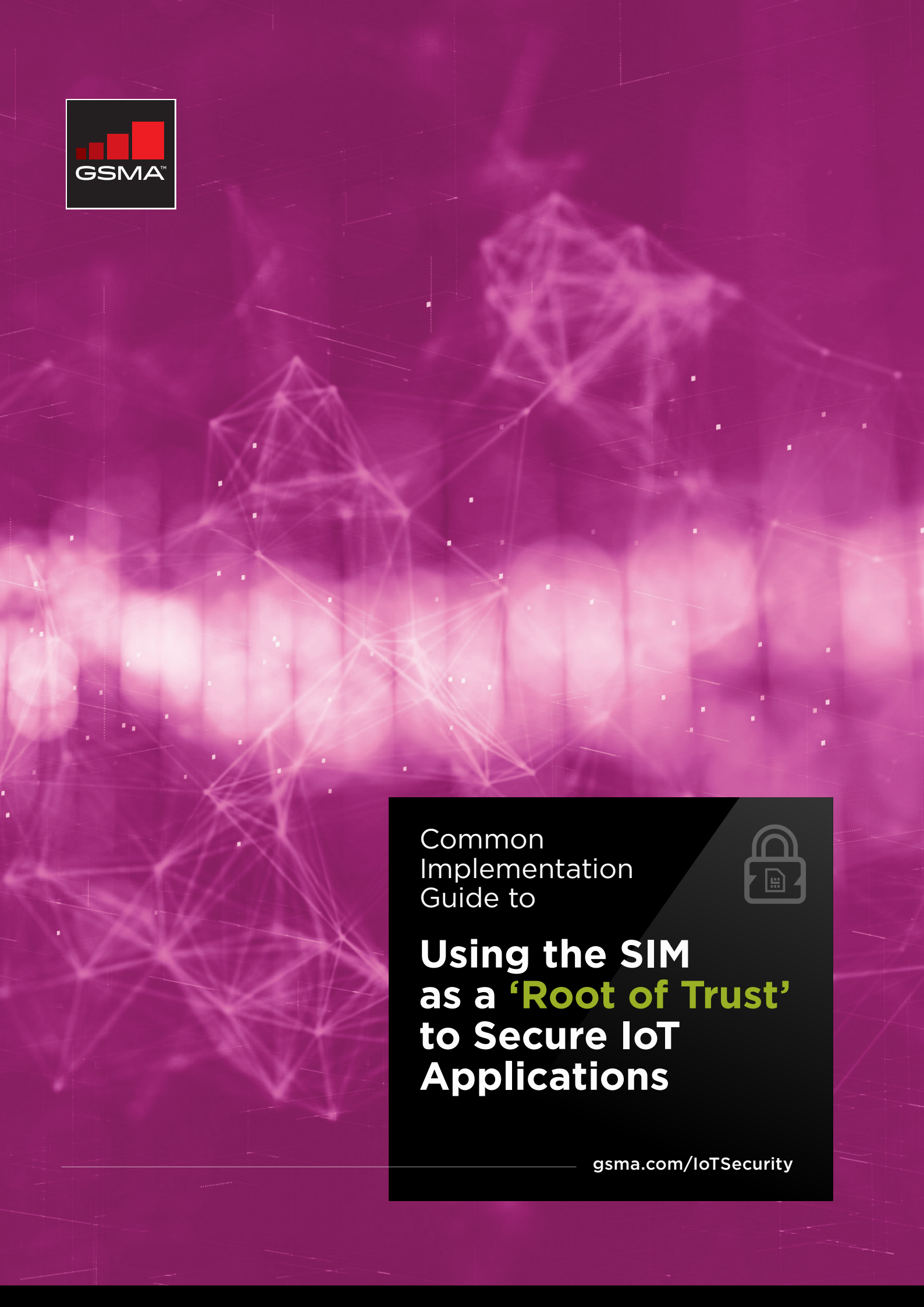

Common Implementation Guide to

Using the SIM as a 'Root of Trust' to Secure IoT Applications

gsma.com/IoTSecurity

Common Implementation Guide to Using the SIM as a 'Root of Trust' to Secure IoT Applications

Version 1.0

03 December 2019

This is a Non-binding Permanent Reference Document of the GSMA

Security Classification: Non-confidential

Access to and distribution of this document is restricted to the persons permitted by the security classification. This document is confidential to the Association and is subject to copyright protection. This document is to be used only for the purposes for which it has been supplied and information contained in it must not be disclosed or in any other way made available, in whole or in part, to persons other than those permitted under the security classification without the prior written approval of the Association.

Disclaimer

The GSM Association ("Association") makes no representation, warranty or undertaking (express or implied) with respect to and does not accept any responsibility for, and hereby disclaims liability for the accuracy or completeness or timeliness of the information contained in this document. The information contained in this document may be subject to change without prior notice.

Antitrust Notice

The information contain herein is in full compliance with the GSM Association's antitrust compliance policy.

Table of Contents

1 Introduction

1.1 Overview

The GSMA's IoT Programme has investigated how to leverage existing mobile network operator assets to secure IoT services. One of the key assets we have considered is the SIM and this has led to the publication of a whitepaper and a case study. The whitepaper "Solutions to Enhance IoT Authentication Using SIM Cards (UICC)" [1] describes the potential capability to use the SIM to secure IoT services and the case study "Leveraging the SIM to Secure IoT Services" [2] describes four mobile network operator proof-of-concepts demonstrations that prove the concepts outlined in the whitepaper are feasible.

This guide defines a common way for IoT applications to use the capabilities of the SIM to enhance the security of several commonly used internet protocols. The internet protocols to be covered as a priority within the document will include Transport Layer Security (TLS), Datagram Transport Layer Security (DTLS) and the 3GPP Generic Bootstrapping Architecture (GBA). The use of Public Key Mechanisms and Pre-Shared Key (PSK) mechanisms for credential establishment are within the scope of work.

More precisely, this guide investigates, and defines, the following:

- A simple microcontroller API that IoT client applications can use to access the secure capabilities of the SIM when using TLS, DTLS and GBA protocols to secure an IP connection to an IoT service platform.
- The properties of a common SIM application where the credentials (e.g. certificates / keys) associated with TLS or DTLS mutual authentication protocols can be stored and used by the IoT application.
- A common way for IoT service providers to initially provision (i.e. one time), and potentially update (i.e. dynamic management) their internet security credentials (e.g. certificates / keys) and policies within the SIM using over-the-air management capabilities.

1.2 Scope

- The guide leverages existing SIM and SIM OTA standards only - it does not look to develop any new SIM / OTA standards or new core technical capabilities.
- This guide is applicable for both UICC and eUICC.
- For simplicity the guide focuses on single eUICC profile scenarios only, and on how to enhance the security of some existing, commonly deployed, internet protocols within low complexity IoT devices.
- The guide focuses on cellular (3GPP) based devices with UICC (i.e. GSM, UMTS, NB-IoT and LTE, LTE-M1, 5G NR) only.
- This guide is compatible with GSMA SGP.02 [3] and/or SGP.22 [4]

1.3 Intended Audience

Technical experts working within mobile network operators, SIM solution providers, IoT device vendors, IoT service providers and IoT developers.

1.4 Definition of Terms

Term	Description
eUICC	A removable or non-removable UICC which enables the remote and/or local management of Profiles in a secure way. NOTE: The term originates from "embedded UICC".
UICC	A secure element platform specified in ETSI TS 102 221.

1.5 Abbreviations

Abbreviation	Description
3GPP	3rd Generation Project Partnership
5G	5th Generation
AEAD	Authenticated Encryption with Associated Data
AKA	Authentication and Key Agreement
ANSSI	Agence Nationale de la Sécurité des Systèmes d'Information
APDU	Application Protocol Data Unit
API	Application Programming Interface
AuC	Authentication Centre
BSF	Bootstrapping Server Function
BSI	Bundesamt für Sicherheit in der Informationstechnik
CA	Certificate Authority
CSR	Certificate Signing Request
DTLS	Datagram Transport Layer Security
ECA	Enrolment Certificate Authority
ECC	Elliptic Curve Cryptography
ECDH	Elliptic Curve Diffie-Hellman
ECDHE	Elliptic Curve Diffie-Hellman Ephemeral
ECDSA	Elliptic Curve Digital Signature Algorithm
EID	eUICC Identifier
ETSI	European Telecommunications Standard Institute
GBA	Generic Bootstrapping Architecture
GBA_ME	Generic Bootstrapping Architecture Mobile Equipment
GBA_U	Generic Bootstrapping Architecture USIM
GSM	Global System for Mobile
GSMA	GSM Association
HKDF	Hash-based Key Derivation Function
HMAC	Hash-based Message Authentication Code
HSM	Hardware Security Module
HSS	Home Subscriber Server
ICCID	Integrated Circuit Card Identifier

Abbreviation	Description
IoT	Internet of Things
IP	Internet Protocol
ISIM	IP Multimedia Subscriber Identify Module
KDF	Key Derivation Function
KMS	Key Management System
LTE	Long Term Evolution
MAC	Message Authentication Code
ME	Mobile Equipment
MSISDN	Mobile Station International Subscriber Directory Number
NAF	Network Application Function
NB-IoT	Narrow Band Internet of Things
NIST	National Institute for Science and Technology
NR	New Radio
OBKG	On Board Key Generation
OBU	On Board Unit
OTA	Over The Air
PKI	Public Key Infrastructure
PRF	Pseudorandom Function
PSA	Platform Security Architecture
PSK	Pre-Shared Key
REST	Representational State Transfer
RNG	Random Number Generator
RSA	Rivest / Shamir / Adleman
SIM	Subscriber Identity Module
TLS	Transport Layer Security
UMTS	Universal Mobile Telecommunications Service
USIM	UMTS Subscriber Identify Module

1.6 References

Ref	Document Number	Title
[1]	-	Solutions to Enhance IoT Authentication Using SIM Cards (UICC). <LINK>
[2]	-	Leveraging the SIM to Secure IoT Services. <LINK>
[3]	GSMA SGP.02	Remote Provisioning Architecture for Embedded UICC Technical Specification <LINK>
[4]	GSMA SGP.22	RSP Technical Specification <LINK>

Ref	Document Number	Title
[5]	RFC 2119	Key words for use in RFCs to Indicate Requirement Levels <LINK>
[6]	-	Java Card Platform Specification <LINK>
[7]	-	GlobalPlatform Technical Overview <LINK>
[8]	ETSI 102 241	Smart Cards; UICC Application Programming Interface (UICC API) for Java Card™ <LINK>
[9]	NIST Special Publication 800-57 Part 1 Revision 4	Recommendation for Key Management Part 1: General <LINK>
[10]	BSI TR-02102-2	BSI "Cryptographic Mechanisms: Recommendations and Key Lengths: Use of Transport Layer Security (TLS)" Version: 2019-1 <LINK>
[11]	ANSSI SDE-NT-35-EN/ANSSI/SDE/NP	ANSSI Security Recommendations for TLS <LINK>
[12]	RFC5246	The Transport Layer Security (TLS) Protocol Version 1.2 <LINK>
[13]	RFC8446	The Transport Layer Security (TLS) Protocol Version 1.3 <LINK>
[14]	RFC5869	HMAC-based Extract-and-Expand Key Derivation Function (HKDF) <LINK>
[15]	GSMA IOT.05	<LINK to be added once the document is published>
[16]	3GPP TS 33.220	Generic Authentication Architecture (GAA); Generic Bootstrapping Architecture (GBA) <LINK>
[17]	3GPP TS 27.007	AT command set for User Equipment <LINK>
[18]	-	u-blox cellular modules AT commands manual <LINK>
[19]	-	Platform Security Architecture Resources <LINK>
[20]	-	PKI Basics - a Technical Perspective <LINK>

1.7 Conventions

The key words "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", and "MAY" in this document SHALL be interpreted as described in RFC 2119 [5].

2 IoT Scenarios that Leverage the SIM for IoT Security

A set of example scenarios is provided below. This is not intended to be an exhaustive list of all possible usage scenarios.

2.1 Scenario 1 (UICC manufactured with asymmetric keys generated at personalisation step)

In this scenario, data from an IoT device is secured using an asymmetric key pair (public/private keys) that is installed onto a UICC at the time of manufacture.

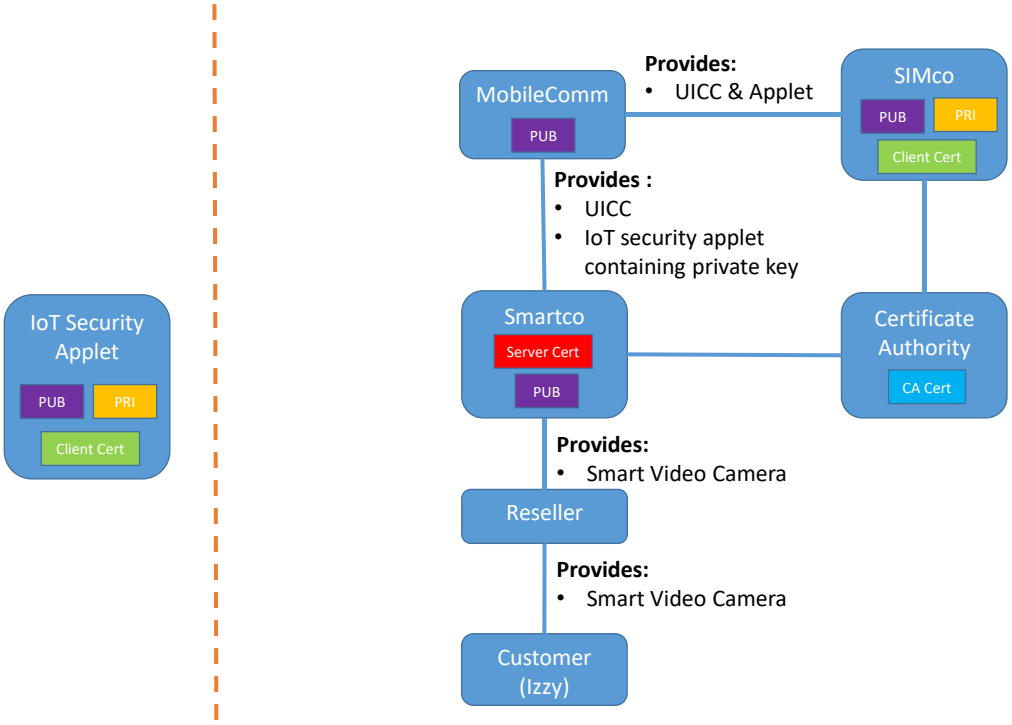

Figure 1: Relationship Between Actors in Scenario 1

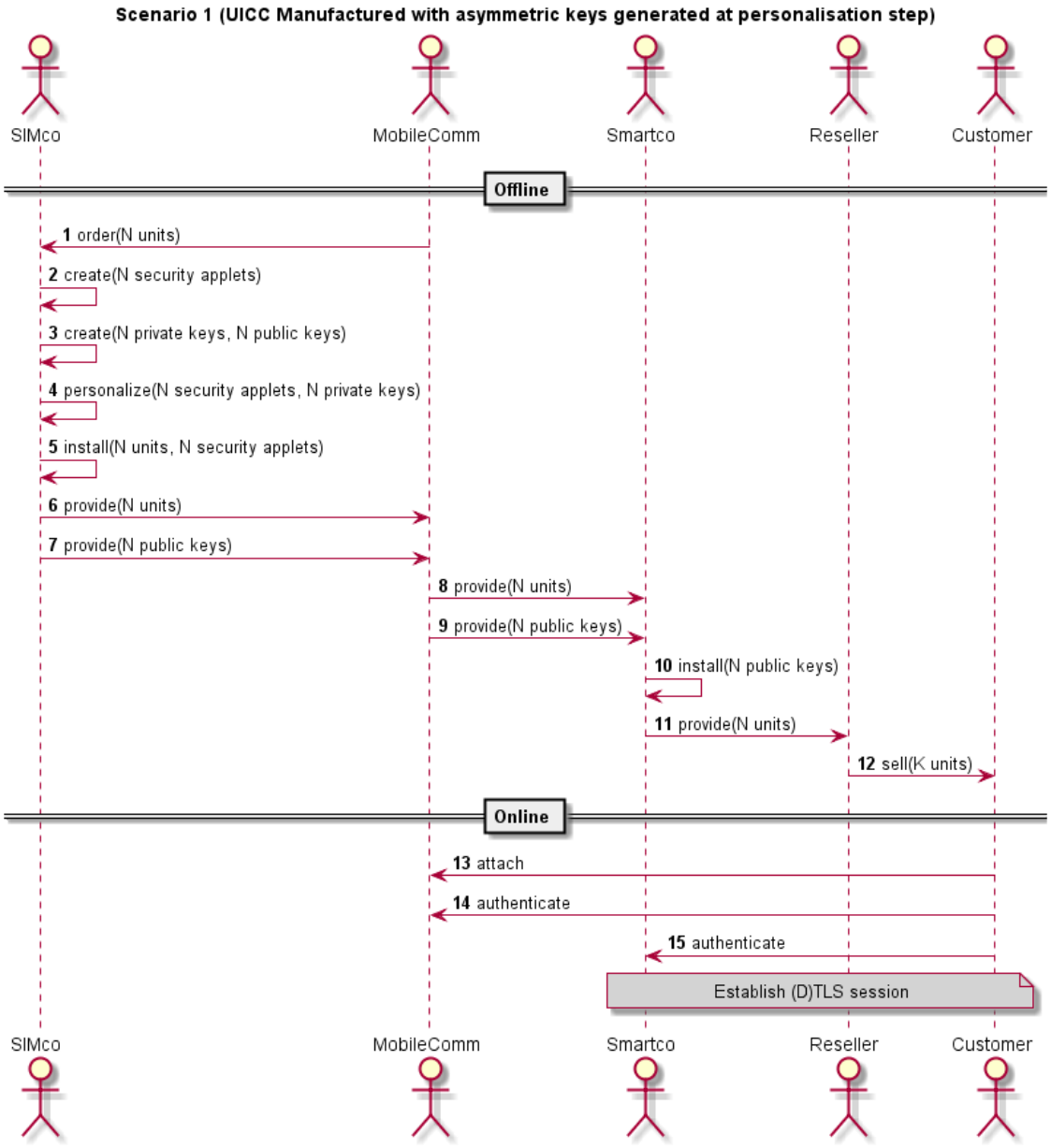

Figure 2: Flow of Events for Scenario 1

'MobileComm", a mobile network operator, procures a batch of UICCs from "SIMco" a UICC vendor. When the UICCs are securely personalised for MobileComm at SIMco's factory, an IoT security applet is installed onto each UICC. One or more randomised private keys are installed within the IoT security applet, depending upon the prospective usage, as part of the personalisation process. The private keys never leave the personalisation area within the factory. The public key (i.e. certificate or raw public key) corresponding to each private key is shared with MobileComm.

MobileComm has a commercial relationship with "Smartco", an IoT service provider selling cellular connected video cameras, for the supply of connectivity and security services. MobileComm supplies UICCs to Smartco. MobileComm also provides Smartco with a copy of the public key that corresponds to the IoT security applet on each UICC.

Smartco installs MobileComm UICCs into its security camera products at its distribution centre. Smartco provisions the public key for each camera/UICC combination onto its service platform. The cameras are then shipped to a reseller.

The reseller sells a camera to a customer called 'Izzy'. Izzy takes the camera home and switches on the camera.

The camera connects to MobileComm's mobile network. The network authenticates the UICC in the camera. The camera establishes an internet connection to Smartco's service platform and initiates a mutual authentication procedure to establish a secure (D)TLS connection with the service platform. The device side (D)TLS mutual authentication steps are performed using the key previously provisioned within the IoT security applet. At the end of the mutual authentication procedure, secure IP communication can take place between the camera and cloud service platform.

The camera is now connected to the service platform and Izzy can use the camera.

2.2 Scenario 2 (eUICC with on-board asymmetric key generation, signing certificate)

In this scenario, data from an IoT device is secured using on-board key generation of asymmetric (public/private) keys and a certificate is installed onto the eUICC "over-the-air" by the mobile network operator.

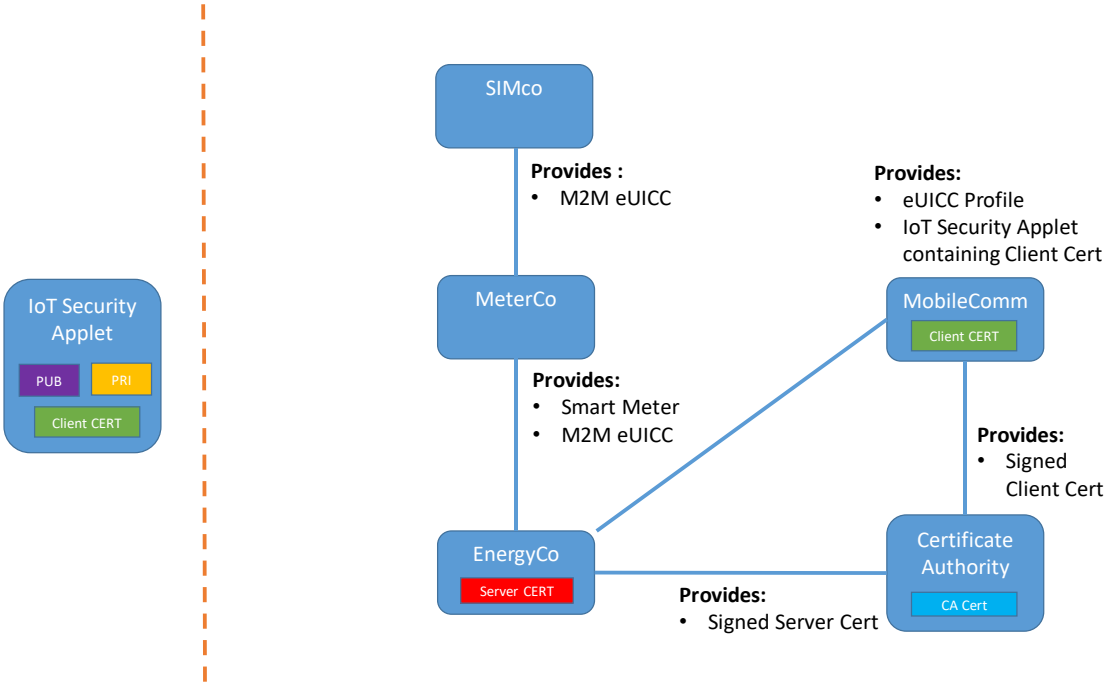

Figure 3: Relationships Between Actors for Scenario 2

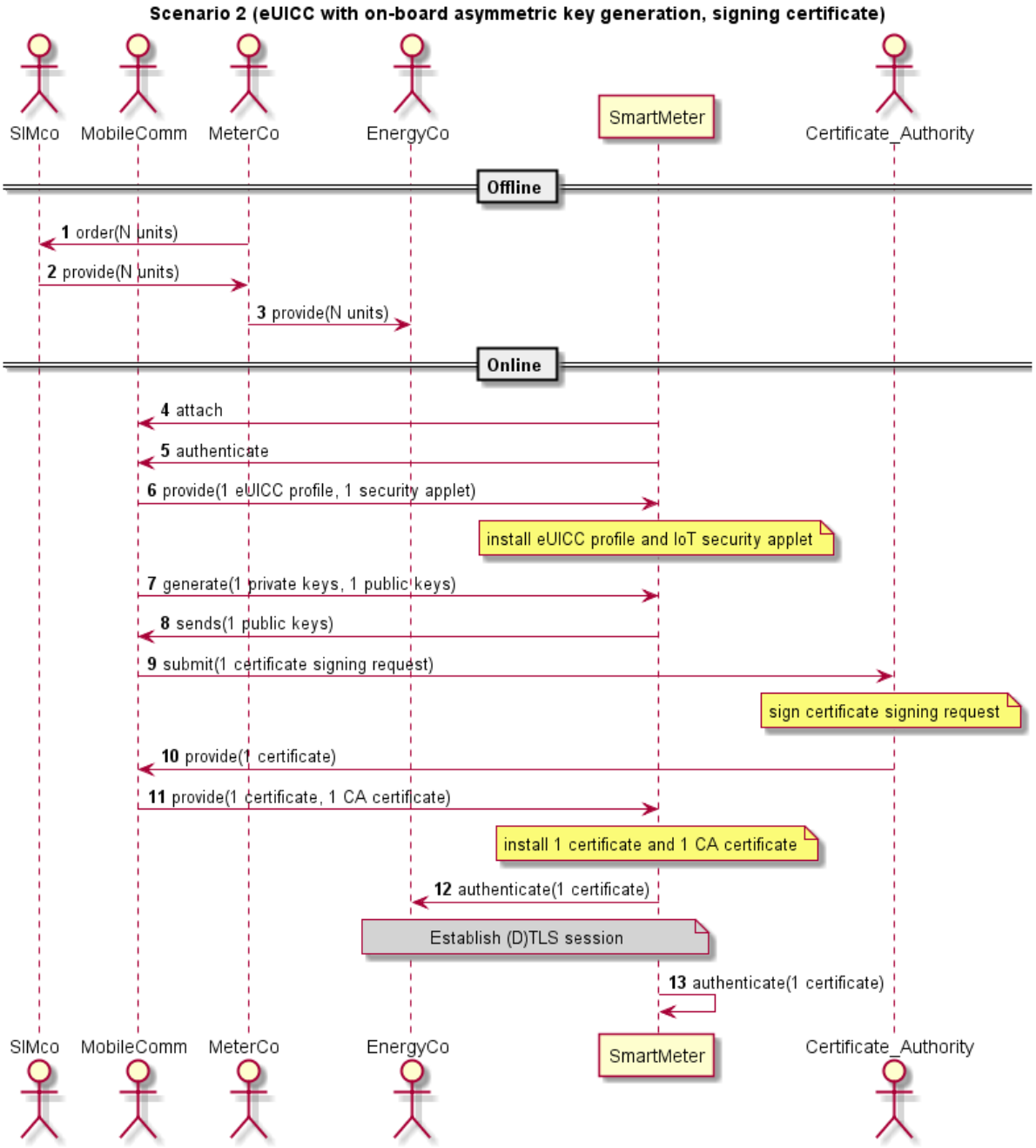

Figure 4: Flow of Events for Scenario 2

'MeterCo", a smart meter vendor, procures a batch of certified industrial grade M2M eUICCs from "SIMco", an eUICC vendor. At their factory, MeterCo solders an eUICC into each of their cellular connected smart meter products. MeterCo sells a batch of smart meters to their customer EnergyCo – an energy provider.

EnergyCo has a commercial relationship with MobileComm, a mobile network operator, to connect their smart meters to the MobileComm network and for the provision of additional security services.

When an EnergyCo smart meter is installed into a building and switched on for the first time, it is provisioned with a MobileComm eUICC profile and IoT security applet. Once the profile is installed and activated, the smart meter securely authenticates and registers to the MobileComm network. Once registered, MobileComm securely requests, over-the-air, the IoT security applet to generate a public/private key pair. The IoT security applet sends the public key to MobileComm. MobileComm generates a certificate signing request to a Certificate Authority chosen by EnergyCo. MobileComm provisions the signed certificate and trusted CA certificate into the IoT security applet.

The smart meter establishes an IP connection to EnergyCo's service platform and initiates a mutual authentication procedure to establish a secure (D)TLS connection with the service platform. The device side (D)TLS mutual authentication steps are performed using the IoT security applet within the eUICC.

The smart meter is now connected to the service platform and can send meter readings to EnergyCo.

During the lifetime of the smart meters MobileComm, acting on the request of EnergyCo, can revoke and renew the credentials stored within each smart meter – for example when a smart meter is moved or at end-of-life of a meter.

2.3 Scenario 3 (PSK#1: UICC manufactured and pre-loaded with set of key-loading keys for application owner to manage their own keys

In this scenario, data from an IoT device is secured using two symmetrical pre-shared keys that are preloaded onto the UICC at point of manufacture – a 'transport key' to secure the transport of the data by the mobile network operator and an 'encryption key' to keep the payload data private from the mobile network operator.

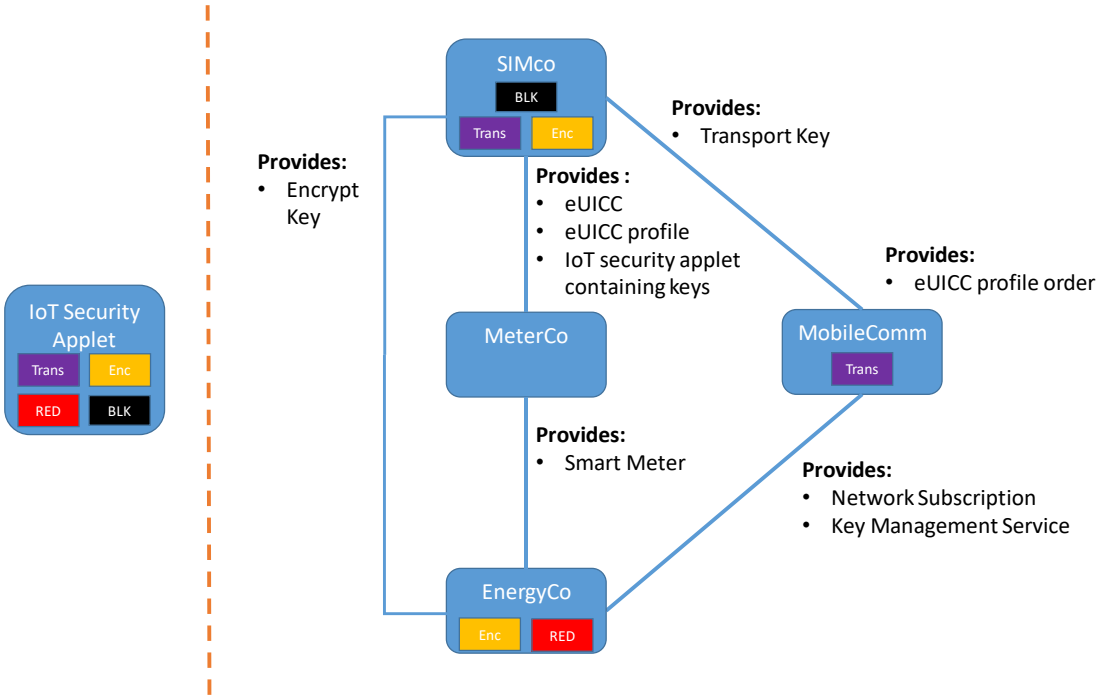

Figure 5: Relationships Between Actors for Scenario 3

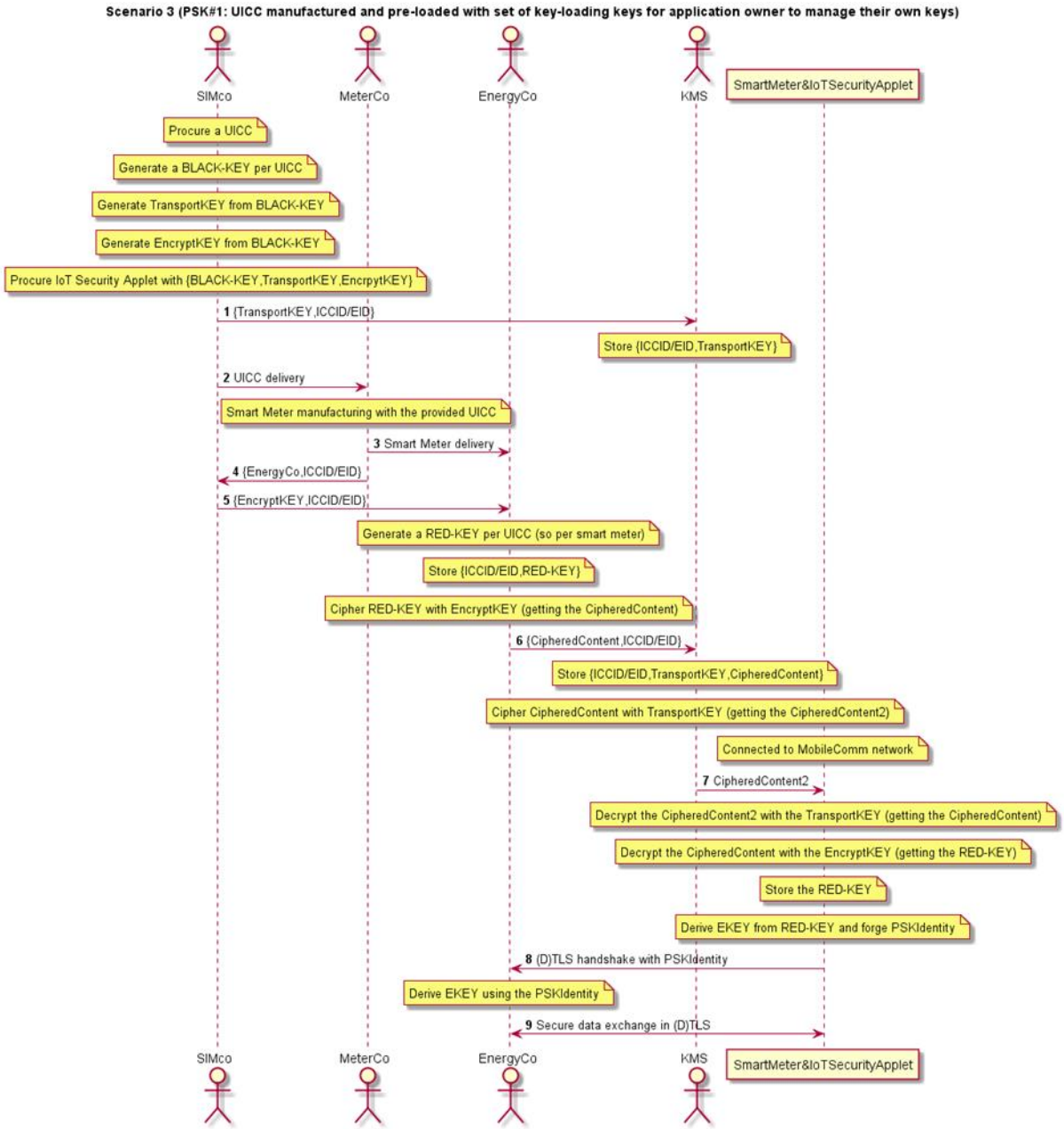

Figure 6: Flow of Events for Scenario 3

MeterCo, a smart meter vendor, procures a batch of UICCs from SIMco, a UICC manufacturer vendor.

SIMco provides a profile for the MobileComm operator and an IoT security applet.

SIMco generates a root AES-128 key (BLACK KEY) for each UICC and derives a set of keys from it:

- TransportKEY = KDF(BLACK-KEY, transport_usage_code)

- EncryptKEY = KDF(BLACK-KEY, encrypt_usage_code)
  (provided the Black key is stored within the IoT security applet new keys could be added as needed)

The usage codes are suitable numeric or text strings.

SIMco provides only the TransportKEY, along with the ICCIDs/EIDs to MobileComm running a KMS (Key Management System) or to a separated KMS operator trusted by MobileComm. The KMS is a part of the IoT security service.

At their factory, MeterCo solders an UICC into each of their cellular connected smart meter products. MeterCo sells a batch of smart meters to their customer EnergyCo – an energy provider, informing SIMco to supply EnergyCo with the EncryptKEYs associated to the Smart Meter UICCs.

EnergyCo has a commercial relationship with MobileComm, a mobile network operator, to connect their smart meters to the MobileComm network and for the provision of KMS services or it has a commercial relationship with MobileComm and a separated KMS operator trusted by MobileComm.

EnergyCo generates at least one AES-128 key (RED KEY) for each smart meter purchased, encrypts it with the EncryptKEY and delivers this *ciphered content,* along with the ICCID/EID to its KMS provider (MobileComm or a separated KMS operator). The KMS provider maps the *ciphered content* to the tuple {ICCID/EID, TransportKEY, *ciphered content*}. The KMS provider can upload the *ciphered content* to the smart meter*,* but cannot access the information that will be exchanged between meters and EnergyCo.

When an EnergyCo smart meter is installed into a building and switched on for the first time, it securely authenticates and registers to the MobileComm network and it is provisioned by the KMS operator with the RED_KEY, by downloading the *ciphered content,* and decrypting it with the TransportKEY. The RED_KEY can now be used by the Smart Meter to securely connect to EnergyCo.

When the smart meter establishes an IP connection to EnergyCo's service platform, it establishes a (D)TLS connection using a key (EKEY) derived from RED KEY. It sends during (D)TLS handshake the PSK identity to allow the EnergyCo service platform to recreate on the server side the same ephemeral key (EKEY) derived by the smart meter.

The smart meter is now connected to the service platform and can send meter readings to EnergyCo.

During the lifetime of the smart meters, the KMS provider, acting on the request of EnergyCo, can replace the RED KEY using the mechanism above.

2.4 Scenario 4 (PSK#2: Trusted third party manages transport and application security on behalf of application owner)

In this scenario, we protect the data generated by an IoT device using a symmetrical pre-shared key stored on the SIM and encryption/decryption service provided by a mobile network operator or Key Management Service provider.

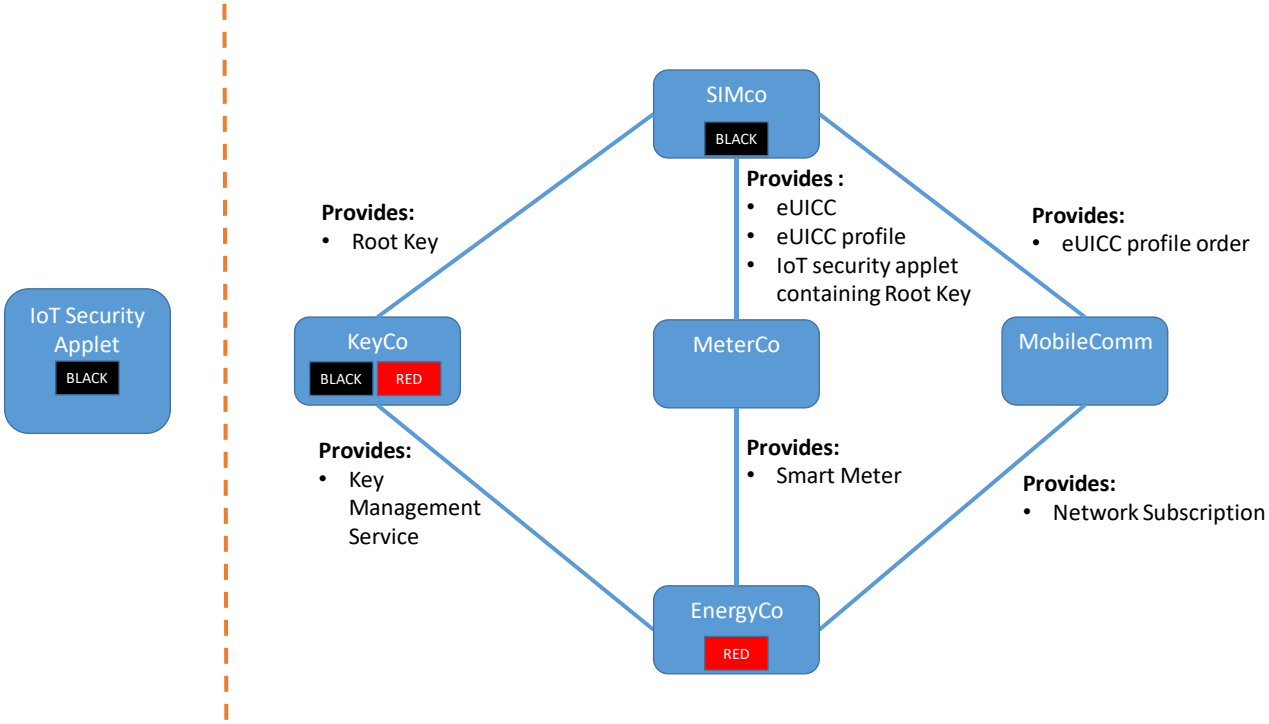

Figure 7: Relationships Between Actors for Scenario 4

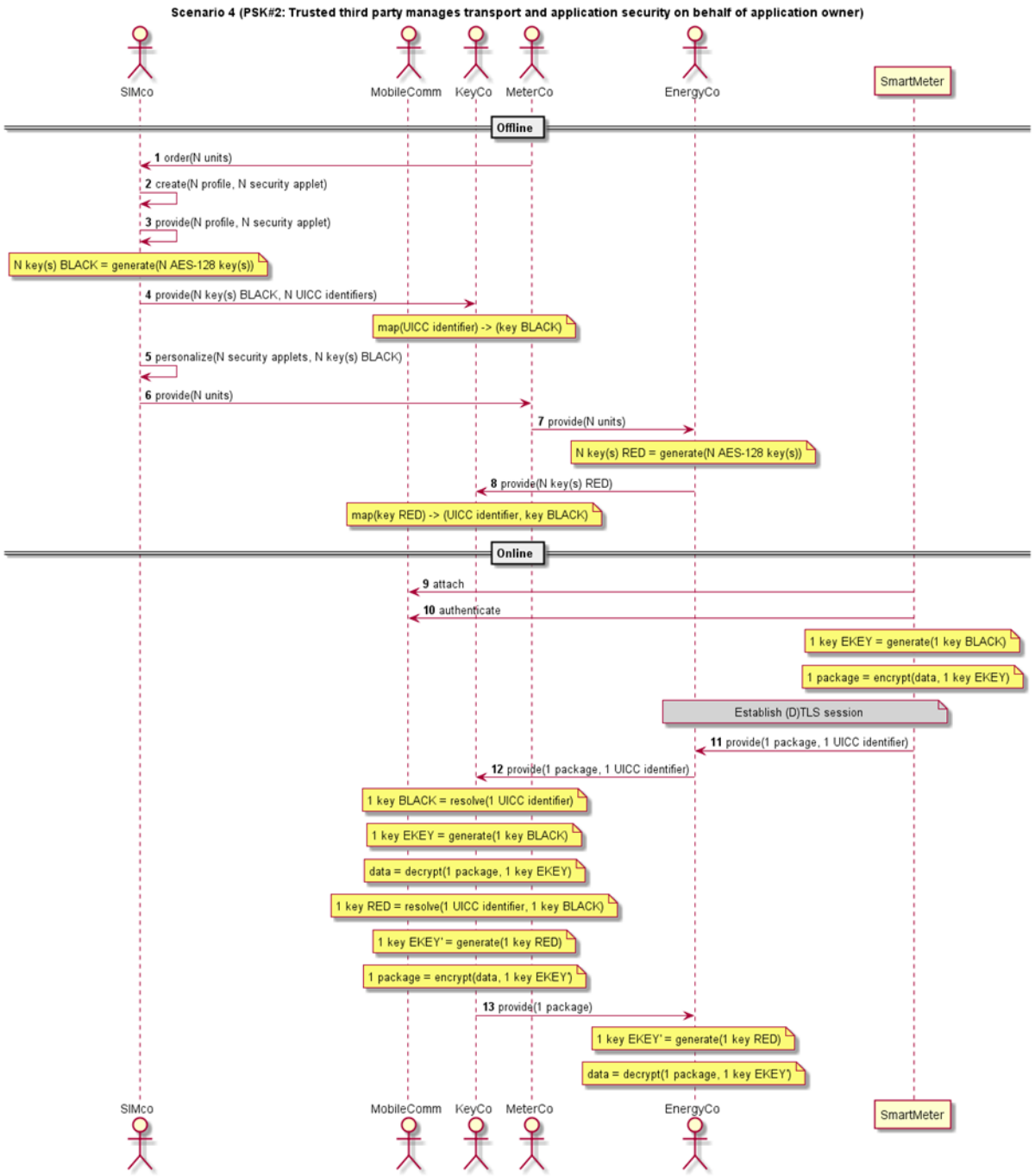

Figure 8: Flow of Events for Scenario 4

The steps are as follows:

MeterCo, a smart meter vendor, procures a batch of UICCs from SIMco, a UICC manufacturer.

SIMco provides eUICCs to MeterCo, each provisioned with an eUICC profile for the MobileComm operator and an IoT security applet.

SIMco generates a root AES-128 key (BLACK KEY) for each UICC and provides this key to MobileComm running a Key Management Service or to a separated KMS operator, KeyCo, trusted by MobileComm. The entity providing the KMS maps the ICCID/EID of the UICC to the corresponding BLACK KEY.

SIMco personalises the IoT security applet associated with each UICC with the BLACK KEY.

At their factory, MeterCo solders an UICC into each of their cellular connected smart meter products. MeterCo sells a batch of smart meters to their customer EnergyCo – an energy provider.

EnergyCo has a commercial relationship with MobileComm, a mobile network operator, to connect their smart meters to the MobileComm network and for the provision of a KMS (or it has a commercial relationship with MobileComm and a separated KMS provider "KeyCo" trusted by MobileComm).

EnergyCo generates a root AES-128 key (RED KEY) for each smart meter purchased and delivers this key to its KMS provider. The KMS provider maps the RED KEY to the tuple {ICCID/EID, BLACK KEY}.

When an EnergyCo smart meter is installed into a building and switched on for the first time, it securely authenticates and registers to the MobileComm network.

The smart meter establishes a (D)TLS protected connection to EnergyCo's service platform and sends to the EnergyCo service platform the meter readings encrypted using a key (EKEY) derived from the BLACK KEY. Along with the encrypted meter readings the necessary data is also provided to allow the recreation on the server side of the key (EKEY) derived by the smart meter (including ICCID/EID).

EnergyCo sends the encrypted payload to the KMS provider.

The KMS provider derives a key (EKEY) from the BLACK KEY using the material provided, decrypts the encrypted payload received from EnergyCo using the derived key, re-encrypts it with a new key (EKEY') derived from the RED KEY and sends it back to EnergyCo together with the material to allow the recreation on the EnergyCo side of the new ephemeral key (EKEY').

EnergyCo derives the new key (EKEY') from the RED KEY and the material provided and therefore can decrypt the meter readings.

During the lifetime of the smart meters the KMS provider, acting on the request of EnergyCo, can replace the RED KEY.

2.5 Scenario 5 (PSK#3: Trusted third party manages security, derived on a per-batch basis, on behalf of application owner)

In this scenario, we protect the data generated by an IoT device using a pre-shared key to derive common symmetrical ephemeral keys on both the IoT device and the server side.

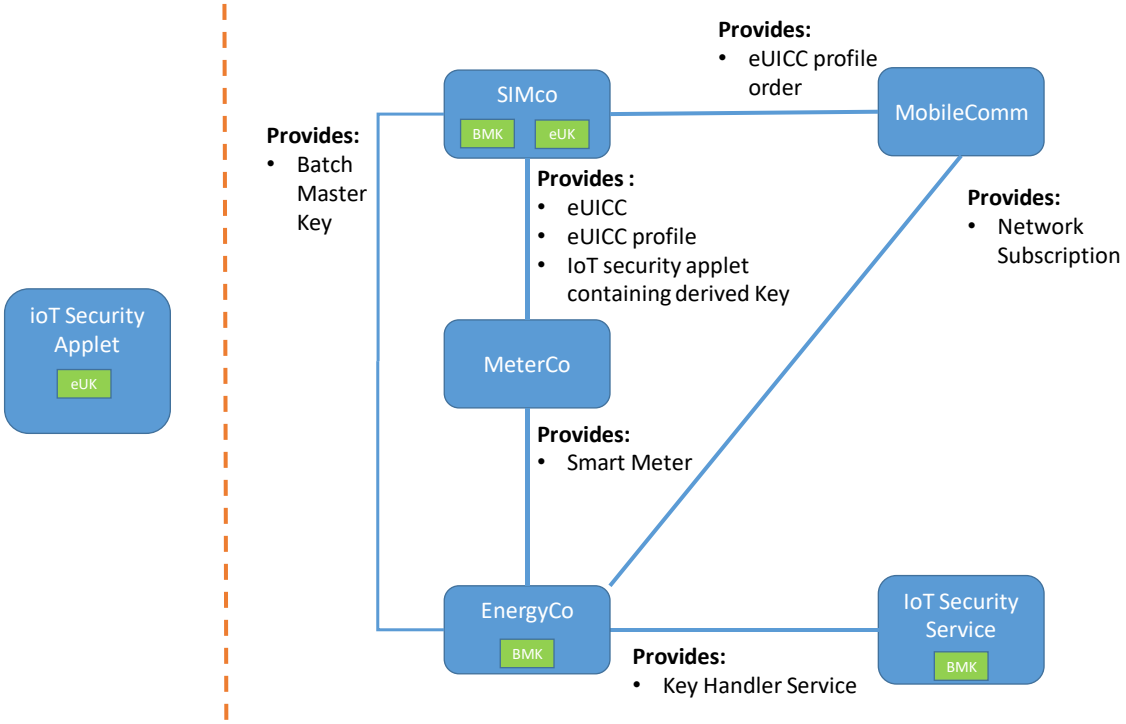

Figure 9: Relationships Between Actors for Scenario 5

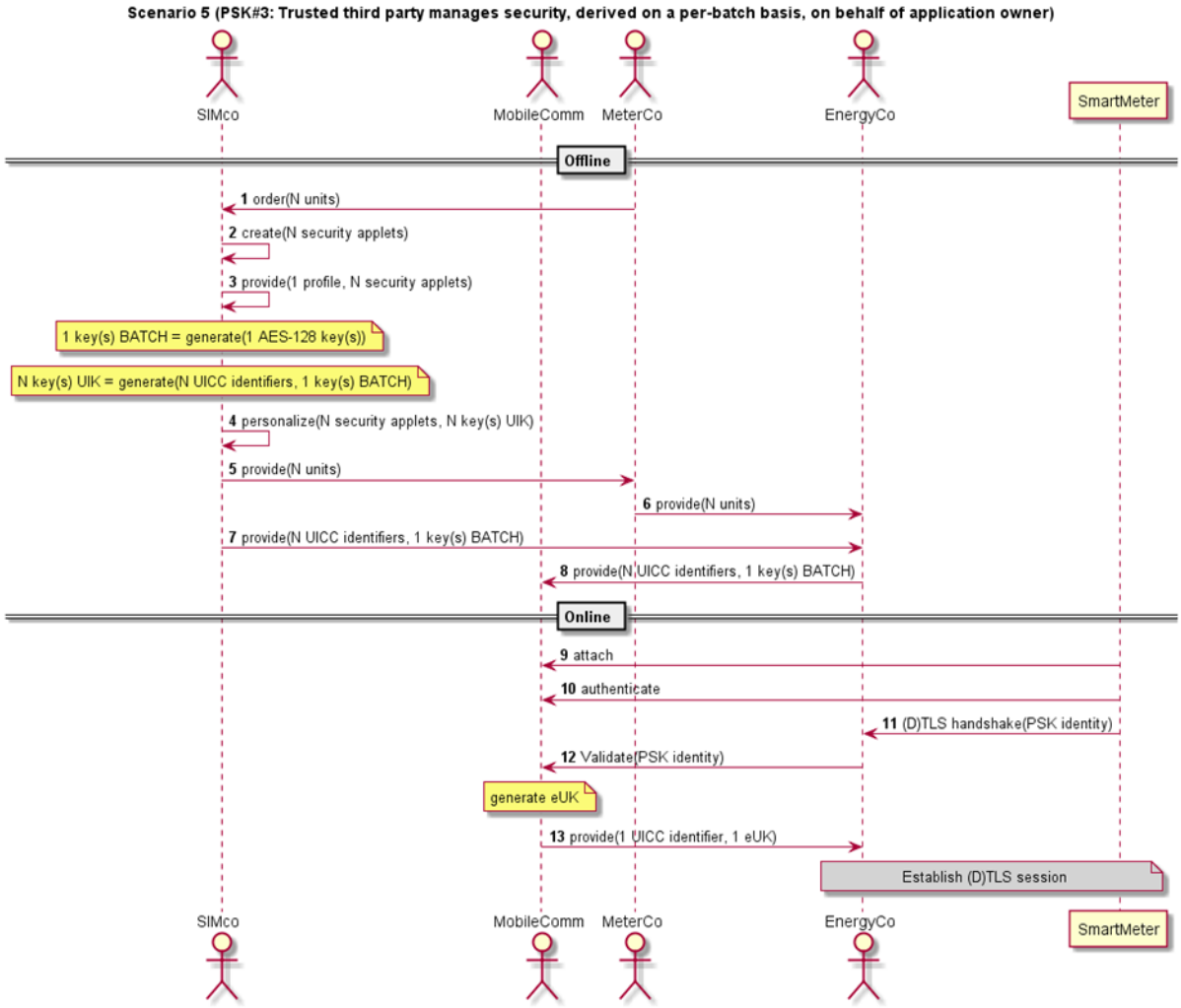

Figure 10: Flow of Events for Scenario 5

EnergyCo, an energy provider, would like to set-up a network of connected Smart Energy Meters. It plans to procure a batch of meters from MeterCo, a smart meter vendor. EnergyCo also establishes a commercial relationship with MobileComm, a mobile network operator, to connect the smart meters to their mobile network. EnergyCo will also establish a commercial relationship with an IoT security service provider of choice to help manage the fleet.

MeterCo, as part of their manufacturing process, procures a batch of certified eUICC's from SIMco (eUICC manufacturer), with the indication of MobileComm for the IoT connectivity.

SIMco provides a profile for the MobileComm operator and an IoT security applet. Upon ordering, SIMco generates a secret key BMK (Batch MASTER KEY), suitable for robust symmetric secrecy. It then personalizes each eUICC with a UICC KEYs, eUK, where eUK is derived from the BMK using a secure Key Derivation Function (KDF):

- eUK = KDF ("MeterCo" || "batch" || ICCID || .. , BMK)

SIMco ships the personalized eUICC to MeterCo. At the factory, MeterCo solders an eUICC into each of its cellular connected smart meter products. MeterCo sells the batch of smart meters to their customer EnergyCo, informing SIMco. Following that, SIMco securely sends the BMK and the list of ICCIDs sent in the last shipment to EnergyCo.

EnergyCo shares this information with MobileComm running an IoT security service, or to a separate IoT security service trusted by EnergyCo. The IoT security service sets up a PSK Key Handler service.

When an EnergyCo smart meter is installed in a building and switched on for the first time, it connects to the MobileComm network.

The smart meter establishes an IP connection to EnergyCo's service platform and initiates a (D)TLS connection with PSK authentication, with a list of only PSK ciphers in the ClientHello message. The client sends the following "PSK identity":

- "MeterCo" || "batch" || ICCID

IoT Service Provider Key Handler validates the identity of the connecting meter, and calculates the symmetric key eUK, that is used to complete the (D)TLS handshake with PSK authentication. The smart meter is now connected to the service platform and can send meter readings to EnergyCo securely.

During the lifetime of the smart meters the IoT security service Provider, acting on behalf of EnergyCo, can change the Key.

Possible option: A very constrained device may not be capable of sustaining (D)TLS with PSK authentication. In such a case, only an IP link is set-up, on top of which a password based authentications, such as SCRAM (RFC5802), or simpler challenge-response authentication (PAP, CHAP) using the "PSK Identity" as user, and the Key as password.

2.6 Scenario 6 (Application Certificate Provisioning)

In addition to the X.509 certificate for (D)TLS authentication that was provisioned on its eUICCs by the UICC manufacturer, EnergyCo uses an additional certificate for application-level authentication. EnergyCo uses a certificate authority to sign both device and server certificates. EnergyCo's server application has previously been provisioned with its own server certificate.

EnergyCo's IoT client application, using the IoT device middleware, requests onboard key generation of a new public/private key pair within the IoT security applet. It generates a certificate signing request for the public key, and it requests the IoT security applet to sign a portion of that request using the private key. The client application sends the CSR to EnergyCo's server, which forwards it to their certificate authority. The CA returns the signed client certificate. EnergyCo's IoT server application sends the new client certificate, its own server certificate, and possibly the CA's self-signed certificate to the IoT client application. The IoT client application, via the IoT device middleware, stores these to the IoT security applet.

The IoT client application and IoT server application are now able to perform mutual authentication using the application certificates stored in the IoT security applet and the HSM within the server.

2.7 Scenario 7 (GBA)

"MobileComm", a mobile network operator, has commercial relationship with "VehicleMan", a vehicle manufacturer. MobileComm provides cellular connectivity to VehicleMan's OBUs (On Board Units), as well as secure communication tunnels between OBUs and VehicleMan's application platforms. VehicleMan operates a ECA (Enrolment Certificate Authority) server for the provisioning of enrolment certificate to vehicles, which allows vehicle devices to securely connect to application servers.

"OBUCo" is the OBU supplier for VehicleMan. With OBUs equipped on each vehicle, they can communicate with other terminals or application platforms through cellular networks to achieve V2X services like vehicle navigation, vehicle monitoring, etc.

MobileComm delivers configured UICCs to VehicleMan in batches. VehicleMan then assembles a UICC onto the OBU of each vehicle. Before delivering OBUs to end users (could be vehicle manufacturers or vehicle users), VehicleMan binds the vehicle ID with OBU device ID as well as the mobile subscriber identity (MSISDN) associated with the UICC and configures the OBU with the IP address of the ECA server.

After each OBU gets access to the cellular network provided by MobileComm, it connects to the ECA server in order to obtain the enrolment certificate. As there are no provisioned security credentials on the OBU initially, the ECA server cannot identify and authorize the OBU, thus it requires the OBU to do authentication by means of GBA. The OBU triggers a bootstrapping procedure using GBA and the core network of MobileComm starts the authentication of the OBU. With successful authentication results obtained, MobileComm generates a session key for the ECA server and the OBU for the use of setting up an end-to-end secure tunnel. Via this secure tunnel, the OBU sends generated enrolment certificate public key and the device ID to the ECA server, in order to apply for the enrolment certificate. The ECA server signs the enrolment certificate accordingly and the OBU can use the secure tunnel for downloading it.

Similarly, OBUs are able to obtain other certificates or credentials from VehicleMan's application platforms using GBA, in order to achieve initial provisioning of other services. There is no need for OBUs to pre-provision certificates or pre-shred keys at the production line.

3 Solution 1 - Use of SIM Applet

3.1 Solution Architecture

3.1.1 Architecture Diagram

This section defines the functional architecture required to support the solution. The basic building blocks of the architecture consist of the functions to be performed and the roles performing the functions.

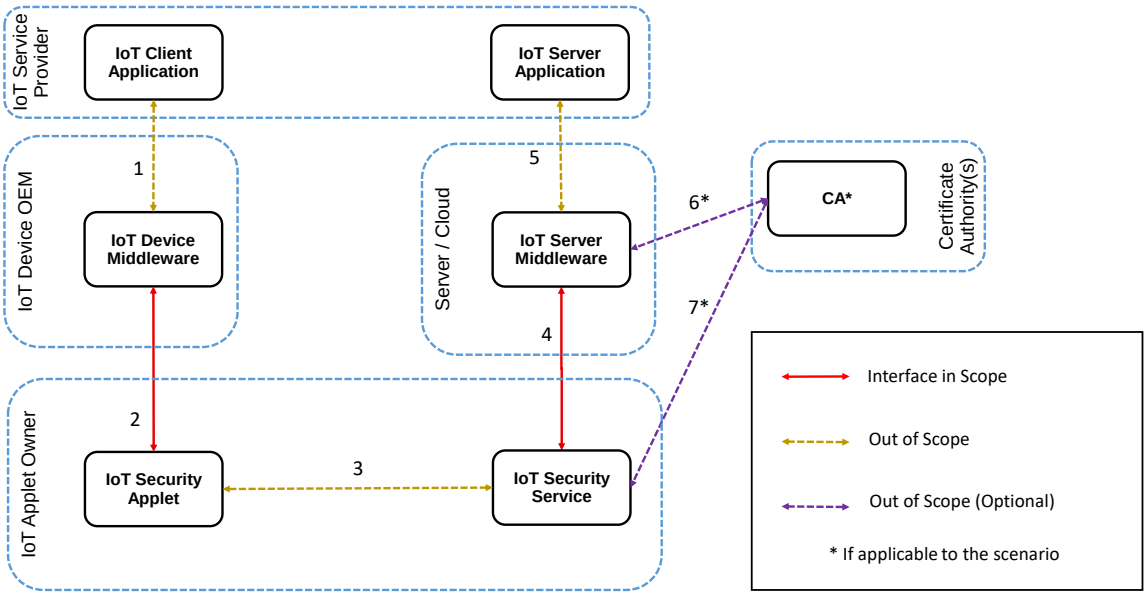

Figure 11: Architecture of Solution 1 (Use of SIM applet)

The architecture shows the following interfaces:

3.1.2 IoT Client Application <> IoT Device Middleware Interface (Out of scope)

This interface allows the IoT device middleware to expose security functions to the IoT client application. See informative Annex B.

3.1.3 IoT Device Middleware <> IoT Security Applet Interface (In-Scope)

Enables the IoT device middleware to communicate with the IoT security applet on the UICC. This interface is in scope because there are expected to be many different IoT device providers and many different IoT security applet providers. For a successful ecosystem to develop it is essential that the IoT device middleware and IoT security applet can interface using a common API.

3.1.4 IoT Security Applet <> IoT Security Service Interface (Out of Scope)

This interface enables the IoT security applet to be managed (i.e. to perform the loading/update/deletion of credentials within the applet) by the IoT security service. This interface is out of scope, as this interface will reuse existing APIs such as those defined by ETSI, 3GPP and GlobalPlatform for Over-the-Air SIM management or GSMA for remote SIM provisioning. Alternatively, if the credential management operations are taking place within a secure SIM production environment, proprietary APIs may be used.

3.1.5 IoT Security Service <> IoT Server Middleware Interface (In Scope)

Some use cases require an exchange of information between the IoT security service and the IoT server middleware. This interface is in scope as there are expected to be many different IoT security service providers and many different IoT server middleware providers. For a successful ecosystem to develop, it is essential that these two entities can interface using a common API.

3.1.6 IoT Server Middleware <> IoT Server Application Interface (Out of Scope)

This interface allows the IoT server middleware to expose security functions to the IoT server application. This interface is out of scope as de-facto security APIs exist and are already implemented by for most server/cloud providers.

3.1.7 IoT Server Middleware <> CA Interface (Out of Scope)

To support the public certificate based use cases in this document, an interface may exist between the IoT server middleware and a Certificate Authority.

This interface is out of the scope of this document as there are a relatively few number of CAs and APIs to these CAs are already well defined.

3.1.8 IoT Security Service <> CA Interface (Out of Scope)

To support the public certificate based use cases in this document, an interface may exist between the IoT security service and a Certificate Authority.

This interface is out of the scope of this document as there are a relatively few number of CAs and APIs to these CAs are already in place.

3.2 Functional Description of the Building Blocks

3.2.1 IoT Client Application

Functional Requirements for the IoT Client Application (Solution 1)	
A1	The IoT client application shall securely communicate to an IoT server application using the flows described in this document.
A2	The IoT client application shall initiate mutual authentication between the IoT device and the IoT server.
A3	The IoT client application shall have an interface to the device middleware.

3.2.2 IoT Device Middleware

Functional Requirements for the IoT Device Middleware (Solution 1)	
B1	The IoT device middleware shall be able to mutually authenticate with the IoT server middleware using the flows described in this document.
B2	The IoT device middleware shall implement a (D)TLS stack.
B3	The IoT device middleware provides an API to the IoT client application to establish (D)TLS connection(s) using the IoT security applet.
B4	The IoT device middleware shall implement the (D)TLS functions which are not mandatory within the IoT security applet to establish a successful (D)TLS session.
B5	The IoT device middleware shall send commands to the IoT security applet.
B6	The IoT device middleware shall support an interface to the IoT security applet on the UICC. This interface shall use the APDU based protocol defined in section 3.3.1 of this document.
B7	The IoT device middleware shall support at least one cryptographic hash operation.

B8	The IoT device middleware shall be able to generate (D)TLS session keys.
B9	The IoT device middleware shall be able to verify server certificates.

3.2.3 IoT Security Applets

Two applets are defined in this section:

- IoT Security Applet Type 1
- IoT Security Applet Type 2

IoT Security Applet Requirements (Solution 1)	
C1	IoT Security Applet Type 1 shall satisfy IoT Security Feature Set 1 (as defined in section 3.2.3.1) and optionally satisfy IoT Security Feature Set 2 (as defined in section 3.2.3.2).
C2	IoT Security Applet Type 2 shall satisfy IoT Security Feature Set 2 (as defined in section 3.2.3.2).

3.2.3.1 IoT Security Applet Feature Set 1

This feature set satisfies asymmetric scenarios 1 and 2 from section 2 of this document.

This feature set optionally satisfies scenarios 3, 4 and 5 from section 2 of this document.

General Requirements	
D1	The applet shall enable the IoT device middleware to securely perform the mutual (D)TLS authentication to an IoT service provider server by supporting X509 and asymmetric keys security scheme.
D2	The considered (D)TLS version shall be version 1.2 & 1.3.
D3	The capabilities of the applet shall be discoverable by the IoT device middleware.
D4	The IoT security applet shall enable credential life cycle management from an IoT security service. These credentials shall be independent of any credentials managed by the device.
D5	The IoT security applet may enable credential life cycle management by the IoT client application. These credentials shall be independent of any credentials managed by the IoT security service.
D6	Not to conflict with reserved channels (i.e. LC0 reserved for telecom operations), the IoT security applet can operate on a dedicated logical channel.
D7	The IoT security applet shall be agnostic from which physical interface is used to communicate with the UICC.

D8	The IoT security applet shall only use APIs defined by JavaCard [6], GlobalPlatform [7] and ETSI 102 241 [7].
D9	The IoT security applet shall provide a list of the available PKI credentials (label and reference).

Cryptographic Requirements	
E1	The applet shall enable signature generation and verification using: • ECDSA with the following curves: NIST P-256. Minimum key length for ECC keys shall be 256bits. Other ECC curves may be supported – such as brainpoolP256r1 and Curve25519. The minimum key size shall be 256 bits.
E2	The applet should enable signature generation and verification using: • RSA keys: 2048bits (or more) for RSA (cf. NIST SP.800-57 [9], BSI TR-02102-2 [10] and ANSSI SDE-NT-35/ANSSI/SDE/NP [11]). and signatures using RSASSA PKCS1 V1.5 and RSASSA PSS.
E3	The applet shall provide storage of client credentials: X509 certificate & associated private key: • X509 certificate stored as plain files (Binary) • Private key is stored in dedicated key objects • Multiple key pairs and certificates must be considered.
E4	The applet shall provide storage of server and issuer credentials: • X509 certificates are stored as plain files (Binary) or • Public key is stored in dedicated key objects. Multiple certificates must be considered.
E5	The applet shall ensure certificates and associated private/public keys can be named and paired with a given unique label per UICC.
E6	The applet shall enable the device to establish a (D)TLS connection with Ephemeral scheme for key agreement using ECDHE and signatures using ECDSA.

Security Services provided by the IoT security applet to the IoT device middleware	
F1	The applet shall provide cryptographically secure pseudo-Random Number Generation (RNG) service. See: RFC5246 [12] Annex D.1 or RFC8446 [13] Annex C.1.

F2	The applet shall enable signature generation and signature verification service for the TLS handshake. Various signature modes might be supported among which: • Hash and padding is done externally (by the device), signature generation or verification done by the applet. • Hash, padding and signature generation or verification done by the applet.
F3	The applet shall enable server certificate validation
F4	The applet shall provide a service to compute ECDHE shared secrets (TLS1.2 & TLS 1.3).
F5	The applet shall provide a service to compute master secrets for pre-shared symmetric keys combined with ECDHE (TLS1.2).
F6	The applet shall provide a service to compute early secrets and handshake secrets (TLS1.3).
F7	The applet shall provide an OBKG service to generate ECDH ephemeral keys for the TLS handshake.

Provisioning and Remote Management by IoT Security Service	
G1	The applet shall support key pair provisioning at the factory with or without certificates
G2	The applet shall support OBKG requests from an administration server. The public key is returned to the administration server, with optionally CSR generated by applet.
G3	The applet shall support: • Injection of certificates from an administration server • Deletion and revoking of keys (private/public) and certificates from an administration server
G4	The applet shall allow the establishment of a remote management session at any time. During a remote management session, the applet shall not block requests from the IoT device middleware. During a remote management session, the applet shall respond to the IoT device middleware with an error if a credential used for the requested operation is in the process of being updated. The applet shall make it possible for the device to know that a remote management session has completed.

Local Provisioning and Management (if supported)	
H1	The applet shall support OBKG requests from the device. The public key is returned to the device, optionally along with a certificate signing request.
H2	The applet shall support: • Injection/storage of certificates from the device • Update/replacement of stored certificates that were generated by the device • Deletion of key pairs and certificates that were generated by the device
H3	The applet shall ensure certificates are signed by a trusted CA before the certificate is stored within the applet
H4	If a CSR is generated at the applet, it shall be additionally signed using one of remotely provisioned key in the applet dedicated to this usage. CSR and attestation signature along with reference of key used for signing shall be returned in the response.

3.2.3.2 IoT Security Applet Feature Set 2

This feature set satisfies symmetric scenarios 3, 4 and 5.

General Requirements	
I1	The applet shall enable the IoT device middleware to securely perform the mutual (D)TLS authentication to an IoT service provider server by supporting PSK (pre-shared keys) security scheme.
I2	The applet shall enable the IoT device middleware to compute shared secrets keeping long term keys secret.
I3	The considered (D)TLS version shall be version 1.2 & 1.3.
I4	The capabilities of the applet shall be discoverable by the IoT device middleware.
I5	The IoT security applet shall enable symmetric key life cycle management from a IoT security service.
I6	Not to conflict with reserved channels (i.e. LC0 reserved for telecom operations), the IoT security applet can operate on a dedicated logical channel.
I7	The IoT security applet shall be agnostic from which physical interface is used to communicate with the UICC.
I8	The IoT security applet shall provide a list of the available PSK credentials (label and object reference).
I9	The IoT security applet shall not allow the modification, deletion or retrieval of the PSK credentials through the device to applet interface.
I10	The IoT security applet shall only use APIs defined by JavaCard [6], GlobalPlatform [7] and ETSI 102 241 [8].

Security Services provided by the IoT security applet to the IoT device middleware	
J1	The applet shall support mutual authentication based on PSK as specified in section 2 of RFC4279 for TLS1.2 and sections 4.2.11 and 7.1 for TLS1.3
J2	The applet shall provide storage for a minimum of 4 PSKs (and respective PSK-Identity): • PSK with length up to 512 bits. (For recommended minimum PSK lengths see [3], [4] and [5]). • PSK-Identity.
J3	The applet shall ensure the IoT device middleware can use object reference of a PSK and PSK-Identity to use associated services.
J4	The applet shall support a PRF service (RFC5246 [12]) for the purpose of shared secrets computation (TLS1.2).
J5	The applet shall support a HKDF service (RFC5869 [14]) for the purpose of shared secrets computation (TLS1.3).

Provisioning and Remote Management by IoT Security Service	
K1	The applet shall support PSK and respective PSK-identity provisioning at factory.
K2	The applet shall support: • Injection of PSK and respective PSK-Identity from the IoT security service. • Deletion of PSK and respective PSK-Identity from the IoT security service.
K3	The applet shall allow the establishment of a remote management session at any time. During a remote management session, the applet shall not block requests from the IoT device middleware. During a remote management session, the applet shall respond to the IoT device middleware with an error if a credential used for the requested operation is in the process of being updated. The applet shall make it possible for the device to know that a remote management session has completed.

3.2.4 IoT Security Service

Functional Requirements for the IoT Security Service (Solution 1)	
L1	The IoT Security service shall provide the IoT client application credentials to the IoT service middleware to allow (D)TLS communication.
L2	The IoT Security service shall be able to securely set up credentials within the IoT security applet.
L3	When setting up asymmetric credentials the IoT security service may use the services of a certificate authority.
L4	The IoT Security service shall be able to securely manage the lifecycle of the client and server credentials within the IoT security applet.

L5	The IoT Security service shall be provisioned with all necessary credentials that have been provisioned into the IoT security applet at the factory.
L6	The IoT Security service shall securely communicate with the IoT security applet over the air
L7	The IoT Security service shall know the UICC identifier associated with the IoT security applet.
L8	The IoT Security service shall provide a secure interface to the IoT server middleware.
L9	The IoT Security service shall know the IoT service provider associated with the IoT security applet.
L10	The IoT Security service shall provision the server certificate to the IoT security applet.
L11	The IoT Security service may be able to act as a registration authority to request a certificate from a certificate authority (CA).

3.2.5 IoT Server Middleware

Functional Requirements for the IoT Server Middleware (Solution 1)	
M1	The IoT server middleware will enable the IoT server application to perform the security procedures defined in section 3.3 of this document (for example performing a (D)TLS connection establishment handshake).
M2	The IoT server middleware will manage the public and private credentials associated with the IoT server application and the public credentials associated with the IoT client application.
M3	The IoT server middleware uses the IoT security service to manage the IoT service provider's credentials within the IoT security applet.
M4	If initiating or renewing PKI credentials the IoT server middleware may use the services of a certificate authority, by acting as a registration authority.
M5	The IoT server middleware has an interface to the IoT security service.
M6	The IoT server middleware may provide an interface to the IoT security service to receive a certificate from a certification authority connected to IoT server middleware.

3.2.6 IoT Server Application

Functional Requirements for the IoT Server Application (Solution 1)	
N1	The IoT server application uses the security services provided by the IoT server middleware to manage (D)TLS connections.
N2	The IoT server application shall have an interface to the IoT server middleware.

3.3 Interface Description

3.3.1 IoT Device Middleware <-> IoT Security Applet Interface (Normative)

The IoT device middleware to IoT security applet interface (interface 2) is defined in the GSMA document IOT.05 [15]

3.3.2 IoT Server Middleware <-> IoT Security Service Interface (Informative)

Interface 4, which represents the relationship between the IoT security service and the IoT server middleware, comprises a set of REST APIs representing groups of services provided by the IoT security service or IoT server middleware, as depicted below:

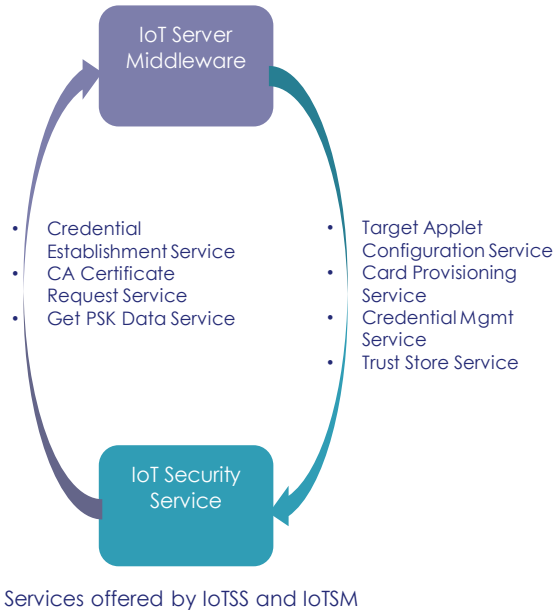

Figure 12: Depiction of IoT Server Middleware <-> IoT Security Service API

This section provides an informative description of the services to be provided on this interface. A normative specification of the REST APIs and associated object models will be provided in a future version of this document.

3.3.2.1 Services provided by IoT Security Service

The IoT security service provides the following services that may be used by the IoT server middleware.

3.3.2.1.1 Target Applet Configuration Service

This service defines operations to manage or view the Target Applet Configuration. A target applet configuration depicts the contents and structure of a card in the field. It can also be defined offline and provisioned to IoT security service after which it may be managed using these services.

A target applet configuration can be overridden by a newer version of it, if compatible.

The target applet configuration service exposes the following APIs:

- Get Target Applet Configuration List: Request list of target applet configuration associated to the account
- Get Target Applet Configuration: See target applet configuration information and its assignment to card/group

- Delete Target Applet Configuration: Remove a target applet configuration that is not assigned to any card/group
- Create Target Applet Configuration: Request to add a new target applet configuration
- Update Target Applet Configuration: General updating a target applet configuration is not supported to avoid complex tracking of history however, it can be made activate/inactive through this API.
- Apply Target Applet Configuration: Make a card ready with the target applet configuration. Create structure inside applet according to the Target Applet Configuration.

3.3.2.1.2 Card Provisioning Service

The IoT security service provides this API to allow the IoT server middleware to maintain the list of cards that it will manage through the IoT security service. This service is intended to be used for establishing card ownership and providing information which will be used once those associated cards come online or request credentials.

The ownership may be proven using any proprietary mechanism agreed between the IoT security service and the IoT card owner. IoT security service upon receiving and validating such information will associate these cards to the IoT server application and will perform lifecycle operations accordingly.

Deleting a card or multiple cards mean the IoT card owner no longer wants to manage those cards using the IoT security service. The IoT security service may then delete all records and remove them from a managed cards list as per its procedures and policies.

All the card operations are performed on a card group.

A card group is assigned a target applet configuration which in turn is valid for each card in the group. The IoT security service uses this information to manage cards accordingly. For example, it uses this information to generate credentials when a client is connected and requests for it using the label or identifier.

The following operations are supported:

- Get Card Group List: See list of card groups that have been defined for this account
- Get Card Group: See list of cards assigned to a certain group and group properties like target applet configuration
- Delete Card Group: Remove a card group which doesn't have any cards assigned.
- Create Card Group: Request to add a new card group
- Update Card Group: Update a card group with updated list of cards and properties. This includes associating or un-associating target applet configuration to the group

3.3.2.1.3 Credential Management Service

This service is used to manage credentials in the IoT security applet. The service may be called directly without first calling the card provisioning service. In such case, implementation will require establishing ownership of cards as well.

The credential services list has operations defined for creating new credential as well. This service is used for a situation where IoT server middleware is responsible to proactively create credentials instead of the IoT security service getting triggers from elsewhere.

The following operations are supported:

- Create Card Credential – request symmetric or asymmetric credential from device
- Request CSR – request certificate signing request based on an asymmetric credential
- Request Credential Status – return transaction status
- Update Card Credential – update credential e.g. block/unblock, can also be used for updating externally generated certificate
- Remove Card Credential – remove credential from server and optionally also from client if accessible
- Get Card Credential – this can be used for already set up credential
- Get Card Credentials – return list of credentials in a card

3.3.2.1.4 Trust Store Service

This service is used to manage all trusted data inside the IoT security applet. The IoT server application can use these services to manage the IoT security applet trust store directly or by using an IoT server middleware.

This service is used by an IoT server middleware to store the server certificate, other security data or device configuration settings.

3.3.2.1.5 Supported Operations

The following operations are supported:

- Get Trust Store Data List: See list of all data stored in applet trust store. This could be filtered by a query parameter
- Get Trust Store Data: See a data stored in applet trust store using identifier. This could be filtered by a query parameter
- Delete Trust Store Data: Remove a data entry from applet trust store
- Add Trust Store Data: Request to add new data in applet trust store
- Update Trust Store Data: Modify data in applet trust store
- Get Transaction Status: Get status of earlier requested operation for applet trust store data i.e. Delete, Add, Update operations

3.3.2.2 Services provided by IoT server middleware

The IoT server middleware provides the following services that may be used by the IoT security service.

3.3.2.2.1 Credential Establishment Service

This service is used to communicate client credentials in the field and associated information (such as a device identifier and/or ICCID/EID) to the IoT server application (or IoT server middleware) by the IoT security service. A credential represents a key on the IoT security applet, either an X.509 certificate corresponding to a private key or a symmetric key.

The IoT security service invokes this API to inform the IoT server middleware of the creation, update, and deletion of client credentials. In addition, the IoT security service may query the IoT server middleware about the existence of supporting information for a credential.

When an IoT solution provider orders UICCs from a mobile network operator (or indirectly through a reseller), the order is fulfilled through the physical delivery of UICCs and the corresponding delivery of the credentials (symmetric or PKI) that may have been provisioned during UICC manufacturing. This information may be configured in the IoT server middleware in one of two ways:

1. The mobile network operator may provide the credentials in some offline form (e.g., a file) to the IoT server application. In such a case, this service is not invoked online by the IoT security service. Instead, the IoT server application (or an administrator who operates the IoT solution) would invoke this API to configure these credentials.
2. As part of the fulfilment of the purchase by the mobile network operator, an instance of the IoT security service offered by the mobile network operator may be associated to an instance of the IoT server middleware. The IoT security service would then invoke this API to configure the credentials in the IoT server middleware. It may also invoke this API to report changes to the credential (update, removal) at later points in its lifetime.

The following operations are supported:

- Add/Get/Update/Delete Device Group: Manage a collection of devices/cards associated with an IoT application
- Add/Get/Update/Delete Device Record: Manage an individual card/device with a device group and its credential(s) (asymmetric or PSK)
- Add/Get/Update/Delete Group Record: Manage a group of cards/devices with related credentials (e.g., same signing CA for all certs)

3.3.2.2.2 CA Certificate Request Service

This service provides a means for the IoT security service to send certificate signing requests through the IoT server middleware to its certificate authority (i.e., when the certificate authority is connected to the IoT server middleware through interface 6 rather than to the IoT security service through interface 7). The following operation is supported:

- Get Certificate: A Request certificate based on CSR generated by the client

3.3.2.2.3 Get PSK Data Service

This service provides a means for the IoT security service to obtain a pre-shared key value from the IoT server middleware for provisioning a specified PSK key within one of the managed cards. The following operation is supported:

- Get PSK Data: service to request PSK from IoT server middleware

4 Solution 2 – Use of GBA

The authentication infrastructure of mobile network operators, including the 3GPP Authentication Centre (AuC), the USIM or the ISIM, and the 3GPP AKA protocol run between them, is a valuable asset for mobile network operators. It not only ensures the mutual authentication between subscribers and the mobile network, but could be leveraged to enable third party application providers to authenticate their users, as well as establish shared keys between them to securely communicate. With this infrastructure, 3GPP has defined GBA (Generic Bootstrapping Architecture) [16], providing the bootstrapping of application security

4.1 Solution Architecture

The diagram below defines the general functional architecture required to support the GBA solution.

At the network side, BSF (Bootstrapping Server Function) is a network element under the control of a mobile network operator, which is defined in 3GPP TS 33.220 [16]. BSF and IoT device middleware mutually authenticate each other based on the 3GPP AKA protocol, afterwards the BSF generates a session key that is applied between the IoT client application and IoT Application Server. The IoT Application Server is the network element using GBA to set up secure communication tunnels between the IoT client and the server, which is defined as NAF (Network Application Function) in 3GPP TS 33.220 [16]. After the bootstrapping has been completed, the IoT client application and IoT Application Server run an application specific protocol based on those session keys generated in the GBA procedure. The HSS is a mobile network operator's network element defined in [16], which stores the set of all user security settings used in GBA.

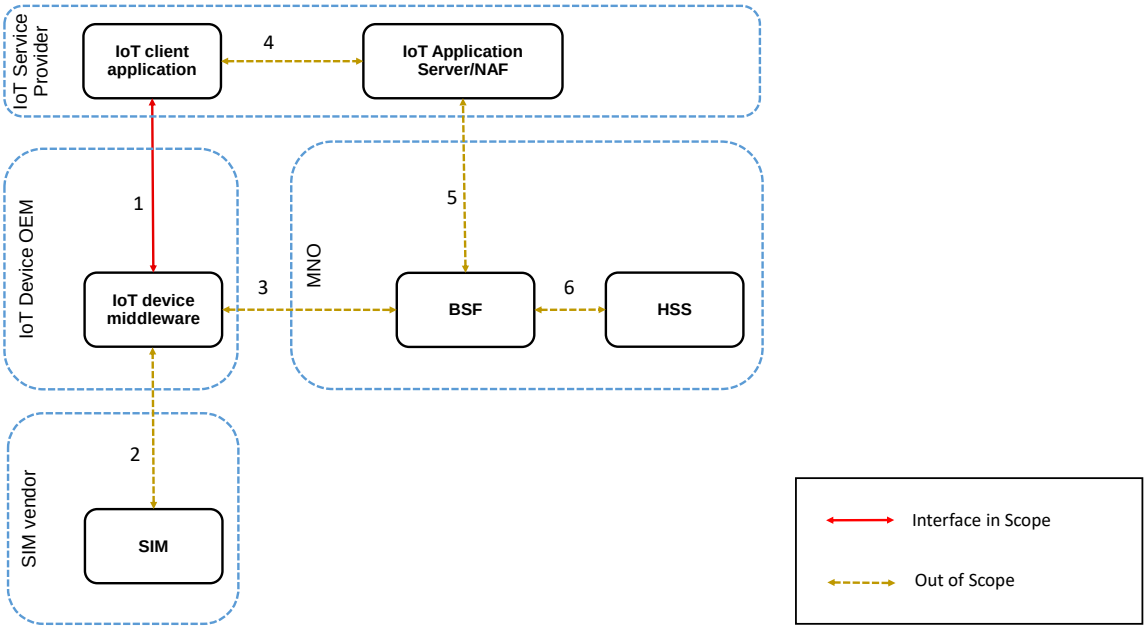

Figure 13: Architecture Diagram for Solution 2 (GBA)

The architecture shows the following interfaces:

4.1.1 IoT Client Application <> IoT device middleware (In-scope)

This interface allows the transmission of a session key generated in SIM (GBA_U) or IoT device middleware (GBA_ME) to an upper layer IoT client application, for the future use of secure tunnel establishment between the IoT client application and an IoT application server. This interface could be seen as a multivendor interface as the developers of IoT applications are assumed to be different from IoT device vendors, thus it has to be defined to ensure multivendor interoperability. Besides, there are also many different IoT device middleware vendors and different OS vendors running IoT applications, thus this interface is in scope and it is essential that IoT device middleware vendors can provide a common API for IoT applications.

4.1.2 IoT Device Middleware<>SIM (Out of Scope)

IoT device middleware uses the APDU protocol to forward parameters for AKA protocol to SIM via this interface. This interface is specified by ISO (APDU protocol), as well as by 3GPP (interface between UICC and ME), thus this interface is out of scope of this document.

4.1.3 IoT Device Middleware<>BSF (Out of Scope)

The interface enables GBA bootstrapping procedure, triggering mutual authentication between IoT device and the network. Since this interface is the same as the Ub interface defined by 3GPP [16], it is out of scope of this document.

4.1.4 IoT Client Application <>IoT Application Server/NAF (Out of Scope)

IoT Client Application initiates the bootstrapping procedure using this interface and also triggers the IoT Application Server/NAF to fetch session keys from BSF. This interface is specified as Ua interface in 3GPP [16], thus it is out of scope.

4.1.5 BSF <> IoT Application Server/NAF (Out of Scope)

This interface is used for NAF to fetch session keys from BSF, in order to further provide a third party application server with these session keys generated by a mobile network operator. Since this interface is specified in 3GPP [16] as Zn, it is out of scope of this document.

4.1.6 BSF <>HSS (Out of Scope)

This interface is used for BSF to fetch authentication vectors from the HSS to accomplish the authentication procedure. This interface is specified in 3GPP [16] as Zh, therefore it is out of scope of this document.

4.2 Functional Description

Detailed functional descriptions and requirements of BSF, IoT Application Server (NAF), HSS, as well as SIM are documented in 3GPP TS 33.220, where both GBA_U and GBA_ME mechanisms are included. The IoT device middleware and the IoT client application are together deemed as ME by 3GPP, however, they are separate blocks in this document, in order to provide multivendor interoperability. The following subsections detail the functional descriptions of the IoT device middleware and the IoT client application.

4.2.1 IoT Device Middleware

Functional Requirements for the IoT Device Middleware (Solution 2)	
A1	The IoT device middleware shall support the GBA bootstrapping procedure, involving mutual authentication with a BSF and generation of a session key Ks. In case of GBA_U the generation and storage of Ks is handled by the SIM, and in case of GBA_ME the generation and storage of Ks is handled by IoT device middleware.
A2	If GBA_ME is used, the IoT device middleware shall derive the key Ks_NAF from Ks as specified in 4.5.2[1] and pass Ks_NAF to the IoT client application.
A3	IoT device middleware shall be able to derive more than one Ks_NAF from one Ks in order to use them towards different IoT applications servers when required.
A4	If GBA_U is used, IoT device middleware shall transmit Ks_ext_NAF from the SIM to the IoT client application.
A5	IoT device middleware shall delete all GBA related keys and corresponding information when conditions in 4.4.11in 3GPP TS 33.220[16] are met.

4.2.2 IoT Client Application

Functional Requirements for the IoT Client Application (Solution 2)	
B1	IoT client application shall interact with the IoT Application Server / NAF to agree whether to use shared keys obtained by means of the GBA
B2	Once the IoT client application and the IoT Application Server / NAF have established that they want to use GBA then every time the IoT client application wants to interact with the IoT Application Server /NAF, it follows the steps of using the bootstrapped security association procedure of GBA.
B3	IoT client application uses Ks_(ext)NAF to set up a secure tunnel between IoT device and IoT application server after the bootstrapping security association procedure of GBA.
B4	IoT client application shall delete all GBA related keys and corresponding information when conditions in 4.4.11in 3GPP TS 33.220[16] are met.

4.3 Procedures

4.3.1 General Procedures of GBA

The detailed procedure of GBA follows 3GPP TS 33.220 [16]. This document gives an outline of the general 3-step procedure of GBA, specifically highlighting the division of IoT device middleware and IoT client application on the terminal side, as well as interactions between them, which are not specified by 3GPP.

4.3.2 Initiation of Bootstrapping

Before communication between the IoT client application and the IoT Application Server can start, they first have to agree whether to use GBA. When an IoT client application wants to interact with the IoT Application Server, but it does not know if the IoT Application Server requires the use of shared keys obtained by means of the GBA, the IoT client application may contact the IoT Application Server for further instructions.

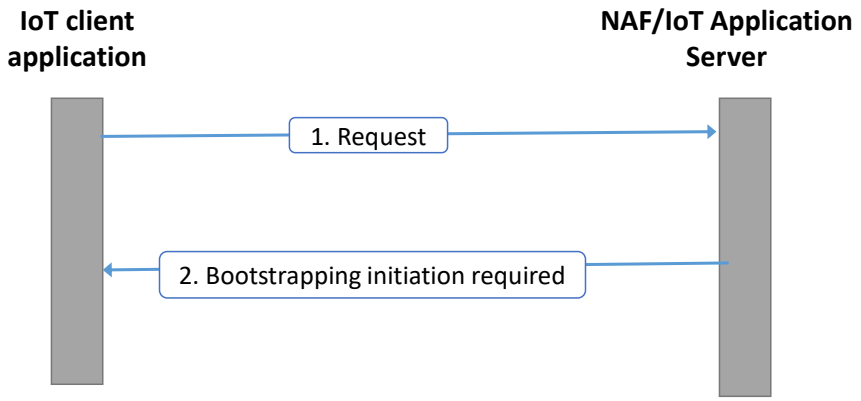

Figure 14: Initiation of Bootstrapping

1. The IoT client application starts communication with the IoT Application Server without any GBA-related parameters.
2. If the IoT Application Server requires the use of shared keys obtained by means of the GBA, it replies with a bootstrapping initiation message.

4.3.3 Bootstrapping Procedure

When an IoT client application wants to interact with an IoT Application Server, and it knows that the bootstrapping procedure is needed, it shall first perform a bootstrapping authentication. The IoT client application notifies the IoT device middleware to trigger the bootstrapping procedure.

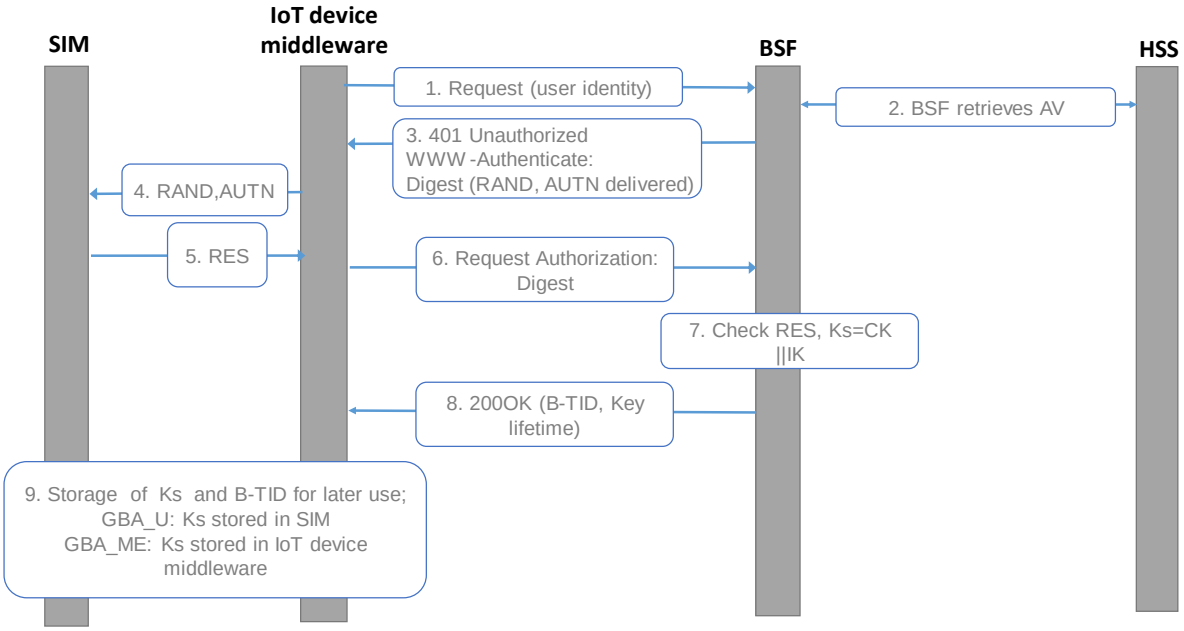

Figure 15: Bootstrapping Procedure

1. IoT device middleware sends an HTTP request towards the BSF.
2. The BSF retrieves the complete set of GBA user security settings and one Authentication Vector (AV, AV = RAND||AUTN||XRES||CK||IK) from the HSS.
3. BSF forwards the RAND and AUTN to the IoT device middleware in HTTP 401 message (without the CK, IK and XRES).
4. The IoT device middleware forwards RAND and AUTN to the SIM, via APDU based protocol.
5. The SIM checks AUTN to verify that the challenge is from an authorised network, then calculates CK, IK and RES. In the case of GBA_U the SIM stores Ks, which is the concatenation of CK and IK. In the case of GBA_ME, CK and IK are delivered to the IoT device middleware that calculates Ks. The SIM returns RES to the IoT device middleware.
6. The IoT device middleware sends another HTTP request, containing the Digest AKA response to the BSF.
7. The BSF authenticates the IoT device middleware by verifying the Digest AKA response. In order to bind the subscriber identity to the keying material, a bootstrapping transaction identifier (B-TID) is also generated by BSF.
8. BSF sends a 200 OK message, including a B-TID, to the IoT device middleware to indicate the success of the authentication. In addition, in the 200 OK message, the BSF shall supply the lifetime of the key Ks.
9. After receiving 200 OK from the BSF B-TID is stored with the Ks for later use. In case of GBA_ME the IoT device middleware handles the storage. In case of GBA_U, the IoT device middleware sends a message to notify the SIM.

4.3.4 Procedure Using the Bootstrapped Security Association

Before the communication between the IoT client application and the IoT Application Server/NAF can start, the IoT client application and the IoT Application Server/NAF first have to agree whether to use shared keys obtained by means of GBA. If the IoT client application does not know whether to use GBA with this IoT Application Server/NAF, it uses the Initiation of Bootstrapping procedure described in clause 4.3.2.

Once the IoT client application and the IoT Application Server/NAF have established that they want to use GBA then every time the IoT client application wants to interact with a IoT Application Server/NAF the following steps are executed.

In GBA_U case, the IoT client application and the IoT Application Server/NAF have to agree, which type of keys to use, Ks_ext_NAF or Ks_int_NAF, or both. The selection of keys and detailed key derivation can be referred in 3GPP TS 33.220 [16]. If used the Ks_ext _NAF is sent to the IoT client application. In GBA_ME case, the Ks_NAF is provided to the IoT client application when available. The following steps are specified in clause 4.5.3/5.3.3 in 3GPP TS 33.220 [16].

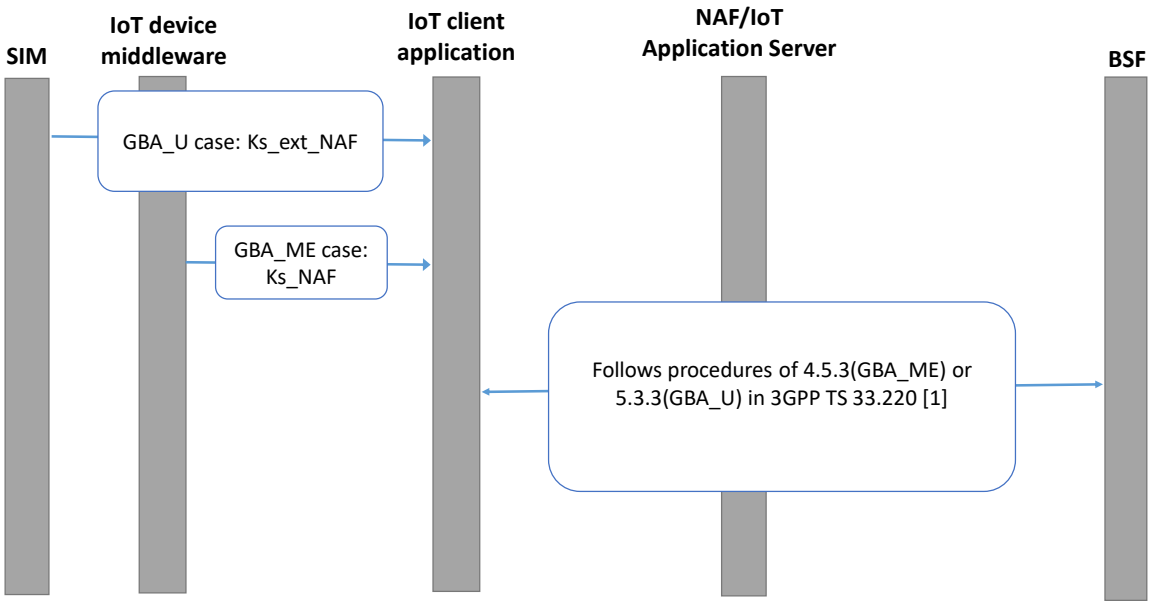

Figure 16: Bootstrapping Usage Procedure

Once the bootstrapping usage procedure is completed, the purpose of bootstrapping is fulfilled as it enabled the IoT client application and IoT Application Server/NAF to securely communicate. The IoT client application can start the communication with the IoT Application Server/NAF using the keys Ks_(ext/int)_NAF, as required.

4.4 Interface Description

4.4.1 IoT Client Application <-> IoT Device Middleware

This interface is used to parse Ks_(ext/int)_NAF and corresponding parameters from IoT device middleware to the IoT client application. When the IoT client application invokes the GBA functionality in the SIM via the IoT device middleware, this interface can be achieved using AT commands or any other protocols.

Annex A Example of Alternate Solution Architecture (Informative)

The use cases described within this document assume the use of a typical PKI backend architecture, such as the architecture described within the OASIS document "PKI Basics - a Technical Perspective" [20].

Alternate backend architectures could be considered and are described in this informative annex.

A.1 Example Blockchain Back End Architecture

According to the same principles described in the document, an additional backend architecture could be implemented to provide secure services (e.g. data encryption, digital signature etc.) for IoT based on a Blockchain concept. This alternative solution leverages the same requirements for the SIM/device interfaces and could be a different implementation with the same level of security compared to a traditional PKI as end to end encryption is provided and the Blockchain is the reference to store and retrieve the keys of the solution. In some conditions, the Blockchain could be adopted for the notarization of collected data from the remote sensors.

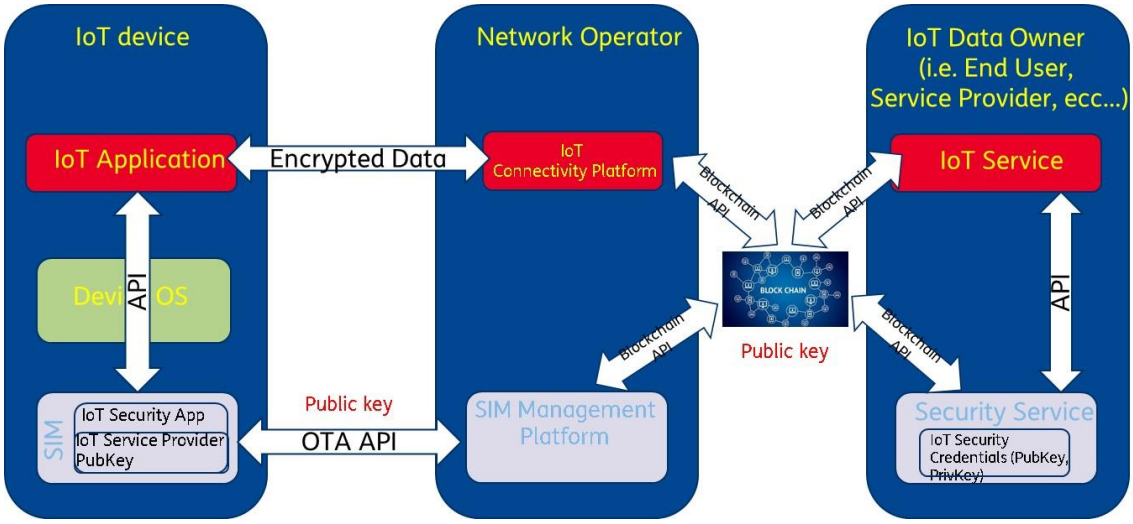

Figure 17: Example of Blockchain Back-end Architecture

In order to understand the components of the solution we can refer to Figure 18 below, where the whole system architecture is provided.

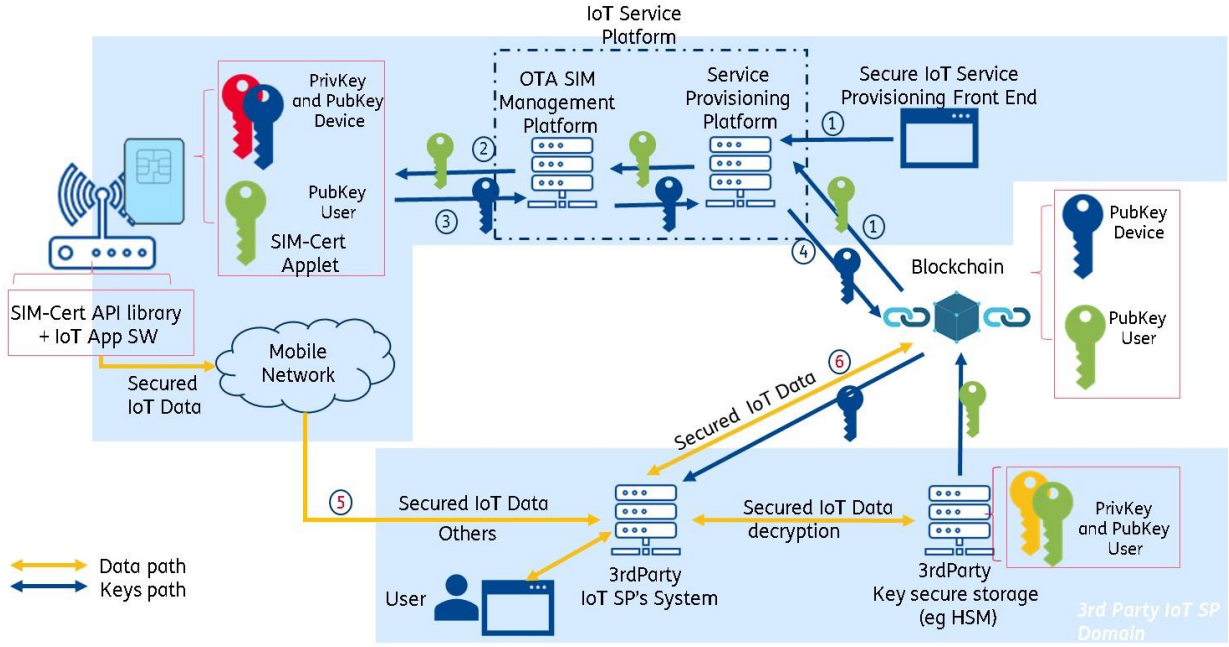

Figure 18: Example System Architecture

Two different domains are highlighted:

- The "**3rd Party IoT Service Provider domain**", comprising e.g. Service Provider's remote systems used to analyze collected IoT data and the Service Provider's security infrastructure.
- The "**Mobile Network Operator Domain**", comprising IoT Service Platform and Service Provisioning front end. This domain includes also the device with the SIM as the remote component of the service where data are collected/encrypted and transmitted towards the Service Provider's domain in a secure mode.

A Blockchain is used to share trusted public keys. The trust of public keys could be assured e.g. by creating one X.509 certificate for each domain, "**3rd Party IoT Service Provider"** and **"Mobile Network Operator**", and using it to sign public keys data before storing them in blockchain (in such way, the signature assures that a specific public key was published by a specific domain owner).

We can explore provisioning phase starting from the ***Secure IoT Service Provisioning front end*** which could be a web app able to collect all data required during service provisioning on a new SIM card or for update key material in SIM card or blockchain. The front end app provides inputs for the ***IoT Service Platform*** (which is composed by the Secure ***Service Provisioning Platform*** and the ***OTA SIM Management Platform***).

Service Provisioning Platform starts the provisioning and update processes: it provides a commands sequence to the OTA SIM Management Platform and interfaces with the blockchain for public key management, while the OTA SIM management platform performs Remote Applet Management, Remote File Management and provides support and API for binary SM-MT / SM-MO management.

Service Provisioning Platform and the OTA SIM Management Platform, are both involved in key material initialization and distribution in the whole system.

- Service Provisioning Platform provides API to Service Front End in order to acquire both information about SIM card to be provisioned and the User's Public Key.
- OTA Platform is the means to securely interact with SIM-Cert Applet. Via the OTA Platform, the Service Provisioning Platform can send User's Public Key to target SIM Card, invoke On Board Key Generation procedure and get the Device's Public Key.

The IoT device (which could be enabled with different sensors) shall be equipped with the target SIM card.

The SIM is the trusted core of the solution as it will contain the IoT security applet, as described in the normative sections of this document, with computing cryptographic mechanisms able to encrypt or sign, depending upon service requirements, the data collected by the device itself.

The IoT security applet provides the following features:

- Generation of Cryptographic Key Pair for digital signature service and Export of Public Key component
- Import of Third Party's Public Key, to be used for data encryption
- Encrypting of IoT device data
- Digital signature of IoT device data

The final IoT App will run on the IoT device's application processor (e.g. collecting data) and call the SIM applet cryptographic API (i.e. a library used to simplify the interaction with the cryptographic applet on the SIM card) before sending secured data to the Service Provider's IoT System.

In the Service Provider Domain, a Key Secure Storage system, which securely manages the Service Provider's private keys, should be included. It could be realized using a Hardware Security Module (HSM), enhanced with blockchain connection for public keys management.

The Blockchain implementation could have different purposes and it is linked to the commercial choices:

- public keys management: to allow the exchange of secure keys for the solution
- IoT data storage for the notarization of collected data where required

The entire solution has the following mandatory requirements which are already part of this document:

- SIM Applet shall be installed and used on SIM card equipped with ECC cryptographic accelerators as described in section 3.2.3.1
- Modem integrated in the IoT device shall provide support for an API (for example the AT+CSIM command – see Annex B) and Card Application Toolkit.

Annex B Example of Device <-> Applet Interface Implementations (Informative)

B.1 Platform Security Architecture (PSA) APIs

The PSA Cryptographic API (Crypto API) [19] provides an interface to modern cryptographic primitives on resource-constrained devices. The interface is user-friendly, while still providing access to the primitives used in modern cryptography. It does not require that the user have access to the key material. Instead, it uses opaque key handles.

The interface is scalable and modular in line with application requirements expressed in section 3.2.2 and for providing a consistent way of addressing the applications as specified in section 3.2.3.

PSA's modularity is such that you should not pay for a functionality that you do not need. You can choose between the applications described in section 3.2.3.1 and 3.2.3.2 depending on the targeted cloud infrastructure. Larger devices implement larger subsets of the same interface, rather than different interfaces, thus implementing only the necessary APIs required.

Because this specification is suitable for very constrained devices, including those where memory is very limited, all operations on unbounded amounts of data allow multipart processing, as long as the calculations on the data are performed in a streaming manner. This means that the application does not need to store the whole message in its memory at one time.

The interface does not expose the representation of keys which are securely stored in the SIM and protects intermediate data, except when required for interchange. This allows developers to choose optimal data representations.

The interface is designed to be as user-friendly as possible, given the aforementioned constraints on suitability for various types of devices and on the freedom to choose algorithms.

The API provides everything needed to establish TLS connections on the device side: asymmetric key management inside a key store, symmetric ciphers, MAC, HMAC, message digests, and AEAD.

Figure 19 demonstrates how an application can call the PSA APIs to leverage the crypto functions supported by the applications specified in section 3.2.3.1 and 3.2.3.2 in an easy way, abstracting the developer from all of the intermediate drivers, hardware layers and the complexity of coding APDU commands.

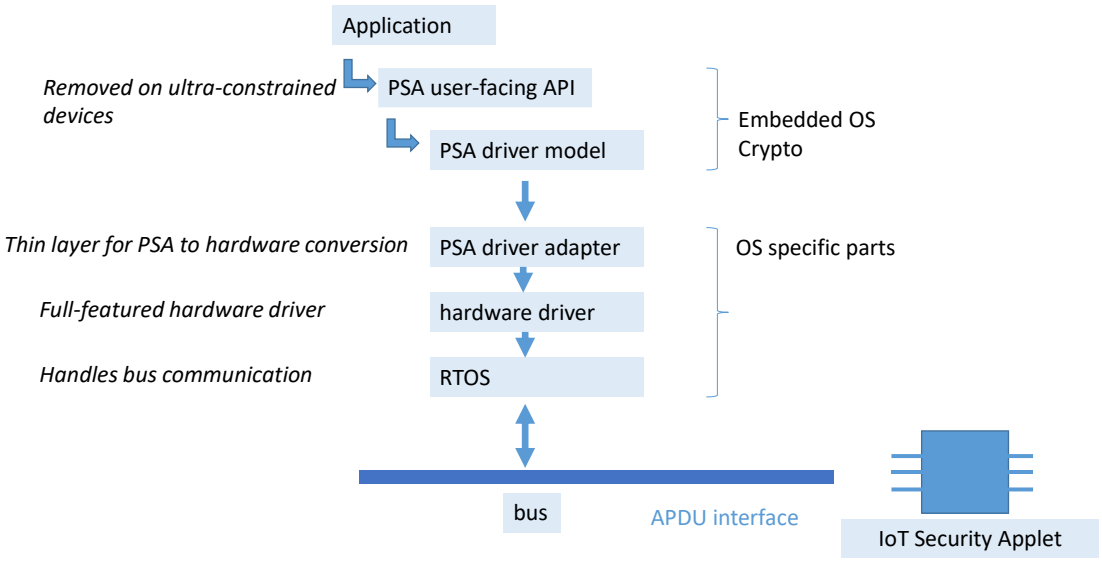

Figure 19: Leveraging PSA APIs to Access the IoT Security Applet

A full set of developer documentation on PSA and the relevant APIs are available online. See reference [19].

B.2 AT Command Interface Examples

The vast majority of SIMs today are physically connected to the cellular baseband modem. Although that may change in the future, currently the use of AT Commands tunnelled through the modem is a common way for an IoT device to interact with the SIM.

The (D)TLS stack can be executed in several configurations within IoT devices, depending on the type of device, the application requirements and designer needs. Two common cases are envisaged, with sub-options:

Case A: The (D)TLS stack is executed inside the device. The (D)TLS stack can be:

1. A separate module, using some commercial, open source, or custom implementation. This configuration may be chosen to reduce the development time, reuse an existing tested and/or certified module, or for other reasons not relevant for this document. We could refer to this configuration as a "*full"* middleware implementation.
2. A software module completely integrated into the IoT client application. This configuration may be chosen to have a highly optimised software (size, performance), or other specific needs not relevant for this document. We could refer to this configuration as a "*light"* middleware implementation.

Case B: The (D)TLS stack is executed inside the cellular baseband modem. In such cases, the IoT device can exploit the modem's computing power, and a well-tested (D)TLS implementation, reducing the computing burden on the rest of the IoT device.

Note: That in either case, the lower level stack will interact with the SIM and/or modem using AT Commands, but for slightly different purposes:

- **In Case A:** the AT Commands allow direct access to the IoT security services offered by the SIM. The interaction with the SIM services is essentially similar for A.1 and A.2.
- **In Case B:** the AT Commands will be directed to the cellular baseband modem to manage the (D)TLS connection, with the exception of some specific services that must be addressed directly to the SIM (e.g. services not related to (D)TLS management).

Cases A and B are shown diagrammatically below. For the sake of brevity we will refer generically to "IoT device middleware" for both "full" and "light" middleware implementations, as this does not affect the AT command behaviour.

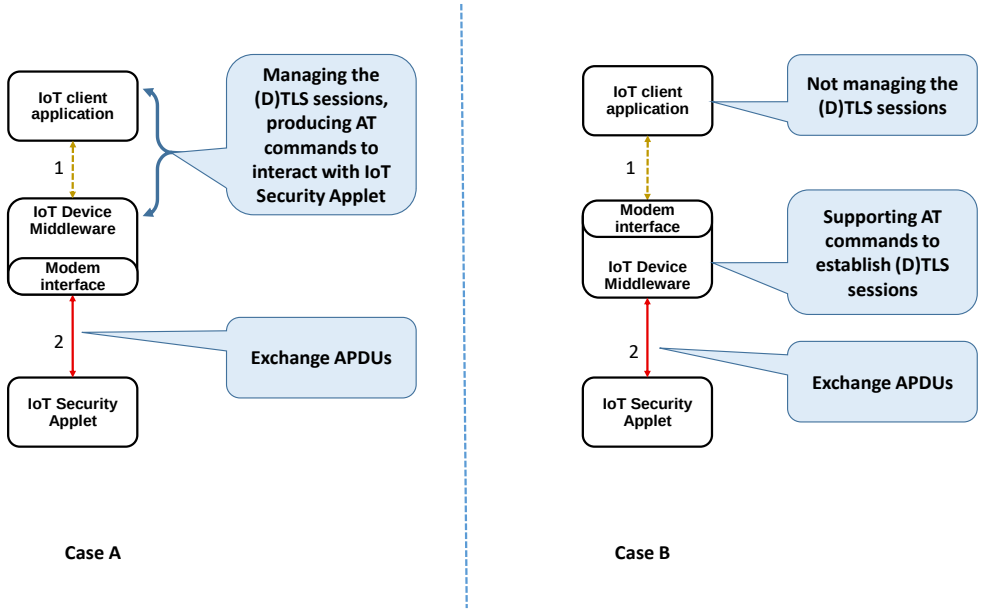

Figure 20: Example Configurations

In sections B.2.1 and B.2.2 we provide an example of the AT commands that might be used for Case A and Case B.

B.2.1 AT Command Example for Case A: IoT client application and IoT device middleware managing the (D)TLS session

In this scenario the IoT client application and IoT device middleware is able to manage the (D)TLS protocol stack and so it uses the modem only as a proxy to reach the IoT security applet.

3GPP TS 27.007 [17] AT commands exposed by the IoT Middleware can be used to exchange APDUs between the IoT client application and the IoT security applet. For example, in the "u-blox cellular modules AT commands manual" [18] you will find the following commands described:

- For services that require a single APDU, the following commands can be used:
    - 16.1: Generic SIM access +CSIM AT command
    - 16.2: Restricted SIM access +CRSM AT command

- For services that requires more than one APDU the logical channel commands can be used:
    - 16.8: Open logical channel +CCHO AT command
    - 16.9: Close logical channel +CCHC AT command
    - 16.10: Generic UICC logical channel access +CGLA AT command
    - 16.11: Restricted UICC logical channel access +CRLA AT command

In the example of u-blox cellular modules, for +CSIM, +CRSM, +CGLA and +CRLA the max length of the command is 255 characters and the max length for the response is 500 bytes.

Other examples can be found based on other cellular modules.

B.2.2 AT Command Example for Case B: IoT client application delegates the (D)TLS session to the modem

In this scenario the IoT client application delegates the (D)TLS session management to the cellular baseband modem with the purpose to create secure TCP sockets on which to exchange data securely. AT commands can be used for that purpose. As this set of commands is not fully standardised, please refer to the AT command list of the specific modem for the technical details.

As an example, u-blox modems support this mode of operation, as described in "u-blox cellular modules AT commands manual" [18]. There the following commands are described:

- 25.3: Create a TCP socket +USOCR AT command
- 25.4: TLS mode configuration +USOSEC AT command
- 26.1.2: TLS certificates and private keys manager +USECMNG AT command
- 26.1.3: TLS security layer profile manager +USECPRF AT command
- 25.9: Connect socket +USOCO AT command
- 25.10: Write socket data +USOWR AT command
- 25.12: Read socket data +USORD AT command
- 25.7: Close socket +USOCL AT command

In the example of u-blox cellular modules, the combination of settings defined in +USECMNG and +USECPRF drives which APDUs will be exchanged between the IoT device middleware and the IoT security applet when the +USOCO AT command is called and so when the end to end TLS session has to established.

The DTLS session establishment can be managed in a similar way but with a different set of AT commands.

Annex C Scenario Examples Elaborated for Interface 4 (Informative)

C.1 Scenario 1 (UICC Manufactured with asymmetric keys generated at personalisation step)

In this scenario, the applets and their initial credentials are installed during SIM personalisation in the factory. This information is passed through an offline channel (such as a file transfer) to the IoT solution provider (SmartCo), so there is no online interaction between the IoT security service in this process. (The IoT security service may participate in the applet configuration and creation of the offline/file data.) The communication of the device credentials to the IoT server middleware may be performed using the interface 4 API by an entity such as an administrator at SmartCo:

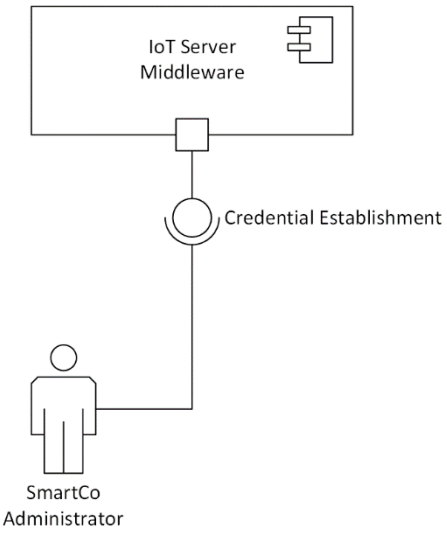

Figure 21: Realisation of Interface 4 for Scenario 1

The following sequence diagram illustrates how this could be accomplished. This is based upon the sequence diagram for scenario 1 in section 2.1 with the following elaborations:

- A new actor called "CamCo" is added to represent the manufacturer of the devices (in this case, security cameras) used by SmartCo, along with steps representing the purchase of these devices
- The actor for SmartCo is represented by two entities: an "administrator" actor representing the human(s) who perform the configuration steps at SmartCo and the IoT server middleware used by the SmartCo solution.
- The flow distinguishes the units provided by SIMco to SmartCo (SIM cards) from the devices SmartCo purchases from CamCo and sells through the Reseller to the Customer.
- Step 10 is represented by three sub-steps

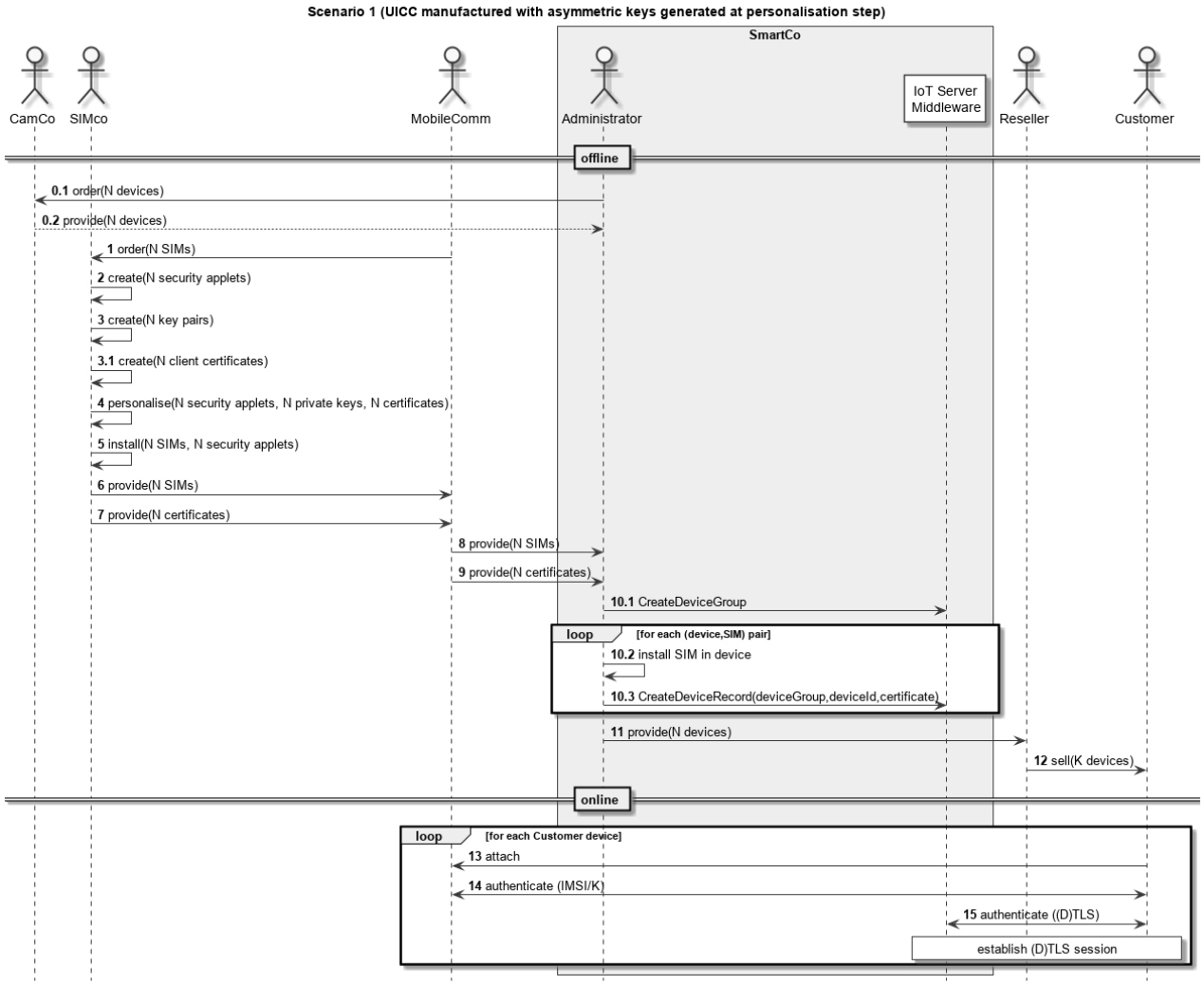

Figure 22: Scenario 1 Flow, Elaborated for Interface 4

Initially (steps 0.1 and 0.2) SmartCo purchases N devices (security cameras).

The ordering and personalisation of the SIMs (steps 1 through 7) proceeds as described in [1].

At step 8, MobileComm delivers N SIM cards to SmartCo. In addition, at step 9 it delivers information about those SIM cards, notably including the certificates they contain, to SmartCo.

At step 10 SmartCo prepares the devices and provisions them in its server. Its devices may be organised into one or more groups for administrative reasons. If the N new devices will be part of a new group, then it creates the device group on the IoT server middleware (step 10.1).

SmartCo has received N devices at step 0.2, N SIM cards at step 8, and the information about the certificates on those cards at step 9. For each of the N devices, a SIM card is inserted into the device (step 10.2) and a device record is added to the IoT server middleware that defines the relationship between the device and the certificate on its SIM

card (step 10.3). The IoT server middleware now has the information it requires to authenticate each device when it performs the (D)TLS handshake at step 15.

The remainder of the steps proceed as described in section 2.1.

C.2 Scenario 2 (eUICC with on-board asymmetric key generation, signing certificate)

In this scenario, eUICCs are installed in devices at the time the devices are manufactured, but they do not contain operational profiles and security applets. Assuming it is an M2M eUICC compliant with SGP.02, it does contain a profile for bootstrap purposes.[1]

The operational profile, security applet, and credentials are provisioned when the device is first switched on. In order for MobileComm to prepare the eSIM profiles, it needs to know the EIDs of all of the devices. In order for MobileComm's IoT security service to provision the security applets, an interface to EnergyCo's IoT Service Middleware must be established, including the EnergyCo's certificate authority. Finally, in order to report the creation of credentials on each device, the IoT security service needs to know the mapping between the eUICC cards and the smart meter devices, i.e., it needs to know the EID and device identifier of each device.

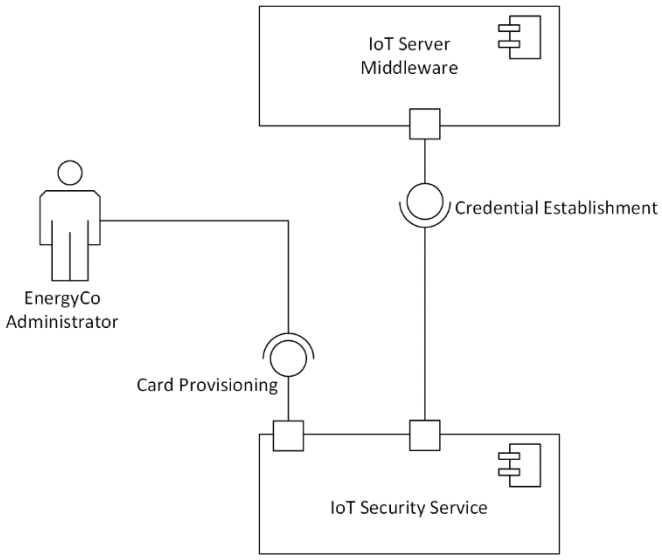

Figure 23: Realisation of Interface 4 for Scenario 2

The following sequence diagram illustrates how this could be accomplished. This is based upon the sequence diagram for scenario 2 in section 2.2 with the following elaborations:

- The actor for MobileComm is represented by three entities: its HSS and other network authentication infrastructure, its eUICC provisioning components (e.g., SM-DP and SM-SR), and its IoT security service

[1] An eUICC compliant with SGP.22 might contain a provisioning profile, or alternatively the device might use some other method for bootstrap connectivity, such as Wi-Fi. Elaboration of those alternatives is left as an exercise for the reader.

- The actor for EnergyCo is represented by two entities: an "administrator" actor representing the human(s) who perform the configuration steps at EnergyCo and the IoT server middleware used by the EnergyCo solution.
- The flow assumes that the units ordered from SIMco are eUICCs and distinguishes the eUICCs and their associated unique eUICC identifiers (EIDs).
- A number of steps in [1] are further elaborated with sub-steps in this flow.

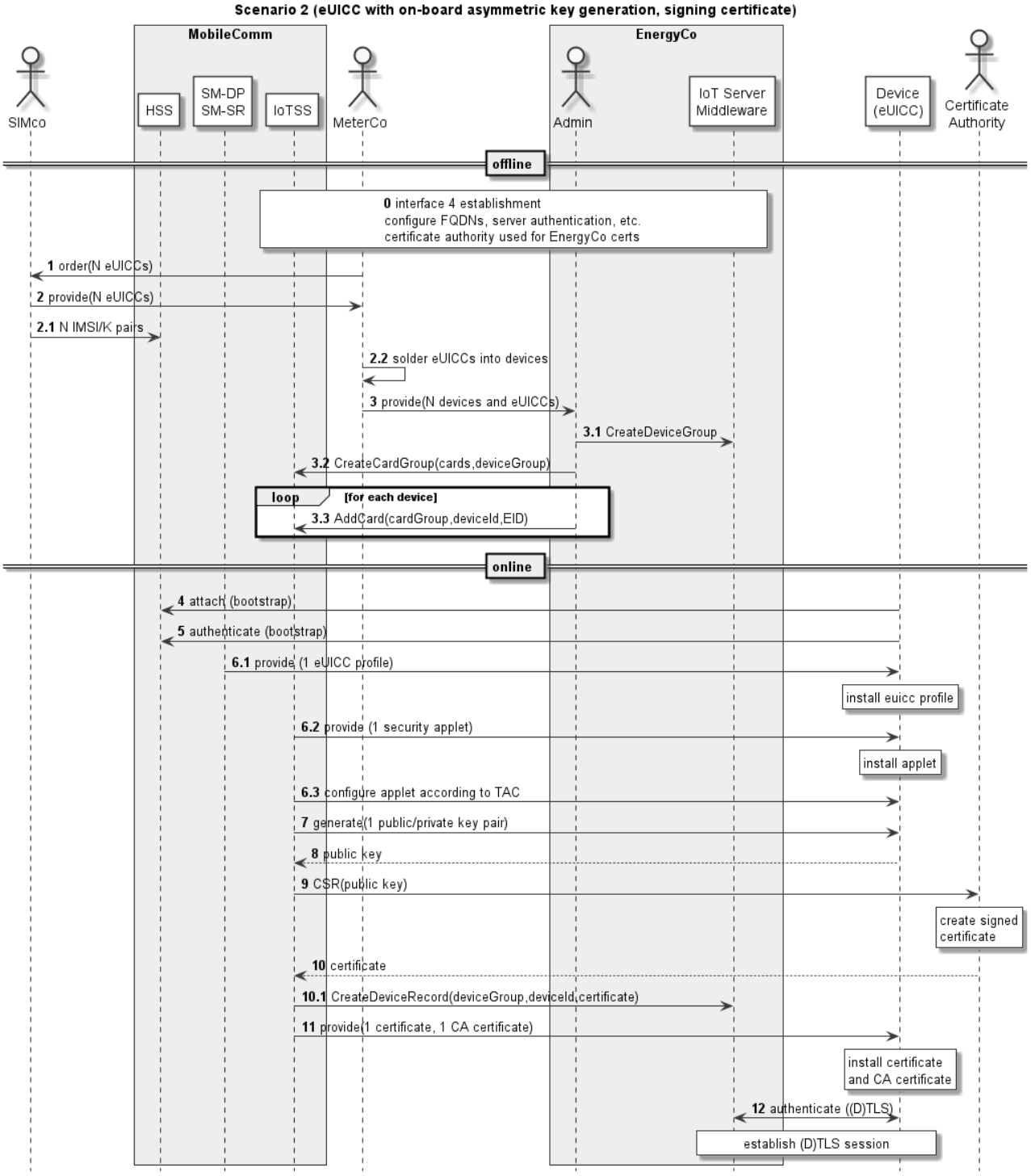

Figure 24: Scenario 2 Flow, Elaborated for Interface 4

Initially the relationship between EnergyCo and MobileComm is established, including the establishment of interface 4 between MobileComm's IoT security service and EnergyCo's IoT server middleware. This process includes the exchange of server FQDNs and credentials for mutual authentication. In addition, it includes the certificate authority that the IoT security service will use to sign all of MobileComm's certificates.

At step 1, MeterCo orders N eUICCs from SIMco. These eUICCs contain a bootstrap profile (assumed here to be for MobileComm), but do not contain operational profiles or security applets. SIMco provides the eUICCs (step 2) to MeterCo and their bootstrap IMSI/K pairs to MobileComm.

MeterCo solders each eUICC into a device (step 2.2) and provides N devices (each with a soldered eUICC) to EnergyCo (step 3).

Depending upon how it organizes its device information, EnergyCo may wish to create a new device group on its IoT server middleware or may simply add the new devices to an existing group (step 3.1). EnergyCo creates a card group in the IoT security service, specifying the Target Applet Configuration and the IoT server middleware device group to which these cards are associated. It then provides information about each card in the group (the card identifier, e.g., EID, and the associated device identifier).

Later, when a device is switched on for the first time, it attaches and authenticates to the MobileComm network using the bootstrap eSIM profile (steps 4 and 5), which SIMco provisioned during eUICC manufacturing. MobileComm provisions an operational profile (step 6.1) to the eUICC and its IoT security service provisions a security applet within that profile (step 6.2). IoT security service then configures security applet (step 6.3) as per the Target Application Configuration defined in step 3.2. It generates required containers and fills them with the fixed or initialization data as necessary.

At step 7, according to the Target Application Configuration for this card, the IoT security service instructs the applet within the eUICC to generate a public/private key pair and receives the public key in response (step 8). It creates a certificate signing request using this public key (and perhaps other information such as the device identifier associated with this EID) and sends the CSR to the certificate authority (step 9). (Depending upon the configuration of interface 4, it may contact the certificate authority directly or may request the CSR through the IoT server middleware.) The CA creates the signed certificate and returns it to the IoT security service (step 10).

After receiving the client certificate, the IoT security service adds a device record on the IoT server middleware that defines the relationship between the certificate and the device identifier (step 10.1). It also installs both the signing certificate and the individual device certificate in the security applet on the eUICC (step 11).

Annex D Plant UML Files for Scenario Sequence Diagrams

This annex contains the Plant UML files for the sequence diagrams found in this document.

D.1 Plant UML Files for the Section 2 Sequence Diagrams.


Scenario 5.txt


Scenario 4.txt


Scenario 3.txt


Scenario 2.txt


Scenario 1.txt

D.2 Plant UML Files for the Annex C Sequence Diagrams

This section contains the UML for the sequence diagrams found in section 2 of this document.


scenario1.puml


scenario2.puml

Annex E Document Management

E.1 Document History

Version	Date	Brief Description of Change	Approval Authority	Editor / Company
V1.0	3rd December 2019	Version 1 of the document.	Technology Group	Ian Smith, GSMA

4.5 Other Information

Type	Description
Document Owner	GSMA IoT Programme
Editor / Company	Ian Smith, GSMA

It is our intention to provide a quality product for your use. If you find any errors or omissions, please contact us with your comments. You may notify us at prd@gsma.com

Your comments or suggestions & questions are always welcome.