Concepts of DPS: <https://docs.microsoft.com/en-us/azure/iot-dps/concepts-auto-provisioning>

TPM Attestation: <https://docs.microsoft.com/en-us/azure/iot-dps/concepts-tpm-attestation>

**Popular types of TPM**

* **DISCRETE TPM (TPM 1.2 & TPM 2.0)**  
  *Discrete TPM provides the highest level of security. The intent of this level is to ensure that the device it’s protecting does not get hacked via even sophisticated methods. To accomplish this, a discrete chip is designed, built and evaluated for the highest level of security that can resist tampering with the chip, including probing it and freezing it with all sorts of sophisticated attacks.*
* **INTEGRATED TPM**  
  *Integrated TPM is the next level down in terms of security. This level still has a hardware TPM but it is integrated into a chip that provides functions other than security. The hardware implementation makes it resistant to software bugs, however, this level is not designed to be tamper-resistant.*
* **FIRMWARE TPM (fTPM)**  
  *Firmware TPM is implemented in protected software. The code runs on the main CPU, so a separate chip is not required. While running like any other program, the code is in a protected execution environment called a trusted execution environment (TEE) that is separated from the rest of the programs that are running on the CPU. By doing this, secrets like private keys that might be needed by the TPM but should not be accessed by others can be kept in the TEE creating a more difficult path for hackers. In addition to the lack of tamper resistance, the downside to the TEE or firmware TPM is that now the TPM is dependent on many additional aspects to keep it secure, including the TEE operating system, bugs in the application code running in the TEE, etc.*
* **SOFTWARE TPM**  
  *Software TPM can be implemented as a software emulator of the TPM. However, a software TPM is open to many vulnerabilities, not only tampering but also the bugs in any operating system running it. It does have key applications: it is very good for testing or building a system prototype with a TPM in it. For testing purposes, a software TPM could provide the right solution/approach.*

# 1. X.509 Attestation

## 1.1 Single enrollment

Scenario:

* + 1. Azure Op creates self-signed or order authorized Root CA. Installs this certificate into DPS
    2. Azure Op, deriving from Root CA, generates list of device certificates using script
    3. Azure Op sends this list to the Factory Op
    4. Azure Op, using provisioning tool, creates Enrollment List in DPS from list of device certificates. Enrollment policy and initial Device Twin should be specified at this stage
    5. Factory Op uses entries in provided list to inject provisioning software and device certificate during production of CanPlugs

Device Input: ScopeID, X.509 PK/Pub, Common Name

Factory Input: ScopeID, List of ( X.509 PK/Pub + Common Name)

HSM implementation complexity: low

Provisioning tool complexity:

Provisioning process complexity:

Provisioning flexibility:

Security Vulnerabilities: ?

Pros: ease of implementation

Cons: Need to synchronize enrollment list between factory and operator. Additional effort needed to manage provisioning policy for each device. Additional script to generate device certificate

## 1.2 Group enrollment

Scenario:

1. Azure Op creates self-signed or order authorized Root CA. Installs this certificate into DPS
2. Azure Op, deriving from Root CA, creates Factory Certificate
3. Azure Op sends this single certificate to Factory Op
4. Based on mutual with Azure Op agreement, Factory Op generates chain of certificates. For instance per model, version, groups of canplugs, production month, etc. List of sub-certificates is sent back to Azure Op

4a) As alternative approach, chain of sub-certificates can be created by Azure Op and sent to Factory Op

1. Azure Op creates Enrollment Group Entries in DPS using sub-certificates. Enrollment policy and Device Twin is common for whole Enrollment Group
2. Factory Op generates Leaf certificates as children of previously created sub-certificates
3. Factory Op injects provisioning software and device certificate during production of CanPlugs

Device Input: ScopeID, X.509 PK/Pub, Common Name

Factory Input: ScopeID, Intermediate certificate(s)

HSM complexity: low

Provisioning tool complexity:

Provisioning process complexity:

Provisioning flexibility:

Security Vulnerabilities: ?

Pros: ease of implementation, selectivity in the enrollment policies

Cons: ?

# 2. Symmetric Key Attestation 2.1 Single enrollment

Scenario:

1. Azure Op uses script to generate list of pairs: Symmetric Key and Registration ID
2. Azure Op uses provisioning tool to create Enrollment List in DPS using list created in previous step
3. Azure Op sends list to Factory Op
4. Factory Op uses provided list to inject provisioning software and SymKey+Registration ID into CanPlug during production

Device Input: ScopeID, Symmetric Key, RegistrationID

Factory Input: ScopeID, Enrollment list

HSM complexity: low

Provisioning tool complexity:

Provisioning process complexity:

Provisioning flexibility:

Security Vulnerabilities: risk of single device faking

Pros: ease of implementation

Cons: Need to synchronize enrollment list between factory and operator. Additional effort needed to manage provisioning policy for each device

## 2.2 Group enrollment

Scenario:

1. Azure Op uses script to create Master Key(s)
2. Azure Op sends Master Key(s) to Factory Op and agrees how those keys should be devoted
3. Azure Op prepares Enrollment Groups in DPS using Master Key(s)
4. Factory Op uses script to prepare list of DDKs using Master Key(s)
5. Factory Op uses list of DDKs to inject provisioning software and DDK+Registration ID into CanPlug during production

Device Input: ScopeID, DDK, RegistrationID

Factory Input: ScopeID, Master Symmetric Key(s)

HSM complexity: low

Provisioning tool complexity:

Provisioning process complexity:

Provisioning flexibility:

Security Vulnerabilities: risk of single device faking. Risk of device spawning in case of Master Symmetric Key leakage

Pros: ease of implementation, simple enrollment policy management

Cons: additional script required for factory to create DDKs

HSM: Software + file system / Firmware

Reprovisioning?

Build Azure IoT Hub SDK for ARM:

/mnt/d/FarmersEdge/auto\_prov/azure\_iot\_sdk\_c/arm# cmake -Duse\_http=ON -Duse\_openssl=OFF -Duse\_amqp=OFF -Dno\_logging=OFF -Ddont\_use\_uploadtoblob=ON -DCMAKE\_TOOLCHAIN\_FILE=./toradex.cmake -Dskip\_samples=ON -Dhsm\_type\_symm\_key:BOOL=ON -Duse\_prov\_client:BOOL=ON -DOPENSSL\_ROOT\_DIR=/usr/local/ssl\_arm -DOPENSSL\_LIBRARIES=/usr/local/ssl\_arm/lib -DCURL\_LIBRARIES=./arm\_env/lib ..

Python SDK:

git clone https://github.com/Azure/azure-iot-sdk-python.git --recursive

cd azure-iot-sdk-python/c

mkdir cmake

cd cmake

cmake -Duse\_prov\_client:BOOL=ON ..

Build Azure IoT SDK for Ubuntu:

git clone <https://github.com/Azure/azure-iot-sdk-c> --recursive

cd azure-iot-sdk-c

mkdir cmake-linux

cd cmake-linux

cmake -Duse\_http=ON -Duse\_openssl=ON -Duse\_amqp=OFF -Dno\_logging=OFF -Ddont\_use\_uploadtoblob=ON -Dskip\_samples=ON -Dhsm\_type\_symm\_key:BOOL=ON -Duse\_prov\_client:BOOL=ON ..

make

Acronyms:

Azure Op – Azure Operator

Factory Op – Factory Operator

TMP – Trusted Platform Module

HSM – Hardware Security Module

DPS – Device Provisioning Service

EK – Endorsement Key

SRK – Storage Root Key

DDK – Derived Device Key