A. Tulshibagwale
Google
M. Scurtescu
Coinbase
A. Backman
Amazon
J. Bradley
Yubico
May 11, 2020

OpenID Shared Signals and Events Profile of IETF Security Events 2.0
openid-sse-profile-2_0

Abstract

This Shared Signals and Events (SSE) profile spec combines the Open-ID RISC Profile [RISC] with the Continuous Access Evaluation Protocol ([CAEP])

It is a general profile for IETF Security Events [1] and it defines:

- Subject Principals
- Subject Identifiers
- Event Types
- Event Properties
- Configuration information and discovery method for Transmitters and Receivers
- a Management API for Event Streams

This spec also directly profiles several IETF Security Events drafts:

- Security Event Token (SET) [SET]
- Push-Based SET Token Delivery Using HTTP [DELIVERYPUSH]
- Poll-Based SET Token Delivery Using HTTP [DELIVERYPOLL]

Table of Contents

1. Introduction

1.1. Notational Conventions

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

2. Profile Version

The following profile versions are recognized in this spec:

2.0
The profile defined in this spec

1.0
The [RISC] profile of Security Events

3. Subject Principals

This SSE profile defines a Subject to be the entity about which an event can be sent by Transmitters and received by Receivers using the SSE profile.

Subject Principals are the management entities associated with Subjects. These may be human or robotic principals or devices or other entities that are managed by Transmitters and Receivers. For example, the same Subject Principal may be referred to by an email address or a phone number. A transmitter or receiver will typically manage subject principals and organize them into sub-groupings such as tenants, groups, organizational units, etc.

Subjects are identified by Subject Identifiers defined below.

3.1. SPAGs

Subject Principals MAY be grouped for administrative purposes into "Subject Principal Administrative Groupings" or SPAGs. For example, all users belonging to one customer of a "multi-tenanted" Transmitter may be in one SPAG. Alternatively, an Organizational Unit or a group of Subjects within a customer of a Transmitter (whether multi-tenanted or not) may be one SPAG. SPAGs may have overlapping sets of Subjects.

A SPAG MAY be the Subject of certain stream update events defined later in this spec, hence is defined as a Subject Type of its own below. A SPAG identifier MAY be included in other subject types to disambiguate the subject.

A SPAG subject identifier is agreed to offline between a Transmitter and a Receiver.

4. Subject Identifiers

A Subject is identified by a structure called a Subject Identifier: a JSON [RFC7159] object containing a set of claims that collectively uniquely identify a subject, according to a simple schema called a Subject Identifier Type.

Every Subject Identifier MUST contain a "subject_type" claim, whose value MUST be a name that uniquely identifies the Subject Identifier Type of the Subject Identifier. Any remaining claims within the

Subject Identifier MUST be interpreted according to the common claims or the definition of the Subject Identifier Type.

4.1. Common Claims in Subject Identifiers

The following claims MAY be present in a subject of any subject type.

4.1.1. SPAG ID

Subjects MAY be identified uniquely without referring to which SPAG they belong to. A transmitter MAY include a SPAG identifier as a part of the subject of any other subject type. If a SPAG identifier claim is included in a SET, then the name of the claim MUST be: "spag_id".

4.1.2. Subject Categories

Subjects may be categorized as users, devices or sessions. To specify the category of a subject, a "category" claim MAY be included. If present, the claim MUST have a value that is one of:

user Specifies that the subject category is a user.

device Specifies that the subject category is a device.

session Specifies that the subject category is a session.

If the "category" claim is not present, the category is assumed to be "user", unless a different default interpretation is specified in the subject type description.

The subject category MUST be used by the Receiver to determine the scope of the event.

4.2. Subject Conformance

A Subject Identifier MUST conform to the Subject Identifier Type identified by its "subject_type" claim, and MUST NOT contain any claims other than the common claims (Section 4.1) and those that are defined by its Subject Identifier Type.

5. Event Properties

Additional claims about an event may be included in the "events" claim. Some of these claims are required and specified as such in the event types spec. (TODO: add reference). If a Transmitter determines that it needs to include additional claims that are not specified in the event types spec, then the name of such claims MUST

be a URI. The discoverability of all additional claims is specified in the Discovery (Section 7) section.

6. Example SETs that conform to SSE profile

The following are hypothetical examples of SETs that conform to the SSE profile.

```
{
  "iss": "https://idp.example.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/risc/event-type/account-\
enabled": {
      "subject": {
        "subject_type": "email",
        "email": "foo@example.com",
      }
    }
  }
}
```

Figure 1: Example: SET Containing a SSE Event with an Email Subject Identifier

```
{
  "iss": "https://idp.example.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/risc/event-type/account-\
enabled": {
      "subject": {
        "subject_type": "iss_sub",
        "iss": "https://issuer.example.com/",
        "sub": "abc1234"
      }
    }
  }
}
```

Figure 2: Example: SET Containing a SSE Event with an Issuer and Subject Identifier

```
{
  "iss": "https://sp.example2.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/ip-address-changed": {
      "subject": {
        "subject_type": "email",
        "email": "foo@example.com",
      }
      "ip_address" : "123.45.67.89"
    }
  }
}
```


Figure 3: Example: SET Containing a SSE Event with a Subject and a Property claim


```
{
  "iss": "https://sp.example2.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/ip-address-changed": {
      "subject": {
        "subject_type": "spag",
        "spag_id" : "https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a",
      }
    }
  }
}
```


Figure 4: Example: SET Containing a SSE Event with a SPAG Subject Type

```
{
  "iss": "https://sp.example2.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/ip-address-changed": {
      "subject": {
        "subject_type": "id_token_claims",
        "cateogry": "device",
        "spag_id" : "https://example.com/v2/Groups/e9e30dba-f08f-4109-8486-d5c6a331660a",
        "phone_number": "+1 (408) 555-1212",
      }
    }
  }
}
```


Figure 5: Example: SET Containing a SSE Event with Common Claims in the Subject

6.1. Subject Identifier Types

A Subject Identifier Type is a light-weight schema that describes a set of claims that uniquely identifies a subject. Every Subject Identifier Type MUST have a unique name registered in the IANA "Security Event Subject Identifier Types" registry established by Section 10.1. A Subject Identifier Type MAY contain more claims than are strictly necessary to uniquely identify a subject, however a Subject Identifier MUST contain all of the claims required by its type (See Privacy Considerations).

The following Subject Identifier Types are registered in the IANA "Security Event Subject Identifier Types" registry established by Section 10.1.

6.1.1. Email Subject Identifier Type

The Email Subject Identifier Type describes a subject by email address as defined in sections 3.2.3 and 3.2.4 of the Internet Message Format [RFC5322] specification. Subject Identifiers of this type MUST contain an "email" claim whose value is a string containing the email address of the subject. The "email" claim MUST NOT be null or empty. The Email Subject Identifier Type is identified by the name "email".

6.1.2. Phone Number Subject Identifier Type

The Phone Number Subject Identifier Type describes a subject by telephone number. Subject Identifiers of this type MUST contain a "phone" claim whose value is a string containing the full telephone number of the subject, including international dialing prefix, formatted according to E.164 [E164]. The "phone" claim MUST NOT be null or empty. The Phone Number Subject Identifier Type is identified by the name "phone".

6.1.3. Issuer and Subject Subject Identifier Type

The Issuer and Subject Subject Identifier Type describes a subject by an issuer and a subject. Subject Identifiers of this type MUST contain an "iss" claim whose value identifies the issuer, and a "sub" claim whose value identifies the subject with respect to the issuer. These claims MUST follow the formats of the "iss" claim and "sub" claim defined by [RFC7519], respectively. Both the "iss" claim and the "sub" claim MUST NOT be null or empty. The Issuer and Subject Subject Identifier Type is identified by the name "iss_sub".

6.1.4. SPAG Subject Identifier Type

The SPAG Subject Identifier Type describes a Subject Principal Administrative Grouping (SPAG (Section 3.1)). Subject Identifiers of this type MUST contain a "spag_id" claim, a unique identifier agreed to offline between a Transmitter and a Receiver that identifies a SPAG uniquely within the Transmitter. The "spag_id" claim MUST NOT be null or empty. The SPAG Subject Identifier Type is identified by the name "spag".

6.1.5. ID Token Claims Subject Identifier Type

The ID Token Claims Subject Identifier Type describes a subject by a subset of the claims from an ID token. Subject Identifiers of this type MUST contain at least one of the following claims:

email
: An "email" claim, as defined in [IDTOKEN].

phone_number
: An "phone_number" claim, as defined in [IDTOKEN].

sub
: A "sub" claim, as defined in [RFC7519].

If the Subject Identifier contains a "sub" claim, it MUST also contain an "iss" claim, as defined in [RFC7519]. The ID Token Claims Subject Identifier Type is identified by the name "id_token_claims".

If the category (Section 4.1.2) claim is not specified, then its value MUST be assumed to be "session"

6.1.6. SAML Subject Identifier Type

The SAML [SAML-CORE] Subject Identifier Type describes a subject by the assertion identifier in the SAML assertion that was used to convey the subject's information to the Receiver. Subject Identifiers of this type MUST contain an "assertion_id" claim. The value of this claim is a string that is equal to the Assertion Identifier in the SAML assertion.

7. Transmitter Configuration Discovery

This section defines a mechanism for Receivers to obtain Transmitter configuration information.

7.1. Transmitter Configuration Metadata

Transmitters have metadata describing their configuration:

issuer REQUIRED. URL using the https scheme with no query or fragment component that the Transmitter asserts as its Issuer Identifier. This MUST be identical to the iss claim value in Security Event Tokens issued from this Transmitter.

jwks_uri REQUIRED. URL of the Transmitter's JSON Web Key Set [RFC7517] document. This contains the signing key(s) the Receiver uses to validate signatures from the Transmitter.

supported_versions OPTIONAL. List of string representations of the numerical profile versions defined in Section 2 supported by this transmitter. If this element is missing in the Transmitter Configuration Metadata, then a single-valued list containing the value "1.0" MUST be assumed.

delivery_methods_supported RECOMMENDED. List of supported delivery method URIs.

configuration_endpoint OPTIONAL. The URL of the Configuration Endpoint.

status_endpoint OPTIONAL. The URL of the Status Endpoint.

add_subject_endpoint OPTIONAL. The URL of the Add Subject Endpoint.

remove_subject_endpoint OPTIONAL. The URL of the Remove Subject Endpoint.

verification_endpoint OPTIONAL. The URL of the Verification Endpoint.

TODO: consider adding a IANA Registry for metadata, similar to Section 7.1.1 of [OAUTH-DISCOVERY]. This would allow other specs to add to the metadata.

7.2. Obtaining Transmitter Configuration Information

Using the Issuer as documented by the Transmitter, the Transmitter Configuration Information can be retrieved.

Transmitters supporting Discovery MUST make a JSON document available at the path formed by inserting the string "/.well-known/sse-configuration" into the Issuer between the host component and the path component, if any. The syntax and semantics of ".well-known" are defined in [RFC5785]. "sse-configuration" MUST point to a JSON document compliant with this specification and MUST be returned using the "application/json" content type.

7.2.1. Transmitter Configuration Request

A Transmitter Configuration Document MUST be queried using an HTTP "GET" request at the previously specified path.

The Receiver would make the following request to the Issuer "https://tr.example.com" to obtain its Configuration information, since the Issuer contains no path component:

```
GET /.well-known/sse-configuration HTTP/1.1
Host: tr.example.com
```

Figure 6: Example: Transmitter Configuration Request (without path)

If the Issuer value contains a path component, any terminating "/" MUST be removed before inserting "/.well-known/sse-configuration" between the host component and the path component. The Receiver would make the following request to the Issuer "https://tr.example.com/issuer1" to obtain its Configuration information, since the Issuer contains a path component:

```
GET /.well-known/sse-configuration/issuer1 HTTP/1.1
Host: tr.example.com
```

Figure 7: Example: Transmitter Configuration Request (with path)

Using path components enables supporting multiple issuers per host. This is required in some multi-tenant hosting configurations. This use of ".well-known" is for supporting multiple issuers per host; unlike its use in [RFC5785], it does not provide general information about the host.

7.2.2. Transmitter Configuration Response

The response is a set of Claims about the Transmitter's configuration, including all necessary endpoints and public key location information. A successful response MUST use the 200 OK HTTP status code and return a JSON object using the "application/json" content type that contains a set of Claims as its members that are a subset of the Metadata values defined in Section 7.1. Other Claims MAY also be returned.

Claims that return multiple values are represented as JSON arrays. Claims with zero elements MUST be omitted from the response.

An error response uses the applicable HTTP status code value.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
  "issuer":
    "https://tr.example.com",
  "jwks_uri":
    "https://tr.example.com/jwks.json",
  "delivery_methods_supported": [
    "https://schemas.openid.net/secevent/risc/delivery-method/push",
    "https://schemas.openid.net/secevent/risc/delivery-method/poll"],
  "configuration_endpoint":
    "https://tr.example.com/risc/mgmt/stream",
  "status_endpoint":
    "https://tr.example.com/risc/mgmt/status",
  "add_subject_endpoint":
    "https://tr.example.com/risc/mgmt/subject:add",
  "remove_subject_endpoint":
    "https://tr.example.com/risc/mgmt/subject:remove",
  "verification_endpoint":
    "https://tr.example.com/risc/mgmt/verification"
}
```

Figure 8: Example: Transmitter Configuration Response

7.2.3. Transmitter Configuration Validation

If any of the validation procedures defined in this specification fail, any operations requiring the information that failed to correctly validate MUST be aborted and the information that failed to validate MUST NOT be used.

The "issuer" value returned MUST be identical to the Issuer URL that was directly used to retrieve the configuration information. This MUST also be identical to the "iss" Claim value in Security Event Tokens issued from this Transmitter.

8. Management API for SET Event Streams

This section defines an HTTP API to be implemented by Event Transmitters and that can be used by Event Receivers to query and update the Event Stream configuration and status, to add and remove subjects and to trigger verification.

This section is based on Management API for SET Event Streams [MGMTAPI].

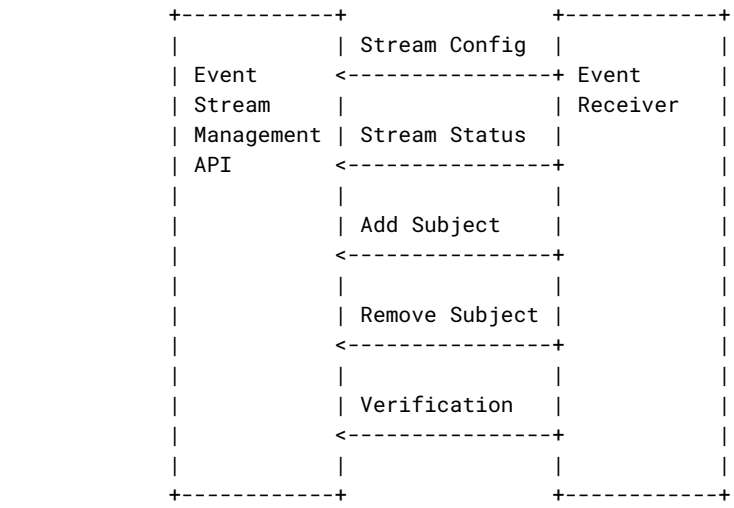

Figure 9: Event Stream Management API

It is OPTIONAL for Transmitters to implement a Management API, but it is RECOMMENDED that they implement it, especially the endpoints for querying the Stream Status and for triggering Verification.

8.1. Event Stream Management

Event Receivers manage how they receive events, and the subjects about which they want to receive events over an Event Stream by making HTTP requests to endpoints in the Event Stream Management API.

A Transmitter and Receiver MAY use the same Event Stream for updates about multiple SPAGs (Section 3.1). The status of the Event Stream MAY be queried and managed independently for each SPAG by Transmitters and Receivers.

The Event Stream Management API is implemented by the Event Transmitter and consists of the following endpoints:

Configuration Endpoint
: An endpoint used to read the Event Stream's current configuration.

Status Endpoint
: An endpoint used to read the Event Stream's current status.

Add Subject Endpoint
: An endpoint used to add subjects to an Event Stream.

Remove Subject Endpoint

An endpoint used to remove subjects from an Event Stream.

Verification Endpoint
An endpoint used to request the Event Transmitter transmit a Verification Event over the Event Stream.

An Event Transmitter MAY use the same URLs as endpoints for multiple streams, provided that the Event Transmitter has some mechanism through which they can identify the applicable Event Stream for any given request, e.g. from authentication credentials. The definition of such mechanisms is outside the scope of this specification.

Within an Event Stream, events related to Subject Principals belonging to different SPAGs MAY be managed independently. A Receiver MAY request all subject principals within a SPAG to be added to or removed from a stream by Updating the Stream Status (Section 8.1.1.2) and specifying the SPAG identifier in the request.

A Transmitter MAY decide to enable, pause or disable updates about a SPAG independently of an update request from a Receiver. If a Transmitter decides to start or stop events for a SPAG then the Transmitter MUST do the following according to the status of the stream

If the stream is:

Enabled the Transmitter MUST send a SPAG stream updated (Section 8.1.5) event respectively to the Receiver within the Event Stream.

Paused the Transmitter SHOULD send SPAG stream updated (Section 8.1.5) after the Event Stream is re-started.

Disabled the Transmitter MAY send SPAG stream updated (Section 8.1.5) after the Event Stream is re-enabled.

If a Subject Principal belongs to more than one SPAG, but the Event Stream status for those SPAGs differ, then the Transmitter MAY decide whether or not to include updates about that Subject Principal within the Event Stream.

8.1.1. Stream Status

8.1.1.1. Reading a Stream's Status

An Event Receiver checks the current status of an event stream by making an HTTP GET request to the stream's Status Endpoint.

An Event Receiver may append a SPAG identifier to the Stream Status URL to obtain the Stream Status for that SPAG.

On receiving a valid request the Event Transmitter responds with a 200 OK response containing a JSON [RFC7159] object with an attribute "status", whose string value MUST have one of the following values:

enabled
: The Transmitter MUST transmit events over the stream, according to the stream's configured delivery method.

paused
: The Transmitter MUST NOT transmit events over the stream. The transmitter will hold any events it would have transmitted while paused, and SHOULD transmit them when the stream's status becomes "enabled".

disabled
: The Transmitter MUST NOT transmit events over the stream, and will not hold any events for later transmission.

The following is a non-normative example request to check an event stream's status:

```
GET /set/status HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 10: Example: Check Stream Status Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "status": "enabled"
}
```

Figure 11: Example: Check Stream Status Response

```
GET /set/status/1234 HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 12: Example: Check Stream Status Request with SPAG ID

The following is a non-normative example response with a SPAG identifier:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "status": "enabled",
  "spag_id": "1234"
}
```

Figure 13: Example: Check Stream Status Response

Errors are signaled with HTTP status codes as follows:

Code	Description
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to read the stream status
404	if there is no Event Stream configured for this Event Receiver, or if the SPAG ID specified is invalid or if the Receiver is not authorized to get status for the specified SPAG ID.

Table 1: Read Stream Status Errors

Examples:

1. If a Receiver makes a request with an invalid OAuth token, then the Transmitter MUST respond with a 401 error status.

2. If the Receiver presents a valid OAuth token, but the Transmitter policy does not permit the Receiver from obtaining the status, then the Transmitter MAY respond with a 403 error status.

3. If the Receiver requests the status for a SPAG ID, but the Transmitter policy does not permit the Receiver to read the status, then the Transmitter MAY respond with a 404 error status in order to not reveal the policy decision.

4. If the specified SPAG ID is invalid then the Transmitter MUST respond with a 404 error status.

8.1.1.2. Updating a Stream's Status

An Event Receiver updates the current status of a stream by making an HTTP POST request to the Status Endpoint. The POST body contains a JSON [RFC7159] object with the following fields:

status
: REQUIRED. The new status of the Event Stream.

spag_id
: OPTIONAL. The SPAG ID to which the new status applies.

authorization
: OPTIONAL. An authorization token specific to this request. The properties of this token, such as format, lifetime and replayability, are determined by the Transmitter.

reason
: OPTIONAL. A short text description that explains the reason for the change.

On receiving a valid request the Event Transmitter responds with a "200 OK" response containing a JSON [RFC7159] representation of the updated stream status in the body.

The following is a non-normative example request to update an Event Stream's status:

```
POST /set/status HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "status": "paused"
}
```

Figure 14: Example: Update Stream Status Request Without Optional Fields

```
POST /set/status HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "status": "paused",
  "spag_id": "1234",
  "authorization": "A0AfH6SMA8kBiQ90qrydNVMKl6R-",
  "reason": "Disabled by administrator action."
}
```

Figure 15: Example: Update Stream Status Request With Optional Fields

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "status": "paused"
}
```

Figure 16: Example: Update Stream Status Response

Errors are signaled with HTTP status codes as follows:

Code	Description
202	if the update request has been accepted, but not processed. Receiver MAY try the same request later in order to get processing result.
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to update the stream status
404	if there is no Event Stream configured for this Event Receiver, or if an invalid SPAG ID is specified.

Table 2: Update Stream Status Errors

Example:

1. If a Receiver makes a request to update the stream to enable it for a specific SPAG ID, and the Transmitter is unable to decide whether or not to complete the request, then the Transmitter MUST respond with a 202 status code.

8.1.2. Stream Configuration

An Event Stream's configuration is represented as a JSON [RFC7159] object with the following properties:

iss
: *Read-Only*, A URL using the https scheme with no query or fragment component that the Transmitter asserts as its Issuer Identifier. This MUST be identical to the "iss" Claim value in Security Event Tokens issued from this Transmitter.

aud
: *Read-Only*, A string or an array of strings containing an audience claim as defined in JSON Web Token (JWT) [RFC7519] that identifies the Event Receiver(s) for the Event Stream. This property cannot be updated. If multiple Receivers are specified then the Transmitter SHOULD know that these Receivers are the same entity.

events_supported
: *Read-Only*, An array of URIs identifying the set of events supported by the Transmitter for this Receiver. If omitted, Event Transmitters SHOULD make this set available to the Event Receiver via some other means (e.g. publishing it in online documentation).

events_requested
: *Read-Write*, An array of URIs identifying the set of events that the Receiver requested. A Receiver SHOULD request only the events that it understands and it can act on. This is configurable by the Receiver.

events_delivered
: *Read-Only*, An array of URIs which is the intersection of "events_supported" and "events_requested". These events MAY be delivered over the Event Stream.

delivery
: *Read-Write*, A JSON object containing a set of name/value pairs specifying configuration parameters for the SET delivery method. The actual delivery method is identified by the special key "method" with the value being a URI as defined in Section 9.2.1.

min_verification_interval
Read-Only, An integer indicating the minimum amount of time in seconds that must pass in between verification requests. If an Event Receiver submits verification requests more frequently than this, the Event Transmitter MAY respond with a 429 status code. An Event Transmitter SHOULD NOT respond with a 429 status code if an Event Receiver is not exceeding this frequency.

subject_type
Read-Write, The Subject Identifier Type that the Receiver wants for the events. If not set then the Transmitter might decide to use a type that discloses more information than necessary.

TODO: consider adding a IANA Registry for stream configuration metadata, similar to Section 7.1.1 of [OAUTH-DISCOVERY]. This would allow other specs to add to the stream configuration.

8.1.2.1. Reading a Stream's Configuration

An Event Receiver gets the current configuration of a stream by making an HTTP GET request to the Configuration Endpoint. On receiving a valid request the Event Transmitter responds with a "200 OK" response containing a JSON [RFC7159] representation of the stream's configuration in the body.

The following is a non-normative example request to read an Event Stream's configuration:

```
GET /set/stream HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 17: Example: Read Stream Configuration Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "iss": "https://tr.example.com",
  "aud": [
      "http://receiver.example.com/web",
      "http://receiver.example.com/mobile"
    ],
  "delivery": {
    "delivery_method":
      "https://schemas.openid.net/secevent/risc/delivery-method/push",
    "url": "https://receiver.example.com/events"
  },
  "events_supported": [
    "urn:example:secevent:events:type_1",
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ],
  "events_requested": [
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3",
    "urn:example:secevent:events:type_4"
  ],
  "events_delivered": [
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ]
}
```

Figure 18: Example: Read Stream Configuration Response

Errors are signaled with HTTP status codes as follows:

Code	Description
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to read the stream configuration
404	if there is no Event Stream configured for this Event Receiver

Table 3: Read Stream Configuration Errors

8.1.2.2. Updating a Stream's Configuration

An Event Receiver updates the current configuration of a stream by making an HTTP POST request to the Configuration Endpoint. The POST body contains a JSON [RFC7159] representation of the updated configuration. On receiving a valid request the Event Transmitter responds with a "200 OK" response containing a JSON [RFC7159] representation of the updated stream configuration in the body.

The full set of editable properties must be present in the POST body, not only the ones that are specifically intended to be changed. Missing properties SHOULD be interpreted as requested to be deleted. Event Receivers should read the configuration first, modify the JSON [RFC7159] representation, then make an update request.

Properties that cannot be updated MAY be present, but they MUST match the expected value.

The following is a non-normative example request to update an Event Stream's configuration:

```
POST /set/stream HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "iss": "https://tr.example.com",
  "aud": [
    "http://receiver.example.com/web",
    "http://receiver.example.com/mobile"
  ],
  "delivery": {
    "delivery_method":
      "https://schemas.openid.net/secevent/risc/delivery-method/push",
    "url": "https://receiver.example.com/events"
  },
  "events_requested": [
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3",
    "urn:example:secevent:events:type_4"
  ]
}
```

Figure 19: Example: Update Stream Configuration Request

The following is a non-normative example response:

```
HTTP/1.1 200 OK
Content-Type: application/json; charset=UTF-8
Cache-Control: no-store
Pragma: no-cache

{
  "iss": "https://tr.example.com",
  "aud": [
    "http://receiver.example.com/web",
    "http://receiver.example.com/mobile"
  ],
  "delivery": {
    "delivery_method":
      "https://schemas.openid.net/secevent/risc/delivery-method/push",
    "url": "https://receiver.example.com/events"
  },
  "events_supported": [
    "urn:example:secevent:events:type_1",
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ],
  "events_requested": [
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3",
    "urn:example:secevent:events:type_4"
  ],
  "events_delivered": [
    "urn:example:secevent:events:type_2",
    "urn:example:secevent:events:type_3"
  ]
}
```

Figure 20: Example: Update Stream Configuration Response

Pending conditions or errors are signaled with HTTP status codes as follows:

Code	Description
202	if the update request has been accepted, but not processed. Receiver MAY try the same request later in order to get processing result.
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to update the stream configuration
404	if there is no Event Stream configured for this Event Receiver

Table 4: Update Stream Configuration Errors

8.1.2.3. Removing a Stream Configuration

An Event Receiver removes the configuration of a stream by making an HTTP DELETE request to the Configuration Endpoint. On receiving a request the Event Transmitter responds with a "200 OK" response if the configuration was successfully removed.

The following is a non-normative example request to remove an Event Stream's configuration:

```
DELETE /set/stream HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
```

Figure 21: Example: Remove Stream Configuration Request

Errors are signaled with HTTP status codes as follows:

Code	Description
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to update the stream configuration

Table 5: Update Stream Configuration Errors

8.1.3. Subjects

An Event Receiver can indicate to an Event Transmitter whether or not the receiver wants to receive events about a particular subject by "adding" or "removing" that subject to the Event Stream, respectively.

8.1.3.1. Adding a Subject to a Stream

To add a subject to an Event Stream, the Event Receiver makes an HTTP POST request to the Add Subject Endpoint, containing in the body a JSON object the following claims:

subject
: REQUIRED. A Subject Identifier identifying the subject to be added.

verified
: OPTIONAL. A boolean value; when true, it indicates that the Event Receiver has verified the Subject Identifier. When false, it indicates that the Event Receiver has not verified the Subject Identifier. If omitted, Event Transmitters SHOULD NOT assume that verification has not occurred.

On a successful response, the Event Transmitter responds with an empty "200 OK" response. The Event Transmitter MAY choose to silently ignore the request, for example if the subject has previously indicated to the transmitter that they do not want events to be transmitted to the Event Receiver. In this case, the transmitter MAY return an empty "200 OK" response or an appropriate error code. See Security Considerations (Section 8.3).

Errors are signaled with HTTP status codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to add this particular subject, or not allowed to add in general
404	if the subject is not recognized by the Event Transmitter, the Event Transmitter may chose to stay silent in this case and respond with "200"
429	if the Event Receiver is sending too many requests in a gvien amount of time

Table 6: Add Subject Errors

The following is a non-normative example request to add a subject to a stream, where the subject is identified by an Email Subject Identifier.

```
POST /set/subjects:add HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "subject": {
    "subject_type": "email",
    "email": "example.user@example.com"
  },
  "verified": true
}
```

Figure 22: Example: Add Subject Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 200 OK
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 23: Example: Add Subject Response

8.1.3.2. Removing a Subject

To remove a subject from an Event Stream, the Event Receiver makes an HTTP POST request to the Remove Subject Endpoint, containing in the body a JSON object with the following claims:

subject
: A Subject Identifier identifying the subject to be removed. REQUIRED.

On a successful response, the Event Transmitter responds with a "204 No Content" response.

Errors are signaled with HTTP status codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
403	if the Event Receiver is not allowed to remove this particular subject, or not allowed to remove in general
404	if the subject is not recognized by the Event Transmitter, the Event Transmitter may chose to stay silent in this case and respond with "204"
429	if the Event Receiver is sending too many requests in a gvien amount of time

Table 7: Remove Subject Errors

The following is a non-normative example request where the subject is identified by a Phone Number Subject Identifier:

```
POST /set/subjects:remove HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=

{
  "subject": {
    "subject_type": "phone",
    "phone_number": "+1 206 555 0123"
  }
}
```

Figure 24: Example: Remove Subject Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 204 No Content
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 25: Example: Remove Subject Response

8.1.4. Verification

In some cases, the frequency of event transmission on an Event Stream will be very low, making it difficult for an Event Receiver to tell the difference between expected behavior and event transmission failure due to a misconfigured stream. Event Receivers can request that a verification event be transmitted over the Event Stream, allowing the receiver to confirm that the stream is configured correctly upon successful receipt of the event. The acknowledgment of a Verification Event also confirms to the Event Transmitter that end-to-end delivery is working, including signature verification and encryption.

An Event Transmitter MAY send a Verification Event at any time, even if one was not requested by the Event Receiver.

8.1.4.1. Verification Event

The Verification Event is a standard SET with the following attributes:

event type
: The Event Type URI is: "https://schemas.openid.net/secevent/risc/event-type/verification".

state
OPTIONAL An opaque value provided by the Event Receiver when the event is triggered. This is a nested attribute in the event payload.

Upon receiving a Verification Event, the Event Receiver SHALL parse the SET and validate its claims. In particular, the Event Receiver SHALL confirm that the value for "state" is as expected. If the value of "state" does not match, an error response of "setData" SHOULD be returned (see Section 2.3 of [DELIVERYPUSH] or [DELIVERYPOLL]).

In many cases, Event Transmitters MAY disable or suspend an Event Stream that fails to successfully verify based on the acknowledgement or lack of acknowledgement by the Event Receiver.

8.1.4.2. Triggering a Verification Event.

To request that a verification event be sent over an Event Stream, the Event Receiver makes an HTTP POST request to the Verification Endpoint, with a JSON [RFC7159] object containing the parameters of the verification request, if any. On a successful request, the event transmitter responds with an empty "204 No Content" response.

Verification requests have the following properties:

state
OPTIONAL. An arbitrary string that the Event Transmitter MUST echo back to the Event Receiver in the verification event's payload. Event Receivers MAY use the value of this parameter to correlate a verification event with a verification request. If the verification event is initiated by the transmitter then this parameter MUST not be set.

A successful response from a POST to the Verification Endpoint does not indicate that the verification event was transmitted successfully, only that the Event Transmitter has transmitted the event or will do so at some point in the future. Event Transmitters MAY transmit the event via an asynchronous process, and SHOULD publish an SLA for verification event transmission times. Event Receivers MUST NOT depend on the verification event being transmitted synchronously or in any particular order relative to the current queue of events.

Errors are signaled with HTTP status codes as follows:

Code	Description
400	if the request body cannot be parsed or if the request is otherwise invalid
401	if authorization failed or it is missing
429	if the Event Receiver is sending too many requests in a gvien amount of time; see related "min_verification_interval" in Section 8.1.2

Table 8: Verification Errors

The following is a non-normative example request to trigger a verification event:

```
POST /set/verify HTTP/1.1
Host: transmitter.example.com
Authorization: Bearer eyJ0b2tlbiI6ImV4YW1wbGUifQo=
Content-Type: application/json; charset=UTF-8

{
  "state": "VGhpcyBpcyBhbiBleGFtcGxlIHN0YXRlIHZhbHVlLgo="
}
```

Figure 26: Example: Trigger Verification Request

The following is a non-normative example response to a successful request:

```
HTTP/1.1 204 No Content
Server: transmitter.example.com
Cache-Control: no-store
Pragma: no-cache
```

Figure 27: Example: Trigger Verification Response

And the following is a non-normative example of a verification event sent to the Event Receiver as a result of the above request:

```
{
  "jti": "123456",
  "iss": "https://transmitter.example.com",
  "aud": "receiver.example.com",
  "iat": "1493856000",
  "events": {
    "https://schemas.openid.net/secevent/risc/event-type/verification":{
      "state": "VGhpcyBpcyBhbiBleGFtcGxlIHN0YXRlIHZhbHVlLgo="
    }
  }
}
```


Figure 28: Example: Verification SET

8.1.5. Stream Updated Event

A Transmitter MAY change the stream status in reference to one or more SPAGs without a request from a Receiver. The Transmitter sends an event of type "https://schemas.openid.net/secevent/risc/event-type/stream-updated" to indicate that it has changed the status of the Event Stream for a specific SPAG.

If a Transmitter decides to change the status of an Event Stream from "enabled" to either "paused" or "disabled", then the Transmitter MUST send this event to any Receiver that is currently "enabled" to receive events for the specified SPAG.

If the Transmitter changes the status of the stream for a specific SPAG from either "paused" or "disabled" to "enabled", then it MAY send this event to any Receiver that has previously been enabled to receive events for the specified SPAG.

The "stream-updated" event contains a "subject" claim of the subject type "spag". It also contains a "properties" claim, which is a JSON object containing the following fields:

status
: REQUIRED. Defines the new status of the stream for the SPAG ID specified in the Subject Type.

reason
: OPTIONAL. Provides a short description of why the Transmitter has updated the status.

```
{
  "jti": "123456",
  "iss": "https://transmitter.example.com",
  "aud": "receiver.example.com",
  "iat": "1493856000",
  "events": {
    "https://schemas.openid.net/secevent/risc/event-type/stream-updated":{
      "id": "1234"
    }
  }
  "properties": {
    "status": "paused",
    "reason": "License no longer valid."
  }
}
```

Figure 29: Example: Stream Updated SET

8.2. Authorization

HTTP API calls from a Receiver to a Transmitter SHOULD be authorized by providing an OAuth 2.0 Access Token as defined by [RFC6750].

The receiver may obtain an access token using the Client Credential Grant [CLIENTCRED], or any other method suitable for the Receiver and the Transmitter.

Event Stream updates for specific SPAGs are authorized using the "authorization" field in the update request.

8.3. Security Considerations

8.3.1. Subject Probing

It may be possible for an Event Transmitter to leak information about subjects through their responses to add subject requests. A "404" response may indicate to the Event Receiver that the subject does not exist, which may inadvertently reveal information about the subject (e.g. that a particular individual does or does not use the Event Transmitter's service).

Event Transmitters SHOULD carefully evaluate the conditions under which they will return error responses to add subject requests. Event Transmitters MAY return a "204" response even if they will not actually send any events related to the subject, and Event Receivers MUST NOT assume that a 204 response means that they will receive events related to the subject.

8.3.2. Information Harvesting

SETs may contain personally identifiable information (PII) or other non-public information about the event transmitter, the subject (of an event in the SET), or the relationship between the two. It is important for Event Transmitters to understand what information they are revealing to Event Receivers when transmitting events to them, lest the event stream become a vector for unauthorized access to private information.

Event Transmitters SHOULD interpret add subject requests as statements of interest in a subject by an Event Receiver, and ARE NOT obligated to transmit events related to every subject an Event Receiver adds to the stream. Event Transmitters MAY choose to transmit some, all, or no events related to any given subject and SHOULD validate that they are permitted to share the information contained within an event with the Event Receiver before transmitting the event. The mechanisms by which such validation is performed are outside the scope of this specification.

8.3.3. Malicious Subject Removal

A malicious party may find it advantageous to remove a particular subject from a stream, in order to reduce the Event Receiver's ability to detect malicious activity related to the subject, inconvenience the subject, or for other reasons. Consequently it may be in the best interests of the subject for the Event Transmitter to continue to send events related to the subject for some time after the subject has been removed from a stream.

Event Transmitters MAY continue sending events related to a subject for some amount of time after that subject has been removed from the stream. Event Receivers MUST tolerate receiving events for subjects that have been removed from the stream, and MUST NOT report these events as errors to the Event Transmitter.

8.3.4. Unauthorized Updates to SPAGs

A SPAG could represent a specific customer in a multi-tenanted environment or an organizational unit within a customer. A malicious administrator could benefit from receiving updates about a SPAG over which they do not have administrative control. Transmitters MUST send updates only about Subject Principals that belong to SPAGs for whom the Receiver is authorized. Transmitters SHOULD use the "authorization" field in the Update Status Request (Section 8.1.2.2) to verify that the Receiver has permission to get updates for a specific SPAG.

9. Profiles

This section is a profile of the following IETF SecEvent specifications:

- "Security Event Token (SET)" [SET]
- Push-Based SET Token Delivery Using HTTP [DELIVERYPUSH]
- Poll-Based SET Token Delivery Using HTTP [DELIVERYPOLL]

The RISC use cases that set the requirements are described in Security Events RISC Use Cases [USECASES].

The CAEP use cases that set the requirements are described in CAEP Use Cases (TODO: Add reference when file is added to repository.)

9.1. Security Event Token Profle

This section provides SSE profiling specifications for the "Security Event Token (SET)" [SET] spec.

9.1.1. Signature Key Resolution

The signature key can be obtained through "jwks_uri", see Section 7.

9.1.2. SSE Event Subject

The subject of a SSE event is identified by the "subject" claim within the event payload, whose value is a Subject Identifier. The "subject" claim is REQUIRED for all SSE events. The JWT "sub" claim MUST NOT be present in any SET containing a SSE event.

9.1.3. SSE Event Properties

The SSE event MAY contain a "properties" claim within the event payload, whose value is a JSON object. The fields in the JSON object depend upon the event type.

```
   {
     "iss": "https://idp.example.com/",
     "jti": "756E69717565206964656E746966696572",
     "iat": 1520364019,
     "aud": "636C69656E745F6964",
     "events": {
       "https://schemas.openid.net/secevent/risc/event-type/account-\
   disabled": {
         "subject": {
           "subject_type": "phone",
           "phone_number": "+1 206 555 0123"
         },
         "reason": "hijacking",
         "cause-time": 1508012752
       }
     }
   }
```

Figure 30: Example: SET Containing a RISC Event with a Phone Number Subject

```
{
  "iss": "https://idp.example.com/",
  "jti": "756E69717565206964656E746966696572",
  "iat": 1520364019,
  "aud": "636C69656E745F6964",
  "events": {
    "https://schemas.openid.net/secevent/caep/event-type/token-revocation": {
      "subject": {
        "subject_type": "email",
        "email": "user@example.com"
      },
      "properties": {
        "token":
"ya29.a0BfC6SMA8kBiQ90qrydNVMKl6R3eoYG7y73gRnqYP_RGxPff_W_zbmlrQKl1-rl4bzaukQeBLMvxqnV6p7dnxSaohz
kXobHFYuP47IFUuSymenultkk4E6Rxx_kQ4Bmg6IsS92AHavHI716y81dOIY5jw_6pYGJqMjD5dTLBxwTR8LGZCqVpIiu6rNr
LJsr2hKayH9a0ewF0TMaMV3w"
      }
      "reason": "hijacking",
      "cause-time": 1508012752
    }
  }
}
```

Figure 31: Example: SET Containing a CAEP Event with Properties

9.1.4. Explicit Typing of SETs

SSE events MUST use explicit typing as defined in Section 2.3 of [SET].

```
{
  "typ":"secevent+jwt",
  "alg":"HS256"
}
```

Figure 32: Explicitly Typed JOSE Header

The purpose is defense against confusion with other JWTs, as described in Sections 4.5, 4.6 and 4.7 of [SET]. While current Id Token [IDTOKEN] validators may not be using the "typ" header parameter, by requiring it for SSE SETs a distinct value is guaranteed for future validators.

9.1.5. The "exp" Claim

The "exp" claim MUST NOT be used in SSE SETs.

The purpose is defense in depth against confusion with other JWTs, as described in Sections 4.5 and 4.6 of [SET].

9.1.6. The "aud" Claim

The "aud" claim can be a single value or an array. Each value SHOULD be the OAuth 2.0 client ID. Other values that uniquely identifies the Receiver to the Transmitter MAY be used, if the two parties have agreement on the format.

More than one value can be present if the corresponding Receivers are known to the Transmitter to be the same entity, for example a web client and a mobile client of the same application. All the Receivers in this case MUST use the exact same delivery method.

If multiple Receivers have the exact same delivery configuration but the Transmitter does not know if they belong to the same entity then the Transmitter SHOULD issue distinct SETs for each Receiver and deliver them separately. In this case the multiple Receivers might use the same service to process SETs, and this service might reroute SETs to respective Receivers, an "aud" claim with multiple Receivers would lead to unintended data disclosure.

```
{
  "jti": "123456",
  "iss": "https://transmitter.example.com",
  "aud": ["receiver.example.com/web", "receiver.example.com/mobile"],
  "iat": "1493856000",
  "events": {
    "https://schemas.openid.net/secevent/risc/event-type/verification":{
      "state": "VGhpcyBpcyBhbiBleGFtcGxlIHN0YXRlIHZhbHVlLgo="
    }
  }
}
```


Figure 33: Example: SET with array 'aud' claim

9.1.7. The "events" claim

The "events" claim SHOULD contain only one event. Multiple event type URIs are permitted only if they are alternative URIs defining the exact same event type.

9.1.8. Security Considerations

9.1.8.1. Distinguishing SETs from other Kinds of JWTs

Of particular concern is the possibility that SETs are confused for other kinds of JWTs. The Security Considerations sesction of [SET] has several sub-sections on this subject. The SSE Profile is asking for further restrioctons:

- The "sub" claim MUST NOT be present, as described in Section 9.1.2.
- SSE SETs MUST use explicit typing, as described in Section 9.1.4.
- The "exp" claim MUST NOT be present, as described in Section 9.1.5.

9.2. SET Token Delivery Using HTTP Profile

This section provides SSE profiling specifications for the [DELIVERYPUSH] and [DELIVERYPOLL] specs.

9.2.1. Stream Configuration Metadata

Each delivery method is identified by a URI, specified below by the "method" metadata.

9.2.1.1. Push Delivery using HTTP

This section provides SSE profiling specifications for the [DELIVERYPUSH] spec.

method "https://schemas.openid.net/secevent/risc/delivery-method/push"

endpoint_url The URL where events are pushed through HTTP POST. This is set by the Receiver.

authorization_header The HTTP Authorization header that the Transmitter MUST set with each event delivery, if the configuration is present. The value is optional and it is set by the Receiver.

9.2.1.2. Polling Delivery using HTTP

This section provides SSE profiling specifications for the [DELIVERYPOLL] spec.

method "https://schemas.openid.net/secevent/risc/delivery-method/poll"

endpoint_url The URL where events can be retrieved from. This is specified by the Transmitter.

10. IANA Considerations

10.1. Security Event Subject Identifier Types Registry

This document defines Subject Identifier Types, for which IANA is asked to create and maintain a new registry titled "Security Event Subject Identifier Types". Initial values for the Security Event Subject Identifier Types registry are given in Section 10.1.2. Future assignments are to be made through the Expert Review registration policy [BCP26] and shall follow the template presented in Section 10.1.1.

10.1.1. Registration Template

Type Name:
The name of the Subject Identifier Type, as described in Section 6.1. The name MUST be an ASCII string consisting only of lower-case characters ("a" - "z"), digits ("0" - "9"), and hyphens ("-"), and SHOULD NOT exceed 20 characters in length.

Type Description:

A brief description of the Subject Identifier Type.

Change Controller:
For types defined in documents published by the OpenID Foundation or its working groups, list "OIDF SSE Working Group". For all other types, list the name of the party responsible for the registration. Contact information such as mailing address, email address, or phone number may also be provided.

Defining Document(s):
A reference to the document or documents that define the Subject Identifier Type. The definition MUST specify the name, format, and meaning of each claim that may occur within a Subject Identifier of the defined type, as well as whether each claim is optional or required, or the circumstances under which the claim is optional or required. URIs that can be used to retrieve copies of each document SHOULD be included.

10.1.2. Initial Registry Contents

- Type Name: "email"
- Type Description: Subject identifier based on email address.
- Change Controller: OIDF SSE Working Group
- Defining Document(s): Section 6.1.1 of this document.

- Type Name: "id_token_claims"
- Type Description: Subject identifier based on OpenID Connect ID Token claims.
- Change Controller: OIDF SSE Working Group
- Defining Document(s): Section 6.1.5 of this document.

- Type Name: "iss_sub"
- Type Description: Subject identifier based on issuer and subject.
- Change Controller: OIDF SSE Working Group
- Defining Document(s): Section 6.1.3 of this document.

- Type Name: "phone"
- Type Description: Subject identifier based on phone number.
- Change Controller: OIDF SSE Working Group
- Defining Document(s): Section 6.1.2 of this document.

- Type Name: "spag"
- Type Description: Subject Principal Administrative Grouping.
- Change Controller: OIDF SSE Working Group
- Defining Document(s): Section 6.1.4 of this document.

10.1.3. Guidance for Expert Reviewers

The Expert Reviewer is expected to review the documentation referenced in a registration request to verify its completeness. The Expert Reviewer must base their decision to accept or reject the request on a fair and impartial assessment of the request. If the Expert Reviewer has a conflict of interest, such as being an author of a defining document referenced by the request, they must recuse themselves from the approval process for that request. In the case where a request is rejected, the Expert Reviewer should provide the requesting party with a written statement expressing the reason for rejection, and be prepared to cite any sources of information that went into that decision.

Subject Identifier Types need not be generally applicable and may be highly specific to a particular domain; it is expected that types may be registered for niche or industry-specific use cases. The Expert Reviewer should focus on whether the type is thoroughly documented, and whether its registration will promote or harm interoperability. In most cases, the Expert Reviewer should not approve a request if the registration would contribute to confusion, or amount to a synonym for an existing type.

10.2. Well-Known URI Registry

TODO: follow steps in Section 5.1 of [RFC5785] to send a registration request.

This specification registers the well-known URI defined in Section 7 in the IANA Well-Known URI registry defined in [RFC5785].

10.2.1. Registry Contents

- URI suffix: "sse-configuration"
- Change controller: OpenID Foundation SSE Working Group - openid-specs-risc@lists.openid.net
- Specification document: Section 7 of this document
- Related information: (none)

11. Privacy Considerations

11.1. Subject Information Leakage

Event issuers and recipients SHOULD take precautions to ensure that they do not leak information about subjects via Subject Identifiers, and choose appropriate Subject Identifier Types accordingly. Parties SHOULD NOT identify a subject using a given Subject Identifier Type if doing so will allow the recipient to correlate different claims about the subject that they are not known to already have knowledge of. Issuers and recipients SHOULD always use the same Subject Identifier Type and the same claim values to identify a given subject when communicating with a given party in order to reduce the possibility of information leakage.

12. References

12.1. Normative References

[BCP26] Cotton, M., Leiba, B., and T. Narten, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 8126, DOI 10.17487/RFC8126, June 2017, <https://tools.ietf.org/html/rfc8126>.

[CAEP] Tulshibagwale, A., "Re-thinking Federated Identity with the Continuous Access Evaluation Protocol", February 2019, <https://cloud.google.com/blog/products/identity-security/re-thinking-federated-identity-with-the-continuous-access-evaluation-protocol>.

[CLIENTCRED] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework - Client Credentials Grant", RFC 6749, DOI 10.17487/RFC6749, October 2012, <https://tools.ietf.org/html/rfc6749#section-4.4>.

[DELIVERYPOLL] Backman, A., Ed., Jones, M., Ed., Hunt, P., Ed., Scurtescu, M., Ansari, M., and A. Nadalin, "Poll-Based SET Token Delivery Using HTTP", April 2018, <https://tools.ietf.org/html/draft-ietf-secevent-http-poll-00>.

[DELIVERYPUSH] Backman, A., Ed., Jones, M., Ed., Hunt, P., Ed., Scurtescu, M., Ansari, M., and A. Nadalin, "Push-Based SET Token Delivery Using HTTP", April 2018, <https://tools.ietf.org/html/draft-ietf-secevent-http-push-00>.

[E164] International Telecommunication Union, "The international public telecommunication numbering plan", 2010, <http://www.itu.int/rec/T-REC-E.164-201011-I/en>.

[IDTOKEN] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 - ID Token", April 2017, <http://openid.net/specs/openid-connect-core-1_0.html#IDToken>.

[MGMTAPI] Scurtescu, M. and A. Backman, "Management API for SET Event Streams", June 2017, <https://tools.ietf.org/html/draft-scurtescu-secevent-simple-control-plane>.

[OAUTH-DISCOVERY] Jones, M., Sakimura, N., and J. Bradley, "OAuth 2.0 Authorization Server Metadata - Version 10", March 2018, <https://tools.ietf.org/html/draft-ietf-oauth-discovery-10>.

[OIDC-DISCOVERY] Sakimura, N., Bradley, J., Jones, M., and E. Jay, "OpenID Connect Discovery 1.0", November 2014, <https://openid.net/specs/openid-connect-discovery-1_0.html>.

[OIDC-SPEC] Sakimura, N., Bradley, J., Jones, M., de Medeiros, B., and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1", November 2014, <https://openid.net/specs/openid-connect-core-1_0.html>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5322] Resnick, P., "Internet Message Format", RFC 5322, October 2008, <https://tools.ietf.org/html/rfc5322>.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, DOI 10.17487/RFC5785, April 2010, <https://www.rfc-editor.org/info/rfc5785>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, DOI 10.17487/RFC6750, October 2012, <https://www.rfc-editor.org/info/rfc6750>.

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, DOI 10.17487/RFC7159, March 2014, <https://www.rfc-editor.org/info/rfc7159>.

[RFC7517] Jones, M., "JSON Web Key (JWK)", RFC 7517, DOI 10.17487/RFC7517, May 2015, <https://www.rfc-editor.org/info/rfc7517>.

[RFC7519] Jones, M., Bradley, J., and N. Sakimura, "JSON Web Token (JWT)", RFC 7519, DOI 10.17487/RFC7519, May 2015, <https://www.rfc-editor.org/info/rfc7519>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RISC] Scurtescu, M., Backman, A., and J. Bradley, "OpenID RISC Profile of IETF Security Events 1.0", April 2018, <https://openid.net/specs/openid-risc-profile-1_0-ID1.html>.

[SAML-CORE] Cantor, S., Kemp, J., Philpott, R., and E. Maler, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0", March 2005, <http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf>.

[SET] Hunt, P., Ed., Jones, M., Denniss, W., and M. Ansari, "Security Event Token (SET)", April 2018, <https://tools.ietf.org/html/draft-ietf-secevent-token-09>.

[USECASES] Scurtescu, M., "Security Events RISC Use Cases", June 2017, <https://tools.ietf.org/html/draft-scurtescu-secevent-risc-use-cases-00>.

12.2. URIs

[1] https://datatracker.ietf.org/wg/secevent/about/

Appendix A. Acknowledgements

Transmitter Configuration Discovery (Section 7) is based on both OpenID Connect Discovery 1.0 [OIDC-DISCOVERY] and OAuth 2.0 Authorization Server Metadata [OAUTH-DISCOVERY].

Authors' Addresses

Atul Tulshibagwale
Google

Email: atultulshi@google.com

Marius Scurtescu
Coinbase

Email: marius.scurtescu@coinbase.com

Annabelle Backman
Amazon

Email: richanna@amazon.com

John Bradley
Yubico

Email: secevemt@ve7jtb.com