JAVA CARD

Java Card 3.1 Unveiled

The major new release tunes the popular Java platform for IoT.

by Nicolas Ponsini

With close to 6 billion Java Card–based devices deployed each year, Java Card is the leading software platform for running security services on secure chips. These are used to protect smartphones, banking cards, government services, and other business needs.

Earlier this year, Oracle released Java Card 3.1—a major version of the Java Card specifications and the Java Card Development Kit. It is the most extensive update to the technology in several years, and it introduces new features to address the IoT market and new secure-elements hardware.

Two articles will detail the content of the 3.1 release. This first article presents the release and focuses on its extensions and services for IoT. An upcoming article will cover features that benefit payment, identity, and cellular connectivity markets.

What Is Java Card 3.1?

Java Card enables secure elements, such as smartcards and other tamper-resistant security chips, to host applications that employ Java technology. Java Card 3.1 is a major release updating all the components for developing Java Card products and applications, including the Java Card specifications and the development and compliance tools.

The specifications provide the basis for cross-platform and cross-vendor application interoperability:

- The Virtual Machine Specification provides the instruction set of the Java Card Virtual Machine, which is a supported subset of the Java language, and the file formats for installing applications and libraries into Java Card devices.
- The Runtime Environment Specification defines the necessary behavior of the runtime environment in any implementation of the technology. The Java Card APIs complement the Java Card Runtime Environment Specification and describe the framework exposed by the technology.
- The Java Card Platform Specification facilitates the development and deployment of secure applications and introduces new functionality to support IoT security.

The Java Card Development Kit is a standalone environment for developing applications. It also includes a simulator and integration with the Eclipse IDE to facilitate the testing. Both the Java Card Platform Specification and the development kit are freely available to developers.

Oracle licenses a Java Card Reference Implementation and the Java Card Technology Compatibility Kit (TCK) to its commercial customers. The TCK ensures application interoperability for a Java Card implementation on a particular platform.

A key goal of version 3.1 is to ensure the availability of security services on a large range of secure hardware, including smartcards, embedded chips, secure enclaves within microprocessor units (MPUs) and microcontroller units (MCUs), and removable SIMs. It was designed to support the growth of existing Java Card markets, such as payment, identity, and connectivity markets, while enabling new IoT use cases with dedicated features. This article covers those features.

IoT Extensions

New features mirror the extended role that a Java Card secure element plays in a connected device. Specifically, the features include a new extensible I/O model and a range of security services to facilitate the design of new security applications. Let's examine these.

Secure elements embedded within a device or integrated within the system on a chip (SoC) of a device have recently been bringing security directly into the heart of devices. This enables use cases that establish a direct channel between a secure element and device peripherals. Making use of this capability requires specialized protocols at the application layer. To that end, Java Card 3.1 introduces an I/O framework, including the `javacardx.framework.event` package and the `javacardx.framework.nio` package, that allows applications to have logical access to device peripherals.

The `javacardx.framework.event` package is the base framework used by platform implementers to extend their platform with specialized APIs defining new I/O protocols or interfaces with peripherals. It contains:

- An `EventSource` that represents any peripheral or I/O interface in the device host or the secure element itself. Sources of events could be, for example, a GPIO pin or port, a UART interface, memory-mapped I/O, an I2C bus, a watchdog timer, and so on.
- An `EventListener` that enables code to handle peripheral or I/O events coming from a given source. The specification provides the default base interface, which is extended by platform implementers.
- An `EventRegistry`, which is a class used by applications to register listeners with a source of events.

The `javacardx.framework.nio` package contains classes for parsing and extracting structured information from raw data in an efficient way. These classes enable access to those data items from the heap and also from external memory (such as from a peripheral).

Numerous use cases can benefit from this I/O framework. For example, a Java Card application can directly read and verify fingerprint data from a biometric sensor. There is no need to go through the host device to transfer data from the biometric sensor to the main processor of the secure element, nor is there a need to tunnel data into application protocol data unit (APDU) commands to overcome associated constraints such as bandwidth, timing, ordering, priorities, execution context, and so on.

In IoT solutions, the enforcement of security policies can benefit from access to device peripherals and from collection of their data for decision-making at the edge. For example, the secure-element application in a smart meter could use localization or a motion sensor to detect abnormal situations and react accordingly. The application could also be used to securely configure attached peripherals and ensure the integrity of the control plane.

Security Services

Security services in Java Card 3.1 include the Certificate API, the Key Derivation API, the Monotonic Counter API, and the System Time API. Let's look at these in more detail.

Certificate API. Cryptographic certificates are critical for security and serve as a basis in a public key infrastructure (PKI) to establish trust between different entities. A notable example of a protocol using

cryptographic certificates is the Transport Layer Security (TLS) protocol. Based on certificate chains, a client (such as an IoT gateway) and a server (for example, an IoT cloud service) can authenticate each other.

The `javacardx.security.cert` package is an efficient way to manage cryptographic certificates such as X.509 certificates for memory- and resource-constrained devices.

With Java Card's Certificate API, it is possible to verify a certificate signature, select and check some of its fields and extensions, and access its public key—without needing to create a dedicated certificate object that is potentially useless in the future. You can also build a certificate object (for example, for root certificates) that will be reused later, while deciding on fields and extensions that need to be associated with this certificate object and storing only useful components of the certificate.

With these mechanisms, an application has an efficient way to verify a certificate chain, check sensitive certificate fields, and keep track of trusted public keys.

Key Derivation API. Pseudorandom functions (PRFs) and key derivation functions (KDFs) are widely used in cryptography to derive sensitive data such as a secret key. They make it possible to stretch a secret or to derive multiple keys from it.

A typical usage is the derivation of a password to store a derived value without needing to store the initial password value. Another common example is the derivation of the shared secret established by a key-agreement operation in Diffie-Hellman (DH) or Elliptic-Curve Diffie Hellman (ECDH) key exchange. Another example is the TLS handshake protocol, which uses a PRF applied to a shared secret in between a client and a server to generate the cryptographic block material used during a TLS session between the two peers.

Java Card 3.1 introduces the `javacardx.security.derive` package. Its class `DerivationFunction` permits the management of both PRF and KDF algorithms, and it is easily extensible. Currently, eight algorithms are proposed that enable support for the International Civil Aviation Organization (ICAO) or TLS protocols, among others.

In addition, the Key Derivation API guarantees both the security of the derivation keys and the derived keys by encapsulating them into trusted objects.

Monotonic Counter API. To prevent replay attacks, numerous security protocols use monotonic counters, which are counters whose value can only increase. Once the value of a monotonic counter has been used, the counter is incremented (typically by 1). Thus, if a counter value has been attached to a given protocol payload at a certain time, it is guaranteed that the same counter value cannot be reused and attached to the same protocol payload later. This allows a third party consuming the protocol payload to know whether it has already been used by checking the counter value.

Device attestation (also known as *remote attestation*) is an example of a payload that needs to be protected against replay attacks. A remote attestation is a signature of software measurements running on a given device, and the attestation is sent to a third party. The third party can check whether a device is running unaltered software and can make sure the attestation is current and not replayed from the past.

External secure storage is another example requiring a monotonic counter. DRM licenses or the number of PIN entry tries are typical sensitive data protected by a Java Card secure element. In some hardware architectures, like the ones with an integrated secure element, such sensitive data might be stored in the memory flash of the host device; that is, in storage external to the secure element hardware itself. Thus, untrusted and rogue applications from the device host might be able to save and restore later such sensitive data. In the case of a DRM license, this risk would mean that access to a content item is granted an unlimited number of times instead of the initial limited times granted

originally. Hence, in addition to integrity and confidentiality, the secure element must guarantee an anti-replay protection to sensitive data stored in external secure storage.

The `javacardx.security.util` package and its `MonotonicCounter` class enable the creation and management of multiple monotonic counters of up to 64 bits each. The Monotonic Counter API guarantees the atomicity of the update of the counter value.

System Time API. Time stamping and time interval calculation are important security operations. Time stamps enable you to record or check the time at which an event occurred. Estimating a time interval allows you to limit the duration of a transaction, for instance.

Java Card 3.1 introduces the package `javacardx.framework.time`, which has two classes:

- `SysTime` serves to retrieve the uptime; that is, the time elapsed since system boot. It does not require an internal clock.
- `TimeDuration` represents a time duration with microsecond resolution. Several operations, such as comparisons and conversions as well as plus and minus operations, are supported.

Java Card's System Time API can support a variety of use cases related to device security. For example, consider an IoT device managing a temperature sensor in the chemical industry. This monitoring system is critical and needs to react to unexpected temperature variations. With the new I/O mechanism introduced in Java Card 3.1 and the System Time API, an application has the ability to retrieve a temperature value securely and to assess the elapsed time since the beginning of the measurement. If the time is too short or too long compared with the average expected value, this is reported to the monitoring system, which will trigger corrective operations such as changing the sensor or checking for a corruption of the IoT device software.

Conclusion

In this article, I described the major new IoT-oriented features of Java Card 3.1 and I detailed some of the related use cases in IoT security. For example, through trusted peripherals, Java Card can secure the "last yard" between devices, gateways, and attached peripherals, enabling trust and the exchange of sensitive data at the very edge. A secure channel can be established between peripherals and security chips to allow out-of-band communication for sensitive data (for example, biometric information or the provisioning of root-of-trust credentials).

Similarly, through device attestation, a Java Card secure element in an IoT device can support multiple proprietary or standard secure boot and device attestation mechanisms without requiring a dedicated security chip. This enables a single secure chip to be used in multiple attestation ecosystems and ensures compatibility with future standards.

In an upcoming article, I will describe the enhanced deployment model and core features as well as cryptographic extensions proposed by the 3.1 release.

Also in This Issue

Nicolas Ponsini

Nicolas Ponsini M.Sc. & CISSP is a seasoned security solutions architect at Oracle. His technical expertise and customer relationships experience are

gathered to serve the Oracle Global Sales Unit for the Business Development of Java and Internet of Things Oracle products. In the past 15 years, he successfully defined, designed, developed and sold security solutions targeting major players in embedded platforms and cloud services. He is an expert in Security, Cryptography, IoT and Trusted Execution Environment and managed successful developments and integrations from the bottom up i.e from SoC to the Cloud. He succeeded in several cutting edge realizations with major players in TEE, DRM, NFC, trusted computing domains. He owns nine patents in related areas.

Share this Page