding and Transport.

Interoperability

OAuth 2.0 provides a rich authorization framework with well-defined security properties. However, as a rich and highly extensible framework with many optional components, on its own, this specification is likely to produce a wide range of non-interoperable implementations.

In addition, this specification leaves a few required components partially or fully undefined (e.g., client registration, authorization server capabilities, endpoint discovery). Without these components, clients must be manually and specifically configured against a specific authorization server and resource server in order to interoperate.

This framework was designed with the clear expectation that future work will define prescriptive profiles and extensions necessary to achieve full web-scale interoperability.

Access Tokens

Access token credentials (as well as any confidential access token attributes) MUST be kept confidential in transit and storage, and only shared among the authorization server, the resource servers the access token is valid for, and the client to whom the access token is issued. Access token credentials MUST only be transmitted using TLS as described in Section *"TLS Version"* above with server authentication as defined by [RFC2818].

When using the implicit grant type, the access token is transmitted in the URI fragment, which can expose it to unauthorized parties.

The authorization server MUST ensure that access tokens cannot be generated, modified, or guessed to produce valid access tokens by unauthorized parties.

The client SHOULD request access tokens with the minimal scope necessary. The authorization server SHOULD take the client identity into account when choosing how to honor the requested scope and MAY issue an access token with less rights than requested.

This specification does not provide any methods for the resource server to ensure that an access token presented to it by a given client was issued to that client by the authorization server.

Credentials-Guessing Attacks

The authorization server MUST prevent attackers from guessing access tokens, authorization codes, refresh tokens, resource owner passwords, and client credentials.

The probability of an attacker guessing generated tokens (and other credentials not intended for handling by end-users) MUST be less than or equal to 2^(-128) and SHOULD be less than or equal to 2^(-160).

The authorization server MUST utilize other means to protect credentials intended for end-user usage.

Cross-Site Request Forgery

Cross-site request forgery (CSRF) is an exploit in which an attacker causes the user-agent of a victim end-user to follow a malicious URI (e.g., provided to the user-agent as a misleading link, image, or redirection) to a trusting server (usually established via the presence of a valid session cookie).

A CSRF attack against the client's redirection URI allows an attacker to inject its own authorization code or access token, which can result in the client using an access token associated with the attacker's protected resources rather than the victim's (e.g., save the victim's bank account information to a protected resource controlled by the attacker).

The client MUST implement CSRF protection for its redirection URI. This is typically accomplished by requiring any request sent to the redirection URI endpoint to include a value that binds the request to the user-agent's authenticated state (e.g., a hash of the session cookie used to authenticate the user-agent). The client SHOULD utilize the "state" request parameter to deliver this value to the authorization server when making an authorization request.

Once authorization has been obtained from the end-user, the authorization server redirects the end-user's user-agent back to the client with the required binding value contained in the "state" parameter. The binding value enables the client to verify the validity of the request by matching the binding value to the user-agent's authenticated state. The binding value used for CSRF protection MUST contain a non-guessable value (as described in Credentials-Guessing Attacks), and the user-agent's authenticated state (e.g., session cookie, HTML5 local storage) MUST be kept in a location accessible only to the client and the user-agent (i.e., protected by same-origin policy).

A CSRF attack against the authorization server's authorization endpoint can result in an attacker obtaining end-user authorization for a malicious client without involving or alerting the end-user.

The authorization server MUST implement CSRF protection for its authorization endpoint and ensure that a malicious client cannot obtain authorization without the awareness and explicit consent of the resource owner.

Misuse of Access Token to Impersonate Resource Owner in Implicit Flow

For public clients using implicit flows, this specification does not provide any method for the client to determine what client an access token was issued to.

A resource owner may willingly delegate access to a resource by granting an access token to an attacker's malicious client. This may be due to phishing or some other pretext. An attacker may also steal a token via some other mechanism. An attacker may then attempt to impersonate the resource owner by providing the access token to a legitimate public client.

In the implicit flow (response_type=token), the attacker can easily switch the token in the response from the authorization server, replacing the real access token with the one previously issued to the attacker.

Servers communicating with native applications that rely on being passed an access token in the back channel to identify the user of the client may be similarly compromised by an attacker creating a compromised application that can inject arbitrary stolen access tokens.

Any public client that makes the assumption that only the resource owner can present it with a valid access token for the resource is vulnerable to this type of attack.

This type of attack may expose information about the resource owner at the legitimate client to the attacker (malicious client). This will also allow the attacker to perform operations at the legitimate client with the same permissions as the resource owner who originally granted the access token or authorization code.

Authenticating resource owners to clients is out of scope for this specification. Any specification that uses the authorization process as a form of delegated end-user authentication to the client (e.g., third-party sign-in service) MUST NOT use the implicit flow without additional security mechanisms that would enable the client to determine if the access token was issued for its use (e.g., audience- restricting the access token).

Use of application/x-www-form-urlencoded Media Type

Excerpted from Appendix B of OAuth 2.0 RFC at https://tools.ietf.org/html/rfc6749#appendix-B

> At the time of publication of OAuth 2.0 specification, the "application/x-www-form-urlencoded" media type was defined in Section 17.13.4 of W3C.REC-html401-19991224 but not registered in the IANA MIME Media Types registry (http://www.iana.org/assignments/media-types). Furthermore, that definition is incomplete, as it does not consider non-US-ASCII characters.
>
> To address this shortcoming when generating payloads using this media type, names and values MUST be encoded using the UTF-8 character encoding scheme [RFC3629] first; the resulting octet sequence then needs to be further encoded using the escaping rules defined in [W3C.REC-html401-19991224].
>
> When parsing data from a payload using this media type, the names and values resulting from reversing the name/value encoding consequently need to be treated as octet sequences, to be decoded using the UTF-8 character encoding scheme.
>
> For example, the value consisting of the six Unicode code points
>
> ```
>      (1) U+0020 (SPACE), (2) U+0025 (PERCENT SIGN),
>      (3) U+0026 (AMPERSAND), (4) U+002B (PLUS SIGN),
>      (5) U+00A3 (POUND SIGN), and (6) U+20AC (EURO SIGN)
> ```
>
> ```
>    would be encoded into the octet sequence below (using hexadecimal notation):
> ```
>
> ```
>      20 25 26 2B C2 A3 E2 82 AC
> ```
>
> and then represented in the payload as:
>
> ```
>      +%25%26%2B%C2%A3%E2%82%AC
> ```

Abbreviations

For the purposes of the present document, the following abbreviations apply:

Abbreviation	**Full Form**
API	Application Programming Interface
CBRS	Citizens Broadband Radio Service
CONOPS	Concept of Operations
DPA	Dynamic Protection Area
EIRP	Effective Isotropic Radiated Power
ESC	Environmental Sensing Capability
FFS	For Further Study
FCC	Federal Communications Commission
JSON	JavaScript Object Notation
HTTP	Hyper Text Transfer Protocol
HRRPS	HTTP plus TLS Protocol
NIIU	Non-Informing Incumbent User; information must be learned
RAT	Radio Access Technology
SAS	Spectrum Access System
TLS	Transfer Layer Security
URL	Uniform Resource Locator
WinnForum	Wireless Innovation Forum

Definitions

Citizens Broadband Radio Service (CBRS)

Wireless operations authorized by the U.S. Federal Communications Commission (FCC) in the 3,550-3,700 MHz frequency band. The CBRS includes Priority Access and General Authorized Access tiers of service.

Environmental Sensing Capability (ESC)

A system that detects and communicates the presence of a signal from an Incumbent User to an SAS to facilitate shared spectrum access consistent with 47 C.F.R. Part 96.

ESC Operator

A legal entity that the FCC has authorized to operate an ESC.

Incumbent User

A federal entity authorized to operate on a primary basis in accordance with the table of frequency allocations, a fixed satellite service operator, or a Grandfathered Wireless Broadband Licensee authorized to operate on a primary basis on frequencies designated in section 96.11.

Protection

To avoid harmful interference from lower-tier user(s) to the upper tier user(s).

SAS Administrator

A legal entity that the FCC has authorized to administer the operation of a SAS.

Spectrum Access System (SAS)

A system that authorizes and manages the use of spectrum for the CBRS in accordance with subpart F of 47 C.F.R. Part 96.

References

WINNF-TS-0016

[n.6] “WINNF-TS-0016 Signaling Protocols and Procedures for Citizens Broadband Radio Service (CBRS): Spectrum Access System (SAS) - Citizens Broadband Radio Service Device (CBSD) Interface Technical Specification v1.1.0”, Wireless Innovation Forum, 18 July 2017

WINNF-TS-0065

[n.3] “WINNF-TS-0065 CBRS Communications Security Technical Specification v1.1.0”, Wireless Innovation Forum, 26 July 2017

SAS-SAS Protocol

[n.5] “WINNF-TS-0096 Signaling Protocols and Procedures for Citizens Broadband Radio Service (CBRS): Spectrum Access System (SAS) - SAS Interface Technical Specification v1.1.0”, Wireless Innovation Forum, 1 August 2017 https://workspace.winnforum.org/higherlogic/ws/public/document?document_id=4517&wg_abbrev=SSC

WINNF-TS-0112-V1.3.0

[n.0] “WINNF-TS-0112 Requirements for Commercial Operation in the U.S. 3550-3700 MHz Citizens Broadband Radio Service Band v1.3.0”, Wireless Innovation Forum, 27 September 2017 https://workspace.winnforum.org/higherlogic/ws/public/document?document_id=4743

RFC-2119

[n.2] Key words for use in RFCs to Indicate Requirement Levels”, March 1997 https://tools.ietf.org/html/rfc2119

RFC 4648

[n.8] The Base16, Base32, and Base64 Data Encodings
https://tools.ietf.org/html/rfc4648

RFC 6749

OAuth 2.0 for automated identification and token rotation
https://tools.ietf.org/html/rfc6749

RFC 7159

The JavaScript Object Notation (JSON) Data Interchange Format
https://tools.ietf.org/html/rfc7159

RFC 7515

[n.7] JSON Web Signature (JWS)
https://tools.ietf.org/html/rfc7515

RFC 7516

JSON web encryption
https://tools.ietf.org/html/rfc7516

RFC 7517

JSON web key
https://tools.ietf.org/html/rfc7517

RFC 7518

JSON web algorithm
https://tools.ietf.org/html/rfc7518

RFC 7519

JSON web token
https://tools.ietf.org/html/rfc7519

RFC 7797

JWS for unencoded payload
https://tools.ietf.org/html/rfc7797