GLOBALPLATFORM®

Web API For Accessing Secure Element

Version 1.0 - September 2016
GlobalPlatform Device Specification

Related links:

- GlobalPlatform
- Github project

Abstract

A Secure Element is a tamper resistant device, providing a secure storage and execution environment for sensitive data and processing. It offers both physical and logical protection against attacks, ensuring integrity and confidentiality of its content. Trusted Execution Environment (TEE) is out of scope of this specification.

This specification defines a communication interface between a web application and a Secure Element. It makes no assumption on the Secure Element type, application domain or physical communication media.

Table of Contents

1. Introduction

The API defined herein allows applications to interact with Secure Elements. Considered Secure Elements are those complying to [ISO7816-4], which defines a command/response protocol, based on structured APDU (Application Protocol Data Unit).

1.1 Technical Background

Secure Elements addressed by this specification are microcontrollers that may come in different form factors, such as:

- Smart cards. The chip is embedded in a plastic card usually of the size of a typical credit card. The card may show physical contacts to communicate with the chip, or the chip may support NFC (Near Field Communication), in which case the plastic card embeds an antenna. Some cards also support both communication methods.
- UICC (Universal Integrated Circuit Card) are smart cards used in cellular telephony, which may be delivered in different sizes. They are often called SIM, which is actually the name of the application hosted by the UICC to access GSM networks. UICC may however host other applications.
- Smart SD cards have a similar form as usual SD cards, but internally include a Secure Element and support an extended set of SD commands to communicate with the Secure Element. Some of these smart SD cards also support NFC.
- Embedded Secure Elements, which are chips directly bonded on the device mother board. Unlike other form factors, this one does not allow interchanging or extracting the Secure Element, it is permanently attached to the device.

Similarly to a computer, a Secure Element may host one or multiple applications. Typical applications are mobile network authentication (SIM cards), payment (credit cards), authentication and signature (corporate badges, eID, etc.), loyalty, ticketing (public transports). These are only examples; many other applications have been and can be deployed.

Applications hosted by the Secure Elements are called **on-card** applications. On-card applications have a limited, if any, user interface. An application can be useful with an off-card application part, which handles the dialog with the user or with external computing resources. Examples of **off-card** applications are ATM for payment, mail applications for signature, access control doors for authentication, etc. This specification defines the API to be used by off-card applications based on web technologies.

1.2 Use Cases

This specification shows how to develop web applications making use of these Secure Element applications. Some typical use cases that applications can address based on this API include:

- **Authentication:** Instead of user name and password, access to an online service may be protected by a strong authentication mechanism, based on credentials stored and processed in a Secure Element. In web-based operating systems, system applications such as VPN (Virtual Private Network) or eMail applications may use the Secure Element to authenticate the user.
- **Digital Signature:** Applications may use the Secure Element to digitally sign a document or any data with a key stored in this Secure Element. The signature operation itself is executed inside the Secure Element, ensuring both the integrity of the signature and the confidentiality of the key used in this process. For instance, this could be used by an eMail application to sign emails sent by the user or could be used by a government web application to sign a online administrative request.
- **Payment:** Online commerce may use widely used smart credit cards or specific payment applications, to enforce the security of online transactions. In a cellular telephony environment, the on-card payment application may be hosted on the UICC, alleviating the need for the user to handle multiple physical devices (such as a mobile plus a debit/credit card).
- **Credential provisioning:** The content of a Secure Element may be updated to install, update or remove an

application or any credential it may host. For example, a public transport application may offer users to credit their NFC-enabled transport card with tickets bought online. In another example, a corporate intranet web application may enable employees to renew online the X.509 certificates hosted in their corporate badge from anywhere just before the certificates expire.

Whatever the form factor listed above, Secure Element considered in this specification implements the same [ISO7816-4] transport protocol. The physical media (USB, NFC or any other wired or wireless technique) used in this communication is abstracted by the API defined in this specification.

1.3 Relationship to W3C APIs

This specification, although addressing some concepts similar to several W3C specifications, has distinct use cases and offer different level of services:

- The current NFC API draft specification [NFC] defines an API allowing to exchange NDEF messages with NFC tags or peers. While the Secure Element API specified herein allows web applications to send commands to Secure Elements wired or plugged in the device or wirelessly connected to the device thanks to NFC technology. The difference between the different communication links (wired or NFC) is only visible through Secure Element type in this API, but does not impact the way applications would interact with the Secure Element. As such there is no overlap of functionalities between the two APIs.
- The Web Cryptography API draft specification [WEBCRYPTO] defines an API allowing a web application to invoke cryptographic services. Its implementation is independent from the underlying layers performing the actual cryptographic operations: it might be pure software or use a dedicated hardware such as a Secure Element or a TPM. As such there is not overlap between the two APIs. Nevertheless one can imagine that a Web Cryptography API implementation may use the Secure Element API directly, but this will be implementation dependent.

2. Conformance

As well as sections marked as non-normative, all authoring guidelines, diagrams, examples, and notes in this specification are non-normative. Everything else in this specification is normative.

The key words MAY, MUST, and MUST NOT are to be interpreted as described in [RFC2119].

This specification defines conformance criteria that apply to a single product: the **user agent** that implements the interfaces that it contains.

Implementations that use ECMAScript to implement the APIs defined in this specification MUST implement them in a manner consistent with the ECMAScript Bindings defined in the Web IDL specification [WEBIDL], as this specification uses that specification and terminology.

3. Dependencies

This specification depends on interfaces and concepts defined in the following specifications.

[HTML]: **event handler**.

[DOM4]: the `Event` and `DOMException` interfaces, the concept of **firing an event**.

[ES6]: the `Promise` object type.

[RFC6454]: The concepts of **origin** and **same-origin**, as well as the algorithm for serializing an origin.

[GP-AC]: The **Access Rules** and **Access Control Enforcer** (ACE) concepts.

[GP]: The **Supplementary Logical Channel** and **Issuer Security Domain** concepts.

Some Secure Element concepts are defined in ISO/IEC specifications:

- **APDU command**, **APDU response**, **class byte**, **instruction byte**, **parameter bytes**, **data field bytes**, **status word**, **extended length**, **logical channel**, **basic logical channel** and **historical bytes** are defined in [ISO7816-4]
- **AID** (Application Identifier) is defined in [ISO7816-5].
- **UICC** is defined in [ETSI-102216]

4. Security and Privacy Considerations

Using a Secure Element may bring additional security to a web application, but is not sufficient to ensure the application is secure. In particular, developers using the Secure Element API should be aware of the following security considerations:

- **Communication** between the Secure Element and the web application has to be secured in order to ensure the confidentiality and integrity of the message exchange. This can either be achieved if the web application using the Secure Element API executes in a trusted execution environment offering guarantees on the integrity and confidentiality of the communication link. Or it can be provided programmatically by encryption and/or MACing of the messages exchanged with the Secure Element (e.g. using GlobalPlatform's secure messaging technology [GP]). The off-card processing of these messages has then to be done in a trusted execution environment, which may be on the device to which the Secure Element is connected, or on a remote device (e.g. the web application's originating server).
- **Interface** between the user and the web application typically consists of displayed text and images (e.g. a transaction confirmation dialog) and user inputs (e.g. a PIN code). Protecting this interface is out of the scope of this specification. If the application requires such guarantee, it should restrict its execution on Trusted Execution Environments.
- **Access** to on-card applications should be restricted to authorized parties. On-card applications usually enforce such control by requiring authentication of the off-card communicating party, for instance by asking user to present a PIN to unlock access or performing a mutual authentication before any sensitive operation.
  There is however a risk of denial of service (DoS) attacks, where an attacker could e.g. either deliberately present multiple invalid PIN values to block the Secure Element or sending burst of commands preventing legitimate applications to execute optimally. In order to mitigate this risk, this specification requires the web application runtime to implement the access control mechanism defined in Access Control section.
- **Traceability** of the user may be facilitated by the unique identifying information the Secure Element may contain. Here again, the access control defined in Access Control section ensures only authorized web applications have access to the Secure Element API.

5. Secure Element Services

Communication with a Secure Element is performed through a `reader`, which is not only able to read content from Secure Element, but also to write or send any application specific command. Given the removable nature of many Secure Elements, a reader may be empty, meaning that no Secure Element is connected to this reader. For instance, a USB smart card reader may be connected to a computer but no smart card is inserted or a device may support NFC interactions with Secure Elements but none is in the field communication range. For this reason, the API provides a means to query the list of available readers and for each of them if they are empty or if a Secure Element is present. In addition, this specification defines events that are triggered when a Secure Element is connected to a reader and also when it is disconnected.

Once a web application is informed that a reader has a connected Secure Element, it can open a `session` with it. Opening a session establishes the communication between the web application and the Secure Element and provides a session object to the application. However a Secure Element is just an application container, the web application still needs to identify the application with which it wants to communicate.

To this end, the session object provides a means to open a `channel` with a specific on-card application, which is uniquely identified by an AID. The web application runtime and the Secure Element may then perform some internal security check to ensure the web application is allowed to connect to this on-card application. If authorized, the returned channel object is the one providing the method to send commands to the on-card application and get the corresponding responses.

The above steps required to send a command to a Secure Element create a set of chained objects. A reader may have several opened sessions, which may have several opened channels. Each object has a `close()` method to release all resources associated with the target object, which invokes the `close()` method on each underlying object in the chain.

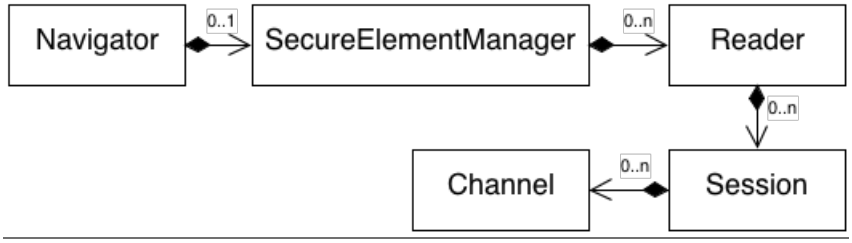

Fig. 1 The figure above represents the class relationships between the Secure Element entities introduced in this specification.

6. Access Control

In order to make sure only authorized web applications can access the Secure Element, the web application runtime MUST use the access control defined in [GP-AC], which defines a simple mechanism that protects legitimate users using non-compromised devices from malicious applications. Note that it does not protect from a compromised device that would not properly implement the Access Control Enforcer, which is addressed by the internal protection of the Secure Element itself (using e.g. PIN or secure messaging). The interface between the web application runtime and the Access Control Enforcer is out of scope of this specification.

6.1 Overall Architecture

To control which applications running on a user device are allowed to access on-card applications, several entities are involved:

- The Secure Element hosts a list of Access Rules. An access rule contains the AID of the on-card application,

the identifier of the requesting off-card application and a filter on authorized APDUs. The access rule may also contain a simple Boolean to authorize all or no communications.

- An Access Control Enforcer is running on the same device as the off-card application. Any attempt to establish a communication with an on-card application from this device triggers this enforcer, which queries the Secure Element to get the access rules (and usually cache them) and check that the requesting off-card application is authorized to communicate with the targeted on-card application.

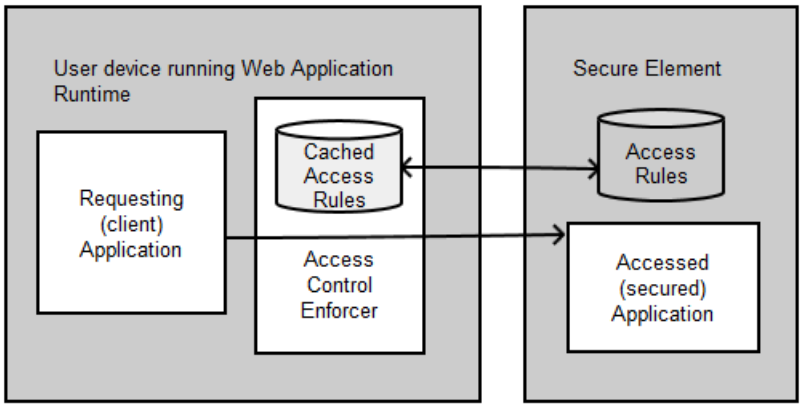

Fig. 2 The figure above shows the overall access control architecture.

Sections below describe how the GlobalPlatform Access Control is applied to web applications. Details of the GlobalPlatform Access Control mechanisms itself are defined in [GP-AC].

6.2 Client Application Identifier

The Access Control Enforcer uses an application identifier to check whether the off-card application is white listed in Access Rules. The web application runtime computes the application identifier using the following algorithm:

- let `origin` be the ASCII serialization of the web application origin as defined in [RFC6454]
- set the application identifier to be the SHA-1 digest of this `origin` value.

> NOTE
> The SHA-1 hash function is used here because the GlobalPlatform Access Control specification for now only supports 20 bytes long identifiers. A stronger algorithm will be used as soon as GlobalPlatform updates its specification to support longer values.

All applications that have the same origin MUST get the same application identifier, hence use the same Access Rules.

The GlobalPlatform Access Control Enforcer will authorize or deny access based on the conditions and algorithm defined in [GP-AC] section 4.

6.3 Additional Security Rules

To be eligible to gain access to the Secure Element API, a web application must meet the following requirements:

- The web application MUST be fetched using HTTPS protocol.
- The TLS/SSL server certificate of this HTTPS connection MUST be trusted and valid:
  - Its subject MUST match the hosting domain name.
  - Its validity dates MUST include the current date provided by the execution runtime.
  - The issuance signature chain MUST be valid.
  - The issuance chain MUST be rooted by a CA trusted by the web application runtime.
- The user MUST NOT be able to bypass these rules. Some browsers offer users to define "exceptions" to allow connections to a HTTPS URL even if SSL server certificate is invalid. Such exception MUST NOT be used to allow access to the Secure Element API.

7. General Rules for Handling of Status Word

This specification defines handling of status word as defined in ISO 7816 specifications, in order to behave correctly with any secure element application across different sectors (banking, transport, identity, etc.).

7.1 Using Transport layer T=1

The API or the underlying implementation MUST handle the protocol T=1 as specified in [ISO7816-3] and returns data (if available) and received status word to the application. The API or underlying implementation MUST never automatically issue GET-RESPONSE APDU on reception of any status word.

7.2 Using Transport layer T=0

Unless otherwise specified, when sending command APDU, this specification requires the following management for status word for all the methods defined in this document:

- For status word ‘61 XX’ the API or underlying implementation MUST issue a GET RESPONSE command as specified by ISO 7816 standard with LE=XX. The same behaviour has to be applied for the scenario where the SE returns overall response data of more than 256 bytes to the APDU. If an Error SW is received to GET RESPONSE command, all data received so far MUST be discarded and the Error SW MUST be returned.
- For the status word ‘6C XX’, the API or underlying implementation MUST reissue the input command with LE=XX as specified by ISO 7816 standard. If an Error SW is received to the reissued command, all data received so far MUST be discarded and the Error SW MUST be returned.
- For any status words different than 61xx and 6Cxx the API or underlying implementation MUST NOT handle the received status words internally (e.g.: it MUST NOT send GET RESPONSE automatically). The API or underlying implementation MUST provide the status word together with data, if data is also received with the SW to the calling web application.

Note that the handling of GET RESPONSE specified above implies that the API or underlying implementation MUST handle response data even if it is more than 256 bytes.

8. `Navigator` Interface

The `Navigator` exposes the Secure Element service.

WebIDL

```
partial interface Navigator {
    readonly attribute SecureElementManager? secureElementManager;
};
```

8.1 Attributes

`secureElementManager` of type *SecureElementManager*, readonly, nullable

When getting the `secureElementManager` attribute, the user agent MUST return the `SecureElementManager` object that provides access to available Secure Element readers. If the user agent doesn't support Secure Element features, it MUST then return `undefined`.

9. `SecureElementManager` Interface

The `SecureElementManager` interface provides access to Secure Element readers and is the source of events notifying the presence of Secure Elements.

WebIDL

```
interface SecureElementManager : EventTarget {
    attribute EventHandler? onsepresent;
    attribute EventHandler? onseremoval;
    Promise<Reader[]> getReaders();
    Promise<void>     shutdown();
};
```

9.1 Attributes

`onsepresent` of type EventHandler, nullable

Event handler for the SE-present event. This event MUST be triggered each time any of the following situations occurs:

- Web application starts while a Secure Element is present in a reader
- Web application is running, a reader which was already listed in the readers attribute but had no Secure Element now detects the presence of a Secure Element.
- Web application is running, a new reader is detected and has a Secure Element present.

The event name is `sepresent` and event type is `ReaderEvent`.

`onseremoval` of type EventHandler, nullable

Event handler for the SE-removal event. This event MUST be triggered when a Secure Element which was present in a reader is no longer present (Meaning it has been unplugged or is out of reach if it was connected through wireless communication). As soon as this event is triggered, all `Session` and `Channel` objects providing access to this Secure Element are marked as closed. The event name is `seremoval` and event type is `ReaderEvent`.

9.2 Methods

`getReaders`

This method MUST return the list of available readers in which a Secure Element may be present. Its value MUST be an empty array if no reader is available. It MUST be `null` if the `shutdown()` method has been called on this `SecureElementManager` object. Several invocations of this method MAY return different array values, because new readers may become available, while others may be disconnected.

`shutdown`

This method closes all sessions and channels opened from this `SecureElementManager` object and releases other potential internal resources. When invoked, the user agent MUST run the following steps:

1. Let *promise* be a newly-created Promise object.
2. Return *promise* and continue the following steps asynchronously.
3. If the `shutdown()` method has already been called on this object, then resolve *promise*.
4. Let *countdown* be the number of readers in the `readers` attribute and *error* an object initially `undefined`.
5. Invoke `closeSessions()` method on each `Reader` object in the array returned by the `readers` attribute.
6. Let *readerpromises* be the set of Promise objects returned by these `closeSessions()` invocations.
7. Set the `readers` attribute value to `null`.
8. If *countdown* is 0, then resolve *promise* with this `SecureElementManager` object.
9. When a *readerpromises* element is fulfilled, *countdown* is decremented. If *countdown* is 0 and *error* is `undefined`, then resolve *promise*. If *countdown* is 0 and *error* is not `undefined`, then reject *promise* with the *error* value.
10. When a *readerpromises* element is rejected, *countdown* is decremented. If *error* is `undefined` then set it to the rejected value. If *countdown* is 0 then reject *promise* with *error* value.

9.3 Events

The `sepresent` and `seremoval` events are of type `ReaderEvent`.

WebIDL

```
[Constructor(DOMString type, optional ReaderEventInit eventInitDict),
 Exposed=Window,
 Worker]
interface ReaderEvent : Event {
    readonly attribute Reader reader;
};
```

9.3.1 Attributes

`ReaderEvent.reader` of type *Reader*, readonly

The reader on which the Secure Element event occurred

9.3.2 Constructor Parameters

WebIDL

```
dictionary ReaderEventInit : EventInit {
    Reader reader;
};
```

ReaderEventInit.reader of type *Reader*
The event's reader attribute value

10. Reader Interface

Readers connected to this device are accessible through the Reader interface. A reader is the connector to a Secure Element. Given the removable nature of some Secure Elements, a reader may or may not have a Secure Element present. A reader MUST have at most one present Secure Element. A reader MAY for instance be a UICC slot, a USB smart card reader, an NFC interface or a mother board slot where a embedded Secure Element is wired.

WebIDL

```
interface Reader {
    readonly attribute boolean           isSEPresent;
    readonly attribute DOMString         name;
    readonly attribute SecureElementType secureElementType;
    readonly attribute boolean           isRemovable;
    Promise<Session> openSession();
    Promise<void>    closeSessions();
    Promise<void>    reset();
};
```

10.1 Attributes

isSEPresent of type boolean, readonly
This attribute MUST return true if a Secure Element is present in this reader. It MUST return false otherwise.

name of type DOMString, readonly
This attribute MUST return the name of the reader. This is an arbitrary name set by the system. It MAY be computed based on reader provided data.

secureElementType of type *SecureElementType*, readonly
This attribute MUST return the SecureElementType value that matches the type of the Secure Element this reader gives access to. This information may be useful for applications that target a specific Secure Element type. It may also be used to build the application user interface, to represent the Secure Element in a realistic way.

isRemovable of type boolean, readonly
This attribute MUST return true if the Secure Element cannot be detached without powering down the device.

10.2 Methods

openSession
This method establishes a communication link with a Secure Element. There may be several sessions opened at the same time: a session MUST NOT lock access to the Secure Element.

closeSessions
This method closes all sessions opened through this reader object. This also closes their associated channels.

reset
This method resets the physical interface of the Secure Element. The enabling process of the physical

interface MUST be replayed. If the physical interface does not support or allow the operation, this method MUST fail with `SEUnsupportedException`. error.

10.3 `SecureElementType` Enum

The `SecureElementType` enum identifies the type of the Secure Element a reader gives access to.

WebIDL

```
enum SecureElementType {
    "uicc",
    "smartcard",
    "ese",
    "sd",
    "other"
};
```

Enumeration description	
`uicc`	The Secure Element is a UICC used by the device to connect to a mobile network.
`smartcard`	The Secure Element is a smart card (contactless, contact or USB token).
`ese`	The Secure Element is embedded in the device.
`sd`	The Secure Element is a SD card and may be unplugged.
`other`	For any other Secure Element type not listed above.

11. `Session` Interface

A `Session` represents a connection to one of the Secure Elements available on the device. These objects can be used to open and close communication channels with an on-card application.

WebIDL

```
interface Session {
    readonly attribute Reader      reader;
    readonly attribute Uint8Array? historicalBytes;
    Promise<Channel> openBasicChannel(Uint8Array? aid, optional octet p2);
    Promise<Channel> openSupplementaryChannel(Uint8Array? aid,
                                              optional octet p2);
    Promise<void>    close();
};
```

11.1 Attributes

`reader` of type *Reader*, readonly

This attribute MUST return the reader object from which this session object was created.

historicalBytes of type Uint8Array, readonly, nullable

This attribute MUST return the historical bytes provided by the physical interface of the Secure Element or `null` if the interface does not provide it.

11.2 Methods

openBasicChannel

This methods opens the basic logical channel and selects an on-card application. Once this channel has been opened by an off-card application, it is considered to be "locked" to other applications: any other call to this method MUST fail with `SENoChannelException` error until this basic channel is closed. Some Secure Elements might always deny opening a basic channel.

If the `aid` parameter is not null, the underlying implementation of this operation MUST send a SELECT command to the Secure Element, as defined in [ISO7816-4], with following header values:

- CLA = '0x00'
- INS = '0xA4'
- P1 ='0x04' (Select by DF name/application identifier)
- P2 = `p2` parameter if present, '0x00' otherwise (First or only occurrence).

If `aid` is null, then no SELECT command is sent, the channel is opened on the default selected on-card application.

If `aid` is an empty array (array of size 0), the Lc and data field of the SELECT command MUST be omitted, so that the Issuer Security Domain is selected as defined in [GP].

This method MUST trigger the Access Control Enforcer to check the requesting off-card application is authorized to open such channel.

For the SELECT command the API MUST handle received status word as defined in section 7, except that for T=0 protocol the GET RESPONSE MUST also be sent in case of SW warning ('62 XX' or '63 XX'). If the status word for the SELECT command indicates that the SE was able to open a channel (status word '90 00' or status words referencing a warning), the API MUST keep the channel open and the channel's `openResponse` attribute MUST return the status word for the SELECT command together with data, if any data is received.

Other status words for the SELECT command which indicate that the SE was not able to open a channel MUST be considered as an error and the corresponding channel MUST NOT be opened.

Parameter	Type	Nullable	Optional	Description
aid	`Uint8Array`	✔	✘	This parameter value MUST either be: • The complete Application Identifier of the targeted on-card application; • A partial Application Identifier matching a set of targeted on-card applications. The application with the first matching AID MUST be selected; • An empty Application Identifier (array of size 0), to select the Issuer Security

				Domain; • `null`, to select the implicitly selected application on this channel.
p2	`octet`	✘	✔	The P2 parameter of the SELECT APDU executed for this channel. Except for specific needs, it is recommended to omit this method parameter. If provided, the following values are supported as defined in [ISO7816-4]: '00', '04', '08', '0C'.

`openSupplementaryChannel`

This methods opens a supplementary logical channel and selects an on-card application. The Secure Element MUST open the next available supplementary logical channel. If no more supplementary logical channels are available, this method MUST fail with `SENoChannelException` error.

If the `aid` parameter is not null, the underlying implementation of this operation MUST send to the Secure Element a SELECT command, as defined in [ISO7816-4], with following header values:

- CLA = '0x01' to '0x03', '0x40 to 0x4F' (as chosen by the Secure Element)
- INS = '0xA4'
- P1 ='0x04' (Select by DF name/application identifier)
- P2 = `p2` parameter if present, '0x00' otherwise (First or only occurrence).

If `aid` is null, then no SELECT is sent, the channel is opened on the default selected on-card application. In the case of a UICC it is recommended that the API rejects the opening of the supplementary logical channel without a specific AID.

If `aid` is an empty array (array of size 0), the Lc and data field of the SELECT command MUST be omitted, so that the Issuer Security Domain is selected as defined in [GP].

This method MUST trigger the Access Control Enforcer to check the requesting off-card application is authorized to open such channel.

For the SELECT command the API MUST handle received status word as defined in section 7, except that for T=0 protocol the GET RESPONSE MUST also be sent in case of SW warning ('62 XX' or '63 XX'). If the status word for the SELECT command indicates that the SE was able to open a channel (status word '90 00' or status words referencing a warning), the API MUST keep the channel open and the channel's `openResponse` attribute MUST return the status word for the SELECT command together with data, if any data is received.

Other status words for the SELECT command which indicate that the SE was not able to open a channel MUST be considered as an error and the corresponding channel MUST NOT be opened.

Parameter	Type	Nullable	Optional	Description
aid	`Uint8Array`	✔	✘	This parameter value MUST either be: • The complete Application Identifier of the targeted on-card application;

				• A partial Application Identifier matching a set of targeted on-card applications. The application with the first matching AID MUST be selected; • An empty Application Identifier (array of size 0), to select the Issuer Security Domain; • `null`, to select the implicitly selected application on this channel.
p2	`octet`	✘	✔	The P2 parameter of the SELECT APDU executed for this channel. Except for specific needs, it is recommended to omit this method parameter. If provided, the following values are supported as defined in [ISO7816-4]: '00', '04', '08', '0C'.

`close`

This method closes the connection session to the Secure Element. This MUST close any channels opened through this session object. Invoking `close()` method on an already closed session has no effect. After invoking this method, calling any other method on this object MUST fail with an SEClosedException error.

12. `Channel` Interface

A `Channel` represents an [ISO7816-4] channel opened to a Secure Element. It can be either a supplementary logical channel or the basic logical channel. It can be used to send commands to and receive responses from an on-card application.

WebIDL

```
interface Channel {
    readonly attribute Session     session;
    readonly attribute ChannelType channelType;
    readonly attribute SEResponse? openResponse;
             attribute int?        timeout;
    Promise<SEResponse> selectNext();
    Promise<SEResponse> transmit(SECommand cmd);
    Promise<Uint8Array> transmitRaw(Uint8Array cmd);
    Promise<void>       close();
};
```

12.1 Attributes

`session` of type *Session*, readonly

This attribute MUST return the session object from which this Channel object was created.

`channelType` of type *ChannelType*, readonly

This attribute MUST return this channel type.

`openResponse` of type *SEResponse*, readonly, nullable

This attribute MUST return the Secure Element's response to the channel opening operation. It MUST be `null` if the channel was opened on the default on-card application (no AID was provided in the open

channel operation).

`timeout` of type int
: The timeout in ms of commands sent through this channel. Any command sent through this channel is guaranteed to be aborted after `timeout` ms.

12.2 Methods

`selectNext`
: Updates the targeted on-card application of this channel to be the next one matching the partial Application Identifier passed when this channel was open. Invoking this method MUST fail with an `SEInvalidStateException` error if this channel was not open with a partial AID or with an `SENoApplicationException` error if there is no next application matching that partial AID. In that case the application associated to this channel is unchanged. If a next application has been found and associated to this channel, this operation succeeds and returns the Secure Element's response.

  For the SELECT command the API MUST handle received status word as defined in section 7, except that for T=0 protocol the GET RESPONSE MUST also be sent in case of SW warning (‘62 XX’ or ‘63 XX’). If the status word for the SELECT command indicates that the SE was able to open a channel (status word ‘90 00’ or status words referencing a warning), the API MUST keep the channel open and the channel's `openResponse` attribute MUST return the status word for the SELECT command together with data, if any data is received.

`transmit`
: This method transmits a command to the Secure Element. The user agent MUST ensure the synchronisation between all the concurrent calls to this method: a command MUST NOT be sent to a Secure Element while a response is still pending on any channel from this same Secure Element.

  This method MUST trigger the Access Control Enforcer to check the requesting off-card application is authorized to send such command on this channel.

  The channel information in the class byte in the APDU command MUST be ignored. The system MAY modify the class byte of the command to ensure the APDU is transported on this channel. To ensure the invoking web application does not exit from the scope of this channel, the user agent MUST reject the following commands with `SEInvalidValueException` error value:

  - MANAGE_CHANNEL (INS=0x70)
  - SELECT by DF Name (INS=0xA4 and P1=04)

Parameter	Type	Nullable	Optional	Description
cmd	`SECommand`	✘	✘	The command to send to the on-card application.

`transmitRaw`
: This method behaves exactly as the `transmit` method above, except that both its parameter and the response are passed as a raw binary data. Before transmitting the command to the Secure Element, the implementation of this method MUST set the logical channel in the class byte of the command so that it fits the channel allocated to this `Channel` object.

  This method MUST trigger the Access Control Enforcer to check the requesting off-card application is authorized to send such command on this channel.

Parameter	Type	Nullable	Optional	Description
cmd	`Uint8Array`	✘	✘	The raw command to send to the on-card application. The channel information in the class byte of the command (first octet of the cmd data) MUST be ignored.

`close`

Closes this channel object. If the method is called when the channel is already closed, this method MUST be ignored. The `close()` method MUST wait for completion of any pending `transmit()` operation before closing the channel.

If the channel is not the basic channel, then a MANAGE CHANNEL Close (P1=0x80) command MUST be sent to the Secure Element for closing the channel.

If the channel is the basic channel, then the following procedure MUST be followed for closing the channel:

1. Send a MANAGE CHANNEL Reset (P1=0x40) command, as defined by [ISO7816-4].
2. If previous command is not supported by the card (i.e. card returns an error SW), then SELECT by DF Name (P1=0x04; P2=0x00) command MUST be sent with an empty AID (Lc=0x00). All data returned for the SELECT by DF Name command MUST be discarded.

In all cases the last SW of the MANAGE CHANNEL Close, MANAGE CHANNEL Reset, or SELECT by DF Name command MUST be ignored and the channel object MUST be closed by the API. After invoking this method, calling any other method on this channel object MUST then fail with an SEClosedException error.

12.3 `ChannelType` Enum

The `ChannelType` enum identifies the type of the Secure Element channel.

WebIDL

```
enum ChannelType {
    "basic",
    "supplementary"
};
```

Enumeration description	
`basic`	Basic logical channel, as defined in [ISO7816-4] (channel number 0).
`supplementary`	Supplementary logical channel, as defined in [GP] (channel number > 0).

13. `SECommand` Interface

The `SECommand` interface represents an APDU command that can be sent to a Secure Element.

WebIDL

```
[Constructor(octet cla, octet ins, octet p1, octet p2, optional Uint8Array data, optional octet
 le, optional boolean isExtended)]
interface SECommand {
    attribute octet          cla;
    attribute octet          ins;
    attribute octet          p1;
    attribute octet          p2;
    attribute Uint8Array?    data;
    attribute unsigned short le;
    attribute boolean        isExtended;
};
```

13.1 Attributes

`cla` of type octet
: Class byte.

`ins` of type octet
: Instruction byte.

`p1` of type octet
: First octet of the parameter bytes.

`p2` of type octet
: Second octet of the parameter bytes.

`data` of type Uint8Array, nullable
: Data field bytes or `null` if command has no data.

`le` of type unsigned short
: The length of the expected response data or `undefined` if application does not require a specific length.

`isExtended` of type boolean
: If set to `true` the command MUST be transmitted using extended length encoding.

14. `SEResponse` Interface

The `SEResponse` interface represents an APDU response received from a Secure Element.

WebIDL

```
[Constructor(Uint8Array raw)]
interface SEResponse {
    readonly attribute Channel    channel;
    readonly attribute octet      sw1;
    readonly attribute octet      sw2;
    readonly attribute Uint8Array data;
    boolean isStatus(octet? sw1, octet? sw2);
};
```

14.1 Attributes

`channel` of type *Channel*, readonly
: This attribute MUST return the channel object that was used to transmit the command which triggered this response object.

`sw1` of type octet, readonly
First octet of response's status word

`sw2` of type octet, readonly
Second octet of response's status word

`data` of type Uint8Array, readonly
The response's data field bytes

14.2 Methods

`isStatus`
Utility method to test the status word of an APDU response. This method MUST return `true` if the parameters match the value of the response.

Parameter	Type	Nullable	Optional	Description
sw1	`octet`	✔	✘	Value to compare to the first octet of response's status word or `null` if this first octet may have any value.
sw2	`octet`	✔	✘	Value to compare to the second octet of response's status word or `null` if this second octet may have any value.

15. Error Types

In the interfaces defined above, some methods return a Promise object. If an error occurs during the execution of any of these methods, the `reject()` method of Promise's resolver MUST be invoked with an error value of one of the following DOMException subtypes:

SESecurityException
The requested operation does not match the Secure Element access conditions, as defined in Access Control section

SEIoException
Communication error

SEInvalidStateException
The target object was not in the proper state to execute the operation

SEInvalidValueException
The method was invoked with an incorrect parameter value

SENoChannelException
The channel could not be opened because no channel is available

SENoApplicationException
The requested on-card application was not found on the Secure Element

SEClosedException
The operation could not be fulfilled because the target object is closed

SEUnsupportedException
The operation is not supported by the reader or Secure Element.

SEUnknownException
Internal error, no further details available

16. Code Example

The javascript code excerpt below shows how a web application can wait for a card to be present and send it an APDU command:

EXAMPLE 1

```
// my application identifier
var myAppId = new Uint8Array(
  [0xA0, 0x00, 0x00, 0x00, 0x18, 0x0C, 0x00, 0x00, 0x01, 0x63, 0x42, 0x00]
);
// my application command
var myAppCmd = new SECommand(0x00, 0xCA, 0x9F, 0x7F, undefined, 0x2A);

// register sepresent event handler
navigator.secureElementManager.onsepresent = (event) => {

  var session;

  // open session
  event.reader.openSession()

  .then( _session => {
    session = _session;
    return session.openBasicChannel(myAppId);
  })

  .then( _channel => {
    return _channel.transmit(myAppCmd);
  })

  .then( _response => {
    if (_response.isStatus(0x90, 0x00)) {
      // process success response
      mySuccessHandler(_response);
    } else {
      // process unexpected response
      myFailureHandler(_response);
    }
  })

  .catch( error => {
    // error handler
    myErrorHandler(error);
  })

  .then( _ => {
    if (session) {
      session.close();
    }
  })

}
```

A. Changes

The complete list of changes can be viewed on Github. You can also check the issues.

B. References

B.1 Normative references

[DOM4]
Anne van Kesteren; Aryeh Gregor; Ms2ger; Alex Russell; Robin Berjon. W3C. *W3C DOM4*. 19 November 2015. W3C Recommendation. URL: https://www.w3.org/TR/dom/

[ES6]
Ecma International. *ECMA-262, 6th Edition / Draft January 20, 2014*. Draft. URL: http://ecma-international.org/ecma-262/6.0/index.html

[ETSI-102216]
ETSI. ETSI. *TR 102 216 : Smart cards; Vocabulary for Smart Card Platform specifications*. URL: https://webapp.etsi.org/workprogram/Report_WorkItem.asp?WKI_ID=18815&curItemNr=1&totalNrItems=1&optDisplay=10&qSORT=HIGHVERSION&qETSI_STANDARD_TYPE=%27TR%27&qETSI_NUMBER=102+216

[GP]
GlobalPlatform. GlobalPlatform. *Card specifications*. URL: https://www.globalplatform.org/specificationscard.asp

[GP-AC]
GlobalPlatform. GlobalPlatform. *Secure Element Access Control v1.1 | GPD_SPE_013*. URL: https://www.globalplatform.org/specificationsdevice.asp

[HTML]
Ian Hickson. WHATWG. *HTML Standard*. Living Standard. URL: https://html.spec.whatwg.org/multipage/

[ISO7816-4]
ISO/IEC. ISO. *7816-4 Identification cards -- Integrated circuit cards -- Part 4: Organization, security and commands for interchange*. URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=54550

[ISO7816-5]
ISO/IEC. ISO. *7816-5 Identification cards -- Integrated circuit cards -- Part 5: Registration of application providers*. URL: http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=34259

[RFC2119]
S. Bradner. IETF. *Key words for use in RFCs to Indicate Requirement Levels*. March 1997. Best Current Practice. URL: https://tools.ietf.org/html/rfc2119

[RFC6454]
A. Barth. IETF. *The Web Origin Concept*. December 2011. Proposed Standard. URL: https://tools.ietf.org/html/rfc6454

[WEBIDL]
Cameron McCormack; Boris Zbarsky. W3C. *WebIDL Level 1*. 8 March 2016. W3C Candidate Recommendation. URL: https://www.w3.org/TR/WebIDL-1/

B.2 Informative references

[ISO7816-3]

ISO/IEC. ISO. *7816-3 Identification cards -- Integrated circuit cards -- Part 3: Cards with contacts -- Electrical interface and transmission protocols*. URL: http://www.iso.org/iso/fr/home/store/catalogue_tc/catalogue_detail.htm?csnumber=38770

[NFC]

W3C Near Field Communications (NFC) Community Group. W3C. *Web NFC*. URL: https://www.w3.org/community/web-nfc/

[WEBCRYPTO]

W3C Web Cryptography Working Group. W3C. *Web Cryptography API*. URL: https://www.w3.org/TR/WebCryptoAPI/