Spotlight on TCG Technologies in Use
Strong Device Identity in the IoT

Yasuyuki Inui, GMO GlobalSign
Overview of GlobalSign

GlobalSign uses its managed PKI & IAM experience and technology to build high-volume, high-scale identity management solutions for the billions of devices, people and things comprising the Internet of Everything.

- The world’s largest pure-play CA
- More than 350 employees distributed in over 15 countries
- More than 5000 Global partners and over 30000 global customers
- More than 10 million identities issued
- 3 Billion OCSP responses delivered daily
lot of risks

Cloud

Impersonation

Gateway

Compromise

Disclosure

Control
Moving towards Higher Assurance Security

- No Identity authentication for Endpoint
- Gateway authentication for Endpoint using ID/Password
- 2-way authentication using mutual TLS
- Strong authentication using Software TPM for key storage
- Strong authentication using Hardware TPM for key storage
PKI / TPM

PKI is a tried and true standard that has been securing machines and devices for decades

PKI

InfoSec “Swiss army knife” - enables a range of information security principles essential to any IIoT product or ecosystem

Authentication – Encryption – Data Integrity

TPMs

Ideal for increasing assurance around the integrity and uniqueness of device identity

Security focused crypto-processors, like TPMs, provide strong hardware-based protection of the device’s private keys from compromise and unauthorized export
Demo

Strong Device ID Enrollment in Manufacturing Environment

TPM
- Endorsement Key
- Endorsement Certificate

Diagnostic Software
- Additional libraries/code to interface with TPM and Provisioning PC
- Diagnostic software, which will facilitate device side activities

Unprovisioned endpoints
- Manufacturing flow
- Generated on TPM, initiated by diagnostic firmware

After provisioning, device has initial software build, it's own identity source from a trusted entity, as well a trust root which tells it what software or what entities to trust

Manufacturing ERP
- Provisioning PC / Test Harness
- Networking Switch with 3G Failover

Local Network
- Public Network
- GlobalSign High Volume Certificate Services
Windows 10

Device Health Attestation (DHA)

Gabe Stocco (gastoco@microsoft.com)
Program Manager
Microsoft OSG, Enterprise and Security R&D
Windows 10 Device Health Attestation (DHA)

- Collects and Provides attested boot state information
- Establishes Attested Link with MDM
- Validates System is in same boot
- Provides a mechanism for remote conditional access based on Device Health
Windows 10 Device Health Attestation (DHA)

- Enroll for AIK Certificate
  - Optional with On-Premises Health Attestation
- Enroll with MDM
- Provide Health Quote to HAS
- Validate Health with MDM
- Request Access to Resources
Obtain AIK Certificate

1- Fuse EK Seed
2- Generate EK Key Pairs (EK_PRIV, EK_PUB) and AIK key Pairs
3- Send EK_PUB to signing server
4- Sign the EK_PUB, issue an EK_CERT
5- Store the EK_CERT on the device
6- Ship the device
7- User purchases the device, turns the device on
8- Device sends the EK_CERT and EK_PUB to AIK provisioning service
9- AIK Provisioning service issues a challenge:
   - Verifies the EK_CERT
   - Issues a challenge:
   - Generates a random value
   - Encrypts it with EK_PUB
   - Sends the encrypted challenge to the device
10- Device decrypts the challenge with EK_PRIV, forward the following to the AIK provisioning service
    - Challenge data in clear format
    - Hash of AIK_PUB to
11- AIK provision service, gets the data:
    - validates if the challenge data are correct
    - Issues a
High Level HAS Flow

- Health Attestation Service
- Mobile Device Management

1. Access please
2. Prove to me you are healthy
3. Request
4. Approved
5. Here is my proof

- Device Health Status (Secure Boot, Device Guard, ...)
- Client Compliance Policies (AV, Firewall, Patch state, ...)

Corporate Managed Assets:
- File Servers
- OneDrive
- Email
- VPN
1. **TPM**
   - Boot Loader
   - Kernel
   - Early Launch Anti-Malware
   - Early Drivers

2. **Device Health CSP**
   - TPM Boot Log
   - PCR

Windows 10 Device
(phone, tablet, laptop, PC, ...)

---

- **Step 1**: Device Measures Boot Components in the TPM
- **Step 2**: DHA-CSP Forwards Measurements to HAS, Gets an Encrypted Report
- **Step 3**: Device Management Solution Gets and Verifies Device Health Report

---

**Microsoft Device Health Attestation Service (DHA-Service)**

**Device Management Solution (MDM)**

---

1. **BIOS / UEFI**
2. **Boot Loader**
3. **Kernel**
4. **Early Launch Anti-Malware**
5. **Early Drivers**

---

2.1. **SSL** (DH Data := TCG Log, Quote (PCR, Counter), cert)
2.2. **SSL** (DH_Boot_Report := Signed (Encrypted (Analyzed DH Data)))
3.1. **SSL** (Verify = DH_QuoteCurrent, State, Nonce, Cert)
3.2. **SSL** (DHA_Verification_Claims := DHA_Boot_Report, Quote (Current_State, Nonce, Cert))
3.3. **SSL** (Verify = DHA_QuoteCurrent, State, Nonce, Cert)
3.4. **SSL** (Device Health Report)
Windows 10 Mobile Device Management (MDM) options

- Windows 10 TPM enabled devices
- Device Health Attestation Service (HAS) options
- Device Management Solution (MDM) options

- **Microsoft Cloud**: ready now
- **On-Prem (2016 Server)**:
  - 1st and 3rd party On-Prem and **Cloud** MDM solutions
Windows 10 Device Health Attestation (DHA)

Sample data points that is evaluated/reported by HAS

- BitlockerStatus
- SecureBootEnabled
- CodeIntegrityEnabled
- ELAMDriverLoaded
- VSMEnabled
- CIPolicyHash
- SBCPPolicyHash
- DEPPolicy State
- SafeMode
- WinPE
- BootDebuggingEnabled
- OSKernelDebuggingEnabled
- TestSigningEnabled
- AIKCertPresent
- Value of PCR 0
- Reset Count (Hibernation)
- Restart Count (Boot/reboot)
- And more ....
Compliance monitoring example: SCCM
Conditional Access – O365

Win 10 Device

Other Device Configuration Service Providers (CSP's)

TPM

Device Health CSP

MDM Client

(A) Get Device Health Certificate
(B) Validate Device Health
(C) Query Device Config - State
(D) Set “IsCompliant” Device Attribute
(E) Request Office 365 Access Token
(F) Access Office 365 Protected Resources

Office Apps

AAD

Office 365 Resource

Bios UEFI
Boot Loader
Kernel
Early Launch Antimalware
Early Drivers

Forward Health Data & Nonce

(D) Set “IsCompliant” Device Attribute

(E) Request Office 365 Access Token

(E4) Validate Device Compliance State

(E5) Forward Token

(E6) Present Token

(A) Get Device Health Certificate

(B) Validate Device Health

(C) Query Device Config - State

(D) Set “IsCompliant” Device Attribute

(E) Request Office 365 Access Token

(F) Access Office 365 Protected Resources

(AAD, TB Plugin/ADAL)

Request Access Token

Access Office 365 Protected Resources

Issue Office 365 Access Token

Present Token

(AAD, TB Plugin/ADAL)

Request Access Token

Access Office 365 Protected Resources
Conditional Access – VPN

Win 10 Device

(A) Get Device Health Certificate
(B) Validate Device Health
(C) Query Device Config - State
(D) Set “IsCompliant” Device Attribute
(E) Request VPN Certificate
(F) Client connects to VPN Server
(G) Access Internal Network Resources
Thank You!
© 2016 Microsoft Corporation. All rights reserved. Microsoft, Windows, Windows Vista and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries. The information herein is for informational purposes only and represents the current view of Microsoft Corporation as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation. MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.
Additional Resources

Device Identifier Composition Engine Prototype
Stefan Thom
Rob Spiger
Device Identifier Composition Engine (DICE)

- New specification development in the TCG Root of Trust for Measurement sub-workgroup (under the Embedded Systems WG)
- Use: Provide a way to know the first mutable code running on a platform
  - Helpful when patches need to be deployed and their installation verified
  - Can help re-establish trust in a platform after a vulnerability has been discovered
Device Identifier Composition Engine (DICE)

- New specification development in the TCG Root of Trust for Measurement sub-workgroup (under the Embedded Systems WG)
- Use: Provide a way to know the first mutable code running on a platform
  - Helpful when patches need to be deployed and their installation verified
  - Can help re-establish trust in a platform after a vulnerability has been discovered

- The Compound Device Identifier is unique to the device and reflects the first mutable code that runs on the platform
Generating the Compound Device Identifier

- DICE computes the Compound Device Identifier:
  
  \[ \text{Compound Device Identifier} = \text{Hash}(\text{Mutable Code Measurement} || \text{Device Secret}) \]

- Immutable code passes the Compound Device Identifier to the first mutable code
- Immutable code has access to the Unique Device Secret, but the mutable code does not
The 3 Dice Platform Requirements

- Immutable device initialization code, ideally ROM or one-time programmable
- Unique device identity of at least 128 bit, ideally one-time programmable
- Device Identity lockout mechanism that resets only with the entire platform
What the STM32L082KZ offers for DICE

- Ultra low cost and low power Cortex-M0+ @32MHz
- 192kB Flash storage, that can be permanently write locked in parts
- 6kB EEPROM for configuration/data storage
- 20kB RAM
- AES-128 hardware engine with DMA support
- True hardware RNG
- Chrystal-less USB 2.0 hardware support
- Memory Firewall support
- Official price ~$1.89 @10K
How is DICE implemented here?

• The ST Firewall functionality is significant more powerful than we need for DICE and is wasting most of this powerful functionality, that actually should be available for the application itself.

• The hardware AES-128 engine is used to calculate CMACs as 128bit *digests*, however a SHA2 hardware engine with 256bit digests would be preferable. The following algorithm substitutions are applied, where the resulting encrypted data is discarded, with the obvious restrictions on operator sizes for the AES algorithm:

  \[
  \text{Digest} = \text{KeyedHash(} \text{KeyIn, DataToHash}) \Rightarrow \\
  \text{Digest} = \text{IV}_{\text{out}} \text{ after AES128-CBC}_{\text{encrypt}}(\text{IV}_{\text{in}}=0, \text{Key} = \text{KeyIn, Data} = \text{DataToHash})
  \]

  \[
  \text{Digest} = \text{HASH(DataToHash)} \Rightarrow \\
  \text{Digest} = \text{IV}_{\text{out}} \text{ after AES128-CBC}_{\text{encrypt}}(\text{IV}_{\text{in}}=0, \text{Key} = \text{DataToHash, Data}=0)
  \]

• The device can be fully locked down by permanently disabling hardware debugging and blowing the read-only fuses for the flash memory hosting the DICE code and Device Identity – making the DICE code immutable.
What is this demo going to show

• The simple DICE *functional concept* that the immutable boot code is the sole consumer of a persistently stored identity value. This demo does not necessarily use the most simple implementation to do that

• The device is using the USB Device Firmware Update profile (DFU) to permit a standardized one-time Identity provisioning and a many-time application payload upgrade process

• A glimpse how the device and compound identities of the factory and software release process can be secured with a TPM 2.0 (on a server in the manufacturer’s environment) and therefore enforce key distribution policies

• While DICE only knows one Device Identity that is extended with the Code Identity to derive the compound identity, this demo explores some additional identities which are derived in different ways to anticipate customers desired scenarios and provide more flexibility
The Main Scenario: The Device Identity

1. Every device is provisioned with a unique DeviceRootIdentity
2. This DeviceRootIdentity is derived into the public DeviceRootName for external identification purposes:
   \[ \text{DeviceRootName} = \text{HASH(DeviceRootIdentity)} \]
3. With the ApplicationPayload the DeviceCompoundIdentity is derived in the form:
   \[ \text{DeviceCompoundIdentity} = \text{KeyedHash(DeviceRootIdentity, ApplicationPayload)} \]
4. And finally the public DeviceCompoundName again for identification purposes as:
   \[ \text{DeviceCompoundName} = \text{HASH(DeviceCompoundIdentity)} \]
Expanded Identity Derivation Overview

- **DICE**
- **CodeBoundManufacturerID**
- **DeviceID**
- **ManufacturerID**
- **DICE**
- **Application Payload**
- **Header**
- **OperatorID**
- **Salt**
- **CodeDigest**
- **Decrypt**
- **Validate**
- **Trust Boundary**
- **Running Application Payload**
- **OperatorID**
- **ManufacturerCompoundID**
- **DeviceCompoundID**

© 2016 Trusted Computing Group
Do Not Redistribute without Permission

© 2016 Microsoft Corporation
Expanded Scenario 1: The Manufacturer Identity

1. At DeviceIdentity Provisioning time the manufacturer may also provision a common ManufacturerRootIdentity that is shared between all devices of a class. This root identity is treated in the same way as the unique DeviceRootIdentity and inaccessible when the application is running.

2. In that case, the ApplicationPayload header has to be signed with that identity for the device to accept it – implementing a form of authenticated boot.

3. The ApplicationPayload header contains a public salt value that is used to derive a specific shared identity between all sibling devices:

   \[
   \text{ManufacturerCompoundIdentity} = \text{KeyedHash(ManufacturerRootIdentity, HeaderSalt)}
   \]

   \[
   \text{ManufacturerCompoundName} = \text{HASH(ManufacturerCompoundIdentity)}
   \]

4. It is in the discretion of the Manufacturer to roll that salt value, because it may remain the same for several firmware releases until he has identified a security relevant issue that needs to break the forward migration of this identity.
Expanded Scenario 2: The Operator Identity

1. A device manufacturer may provide his device to multiple operators that need to run isolated from each other, while they share the same ManufacturerIdentity derived subsequent identities from that family, the operator wants to make sure that devices deployed in his realm are not compromised by identical devices running with different operators.

2. For that purpose the operator has his own identity that has to be provided securely to the ApplicationPayload without recompiling the payload itself or needing to specify that identity at the initial provisioning time.

3. The manufacturer will derive a manufacturer specific code identity that is the same on all devices with the same ApplicationPayload and then encrypt the operator identity and insert that value into the signed application payload header:

   \[
   \text{ManufacturerCompoundCodeIdentity} = \text{KeyedHash}(\text{ManufacturerRootIdentity}, \text{ApplicationPayload})
   \]

   \[
   \text{EncryptedOperatorIdentity} = \text{AES128CBC}_{\text{Encrypt}}(\text{IV}=0, \text{ManufacturerCompoundCodeIdentity}, \text{OperatorIdentity})
   \]

4. Since the encrypted value is covered by the application payload header signature it may not be arbitrarily changed after signing and the manufacturer can calculate a specific encrypted Operator identity for all operators he works with and issuing a customized firmware to them for deployment. This mechanism even permits the manufacturer to prevent an operator to consume a particular firmware version, because if without the explicit code bound encryption that device would not have that identity available.
Now to the actual Demo...

1. The Factory programs the DICE bootloader and provisions the DeviceID and the ManufacturerID
2. The Application Developer creates the application payload and provides to the manufacturer
3. The Manufacturer encrypts the OperatorID and the completes the application payload header and signs it
4. The Operator (or his customers) receive the device and program the firmware package
5. For application payload update process continue at 2.
The Factory programs the DICE bootloader

Keil uVision

Compile

L0Dice.hex

Load

Write

STM32L082KZ

© 2016 Microsoft Corporation
© 2016 Trusted Computing Group
Do Not Redistribute without Permission
The Factory creates an identity package

1. DicePrep.exe CDP [DevId] [ManId]
2. Creates a update package DiceIdPkt.dfu
3. Both identities are also imported on the local TPM 2.0 (*.TPMKEY) with a predetermined set of usage and migration policies that prevent these key from leaking
Provision the device identities

1. The device is booted and since it is uninitialized it will enter automatically the USB DFU mode
2. Using the STMicro DfuSe tool to install the update DiceIdPkt.dfu package on the MCU
3. Shutdown the device
4. The device can now be transferred to the manufacturer and the TPM protected keys can be migrated to his authority
The Application provider builds the payload

1. The application payload is relocated behind the DICE code and data
2. The payload.hex file is transferred to the manufacturer
The Manufacturer Integrates the Payload

1. DicePrep.exe CAP [hex file] [ManID] [Salt] [OprID]
2. Open the specified manufacturerID in the TPM 2.0
3. Encrypt the provided OperatorID and insert in header
4. Calculate code digest and insert in header
5. Copy salt into header
6. Sign header with manufacturerID
7. Prepend the header to the code
8. Generate the DiceFwPkg.dfu package
Program the Application Payload

1. The device is booted and since the application payload is uninitialized it will still enter automatically the UDB DFU mode
2. Using the STMicro DfuSe tool to install the update DiceFwPkt.dfu package
3. Reboot the device
4. The device is now operational
Pre-Calculating the CompoundDeviceID

1. DicePrep.exe DTI [rootName] [DerivationID]
2. Use the TPM held DeviceRootIdentity to derive the expected DeviceCompoundIdentity for the application payload on the targeted device
3. Import the new compound device identity into the TPM and store as *.TPMKEY file
4. Calculate the expected DeviceCompoundName
Booting the application payload: Debug

When the device boots, the DICE prototype dumps diagnostic information to the serial console:
This first portion shows some generic device information that was provisioned by the manufacturers
DiceIdPkg.dfu. It also shows that the device is locked to a manufacturerID, so Secure Boot is
enabled.
Alternatively the device may be locked to a particular application payload rather all payloads of a
given authority, to lock it even further down.
Booting the application payload: Debug

1. The ApplicationInfo shows the application payload signed header data that is stored in the application flash area on the device
2. The VolatileData section shows the root and derived identity names. These match with the provisioned names from the previous steps, so DICE is holding the correct identities
3. The header signature is validated to enforce Secure Boot
4. The DICE root identity data is locked out and secured
Booting the application payload: Debug Spew

1. DICE bootloader is torn down
2. Execution is transferred to application payload located at 0x08008200 in flash
3. Application payload checks that DICE persisted data is unavailable
4. The application can read its own payload header
5. The application payload can read all the names for attestation purposes, however only the compound identities are still residing in memory and the root identities have been zeroed out and are unavailable
6. The application payload does its thing
What’s next?

• The demo code currently does not blow the flash rewrite protection fuses and hardware debugging is not disabled; so there are no irreversible changes on the MCU. In order to consider the solution to provide actual security, this needs to be added.

• The STM32L476RG is a more powerful MCU with additional hardware capabilities that the prototype code is also supposed to be running on.

• Merging this demo code with the RazorClam to secure the MCU connection to a discrete TPM 2.0.

© 2016 Microsoft Corporation

© 2016 Trusted Computing Group
Do Not Redistribute without Permission
How to get involved

- Join TCG!
- Participate in new the DICE Architectures workgroup
- Help defined use cases and software specifications for using the compound device identity
- Help compare security benefits to existing TCG technologies like the TPM
References

- **RIoT** – A Foundation for Trust in the Internet of Things
- STMicro [AN4729]: STM32L0/L4 FIREWALL overview
- STMicro [STSW-STM32080]: DfuSe USB device firmware upgrade
- STMicro [STM32L082KZ]: Ultra-low-power ARM Cortex-M0+ MCU with 192-Kbytes Flash, 32 MHz CPU, USB, AES
- STMicro [STM32L476RG]: Ultra-low-power with FPU ARM Cortex-M4 MCU 80 MHz with 1 Mbyte Flash, LCD, USB OTG
- **RazorClamMCU**: Securely binding the MCU and the TPM together
Thank You for Joining Us!

Thank you to our speakers and
Mr. Jungsu Song, MSIP

To learn more about trusted computing, please visit
www.trustedcomputinggroup.org
For more information on the benefits of membership, please visit
www.trustedcomputinggroup.org/membership