On-board Credentials

N. Asokan
Kari Kostiainen

Joint work with Jan-Erik Ekberg, Pekka Laitinen, Aarne Rantala (VTT)
Outline

• On-board Credentials (ObCs): What and Why
• ObC Architecture
• Secure Provisioning of ObCs
• Instantiations of the Architecture
• Deployment Considerations
• ObCs in Action
• Status
On-board Credentials: What and Why
On-board Credentials (ObCs)

An credential platform that leverages on-board trusted execution environments

Secure yet inexpensive
ObCs: what and why

On-board Credentials

**SW-only credentials**
- Easy, cheap, flexible
- Insecure

**Dedicated HW credentials**
- Secure, intuitive
- Expensive, inflexible, single-purpose

*Like multi-application smartcards, but without issuer control.*
ObCs: design goals

- Credential programs can be **executed securely**
- Credential **secrets** can be stored securely
- Anyone can create and use new credential types
  - Need a security model to strongly isolate credential programs from one another
- Anyone can provision credential **secrets** securely to a credential program
  - Need a mechanism to create a secure channel to the credential program
- Protection of asymmetric credentials is **attestable to anyone**
  - Anyone can verify that a private key is protected by the TEE

Credential = program + secret
ObC Architecture

On Trusted Execution Environments (TEEs) with

- Secure execution (within TEE)
- Secure storage (secret key OPK in TEE)
- Certified device keypair (PK_{dev}/Sk_{dev} in TEE)
- Source of randomness

```
function main()
    read_array(IO_PLAIN_RW, 0, data)
    read_array(IO_SEALED_RW, 1, key)
    aesenc(cipher, data, key)
    write_array(IO_PLAIN_RW, 0, cipher)
    return 0
end
```

More in ACM ASIACCS ‘09 paper

Credential = program + secret
Isolation of ObC Programs

Isolating the platform from programs
- Constraining the program counter, duration of execution, …

Isolating programs from one another
- Only one ObC program can execute at a time
- An ObC program can “seal” data for itself
  - Sealing key is different for every independent ObC program
    Sealing-key = KDF (OPK, program-hash)
  - A program can invoke functions like “seal(data)” (unsealing happens automatically on program loading)

Programming language with single type
- No need for complicated type-safety verification
Secure Provisioning of ObCs
Requirements for Provisioning Credential Secrets

• Provisioning protocols typically focus on **user authentication** only
  • CT-KIP, Open Mobile Alliance Device Management (OMA DM), …
• Dynamic Symmetric Key Provisioning Protocol (DSKPP) (IETF RFC 6063)
  • Allows **device authentication** as well
• We need more…
  • provision a key so that it can be accessed by **specific credential programs**
• Subject to…
  • “Anyone can provision credential secrets securely to a credential program”
  • Support for multiple versions of credential programs
  • Support for several co-operating credential programs
Provisioning credential secrets (1/4)

Idea: a **family** of credential secrets + credential programs endorsed to use them
“family” = dynamic trust domain; **same-origin** authorization policy
Provisioning credential secrets (2/4)

• Provision a family **root key** to the device
  • using **authentic device public key** $PK_{\text{Dev}}$

• Transfer encrypted credential secrets
  • using authenticated encryption (AES-EAX) with RK

• Endorse credential programs for family membership
  • Program ID is a cryptographic hash of program text
  • using authenticated encryption (AES-EAX) with RK
Provisioning credential secrets (3/4)

- Anyone can define a family by provisioning a root key ("Same Origin" policy)
- Multiple credential secrets and programs can be added to a family
- Credential Programs can be encrypted as well
Asymmetric ObCs

Provisioning

Server E

PK, SKAE, Cert₇₇

Cert = Sigₜₑ(PK, ...)

Client Application

CreateKeyPair(credID)

credID

GetPK (credID)

PK

GetKeyPairAttestation (credID)

SKAE for PK

importCert(credID, Cert)

SignMessage(credID, msg, ..)

Sig

Credential Manager

Cert₇₇ (Device certificate) Certificate for PK₇₇ issued by manufacturer

SKAE (Subject Key Attestation Evidence) for PK: Signature on PK issued by SK₇₇, attesting that SK is within the TEE

"Key Attestation from Trusted Execution Environments", Kostiainen et al, TRUST 2010
ObCs: design goals revisited

• Credential programs can be executed securely
  • Use a trusted execution environment (TEE)

• Credential secrets can be stored securely
  • Use a device-specific secret in TEE for secure storage

• Anyone can create and use new credential types
  • Need a security model to strongly isolate credential programs from one another
  • Avoid the need for centralized certification of credential programs

• Anyone can provision credential secrets securely to a credential program
  • Need a mechanism to create a secure channel to the credential program
  • (certified) device keypair; unique identification for credential programs

• Protection of asymmetric credentials is attestable to anyone
  • Anyone can verify that a private key is protected by the TEE
  • Subject key attestation evidence

Credential = program + secret
Instantiations of the Architecture
M-Shield™: Example hardware TEE #1

M-Shield provides

- Secure boot
- Chip-specific secret key (e-fuse)
- Secure execution of certified “Protected Applications” (PAs)
- On-chip RAM for PAs
- … (hardware RNG, crypto accelerators, …)

ObC on Symbian/M-Shield secure h/w (2007-2009)

- M-Shield secure boot used for validation of OS
- Interpreter, Provisioning subsystem are PAs
  - Use on-chip RAM
- OPK from chip-specific secret
- Device key pair
  - generated by Prov. PA
  - protected by chip-specific secret key
  - [certified by manufacturer]
TPM: Example hardware TEE #2

TPM provides

- Authenticated boot
  - Components during boot measured and recorded in Registers (PCRs) within TPM
  - A set of PCR values = a “configuration”
- Secure storage for keys bound to a specific configuration
- Ability to seal arbitrary data bound to a specific configuration
- Secure execution of selected cryptographic operations
- … (remote attestation, …)
ObC using Linux/TPM (2006, 2009)

- Interpreter in kernel module on InitRD
- KeyInitializer in InitRD creates OPK on first use and seals for current configuration
- KeyInitializer unseals OPK on subsequent invocations.

MSc thesis work:
ObC on Maemo/TrustZone secure h/w (2009-2010)

- Linux user space
  - Client Application
  - Client Application
  - ...
  - Qt API (libDeviceEngine.so, libKeyPairEngine.so)
  - Low level C API (libacclib.so)
- Credentials Manager API
- Process boundary
- ObC Daemon (obcsrv)

- Linux kernel space
  - BB5 Security Driver
- TrustZone TEE
  - NOPPA PA
  - ObC program
  - ObC program
  - ...
  - Interpreter PA
  - Crypto Library
  - SK\textsubscript{dev}
  - OPK

Device specific ObC Database

App specific ObC Database

Maemo Lock with OnBoard Credentials

- Close app
- Open door
- Configure

Door select

© 2007-2012 Nokia, ObC-overview-public-for-researchers-jul2012 NA, JEE, KKo
ObC for other platforms

- ObC for MeeGo Harmattan (N9) available in partially emulated mode (see later)
- Other ports yet to be announced publicly
Deployment considerations
1. ObC: Full use of secure hardware

- ObC secret and algorithm (ObC program) protected by hw TEE
  - $PK_{Dev}$ to protect provisioning or attestation
  - Secrets not accessible to OS
  - Cannot be copied between devices
  - Hardware attack typically destructive and device-specific

- Encrypted secret stored in Credentials Manager database
  - Can be backed up

- Example: Symbian devices (N8 and newer, OS version Anna and later)
2. ObC: Partial use of secure hardware

- ObC PAs emulated in the Credential Manager (OS process)
- Secure HW used to enable secure storage and device authentication
- ObC program runtime execution protected by OS platform security

Example: MeeGo Harmattan (N9)
3. ObC: Fully emulated

- ObC PAs emulated in the Credential Manager (OS process)
- Secure HW may be used for secure boot
- Storage ObC secrets and ObC program runtime execution protected by OS platform security
- No device authentication
- For debugging/development
ObC implementation supports all 3 variants

- Implementation contains code for emulating TEE PAs (interpreter+provisioning+crypto)
- Same software package can be installed in any device of the same type
  - automatically decides the variant to use
- (“PA” = “Protected Application” refers to code that runs in hardware TEE)
ObCs in action
Benefits of ObC

- Systematic means to expose useful TEE features (e.g., device authentication) to applications
- Portable programming platform over different chipset technologies for TEE code
- Means for 3rd-party development of credentials for TEE-equipped platforms
ObC Features

Custom Credentials
- Secure key/code provisioning

Built-in Credentials
- Key attestation or Secure key Provisioning

Device Certification
- Validate device platform

Device Authentication

Application Authentication

Content attestation

Secure user credentials

Platform authentication
Target usage scenarios: Platform Authentication

Prove to a third party (e.g., external server)

- **Device authentication**: identity of device
  - E.g., CAPTCHA-avoidance, Comes-with-XYZ

- **Application authentication**: identity of application/process
  - E.g., Extended Web Service APIs for trusted apps

- **Content attestation**: type of content
  - E.g., Enforcing driver distraction rules in MirrorLink
Remote attestation problem

Attesting device

Verifier

What kind of software you are running?

Here is a certified statement of my current configuration (~ “measurements”)

Access control decision

Example: MirrorLink system

Attesting properties, rather than configuration, is more useful
Traditional property-based attestation

Attestation protocol

"Practical property-based attestation", Kostiainen et al, TRUST 2011

App1  P1, P2
App2  P3
...

TEE
Attestation service
Attested application
Verifier

Pick random nonce n

Check application identifier
Verify property p

Pick property p to attest

Attest(n, p || Hash(PKA))

sig ← Sign(SKD, n || p || Hash(PKA))

Attest(n, p, PKA)

sig, CertD

p, sig, CertD, PKA

Verify CertD and sig
Check property p
Save PKA

appSig ← Sign(SKA, appData)

appData, appSig

Verify appSig
Target usage scenarios: User Credentials

- Problem: provide the means to securely provision and store user credentials to user’s personal device
- User benefits:
  - “no need to a bunch of different security tokens”;
  - “digital credentials provisioned easily” (http, e-mail, …)
- Transport ticketing
- “Soft” tokens: embedded SIM, embedded SecurID
- Phone-as-smartcard: use device-resident credentials from legacy PC apps (e.g., browsers, Outlook, VPN clients)
- Physical access control (opening doors)
- …
An Example ObC: SecurID one-time password authentication

Joint research project with RSA security
Phone as smartcard (PASC)

- Applications use public key (PK) cryptography via standard frameworks
  - Crypto API (Windows), Cryptoki (Linux, Mac), Unified Key/cert store (Symbian)
  - Agnostic to specific security tokens or how to communicate with them

→ Any PK-enabled smartcard can be used seamlessly with PK-aware applications!

What if mobile phone can present itself as a PK-enabled smart card?

"Can hand-held computers still be better smartcards?", Tamrakar et al, INTRUST 2010
ObC Status
ObC Status (1/2)

• Available on off-the-shelf Symbian devices
• Development environment for ObC programs (Windows, Linux)
  • Credential Manager and interfaces (native, javascript)
  • Available from Nokia under limited license agreement for research and testing
• Available as an installable software package for MeeGo (N9)
  • distributed as part of the same LLA
• Other platforms in the works
ObC Status (2/2)

• Related research
  • Support for piece-wise execution, sub-routines etc. (Ekberg et al, STC 2009 paper)
    • How to split up ObC programs into smaller pieces securely?
  • Considerations of implementing crypto primitives (Ekberg et al, TRUST 2012 paper)
    • Is authenticated encryption secure even in pipelined mode?
  • Credential Migration, backup/restore (Kostiainen et al, ACNS 2011 paper)
    • Balancing usability/security?

• Useful for several applications
  • Device authentication, financial services, secure messaging, …
  • Pragmatic means to solve otherwise hard privacy/security problems in distributed computing (e.g., secure multi-party computation)
Emerging standardization

- **Global Platform Device Specifications** define standard APIs for TEE applications
- Trusted applications and their data can be provisioned remotely
  - “credential provisioning”
- Modeled after smartcard application provisioning
  - Centralized provisioning
    - TEE supports a hierarchy of protection domains
    - Provisioned TAs must be authenticated using a cert chain
  - No “open provisioning”

Figure 3-1: TEE System Architecture

Figure taken from GlobalPlatform Device Technology TEE System Architecture Version 1.0, December 2011
Limitations

- Open provisioning model
  - Liability and risk management
  - User interaction issues: e.g., Credential migration

- Certification and tamper resistance
  - Not comparable to high-end smart cards

- Will open-provisioning emerge as an alternative to centralized provisioning?
Standing on the shoulders of giants

MTM
Late launch
2010
Mobile OS security architectures

Trusted Platform Module (TPM)
JavaCard platform
Java security architecture
1990

ARM TrustZone
TI M-Shield
Mobile hardware security architectures

Hardware-assisted secure boot

1980
Simple smart cards

On-board Credentials

1970
Protection rings

Cambridge CAP
VAX/VMS

Reference monitor

PC security
Mobile security
Smart card systems
Summary

- On-board Credentials platform
  - inexpensive
  - open
  - secure

- Open provisioning systems can be a viable alternative to traditional closed systems

- **Available for you to build on**
  - http://obc.nokiaresarch.com

- A step towards the vision of a personal trusted device

---

2. Forthcoming Dr. Tech dissertation, Jan-Erik Ekberg, Aalto University
How to make it possible to build trustworthy information protection mechanisms that are simultaneously easy-to-use and inexpensive to deploy while still guaranteeing sufficient protection?