Workgroup: TIGRESS
Internet-Draft: draft-art-tigress-01
Published: 9 November 2022
Intended Status: Standards Track
Expires: 13 May 2023
Authors: D. Vinokurov
Apple Inc
M. Byington
Apple Inc
M. Lerch
Apple Inc
A. Pelletier
Apple Inc
N. Sha
Alphabet Inc

Transfer Digital Credentials Securely

Abstract

This document describes a mechanism to transfer digital credentials securely between two devices. Secure credentials may represent a digital key to a hotel room, a digital key to a door lock in a house or a digital key to a car. Devices that share credentials may belong to the same or two different platforms (e.g. iOS and Android). Secure transfer may include one or more write and read operations. Credential transfer needs to be performed securely due to the sensitive nature of the information.

Discussion Venues

This note is to be removed before publishing as an RFC.

Source for this draft and an issue tracker can be found at https://github.com/dimmyvi/tigress.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 13 May 2023.

Table of Contents

1. Introduction

Today, there is no standard way of transferring digital credentials securely between two devices belonging to the same platform or two different platforms. This document proposes a solution to this problem by introducing a Relay server which allows two devices to exchange encrypted Provisioning Information securely. The Relay server solves this problem by creating and managing temporary mailbox storage.

Each mailbox can be referenced by devices using a unique mailbox identifier in a URL. The URL pointing to encrypted Provisioning Information is to be passed between devices directly over various channels (e.g. SMS, email, messaging applications). The Security Considerations section provides recommendations on passing the URL and the Secret securely.

This document describes a Hypertext (HTTP) Application Programming Interface (API) that allows Sender and Receiver devices to interact with a Relay server in order to perform secure credential transfer.

2. Terminology

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

General terms:

- Relay Server - Web application exposing Tigress API to devices. It serves to securely transfer Provisioning Information between two devices (Sender and Receiver).

- Sender device - a device initiating a transfer of Provisioning Information to a Receiver device so that Receiver can register or provision this credential.

- Receiver device - a device that receives Provisioning Information from Sender device and uses it to register or provision Credential Information.

- Provisioning Partner - an entity which facilitates Credential Information lifecycle on a device. Lifecycle may include provisioning of credential, credential termination, credential update. API to Provisioning Partner is out of scope for this document.

- Provisioning Information - a set of data fields, allowing a device to generate Credential Information or receive it from Provisioning Partner and install it locally. The entire content of Provisioning Information is encrypted by Sender or Receiver device. Therefore, it is not visible to the Relay Server. The structure of Provisioning Information is specific to Provisioning Partner or type of Credential and out of the scope of this document.

- Credential Information - a set of data fields used to facilitate registration or provisioning of Credential Information on the Receiver's device.

- Secret - a symmetric encryption key shared by a pair of Sender and Receiver devices, used to encrypt Provisioning Information stored on the Relay server. Secret stays the same for the entire credential transfer flow (one Secret per complete transfer). Provisioning Information stored on Relay server is always encrypted using the Secret. In Stateful flow all information exchanged by Sender and Receiver devices through Relay server is encrypted with the same Secret. Thus, effectively, Secret has a one-to-one relation with the mailbox.

* Credential Vertical - The broad industry vertical that the credential belongs to. For example, the credential could belong to the car or home vertical.

API parameters:

* Device Claim - a unique token allowing the caller to read from / write data to the mailbox. Exactly one Sender device and one Receiver device **SHOULD** be able to read from / write secure payload to the mailbox. Sender device provides a Device Claim in order to create a mailbox. When the Relay server, having received a request from the Sender device, creates a mailbox, it binds this Sender's Device Claim to the mailbox. When the Receiver device first reads data from the mailbox it presents its Device Claim to the Relay Server, which binds the mailbox to the given Receiver device. Thus, both Sender and Receiver devices are bound to the mailbox (allowed to read from / write to it). Only Sender and Receiver devices that present valid Device Claims are allowed to send subsequent read/update/delete calls to the mailbox. The value **SHALL** be a unique UUID [RFC4122]. Sender and Receiver **MUST** use different values for Device Claim. Implementation **SHOULD** assign unique values for new mailboxes (avoid re-using values).

* Notification Token - a short or long-lived unique token stored by the Sender or Receiver device in a mailbox on the Relay server, which allows Relay server to send a push notification to the Sender or Receiver device, informing them of updates in the mailbox.

* MailboxIdentifier - a unique identifier for the given mailbox, generated by the Relay server at the time of mailbox creation. The value is a UUID [RFC4122].

3. Credential transfer workflows

We define two flows for credential transfer: 1. Stateless (Relay server facilitates a single credential data transfer: Sender -> Relay -> Receiver) and 2. Stateful (Relay server facilitates additional data transfers - there are multiple data transfers in this flow to prepare credential data for registering or provisioning by Receiver). Relay server does not limit the number of such data tranfsfers between Sender and Receiver devices. The details are provided below.

Both stateless and stateful share the following common steps. The processes start with a Sender device composing a set of Provisioning Information, encrypting it with a Secret and storing encrypted Provisioning Information on a Relay server in a mailbox. A unique Mailbox Identifier is generated by the Relay server as a part of

CreateMailbox call, created using a good source of entropy (preferably hardware-based entropy). Sender device generates a unique token - a Sender Device Claim - and stores it to the mailbox. Device Claim allows the Sender device presenting it to read and write data to / from the mailbox, thus binding it to the mailbox.

Sender device calls CreateMailbox API endpoint on a Relay server in order to create a mailbox. Once a mailbox is created, it has limited livetime. When expired, the mailbox **SHALL** be deleted - refer to DeleteMailbox endpoint. Mailbox configuration has a required "expiration" parameter in the request for the CreateMailbox call (refer to mailboxConfiguration request parameter). Relay server is responsible to periodically check for mailboxes that are past the expiration time and delete them.

Relay server builds a unique URL link to a mailbox (for example, "https://relayserver.example.com/v1/m/1234567890") and returns it to the Sender device, which sends the link directly to the Receiver device over communication channel (e.g. SMS, email, iMessage). Please refer to section "Security Considerations" for more details.

Receiver device, having obtained both the URL link and the Secret, generates a unique token - a Receiver Device Claim - and passes it to the Relay server in order to read the encrypted Provisioning Information from the mailbox.

Relay server has finally a given pair of Sender and Receiver devices bound to the mailbox by provided Sender (at the time of mailbox creation) and Receiver (at the time of reading secure content from the mailbox) Device Claims. Only bound devices are allowed to read or write data to the mailbox or to delete the mailbox.

3.1. Stateless workflow

The stateless workflow completes the common steps described in "Credential transfer workflows" section, then finishes the transfer completing the following steps. Receiver device, having read the encrypted Provisioning Information from the Relay mailbox, decrypts it with the Secret received from the Sender and starts credential registering or provisioning process on the device. Once the Receiver device has successfully provisioned credentials, it deletes the mailbox by sending a DeleteMailbox call to the Relay server.

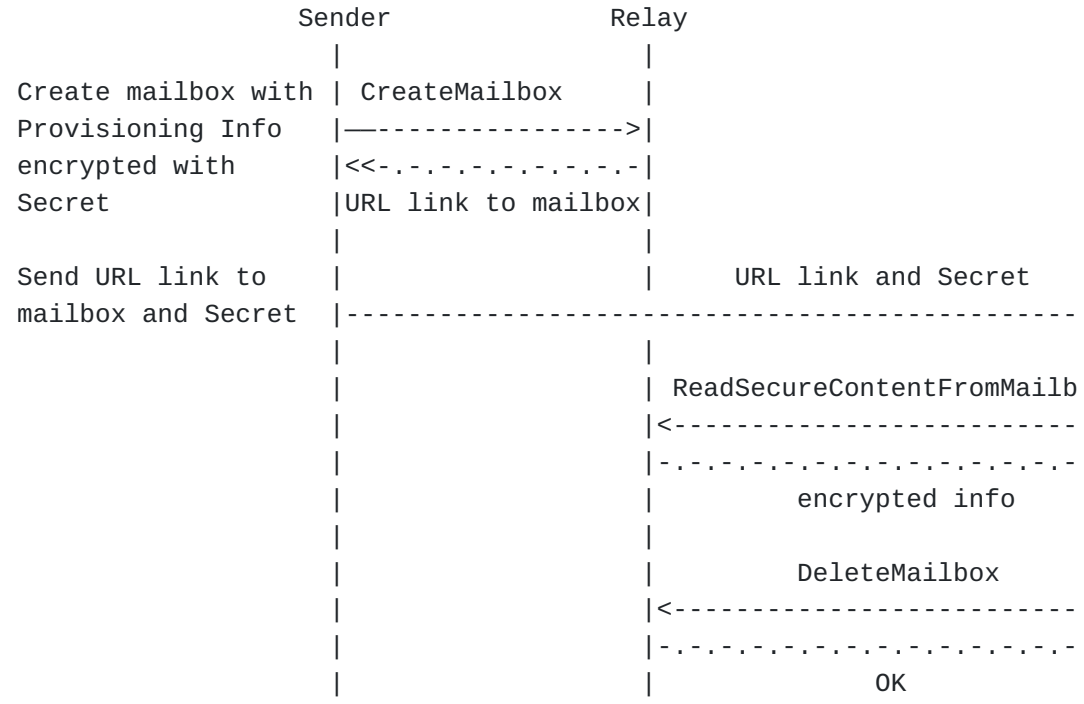

Figure 1: Sample stateless workflow

3.2. Stateful workflow

The stateful workflow completes the common steps described in "Credential transfer workflows" section, then finishes the transfer completing the following steps.

Then the Receiver device, having downloaded the encrypted Provisioning Information from the mailbox by URL and decrypted it with the Secret, generates a new structure of Provisioning Information, e.g. a digital key, and encrypts it with the same Secret, received from the Sender device. It then stores the payload in the same mailbox on the Relay server. In addition to the encrypted payload, Receiver stores a Receiver Notification Token in the given mailbox.

Having received the encrypted Provisioning Information, the Relay server sends a Notification to the Sender device using the Sender Notification Token.

Sender device, having received the notification from the Relay server, reads secure content from the mailbox and decrypts all using the same Secret. Sender device generates new Provisioning Information, encrypts all fields using the Secret and stores all data in the same mailbox on the Relay server.

Relay server, having stored the data above, sends a notification to the Receiver device using Receiver Notification Token. Receiver

device, having received the notification, reads the encrypted Provisioning Information, decrypts the data using the same Secret and uses this data to finalize credential registration or provisioning on device.

Once the Receiver device has successfully registered or provisioned credentials, it deletes the mailbox by sending a DeleteMailbox call to the Relay server. Sender device may terminate the secure credential transfer by deleting the mailbox it created at any time. Deletion of the mailbox on the Relay server stops any on-going credential transfer process.

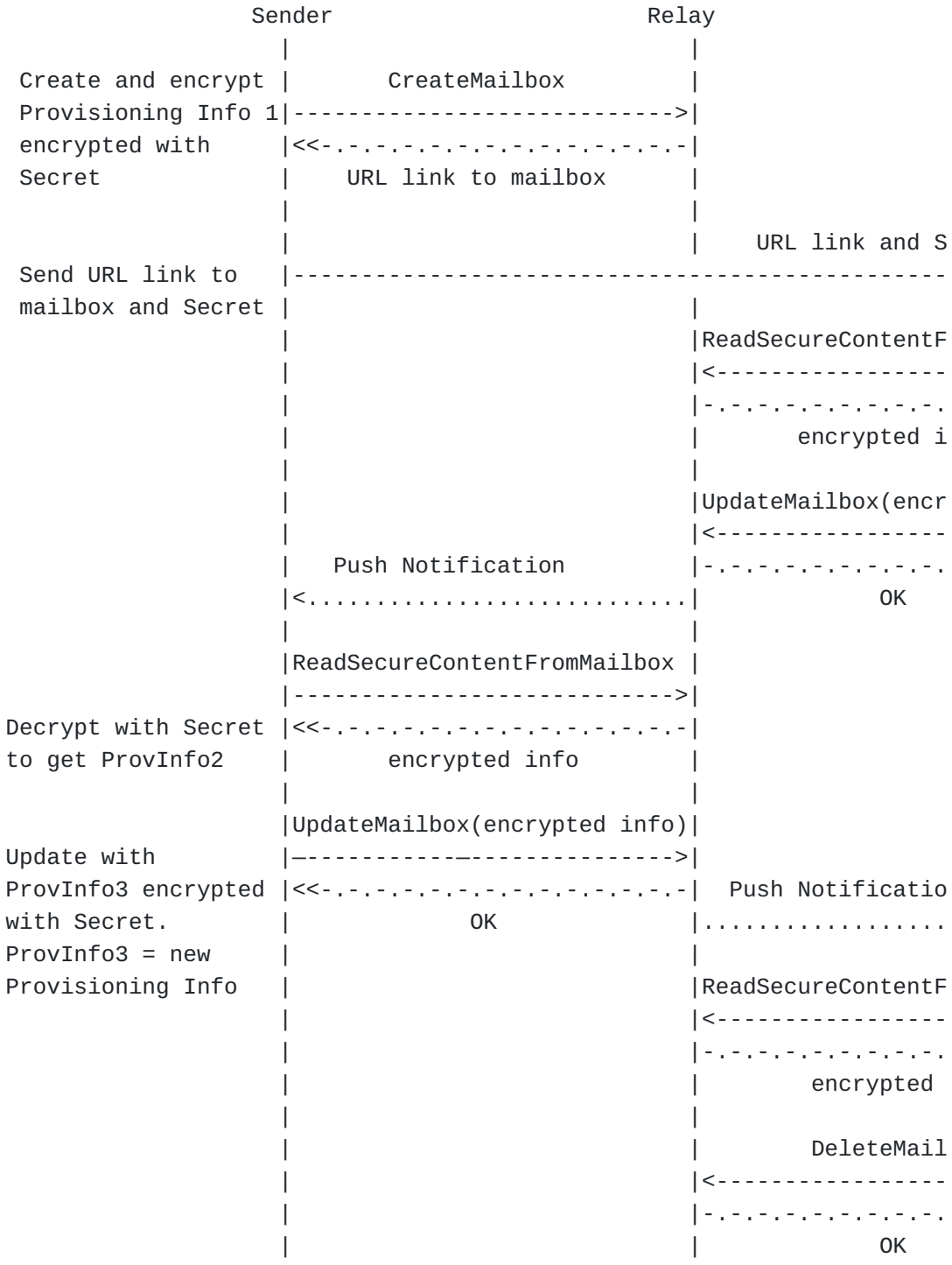

Figure 2: Sample stateful workflow

3.3. Provisioning Information Structure

The Provisioning Information is the data transfered via the Relay Server between the Sender device and Receiver device. Each use case defines its own specalized Provisioning Information format, but all formats must at least adhear to the following structure. Formats are

free to define new top level keys, so clients shouldn't be surprised if a message of an unexpected format has specialized top level keys.

Key	Type	Required	Description
format	String	Yes	The Provisioning Information format that the message follows. This is used by the Sender device and Receiver device to know how to parse the message.
content	Dictionary	Yes	A dictionary of content to be used for the credential transfer. See each format's specification for exact fields.

Table 1

3.3.1. Provisioning Information Format

Each Provisioning Information format must have the message structure defined in an external specification.

Format Type	Spec Link	Description
digitalwallet.carkey.ccc	[CCC-Digital-Key-30]	A digital wallet Provisioning Information for sharing a car key that follows the Car Connectivity Consortium specification.
digitalwallet.generic.authorizationToken	[ISO-18013-5]	A digital wallet Provisioning Information for sharing a generic pass that relies solely on an authorization token.

Table 2

```
{
   "format" : "digitalwallet.carkey.ccc",
   "content": {
      // Format specific fields
   }
}
```

Figure 3: Provisioning Information format

3.3.2. Provisioning Information Encryption

Provisioning Information will be stored on the Relay Server encrypted. The Secret used to encrypt the Provisioning Information should be given to the Receiver Device via a "Share URL" (a URL link to a mailbox). The encrypted payload should be a data structure having the following key-value pairs:

* "type" (String, Required) - the encryption algorithm and mode used.

* "data" (String, Required) - Base64 encoded binary value of the encrypted Provisioning Information, aka the ciphertext.

Please refer to [RFC5116] for the details of the encryption algorithm.

The following algorithms and modes are mandatory to implement:

* "AEAD_AES_128_GCM": AES symmetric encryption algorithm with key length 128 bits, in GCM mode with no padding. Initialization Vector (IV) has the length of 96 bits randomly generated and tag length of 128 bits.

* "AEAD_AES_256_GCM": AES symmetric encryption algorithm with key length 256 bits, in GCM mode with no padding. Initialization Vector (IV) has the length of 96 bits randomly generated and tag length of 128 bits.

```
{
    "type" : "AEAD_AES_128_GCM",
    "data" : "IV  ciphertext  tag"
}
```

Figure 4: Secure Payload Format example

3.4. Share URL

A "Share URL" is the url a Sender device sends to the Receiver device allowing it to retrieve the Provisioning Information stored on the Relay Server. A Share URL is made up of the following fields:

```
https://{RelayServerHost}/v{ApiVersion}/m/{MailboxIdentifier}?v={Credent
```

Figure 5: Share URL example

Field	Location	Required
RelayServerHost	URL Host	Yes
ApiVersion	URI Path Parameter	Yes
MailboxIdentifier	URI Path Parameter	Yes
CredentialVertical	Query Parameter	No
Secret	Fragment	No

Table 3

3.4.1. Credential Vertical in Share URL

When a user interacts with a share URL on a Receiver device it can be helpful to know what Credential Vertical this share is for. This is particularly important if the Receiver device has multiple applications that can handle a share URL. For example, a Receiver device might want to handle a general access share in their wallet app, but handle car key shares in a specific car application.

To properly route a share URL, the sender can include the Credential Vertical in the share URL as a query parameter. The Credential Vertical can't be included in the encrypted payload because the Receiver device might need to open the right application before retrieving the secure payload. The Credential Vertical query parameter uses the "v" key and supports the below types. If no Credential Vertical is provided it will be assumed that this is a general access share URL.

Vertical	Value
General Access	a or *None*
Home Key	h
Car Key	c

Table 4

```
https://relayserver.example.com/v1/m/2bba630e-519b-11ec-bf63-0242ac13000
```

Figure 6: Car Key Share URL example

The Credential Vertical query parameter can be added to the share URL by the Sender device when constructing the full share URL that is going to be sent to the Receiver device.

4. API connection details

The Relay server API endpoint **MUST** be accessed over HTTP using an https URI [RFC2818] and **SHOULD** use the default https port. Request and response bodies **SHALL** be formatted as either JSON or HTML (based on the API endpoint). The communication protocol used for all interfaces **SHALL** be HTTPs. All Strings **SHOULD** be UTF-8 encoded (Unicode Normalization Form C (NFC)). An API version **SHOULD** be included in the URI for all interfaces. The version at the time of this document's latest update is v1. The version **SHALL** be incremented by 1 for major API changes or backward incompatible iterations on existing APIs.

5. HTTP Headers

5.1. Mailbox-Request-ID

All requests to and from Relay server will have an HTTP header "Mailbox-Request-ID". The corresponding response to the API will have the same HTTP header, which **SHALL** echo the value in the request header. This is used to identify the request associated to the response for a particular API request and response pair. The value **SHOULD** be a UUID [RFC4122]. The request originator **SHALL** match the value of this header in the response with the one sent in the request. If response is not received, caller may retry sending the request with the same value of "Mailbox-Request-ID". Relay server **SHOULD** store the value of the last successfully processed "Mailbox-Request-ID" for each device based on the caller's Device Claim. A key-value pair of "Device Claim" to "Mailbox-Request-ID" is suggested to store the last successfully processed request for each device. In case of receiving a request with duplicated "Mailbox-Request-ID", Relay **SHOULD** respond to the caller with status code 201, ignoring the duplicate request body content.

5.2. Mailbox-Device-Claim

All requests to CreateMailbox, ReadSecureContentFromMailbox and UpdateMailbox endpoints **MUST** contain this header. The value represents "Device Claim" (refer to Terminology)

5.3. Mailbox-Device-Attestation

Request to CreateMailbox **MAY** contain this header. The value represents a Device Attestation (String, Optional) - optional remote OEM device proprietary attestation data

6. HTTP access methods

6.1. CreateMailbox

An application running on a remote device can invoke this API on Relay Server to create a mailbox and store secure data content to it (encrypted data specific to a provisioning partner). MailboxIdentifier is created by the Relay server as an UUID [RFC4122], using cryptographic entropy. A URL to the created mailbox to be returned to the caller in the response.

6.1.1. Endpoint

POST /{version}/m

6.1.2. Request Parameters:

Path parameters

*version (String, Required) - the version of the API. At the time of writing this document, "v1".

Header parameters

*Mailbox-Device-Attestation (String, Optional) - optional remote OEM device proprietary attestation data.

*Mailbox-Device-Claim (String, UUID, Required) - Device Claim (refer to Terminology).

6.1.3. Consumes

This API call consumes the following media types via the Content-Type request header: application/json

6.1.4. Produces

This API call produces the following media types via the Content-Type response header: application/json

6.1.5. Request body

Request body is a complex structure, including the following fields:

*payload (Object, Required) - for the purposes of Tigress API, this is a data structure, describing Provisioning Information

specific to Credential Provider. It consists of the following 2 key-value pairs:

1. "type": "AEAD_AES_128_GCM" (refer to Encryption Format section).

2. "data": BASE64-encoded binary value of ciphertext.

*displayInformation (Object, Required) - for the purposes of the Tigress API, this is a data structure. It allows an application running on a receiving device to build a visual representation of the credential to show to user. The data structure contains the following fields:

1. title (String, Required) - the title of the credential (e.g. "Car Key")

2. description (String, Required) - a brief description of the credential (e.g. "a key to my personal car")

3. imageURL (String, Required) - a link to a picture representing the credential visually.

*notificationToken (Object, Optional) - optional notification token used to notify an appropriate remote device that the mailbox data has been updated. Data structure includes the following (if notificationToken is provided it should include both fields):

1. type (String, Required) - notification token name. Used to define which Push Notification System to be used to notify appropriate remote device of a mailbox data update. (E.g. "com.apple.apns" for APNS)

2. tokenData (String, Required) - notification token data (data encoded based on specific device OEM notification service rules - e.g. HEX-encoded or Base64-encoded) - application-specific - refer to appropriate Push Notification System specification.

*mailboxConfiguration (Object, Optional) - optional mailbox configuration, defines access rights to the mailbox, mailbox expiration time. Required at the time of the mailbox creation. OEM device may provide this data in the request, Relay server shall define a default configuration, if it is not provided in the incoming request. Data structure includes the following:

1. accessRights (String, Optional) - optional access rights to the mailbox for Sender and Receiver devices. Default access to the mailbox is Read and Delete. Value is defined as a

combination of the following values: "R" - for read access, "W" - for write access, "D" - for delete access. Example" "RD" - allows to read from the mailbox and delete it.

2. expiration (String, Required) - Mailbox expiration time in "YYYY-MM-DDThh:mm:ssZ" format (UTC time zone) [RFC3339]. Mailbox has limited livetime. Once expired, it **SHALL** be deleted - refer to DeleteMailbox endpoint. Relay server **SHOULD** periodically check for expired mailboxes and delete them.

```
{
   "notificationToken": {
        "type":"com.apple.apns",
        "tokenData":"APNS1234...QW"
    }
}
```

Figure 7: Apple Push Token Example

```
{
    "displayInformation" : {
        "title" : "Hotel Pass",
        "description" : "Some Hotel Pass",
        "imageURL" : "https://example.com/sharingImage"
    },
    "payload" : {
        "type": "AEAD_AES_128_GCM",
        "data": "FDEC...987654321"
    },
    "notificationToken" : {
        "type" : "com.apple.apns",
        "tokenData" : “APNS...1234"
    },
    "mailboxConfiguration" : {
        "accessRights" : "RWD",
        "expiration" : "2022-02-08T14:57:22Z"
    }
}
```

Figure 8: Create Mailbox Request Example

6.1.6. Responses

200 Status: "200" (OK)

ResponseBody:

*urlLink (String, Required) - a full URL link to the mailbox including fully qualified domain name and mailbox Identifier. Refer to "Share URL" section for details.

*isPushNotificationSupported (boolean, Required) - indicates whether push notification is supported or not. The device uses this field to decide whether it should listen on the push topic or do long-polling.


```
{
 "urlLink":"https://relayserver.example.com/m/12345678-9...A-BCD",
 "isPushNotificationSupported":true
}
```


Figure 9: Create Mailbox Response Example

201 Status: "201" (Created) - response to a duplicated request (duplicated "Mailbox-Request-ID"). Relay server **SHALL** respond to duplicated requests with 201 without creating a new mailbox. "Mailbox-Request-ID" passed in the first CreateMailbox request's header **SHOULD** be stored by the Relay server and compared to the same value in the subsequent requests to identify duplicated requests. If duplicate is found, Relay **SHALL** not create a new mailbox, but respond with 201 instead. The value of "Mailbox-Request-ID" of the last successfully completed request **SHOULD** be stored based on the Device Claim passed by the caller.

400 Bad Request - invalid request has been passed (can not parse or required fields missing).

401 Unauthorized - calling device is not authorized to create a mailbox. E.g. a device presented an invalid device claim or device attestation.

6.2. UpdateMailbox

An application running on a remote device can invoke this API on Relay Server to update secure data content in an existing mailbox (encrypted data specific to a Provisioning Partner). The update effectively overwrites the secure payload previously stored in the mailbox.

6.2.1. Endpoint

PUT /{version}/m/{mailboxIdentifier}

6.2.2. Request Parameters

Path parameters:

- *version (String, Required) - the version of the API. At the time of writing this document, "v1".
- *mailboxIdentifier(String, Required) - MailboxIdentifier (refer to Terminology).

Header parameters:

- *Mailbox-Device-Attestation (String, Optional) - optional remote OEM device proprietary attestation data.
- *Mailbox-Device-Claim (String, UUID, Required) - Device Claim (refer to Terminology).

6.2.3. Consumes

This API call consumes the following media types via the Content-Type request header: application/json

6.2.4. Produces

This API call produces following media types via the Content-Type request header: application/json

6.2.5. Request body

Request body is a complex structure, including the following fields:

- *payload (Object, Required) - for the purposes of Tigress API, this is a data structure, describing Provisioning Information specific to Credential Provider. It consists of the following 2 key-value pairs:
  1. "type": "AEAD_AES_128_GCM" (refer to Encryption Format section).
  2. "data": BASE64-encoded binary value of ciphertext.
- *notificationToken (Object, Optional) - optional notification token used to notify an appropriate remote device that the mailbox data has been updated. Data structure includes the following (if notificationToken is provided it should include both fields):
  1. type (String, Required) - notification token name. Used to define which Push Notification System to be used to notify

appropriate remote device of a mailbox data update. (E.g. "com.apple.apns" for APNS)

2. tokenData (String, Required) - notification token data (data encoded based on specific device OEM notification service rules - e.g. HEX-encoded or Base64-encoded) - application-specific - refer to appropriate Push Notification System specification.

```
{
     "payload" : {
        "type": "AEAD_AES_128_GCM",
        "data": "FDEC...987654321"
    },
    "notificationToken":{
        "type" : "com.apple.apns",
        "tokenData" : “APNS...1234"
    }
}
```

Figure 10: Update Mailbox Request Example

6.2.6. Responses

ResponseBody:

*isPushNotificationSupported (boolean, Required) - indicates whether push notification is supported or not. The device uses this field to decide whether it should listen on the push topic or do long-polling.

```
{
    "isPushNotificationSupported":true
}
```

Figure 11: Create Mailbox Response Example

200 Status: "200" (OK)

201 Status: "201" (Created) - response to a duplicate request (duplicate "Mailbox-Request-ID"). Relay server **SHALL** respond to duplicate requests with 201 without performing mailbox update. "Mailbox-Request-ID" passed in the first UpdateMailbox request's header **SHALL** be stored by the Relay server and compared to the same value in the subsequent requests to identify duplicate requests. If duplicate is found, Relay **SHALL** not perform mailbox update, but respond with 201 instead. The value of "Mailbox-Request-ID" of the

last successfully completed request **SHALL** be stored based on the Device Claim passed by the caller.

400 Bad Request - invalid request has been passed (can not parse or required fields missing).

401 Unauthorized - calling device is not authorized to update the mailbox. E.g. a device presented the incorrect Device Claim.

404 Not Found - mailbox with provided mailboxIdentifier not found.

6.3. DeleteMailbox

An application running on a remote device can invoke this API on Relay Server to close the existing mailbox after it served its purpose. Receiver or Sender device needs to present a Device Claim in order to close the mailbox.

6.3.1. Endpoint

DELETE /{version}/m/{mailboxIdentifier}

6.3.2. Request Parameters

Path parameters:

* version (String, Required) - the version of the API. At the time of writing this document, "v1".

* mailboxIdentifier(String, Required) - MailboxIdentifier (refer to Terminology).

Header parameters:

* Mailbox-Device-Attestation (String, Optional) - optional remote OEM device proprietary attestation data.

* Mailbox-Device-Claim (String, UUID, Required) - Device Claim (refer to Terminology).

6.3.3. Responses

200 Status: "200" (OK)

401 Unauthorized - calling device is not authorized to delete a mailbox. E.g. a device presented the incorrect Device Claim.

404 Not Found - mailbox with provided mailboxIdentifier not found. Relay server may respond with 404 if the Mailbox Identifier passed

by the caller is invalid or mailbox has already been deleted (as a result of duplicate DeleteMailbox request).

6.4. ReadDisplayInformationFromMailbox

An application running on a remote device can invoke this API on Relay Server to retrieve public display information content from a mailbox. Display Information shall be returned in OpenGraph format (please refer to https://ogp.me for details). OpenGraph-formatted display information is required to display a preview of credential in a messaging application, e.g. iMessage or WhatsApp.

6.4.1. Endpoint

GET /{version}/m/{mailboxIdentifier}

6.4.2. Request Parameters

Path parameters:

* version (String, Required)- the version of the API. At the time of writing this document, "v1".

* mailboxIdentifier(String, Required) - MailboxIdentifier (refer to Terminology).

6.4.3. Produces

This API call produces the following media types via the Content-Type response header: text/html

6.4.4. Responses

200 Status: "200" (OK)

ResponseBody :

* displayInformation (Object, Required) - visual representation of digital credential in OpenGraph format (please refer to https://ogp.me for details).

```
 "<html prefix="og: https://ogp.me/ns#">
  <head>
  <title>Hotel Pass</title>
  <meta property="og:title" content="Hotel Pass" />
  <meta property="og:type" content="image/jpeg" />
  <meta property="og:description" content="Some Hotel Pass" />
  <meta property="og:url" content="share://" />
  <meta property="og:image" content="https://example.com/photos/photo
  <meta property="og:image:width" content="612" />
  <meta property="og:image:height" content="408" /></head>
  </html>"
```

Figure 12: Read Display Information Response Example

404 Not Found - mailbox with provided mailboxIdentifier not found.

6.5. ReadSecureContentFromMailbox

An application running on a remote device can invoke this API on Relay Server to retrieve secure payload content from a mailbox (encrypted data specific to a Provisioning Information Provider).

6.5.1. Endpoint

POST /{version}/m/{mailboxIdentifier}

6.5.2. Request Parameters

Path parameters:

* version (String, Required) - the version of the API. At the time of writing this document, "v1".

* mailboxIdentifier(String, Required) - MailboxIdentifier (refer to Terminology).

Header parameters:

* Mailbox-Device-Attestation (String, Optional) - optional remote OEM device proprietary attestation data.

* MAilbox-Device-Claim (String, UUID, Required) - Device Claim (refer to Terminology).

6.5.3. Produces

This API call produces the following media types via the Content-Type response header: application/json

6.5.4. Responses

200 Status: "200" (OK)

ResponseBody :

*payload (String, Required) - for the purposes of Tigress API, this is a JSON metadata blob, describing Provisioning Information specific to Credential Provider.

*displayInformation (Object, Required) - for the purposes of the Tigress API, this is a JSON data blob. It allows an application running on a receiving device to build a visual representation of the credential to show to user. Specific to Credential Provider.

*expiration (String, Required) - the date that the mailbox will expire. The mailbox expiration time is set during mailbox creation. Expiration time should be a complete [RFC3339] date string in "YYYY-MM-DDThh:mm:ssZ" format (UTC time zone), and can be used to allow receiving clients to show when a share will expire.

```
{
    “displayInformation" : {
        "title" : "Hotel Pass",
        "description" : "Some Hotel Pass",
        "imageURL" : "https://example.com/sharingImage"
    },
    "payload" : {
        "type": "AEAD_AES_128_GCM",
        "data": "FDEC...987654321"
    },
    "expiration": "2021-11-03T20:32:34Z"
}
```

Figure 13: Read Secure Content Response Example

401 Unauthorized - calling device is not authorized to read the secure content of the mailbox. E.g. a device presented the incorrect Device Claim.

404 Not Found - mailbox with provided mailboxIdentifier not found.

6.6. RelinquishMailbox

An application running on a remote device can invoke this API on Relay Server to relinquish their ownership of the mailbox. Receiver device needs to present the currently established Receiver Device Claim in order to relinquish their ownership of the mailbox. Once

relinquished, the mailbox can be bound to a different Receiver device that presents its Device Claim in a ReadSecureContentFromMailbox call.

6.6.1. Endpoint

PATCH /{version}/m/{mailboxIdentifier}

6.6.2. Request Parameters

Path parameters:

* version (String, Required) - the version of the API. At the time of writing this document, "v1".

* mailboxIdentifier(String, Required) - MailboxIdentifier (refer to Terminology).

Header parameters:

* Mailbox-Device-Attestation (String, Optional) - optional remote OEM device proprietary attestation data.

* Mailbox-Device-Claim (String, UUID, Required) - Device Claim (refer to Terminology).

6.6.3. Responses

200 Status: "200" (OK)

201 Status: "201" (Created) - response to a duplicate request (duplicate "Mailbox-Request-ID"). Relay server **SHALL** respond to duplicate requests with 201 without performing mailbox relinquish. "Mailbox-Request-ID" passed in the first RelinquishMailbox request's header **SHALL** be stored by the Relay server and compared to the same value in the subsequent requests to identify duplicate requests. If duplicate is found, Relay **SHALL** not perform mailbox relinquish, but respond with 201 instead. The value of "Mailbox-Request-ID" of the last successfully completed request **SHALL** be stored based on the Device Claim passed by the caller.

401 Unauthorized - calling device is not authorized to relinquish a mailbox. E.g. a device presented the incorrect Device Claim, or the device is not bound to the mailbox.

404 Not Found - mailbox with provided mailboxIdentifier not found. Relay server may respond with 404 if the Mailbox Identifier passed by the caller is invalid.

7. Security Considerations

The following threats and mitigations have been considered:

* Sender shares with the wrong receiver

  - Sender **SHOULD** be encouraged to share Secret over a channel allowing authentication of the receiver (e.g. voice).

  - Provisioning Partners **SHALL** allow senders to cancel existing shares.

* Malicious receiver forwards the share to 3rd party without redeeming it or the Receiver's device is compromised.

  - No mitigation, the Sender **SHOULD** only share with receivers they trust.

* Malicious receiver attempts re-use share

  - Provisioning Partners **SHALL** ensure that the Provisioning Information of a share can only be redeemed once.

* Share URL accidental disclosure. (e.g. share URL sent as a message which gets displayed on a locked screen)

  - Knowledge of Secret is required to access Provisioning Information and it **SHOULD** have been sent in a separate channel.

  - Device Claim is required (if sender and receiver have already both contacted the Relay server)

* Network attacks

  - Machine-in-the-middle: Relay server **SHALL** only allow TLS connections. URLs displayed to user **SHOULD** include the https scheme.

  - MailboxIdentifier guessing: the MailboxIdentifier is a version 4 UUID [RFC4122] which **SHOULD** contain 122-bits of cryptographic entropy, making brute-force attacks impractical.

* Risk of hosting malicious or untrusted scripts by relay server preview page (ReadDisplayInformationFromMailbox)

  - Relay server should either not allow hosting a third party JavaScripts on a preview page or implement a policy and utilize tools to maintain the trust of such scripts (e.g.

force client to verify the script against a good known hash of it).

7.1. Sender/Receiver privacy

*At no time Relay server **SHALL** store or track the identities of both Sender and Receiver devices.

*The value of the Notification Token shall not contain information allowing the identification of the device providing it. It **SHOULD** also be different for every new share to prevent the Relay server from correlating different sharing.

*Notification token **SHOULD** only inform the corresponding device that there has been a data update on the mailbox associated to it (by Device Claim). Each device **SHOULD** keep track of all mailboxes associated with it and make read calls to appropriate mailboxes.

*Both Sender and Receiver devices **SHOULD** store the URL of the Relay server they use for an active act of credential transfer.

*The value of Mailbox-Device-Attestation header parameter **SHALL** not contain information allowing the identification of the device providing it. It **SHOULD** also be different for every new share to prevent the Relay server from correlating different sharing.

*Display Information is not encrypted, therefore, it **SHOULD** not contain any information allowing to identify Sender or Receiver devices.

7.2. Credential's confidentiality and integrity

*Content of the mailbox **SHALL** be only visible to devices having Secret.

*It is recommended to send URL to the mailbox and the Secret over different channels (out-of-band) from Sender device to Receiver device (e.g. send URL over SMS and Secret over iMessage).

*Relay server **MUST** not receive the Secret with the MailboxIdentifier at any time.

*Content of the mailbox **MUST** guaranty its integrity with cryptographic checksum (e.g. MAC, AES-GCM tag).

*Relay server **SHALL** periodically check and delete expired mailboxes (refer to expiration parameter in the CreateMailbox request).

*If the Sender device sends both URL and the Secret over the same channel as a single URL, the Sender **MUST** append the Secret as URI fragment [RFC3986], so that the resulting URL shall look as in the example below. Receiver device, upon receipt of such URL, **MUST** remove the Fragment (Secret) before calling the Relay server API.

“https://relayserver.example.com/v1/m/{mailboxIdentifier}#{Secret}”

Figure 14: Example of URL with Secret as URI Fragment

7.3. Second factor authentication for Receiver credential provisioning

*Provisioning Partner shall require an additional security confirmation (PIN code) from Receiver Device at the time of credential provisioning.

*PIN code shall be generated by the Sender Device at the time when it creates a new Mailbox with Provisioning Information on a Relay Server.

*PIN code shall be sent from Sender device to Receiver device in a secure way (preferrably over encrypted channel) out-of-band with the Mailbox URL and Secret

*If Sender device can not send the PIN code over secure channel, it may include it into encrypted Payload stored on the relay server so that Receiver device can decrypt it and use to get Provisioning Information from Provisioning Partner.

*Provisioning Partner shall limit the number of PIN code entry attempts at the time when Receiver device calls it in order to receive Provisioning Information.

*The way PIN code is transferred between Sender device and Receiver device is defined by specific implementation and out of scope of this document.

8. IANA Considerations

This document registers new headers, "Mailbox-Request-ID", "Mailbox-Device-Claim" and "Mailbox-Device-Attestation" in the "Permanent Message Header Field Names" <https://www.iana.org/assignments/message-headers>.

Header Field Name	Protocol	Status	Reference
Mailbox-Request-ID	http	std	This document
Mailbox-Device-Claim	http	std	This document
Mailbox-Device-Attestation	http	std	This document

Figure 15: Registered HTTP Header

9. References

9.1. Normative References

[CCC-Digital-Key-30] Car Connectivity Consortium, "Digital Key Release 3", July 2022, <https://carconnectivity.org/download-digital-key-3-specification/>.

[ISO-18013-5] Cards and security devices for personal identification, "Personal identification — ISO-compliant driving licence — Part 5: Mobile driving licence (mDL) application", September 2021, <https://www.iso.org/standard/69084.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/rfc/rfc2119>.

[RFC3339] Klyne, G. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, DOI 10.17487/RFC3339, July 2002, <https://www.rfc-editor.org/rfc/rfc3339>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <https://www.rfc-editor.org/rfc/rfc3986>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, DOI 10.17487/RFC4122, July 2005, <https://www.rfc-editor.org/rfc/rfc4122>.

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, DOI 10.17487/

RFC5116, January 2008, <https://www.rfc-editor.org/rfc/rfc5116>.

[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/rfc/rfc8174>.

9.2. Informative References

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, DOI 10.17487/RFC2818, May 2000, <https://www.rfc-editor.org/rfc/rfc2818>.

Appendix A. Contributors

The following people provided substantive contributions to this document:

* Ben Chester
* Casey Astiz
* Jean-Luc Giraud
* Yogesh Karandikar
* Alexey Bulgakov
* Tommy Pauly
* Crystal Qin
* Adam Bar-Niv
* Manuel Gerster
* Igor Gariev

Appendix B. Acknowledgments

TODO acknowledge.

Authors' Addresses

Dmitry Vinokurov
Apple Inc

Email: dvinokurov@apple.com

Matt Byington
Apple Inc

Email: mbyington@apple.com

Matthias Lerch
Apple Inc

Email: mlerch@apple.com

Alex Pelletier
Apple Inc

Email: a_pelletier@apple.com

Nick Sha
Alphabet Inc

Email: nicksha@google.com