Universal Key Ring – The Time Has Come!

Anders Rundgren, WebPKI.org

This presentation outlines a security architecture, provisioning, and management scheme for secure cryptographic keys, targeting a wide variety of applications including Virtual SIMs, On-line Banking, Payments, e-Government Access, and Enterprise Login.

In the core of the architecture there is component coined SKS (Secure Key Store), which leverages the TEE (possibly aided by a local security processor).

To facilitate easy enrollment of SKS keys, a matching browser-based provisioning protocol called KeyGen2 has been developed as well.

Since cryptographic keys (unlike files), usually represent “relationships” to external parties, the scheme provides extensive support for different policies including an ACL system which through the OS/TEE layers, governs which applications a key may be used with.

A side-effect of this arrangement is that cryptographic keys become first-class OS objects like files.

This effort is complementary to FIDO alliance. In fact, it seems quite feasible building FIDO alliance products and SKS/KeyGen2 on the very same security platform.
Core OS Objects

- Devices
- Files
- Users
- Processes
- Keys
Typical Applications

Income declaration

Year: 2016
Name: Marion Anderson
Citizen code: 19950710-1518
Declared income: Fair ($30000-$99999)

Secure payments on the web as well as in brick and mortar shops

eGovernment signature applications like income declarations, change of address and permits requests

Virtual SIM-cards enable you to buy, carry and use multiple subscriptions in an easy way

Virtualized SKS SIM Credentials =
Secure Key Store – A Three Layer Architecture

The SE only holds static data: a Device Certificate, a matching Attestation Key and a Master Key which is used for wrapping user keys. The Attestation Key signs session keys which KeyGen2 uses for secure key provisioning and management. The SE generates wrapped keys and as well as performing standard cryptographic operations on wrapped keys. That is, user keys are never exposed in clear.

The TEE performs all access control to keys as well as having exclusive access to the SE. Core key data is stored in the TEE while encrypted key material, logotypes and attributes are stored in the Credential Database.

The Operating System invokes the TEE and also provides the TEE with User and Application data required for key access control based on ACLs attached to key entries.

The SKS Native API

Key ID | ACL | User | PIN | Status | Pointer
---|---|---|---|---|---

The SKS Driver Module

Credential Database

Key Entries

Device Certificate | Attestation Private Key | Symmetric Master Key | Crypto Processor
---|---|---|---

SKS - Secure Key Store

SKS Native API

Operating System

TEE – Trusted Execution Environment (like ARM TrustZone™)

SE- Security Element (Optional Crypto Peripheral inside of the CPU)
Key + “Decoration” = Credential

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key</td>
<td><em>Mandatory</em>: Asymmetric (private) or Symmetric (secret) key</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Certificate</td>
<td><em>Mandatory</em>: X.509 certificate having two uses:</td>
</tr>
<tr>
<td></td>
<td>• Support for PKI-based applications</td>
</tr>
<tr>
<td></td>
<td>• Providing a “name” for key management operations</td>
</tr>
<tr>
<td>Algorithms</td>
<td><em>Optional</em>: Set of algorithms permitted to use with the key</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Images</td>
<td><em>Optional</em>: For usage in GUIs. Type information enable selecting appropriate images for different scenarios</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>PIN</td>
<td><em>Optional</em>: For key unlock. May be substituted or complemented with biometrics if the hardware supports that</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Attributes</td>
<td><em>Optional</em>: Arbitrary text and binary properties containing things like <em>URLs</em>, <em>Public keys</em>, and <em>Constants</em> to be used by associated applications</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ACL</td>
<td><em>Optional</em>: Access Control List protecting keys from illicit access</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>Code</td>
<td><em>Not Supported.</em> Trusted Credentials != Trusted Applications</td>
</tr>
</tbody>
</table>
Demo - Saturn (Payment Authorization)
Bank-to-Bank Payments
State Diagram

Request Signature

Merchant

User + Wallet

Virtual Cards

Virtual Card Properties
- Signature Key
- Encryption Key
  - Account Type URI
  - URL to User Bank
  - User Account ID
  - PIN
  - Card Logotype
  - ...

“Checkout/Pay” ①
(Scenario-dependent)

“PaymentClientRequest” ②
(Scenario-dependent “channel" technology used for Merchant to Wallet communication)

Select Card

Authorize

PIN

User Bank

User Bank Lookup*
(Signed & Encrypted)

Commit Signature

User Bank Verification*

Merchant Bank

User Bank

User Bank Lookup*

Decryption Key

“PayerAuthorization” ③

“AuthorizationRequest” ④
(HTTP POST)

Existing Payment Rails
(Bank-to-Bank Payments)

“AuthorizationResponse” ⑤
(HTTP Response)

All transaction steps are now available in a single object where each layer is signed and embeds inner layers

Sample application that was built using SKS and KeyGen2 for Storing/Using respectively Issuing Virtual Cards

Commit Signature

User Bank Verification*

“Accept”
The Missing Link – Credential Provisioning

KeyGen2: End-to-end-secured credential provisioning and management protocol specifically designed for SKS

Application

Bank

Government

Etc...

Credential Issuers (CAs or IdPs)

Application Level

Standard Crypto API

KeyGen2 Proxy

Browser

Networking

JSON Processing

Content Aggregation

User Interaction

SKS Native API
Demo – Enrollment using KeyGen2
Project Status – February 2017

• SKS software emulator in Java
• Android “App” implementing SKS, KeyGen2 and two test applications available on PlayStore
• Public test applications on the Web
• Extensive documentation
• Published on GitHub: https://github.com/cyberphone

Currently Missing

• SKS/TEE integration
• Browser integration
• and most of all, device vendor partners…
Related Standardization Efforts

JCS – JSON Clear-text Signature. Fully implemented reference implementation in Java. JCS also runs in browsers and Node.js

```json
{
  "myProperty": "Some data",
  "signature": {
    "algorithm": "ES256",
    "publicKey": {
      "type": "EC",
      "curve": "P-256",
      "x": "vlYxD4dtFJOp1_8 QUcieWCW-4KrLMmFL2rpkY1bQDs",
      "y": "fxEF70yJenP3PHM9hv-EnvhG6nXr3_S-fDqoj-F6yM"
    },
    "value": "gNfr9Es0cnc263tmOYMscBh ... Qd2h8QSePPGsKdkLILVJDBlAbkQ1eA"
  }
}
```