Java Card

Introducing Java Card Device IO to secure peripherals on I3C bus

Patrick Van Haver
Security Solutions Architect
Oracle - Java Platform Group

Dec. 2021
Safe Harbor Statement

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, timing, and pricing of any features or functionality described for Oracle’s products may change and remains at the sole discretion of Oracle Corporation.
Java Card
Reference runtime for Secure Elements

Access Control
- Biometry
- Authorization
- Data Confidentiality
- Secure communication
- Data Integrity
- Secure storage
- Secure transactions

Authentication
- Device Attestation
- Identification
- Biometry
- Secure transactions
- Device Attestation
- Secure transactions
- Home automation
- Connected cars
- Manufacturing
- Connectivity, mobile payment, e-ticket, smart-key...
- Security tokens
- Smart-city
- Smart-metering
- Identification
- Secure Element
- SIM card
- Payment card
- ID and access cards
- ePassport

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.
Java Card
Reference runtime for Secure Elements

- Identification
  - Biometry
  - Authorization
- Access Control
  - Root of trust
  - Data Integrity
- Authentication
  - Secure communication
  - Secure storage
- Secure transactions
- Device Attestation

Secure Element connectivity, mobile payment, e-ticket, smart-key...
Secure Element security-tokens
SMART-card
SIM card
ID and access cards
Payment card
ID and access cards
Security tokens
Secure Element
Connected cars
Connected Smart-cars
Home automation
Secure Element
Secure transactions
Secure communication
Secure storage
Device Attestation
Root of trust
Data Integrity
Authorization
Biometry
Identification
Secure Element
Copyright © 2021, Oracle and/or its affiliates. All rights reserved.
Integration of the Secure Element on the I/O bus

Example using I3C bus
I3C used as the main interface with host

To support existing SE use-cases over I3C I/O interface

- The SE is an I3C target
  - Replies to requests received from host

- Benefit from I3C interface
  - Simplified integration and wiring with host
  - Lower power consumption
  - Higher data rates

- No impacts on existing applications
  - Only physical layer up to transport layer are updated
  - Encapsulated in APDU class

Device

Host

Device Application

OS

Host CPU

I3C host controller

SE

Applet

Java Card Platform

APDU stack

Secure CPU

I3C interface

I3C target

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.
I3C interface used by Applets to access peripherals

*To implement new use-cases using shared peripherals*

- The SE becomes (secondary) controller
  - Accessing peripherals (I3C targets)

- SE takes control over peripherals
  - To implement new scenarios: configure sensors, read or write data
  - To augment existing use-cases with information coming from sensors (e.g. location info for authentication)

- Requires API extensions
  - Device IO API
I3C interface used by Applets to access peripherals

*To implement new use-cases using dedicated peripherals*

- Security Subsystem
  - To enhance security – exclusive control and access to a peripheral
  - To lower power consumption – segmented subsystems (with Hot-Join) independently powered only when needed
Design also relevant for standalone secure elements
Secure peripherals use-cases

• **Smart-metering**
  Java Card application uses some sensors to detect tampering and access meter data to enforce measurement integrity.

• **Access management (smart-lock, car key, ...)**
  Java Card application is triggered when pressing a button and generates a temporary code to open a lock.

• **Biometric identification**
  Java Card application uses biometric sensor to securely capture fingerprint and perform matching

• **Display control**
  Java Card application uses a peripheral to control data to be displayed (dynamic CVV, OTP, crypto-wallet balance, ...)

Copyright © 2021, Oracle and/or its affiliates. All rights reserved.
Java Card Device IO API
## Interface Summary

<table>
<thead>
<tr>
<th>Interface</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Device</td>
<td>The Device interface represents devices in the system.</td>
</tr>
<tr>
<td>DeviceConfig</td>
<td>The DeviceConfig class is the base interface for all device configuration classes.</td>
</tr>
<tr>
<td>GPIOPin</td>
<td>The GPIOPin interface provides methods for controlling a GPIO pin.</td>
</tr>
<tr>
<td>GPIOPinListener</td>
<td>The GPIOPinListener interface defines methods for getting notified of GPIO pin value changes.</td>
</tr>
<tr>
<td>I2CDevice</td>
<td>The I2CDevice interface provides methods for an I2C controller to send and receive data to/from an I2C peripheral.</td>
</tr>
<tr>
<td>I3CDevice</td>
<td>The I3CDevice interface provides methods for an I3C controller to send and receive data to/from an I3C peripheral.</td>
</tr>
<tr>
<td>SPIDevice</td>
<td>The SPIDevice interface provides methods for an SPI controller to send and receive data to/from an SPI peripheral.</td>
</tr>
<tr>
<td>UARTDevice</td>
<td>The UARTDevice interface provides methods for controlling and accessing a UART (Universal Asynchronous Receiver/Transmitter).</td>
</tr>
<tr>
<td>UARTEventListener</td>
<td>The UARTEventListener interface defines methods for getting notified of events fired by devices that implement the UARTDevice interface.</td>
</tr>
</tbody>
</table>

## Class Summary

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceManager</td>
<td>The DeviceManager class provides methods to create and configure Device instances.</td>
</tr>
<tr>
<td>GPIOPinConfig</td>
<td>The GPIOPinConfig class encapsulates the static and dynamic configuration parameters of a GPIO pin. An instance of GPIOPinConfig is immutable and can be used to create a GPIOPin with the specified configuration parameters.</td>
</tr>
<tr>
<td>I2CDeviceConfig</td>
<td>The I2CDeviceConfig class encapsulates the static and dynamic configuration parameters of an I2C device. An instance of I2CDeviceConfig is immutable and can be used to create an I2CDevice with the specified configuration parameters.</td>
</tr>
<tr>
<td>I3CDeviceConfig</td>
<td>The I3CDeviceConfig class encapsulates the static and dynamic configuration parameters of an I3C device. An instance of I3CDeviceConfig is immutable and can be used to create an I3CDevice with the specified configuration parameters.</td>
</tr>
<tr>
<td>SPIDeviceConfig</td>
<td>The SPIDeviceConfig class encapsulates the static and dynamic configuration parameters of an SPI device. An instance of SPIDeviceConfig is immutable and can be used to create a SPIDevice with the specified configuration parameters.</td>
</tr>
<tr>
<td>UARTDeviceConfig</td>
<td>The UARTDeviceConfig class encapsulates the static and dynamic configuration parameters of a UART. An instance of UARTDeviceConfig is immutable and can be used to create a UARTDevice with the specified configuration parameters.</td>
</tr>
</tbody>
</table>

---

**Interfaces to interact with connected devices**
- GPIO
- I2C
- I3C
- SPI
- UART

**Configuration parameters for each device type**

---

To create and configure Device instances
Example: verifying fingerprint

```java
try {
    // get a device instance with specified configuration
    device = DeviceManager.getInstance(IO_ID1, I3DDeviceConfig.build(fpaddress, DTM_SDR0));

    // open communication with fingerprint sensor
    device.open();

    // send command to capture fingerprint
    device.write(cmd_get_fp);

    // read fingerprint and check
    len = device.read(rxbuffer);
    res = bioMather.initMatch(rxbuffer.array(), (short)0, len);

    while (res == MATCH_NEED_MORE_DATA) {
        len = device.read(rxbuffer.clear());
        res = bioMather.match(rxbuffer.array(), (short)0, len);
    }

    if ((res >= MINIMUM_SUCCESSFUL_MATCH_SCORE) {
        // success ...
    }
} finally {
    // close device when done
    device.close();
}
```
Conclusion
Takeways

• Secure Elements integration into devices requires new IO interface types

• I2C and SPI are already used for the main IO interface of the SE, I3C is on the way

• Secure Elements applications can also access peripherals connected on the same bus, addressing new use-cases and offering effective solutions to enhance device security

• Device IO API is a mean for Java Card applications to access peripherals connected to the secure element and achieve higher security
Questions ?
More Information

https://www.oracle.com/java/technologies/java-card-tech.html

Java Card 3.1 Documentation
Includes all documentation for the Java Card platform and Development Kit.

Java Card Platform Specification 3.1
Latest release of the Java Card specification and the reference for Java Card products.

Java Card Development Kit Tools
The Java Card Development Kit Tools are used to convert and verify Java Card applications. The Tools can be used with products based on version 3.1, 3.0.5 and 3.0.4 of the Java Card Specifications.

Java Card Development Kit Simulator
The Java Card Development Kit Simulator includes a simulation component and Eclipse plug-in. Combined with the Java Card Development Kit Tools, it provides a complete, stand-alone development environment.

contact: patrick.van.haver at oracle.com