Security framework for use of identity-based cryptography in support of IoT services over Telecom networks

Summary

The traditional certificate-based solution involves heavyweight key management operations including certificate issuing, querying, and revocation. Such systems face great difficulty to keep up with the pace of device increment of IoT at the same time maintaining decent performance.

Identity-based cryptography (IBC) technology is another type of public-key technology which uses an entity's identity as a public key. As one of the essential features of IoT, everything has a unique identifier. Using such identifiers as public keys, no certificates are required. Consequently, IBC security solution utilizes simpler key management, enables distributed authorities controlling their own devices and scales well to both the high number of endpoints and the diversity of the devices.

This Recommendation provides security framework for use of IBC public key technology in support of IoT services over Telecom networks including mechanisms of identity management, key management architecture, key management operations and authentication.

Keywords

IoT, Identity-based cryptography (IBC), user data security, security framework
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Scope</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>References</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>Definitions</td>
<td>5</td>
</tr>
<tr>
<td>3.1</td>
<td>Terms defined elsewhere</td>
<td>5</td>
</tr>
<tr>
<td>3.2</td>
<td>Terms defined in this Recommendation</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Abbreviations and acronyms</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>Conventions</td>
<td>8</td>
</tr>
<tr>
<td>6</td>
<td>Overview</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>System reference architecture for IoT services over Telecom networks</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>Framework of using IBC for IoT services over Telecom networks</td>
<td>11</td>
</tr>
<tr>
<td>8.1</td>
<td>IoT system architecture with IBC</td>
<td>11</td>
</tr>
<tr>
<td>8.2</td>
<td>Key management architecture</td>
<td>14</td>
</tr>
<tr>
<td>8.3</td>
<td>Identity naming</td>
<td>16</td>
</tr>
<tr>
<td>8.4</td>
<td>Key management</td>
<td>16</td>
</tr>
<tr>
<td>8.5</td>
<td>Authentication</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>Security requirements</td>
<td>18</td>
</tr>
<tr>
<td>Annex A</td>
<td>IBC generic formulation and supported IBC algorithms</td>
<td>20</td>
</tr>
<tr>
<td>Annex B</td>
<td>IBC key data definition</td>
<td>23</td>
</tr>
<tr>
<td>Annex C</td>
<td>Key management operations</td>
<td>36</td>
</tr>
<tr>
<td>C.1</td>
<td>System initialization</td>
<td>36</td>
</tr>
<tr>
<td>C.2</td>
<td>Device initialization</td>
<td>45</td>
</tr>
<tr>
<td>C.3</td>
<td>Public parameter lookup</td>
<td>47</td>
</tr>
<tr>
<td>C.4</td>
<td>Identity/key provisioning</td>
<td>47</td>
</tr>
<tr>
<td>C.5</td>
<td>Identity/key revocation</td>
<td>52</td>
</tr>
<tr>
<td>Annex D</td>
<td>Authentication</td>
<td>59</td>
</tr>
<tr>
<td>D.1</td>
<td>One-pass secret transport protocol</td>
<td>59</td>
</tr>
<tr>
<td>D.2</td>
<td>TLS-IBS</td>
<td>60</td>
</tr>
<tr>
<td>D.3</td>
<td>EAP-TLS-IBS</td>
<td>64</td>
</tr>
</tbody>
</table>
D.4 EAP-PSK-ECCSI.................................................................................................................. 66
D.4.1 Attach............................................................................................................................. 68
D.4.2 EAP-PSK--ECCSI First Message (Message 3 in Figure D.4-1)................................. 68
D.4.3 EAP-PSK--ECCSI Second Message (Message 5 in Figure D.4-1)............................ 70
D.4.4 EAP-PSK--ECCSI Third Message (Message 10 in Figure D.4-1).............................. 71
D.4.5 EAP-PSK--ECCSI Fourth Message (Message 12 in Figure D.4-1)......................... 71
**Introduction**

The traditional certificate-based solution involves heavyweight key management operations including certificate issuing, querying, and revocation. Such systems face great difficulty to keep up with the pace of device increment of IoT at the same time maintaining decent performance.

Identity-based cryptography (IBC) technology is another type of public-key technology which uses an entity's identity as a public key. As one of the essential features of IoT, everything has a unique identifier. Using such identifiers as public keys, no certificates are required. Consequently, IBC security solution utilizes simpler key management, enables distributed authorities controlling their own devices and scales well to both the high number of endpoints and the diversity of the devices.

This Recommendation provides security framework for use of IBC public key technology in support of IoT services over Telecom networks including mechanisms of identity management, key management architecture, key management operations and authentication.
Draft Recommendation ITU-T X.ibc-iot

Security framework for use of identity-based cryptography in support of IoT services over Telecom networks

1 Scope
This Recommendation provides security framework for use of IBC technology in support of IoT services over Telecom networks. This security framework includes mechanisms for device identifying, private key issuing, public parameter lookup, and authentication protocols etc.

2 References
None.

3 Definitions

3.1 Terms defined elsewhere

3.1.1 EAP [RFC 3758]: Extensible Authentication Protocol, a generic authentication protocol framework, different authentication method are supported with the defined framework.

3.2 Terms defined in this Recommendation

3.2.1 Identity Domain: A collection of entities which share the same set of public parameters and identity naming rules.

3.2.2 Identity Provider: An entity which creates, maintains, and manages identity information.

3.2.3 Identifier: A string that represents an identity.

3.2.4 Master Secret Key: The secret value used by the private key generator to compute private keys corresponding to an identifier.

3.2.5 Master Public Key: The public value uniquely determined by the corresponding master secret key.

3.2.6 Private Key Generator: An entity or function which generates a set of private keys.

3.2.7 Public Parameters: Parameters for cryptographic computation including a selection of a particular cryptographic scheme or function from a family of cryptographic schemes or functions, or from a family of mathematical spaces and the master public key.

3.2.8 Public Parameter Server: An entity that provides public parameters upon request.

3.2.9 Security Module: A piece of software or hardware or the composition of software and hardware which securely implements the cryptographic mechanisms and provides purported security services.

3.2.10 Prime Field F_p: A finite field of prime characteristic p.

3.2.11 Field Extension F_p^k: A field extension of field F_p with degree k.
### 4 Abbreviations and acronyms

This Recommendation uses the following abbreviations and acronyms:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AuC</td>
<td>Authentication Center</td>
</tr>
<tr>
<td>AGW</td>
<td>Aggregate Gateway</td>
</tr>
<tr>
<td>AN</td>
<td>Access Node</td>
</tr>
<tr>
<td>AS</td>
<td>Access System</td>
</tr>
<tr>
<td>AU</td>
<td>Authentication Unit</td>
</tr>
<tr>
<td>BN</td>
<td>Barreto-Naehrig curves</td>
</tr>
<tr>
<td>BLS-12</td>
<td>Barreto-Lynn-Scott curves with embedding degree 12</td>
</tr>
<tr>
<td>BLS-24</td>
<td>Barreto-Lynn-Scott curves with embedding degree 24</td>
</tr>
<tr>
<td>CA</td>
<td>Certificate Authority</td>
</tr>
<tr>
<td>CRL</td>
<td>Certificate Revocation List</td>
</tr>
<tr>
<td>EAP</td>
<td>Extensible Authentication Protocol</td>
</tr>
<tr>
<td>EID</td>
<td>eUICC-ID</td>
</tr>
<tr>
<td>EIS</td>
<td>eUICC Information Set</td>
</tr>
<tr>
<td>eUICC</td>
<td>Embedded UICC</td>
</tr>
<tr>
<td>EUM</td>
<td>eUICC Manufacture</td>
</tr>
<tr>
<td>GW</td>
<td>GateWay</td>
</tr>
<tr>
<td>HSM</td>
<td>Hardware Security Module</td>
</tr>
<tr>
<td>IBAKA</td>
<td>Identity-Based Authenticated Key Agreement</td>
</tr>
<tr>
<td>IBC</td>
<td>Identity-Based Cryptography</td>
</tr>
<tr>
<td>IBE</td>
<td>Identity-Based Encryption</td>
</tr>
<tr>
<td>IBS</td>
<td>Identity-Based Signature</td>
</tr>
<tr>
<td>ib.mpk</td>
<td>Identity Based master public key</td>
</tr>
</tbody>
</table>
ib.msk Identity Based master secret key

ib.pubparam Identity Based public parameters including identity based system parameters and master public key

ib.prk Identity Based private key

ib.sysparam Identity Based system parameters

ib.puk Identity-Based public key

IdP Identity Provider

IoT Internet of Things

ISP IoT Service Platform

IRL Identity Revocation List

ISD Issuer Security Domain

KDF Key Derivation Function

KMIP Key Management Interoperability Protocol

KMS Key Management Service

KSS-16 Kachisa-Schaefer-Scott curves with embedding degree 16

KSS-18 Kachisa-Schaefer-Scott curves with embedding degree 18

MNO Mobile Network Operator

OCSP Online Certificate Status Protocol

OISP Online Identity Status Protocol

PKC Public Key Cryptography

PKG Private Key Generator

PKI Public Key Infrastructure

PPS Public Parameter Server

RSF Revocation Server Function

SecM Security Module
SM-DP Subscription Manager Data Preparation
SM-SR Subscription Manager Secure Routing
TLS Transport Layer Security
TVP Time-variant parameter such as a random number or a timestamp
PROV.ID Provisional identity
PROV.CRED Provisional credential
IdP.ID Identity provider's key identity
IdP.PUK Identity provider's public key

5 Conventions
None.

6 Overview
The Internet of Things (IoT) "can be viewed as a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies (ICT)" [b-ITU-T Y.2060]. Security of the IoT is one of the foremost concerns due to the ubiquitous nature of the devices coupled with the increasing sensitivity of user data. [ITU-T Y.4100] describes in high level the common security requirements of the IoT including communication security, data management security, service provision security, mutual authentication and authorization, etc. [b-ITU-T X.1361] further analyses security threats and challenges in an IoT environment, and describes capabilities that could address and mitigate these threats and challenges. These necessary security capabilities includes

- "a secure communication capability for supporting secure, trusted and privacy protected communication";
- "a secure key management capability for supporting secure communications";
- "a secure data management capability for providing secure, trusted and privacy protected data management";
- "an authentication capability for authenticating devices";
- "an authorization (access control) capability for authorizing devices";
- "a capability to implement secure protocols based on lightweight cryptographic algorithms";

and many others.

IoT devices are characterized by the constraint of resources such as computation and communication capabilities. The nature of IoT devices brings new challenges to satisfy the security requirements in an IoT system. Particularly easy-deployment, lightweight management operations and distributed authority are among the key factors when considering security solutions for the IoT.
As shown in [ITU-T X.1361], authentication, access control, and data integrity and confidentiality are among the essential services required for securing the IoT. Both symmetric-key and public-key cryptography mechanics can be exploited for providing such services.

The symmetric-key based security solution is relatively simple. However, it does not fit well the peer-to-peer scenarios such as the machine-to-machine (M2M) applications in the IoT without an online service playing as a trust broker or without pre-sharing a secret pairwisely among devices. Cross-system secure communication is also complicated without exposing user secret to peer parties.

The traditional certificate-based public key cryptography solution involves heavyweight key management operations including certificate issuing, querying, distributing, verification, and revocation. Such systems face significant challenges to keep up with the pace of device increment of the IoT at the same time maintaining decent performance. The overhead of exchanging certificates in security protocols also causes problems, particularly in the NB-IoT networks which have narrow communication band and small packet data unit.

Identity-based cryptography technology is another type of public-key technology, which uses an entity's identity as a public key. As one of the essential features of the IoT, everything has a unique identifier. By using such identifier as public keys, no certificates are required. Consequently, the IBC security solution utilizes simpler key management, enables distributed authorities controlling each own devices and scales well to both the high number of endpoints and the diversity of the devices. As no certificates are transmitted, security protocols can be conducted more efficiently.

In an IBC system, there is a trusted party called Key Management Service (KMS) that is in charge of generating each entity's private key. Before providing the key generation service, the KMS starts a system initialization process by invoking an \texttt{IBSetup} function, which given a security parameter determines a set of system parameters and generates a master secret key and a master public key. Note that the KMS has the same function as the PKG. Thus, for the convenience of expression, we use KMS and PKG interchangeably in this document. In this Recommendation, the combination of system parameters and the master public key is referred to as the public parameters. The KMS keeps the master secret key strictly confidential and makes the public parameters public available. If necessary, the public parameters may be published by a dedicated service Public Parameter Server (PPS).

A typical IBC security system may use an arrange of identity-based cryptography mechanisms including Identity-Based Encryption (IBE), Identity-Based Signature (IBS) and Identity-Based Authenticated Key Agreement (IBAKA) to provide various security services including data confidentiality, entity authentication, secure channel establishment, etc. All these IBC algorithms can be deemed as the composition of two sets of functions. One is the key generation functions, which generate identity-based public/private key pairs. The private key generation function (\texttt{IBExtract}) generates a private key from an identifier, the master secret key, and the public parameters. The identity public key derivation function (\texttt{IBDerivate}) computes a public key from an identifier and the public parameters. The other set of functions like encryption/decryption (\texttt{IBEnc/IBDec}), signing/verification (\texttt{IBSign/IBVerify}), and authenticated session key establishment protocol, etc., use the generated key pairs to complete corresponding cryptographic operations.

IBC technology has been standardized by various standard development organization, including ISO, IETF, IEEE, CCSA etc. The list of relative standards developed by these organizations can be found in Annex A. OneM2M is also considering to use IBC technologies for IoT networks in Release 4. Security analysis for Release 4 can be found in TR 0008 [b-TR 0008].
This Recommendation describes the security framework for using the IBC technology to provide security capabilities for IoT services over Telecom networks. The framework covers the aspects of identity management, key management architecture, key management operations and authentication and key agreement protocols of using IBC.

7 System reference architecture for IoT services over Telecom networks

This section presents a general system reference architecture for IoT services over Telecom networks. Figure 7.1 shows a conceptual system reference architecture for IoT services. The system consists of three domains: IoT device, Access system, and IoT service platform.

![Conceptual system architecture for IoT services](image)

Figure 7.1: Conceptual system architecture for IoT services

IoT devices are responsible for data collecting or action performing. For most of IoT devices, they can establish a connection with Telecom system and communicate with IoT service platform. Nowadays, majority of the IoT devices connect to IoT service platform via the wireless link established with Telecom networks. For the access system, in this Recommendation, it refers to the Telecom networks. Usually it consist of two parts, access network and core network. Core network can be further divided into two parts, control plane and user plane, responsible for control signalling and data transmission respectively.

Telecom network, as a traditional wireless connection, has already been used for several generations. Historically, Telecom networks are designed to support mobile communications for human beings with seamless roaming features. In recent years, starting from 4G LTE networks, supporting of IoT devices are also considered in the design. For example, in 4G networks, LTE-M and NB-IoT technologies are developed to support IoT devices.

Majority of today’s Telecom systems consist of three components, i.e. the terminals, usually denoted as UE, an access network, denoted as AN, and core networks. Here, we assume that both the AN and core networks belong to the Access System, demonstrated in Figure 7.1. IoT services are usually outside the Telecom networks with some interfaces for data transmission and service management. To provide better support for IoT services, Telecom networks are including more and more IoT specific design in its system specification. The integration between Telecom networks and IoT services are tightened in recent years.

With the system specification developed for 5G networks, public key technologies are supported for IoT services, including network access authentication. As mentioned in the overview section, comparing to other public key technologies, IBC is simple in management and efficient in
transmission. Therefore, using IBC for IoT services over Telecom networks shall be specified as a complement standard to the existing specifications.

8 Framework of using IBC for IoT services over Telecom networks

In this section, a framework of using IBC public key technologies for IoT services in Telecom networks is provided. The framework contains a system architecture by including necessary network components required when using IBC technologies. Furthermore, a key management framework for IBC is specified as it is essential for system using IBC technology. Other critical issues such as key management, identity naming and authentication protocols etc. are also addressed in this framework.

8.1 IoT system architecture with IBC

For IoT services run over the Telecom networks, IBC can be either used for network access authentication or used for service access authentication, or used in both scenarios. Network access authentication addresses the issue of whether a device is allowed to access the network while service access authentication addresses the issue of whether a device can access IoT service platform or not.

IoT devices can access the Telecom network directly or indirectly. Hence, there are two access models:

- Direct-connected model: IoT devices connect to the access system directly.
- Undirect-connected model: IoT devices connect to the access system via an Aggregate Gateway.

The Figure 8.1-1 presents an IoT system reference architecture with IBC used for both Access System (AS) and IoT Service Platform (ISP) security protection. From the security point of view, both the AS and ISP may have their own security requirements on the IoT services. Considering the security credentials may be provided by either the AS or the ISP, there are three scenarios using IBC for IoT networks. The three scenarios are described as follows:

- Using IBC in AS security protection scenario.
  
  For this scenario, the security credentials for network access stored in IoT devices are provided and managed by the AS. The IoT device are authenticated by the AS when connecting to the AS. For example, the IoT device computes the IBS signature based on the private key provided by AS, and sends the signature to the Access System. Accordingly, AS could authenticate the IoT device based on the IBS signature provided in the authentication messages. And if the verification is success, AS sends the data from the IoT device to the IoT server thereafter.

- Using IBC for ISP security protection.
  
  Security credential stored in IoT devices are provided and managed by the ISP for service access. IoT devices are authenticated by the ISP based on the signature generated with IBC credentials.

- Using IBC in both AS and ISP security protection scenario.
  
  The IBC security credentials stored in IoT devices are provided and managed by either the AS or by the ISP, or can be managed by both jointly. The IoT device can be authenticated by both the AS and ISP with the same set of credentials.
The three scenarios described above cover most of the use cases with IBC for network and service access. However, there could be other scenarios. In this Recommendation, we only focus on the above three scenarios. Others may be covered in the future recommendations.

The IoT system architecture based on IBC consists of the following network functions (NF) and devices.

- **Access System (AS):** an access system for IoT devices or AGW, including the Access Node (AS.AN), Key Management System Function (AS.KMS), Authentication Unit (AS.AU), Revocation Server Function (AS.RSF), and GateWay (AS.GW).

- **IoT Service Platform (ISP):** a platform for IoT service management, including Key Management System Function (ISP.KMS), Authentication Unit (ISP.AU), Revocation Server Function (ISP.RSF), GateWay (ISP.GW), and IoT server. The ISP shall support the key management, distribution, the identity authentication, encryption/decryption, and signature signing/verification, etc.

- **Aggregate Gateway (AGW):** an aggregation node, responsible for IoT device connection, aggregating and sending all IoT devices’ data to the access system. The aggregate gateway plays the proxy role of data transmission between the IoT devices and the AN.

- **Access Node (AN):** access node for IoT devices or AGW, can be a wireless or fixed network access point.

- **Key Management System Function (KMS):** a management system, responsible for key generation, distribution and update of the IBC keys and parameters for IoT devices and network functions.

- **Authentication Unit (AU):** an authentication unit authenticates the IoT device based on IBC system.

- **Revocation Server Function (RSF):** a server maintains an identity revocation list. Public keys or Identities in the revocation list are excluded from usage.

Note: both AS and ISP may have their own KMS, AU and RSF.
• Access System GateWay (AS.GW): a network element connected to the IoT GW, responsible for IoT user data transmission.

• IoT Gateway (IoT GW): a gateway responsible for forwarding/aggregating data and transmitting data to IoT server, or forwarding data/signaling from IoT server to IoT devices.

• IoT server: a server locates in the IoT service provider side, collecting IoT data from the IoT GW.

• IoT device: an end device, used for data collecting and establish connection with access network and IoT server, providing data protection, including key negotiation, encryption/decryption, and signature signing/verification, etc.

The function of the reference points shown in the figure are described as follows:

- G1: reference point between IoT device and aggregate gateway, used for authentication and security communication.

- G2: reference point between AGW and AN, used for signalling and data communication between AGW and AN.

- T0: reference point between IoT devices, used for signalling and data exchange.

- T1: reference point between IoT devices and AN, used for authentication and security communication.

- T2: reference points between AS.GW and ISP.GW, providing a user plane data tunnel between AS.GW and ISP.

- T3: reference points between AS.AU and ISP.AU, for signalling exchange including the identity exchange or key provision.

- A1: reference points between AN and AS.GW, for the user plane data tunnelling.

- A2: reference points between AS.AU and AN, for the control plane signalling.

- A3: reference points between AS.AU and AS.GW, for the gateway allocation and management protocol in the access system.

- A4: reference points between AS.AU and AS.KMS, for the key provision protocol in the access system.

- A5: reference points between AS.AU and AS.RSF, for the identity/key revocation protocol in the access system.

- I1: reference points between IoT server and ISP.GW, for the user plane data tunnelling.

- I2: reference points between ISP.AU and ISP.GW, for the gateway allocation and management protocol in the IoT service platform.

- I3: reference points between ISP.AU and IoT server, for information exchange, such as service related subscription information transferred from IoT server to ISP.AU, authentication notification message from ISP.AU to the IoT server.
8.2 Key management architecture

This section describes the functional architecture required to support the key management when using IBC mechanisms in the IoT. Based on whether an IoT device has an eUICC component, two types of scenarios are considered, namely, the key management architecture of using IBC in IoT devices with eUICC and the key management architecture of using IBC in non-eUICC IoT devices.

In the case of using IBC in IoT devices with eUICC, the architecture follows the general architecture of eUICC remote provision defined in [b-eUICC] by adding two new function entities, i.e., KMS and PPS. Depending on the location of KMS, this case is further separated into two subcases.

Case 1. KMS is managed by the entity that is also in charge of MNO.

Case 2. KMS is managed by the entity that is in charge of SM-DP.

Figure 8.2-1 and 8.2-2 represent these two sub-cases respectively. In both subcases, the keys including the private key and public parameters are generated when MNO places an order of profile. The keys are then provided remotely to eUICC devices as those keys installed following the current remote key provision specification in [b-eUICC]. Details of the roles, the associated functions and interfaces of eUICC remote provision can be found in [b-eUICC]. The specification of the profile and the storage format and the usage of these keys in eUICC is out of the scope of this Recommendation.

Figure 8.2-1: IBC key management architecture A for IoT devices with eUICC
In the case of using IBC in non-eUICC IoT devices, a general architecture is depicted in Figure 8.2-3. The building blocks include

- **SecM.** A security module represents an element that can securely store keys and execute security mechanisms with the stored keys to complete security operations. An IoT device should have a security module.

- **IdP.** An identity provider is an entity that creates, maintains, and manages identity information.

- **AuC.** An authentication center offers entity authentication as a service.

The IdP relies on the authentication service provided by the AuC to authenticate IoT devices. After the initial authentication process, the IdP provides identity provision service to the SecM including identity creation, designation, replacement, and revocation. After a new identity is created and designated to an IoT device, the IdP invokes the private key generation service provided by the KMS to generate the private key corresponding to the newly allocated identifier and securely distribute the keys to the SecM. The IdP also extracts public parameters from the KMS and feed them into the PPS which publishes the public parameters to external entities. The IdP may also provide authentication service to other entities by executing with the SecM specific authentication protocols including those defined in this Recommendation.
8.3 Identity naming

When using IBC technology for IoT services over Telecom network, the identity naming is important as it can provide useful information to help operators to manage the network. Various information such as service type, location, device ID, valid time, etc. can be embedded in an identity. Part of the information is necessary when using IBC technology, e.g. the valid time. With the identity information, operator can optimize the network management, e.g. by allocating a connection to a specific network slices based on its service type. It is also easy to locate the device based on its location information. An example definition of the identity is presented in Appendix I.

8.4 Key management

Apart from the identity value, an IBC system involves three types of cryptographic key values: the master secret key, the public parameters, and the private key. The ASN.1 definition of these key structures is given in Annex B: IBC key data definition.

To manage these keys, the IBC system makes use of following five key management operations:

- System initialization operation
- Device initialization
- Public parameter lookup
- Identity and key provisioning
- Identity and key revocation

Key Management Interoperability Protocol [b-KMIP] can be utilized to exchange messages between the management entity and the KMS. However, necessary extension to KMIP to satisfy the new requirements of the IBSetup and IBExtract function shall be defined. For IoT devices with eUICC, standard procedures for remote key provision are used [b-eUICC]. For non-eUICC IoT devices, the protocols for interactions between the security modules and the management entities are defined based on HTTP. The specifications of these operations are defined in Annex C.
System initialization operation is to initialize an IBC system by generating the master secret key and public parameters. A management entity such as IdP or SM-DP or MNO is assumed in charge of IBC system initialization process. It establishes a secure channel with a KMS entity which implements the \textbf{IBSetup} function. The two parties execute the KMIP with the Create Key Pair operation. The management entity provides necessary information for the KMS to invoke \textbf{IBCSetup} function and generate the master secret key and public parameters. The KMIP is extended to support the setup functions including of various standardized IBC algorithms. Details of this operation are specified in Annex C.1.

Device initialization operation is to prepare an IoT device for identity and key provision. There are two cases: initialization of IoT devices with eUICC and initialization of non-eUICC IoT devices. For devices with eUICC, it is required that eUICC completes the Registration at SM-SR and so is ready for Profile download [b-eUICC]. There is no extra operation required for standard eUICC devices. For non-eUICC IoT devices, the security module should first register with AuC to get a provision ID (PROV.ID) and provision credential (PROV.CRED). This pair of PROV.ID/PROV.CRED is used for entity authentication in the identity/key provision process. If IoT devices cannot establish a secure channel with IdP using TLS, it is further required that a key identity IdP.ID and a related public key IdP.PUK belonging to IdP or the public parameter should be installed in SecM during the device initialization process. Details of the operation are specified in Annex C.2.

Public parameter lookup operation is to retrieve the IBC public parameters. An IoT device shall use the identity and key provision procedure to get the public parameters of the IBC system that it belongs to. It may follow the specification defined in Section 4 Public Parameter Lookup in [b-RFC 5408] to retrieve the public parameters of another IBC system from the known PPS. Details of the operation are specified in Annex C.3.

Identity and key provision operation includes the identity assignment, the private key extraction, and the key distribution procedure. IoT devices after the initialization process only have a provisional identity. The IdP or SM-DP or MNO shall determine which identity to be assigned to the requesting device and then communicate with the KMS to generate the corresponding private key and finally distribute the identity, the private key and the public parameters to the device securely. Details of the operation are specified in Annex C.4.

Identity and key revocation is used when stringent security policy requires that an identity should be revoked timely. If an identity is revoked, the identity shall be set in the revoked status. If an entry queries the status of a revoked identity, IdP or SM-DP or MNO shall return the correct value as defined in the Online Identity Status Protocol (OISP). To check a batch of identity statuses more efficiently, an entity can retrieve the Identity Revocation List (IRL) from IdP or SM-DP or MNO regularly and store it locally, and the entity can check with the fresh IRL to determine if an identity is revoked or not without querying status online for each identity. Details of the operation are specified in Annex C.5.

8.5 Authentication

Authentication is the process of determining whether an entity (a device or a user) has the right to access certain resources. In Telecom networks, there are two types of authentication related to IoT devices, which are network access authentication and service authentication. Network access authentication addresses the issue whether a device is allowed to access the network, while the service authentication addresses the issue whether a device can access IoT service platform or not.
Authentication protocols built based on IBC technologies are suitable for IoT authentications in Telecom Networks. This is due to the fact that IBC can greatly reduce the burden on identity and key management for massive number of IoT devices. Another advantage of IBC is that it enables distributed authentication, which not only greatly reduce the authentication time but also enables new application scenarios (such as device to device authentication, vehicle to vehicle authentication). For the current Telecom networks, such as 4G LTE networks, IBC can be used in the authentication between IoT devices and IoT service platforms. For the next generation cellular networks, i.e., the 5th generation cellular networks, IBC can be used for both network access authentication and services access authentication. In the current specification for 5G security, [b-TS 33.501], it defines a unified authentication framework that supports EAP authentication methods. In the annex of [b-TS 33.501], it further specifies how to use EAP-TLS in 5G for IoT networks.

The EAP authentication framework is open and supports many authentication protocols, including EAP-TLS. Both symmetric and asymmetric keys are supported by EAP authentication methods. As a relatively new public key technology, IBC is not well supported in existing authentication protocols. Therefore, in the Annex D of this Recommendation, four existing protocols are amended to support IBC in authentication:

- Annex D.2: TLS with raw public [b-RFC 8446] is amended to support IBC
- Annex D.3: EAP-TLS [b-RFC 5216] is amended to support IBC.
- Annex D.4: EAP-PSK [b-RFC 4764] is amended to support IBC.

9 Security requirements

This Recommendation only focuses on the security requirements of using IBC in IoT. General security threats and requirements for IoT are specified in [b-ITU-T X.1361]. As a cryptosystem, the paramount security concerns are of the integrity and authenticity of used public keys and secrecy of the used long-term and ephemeral secret keys. An IBC system involves following types of keys: the master secret key, public parameters, identifiers, private keys, and the ephemeral secrets used in the cryptographic operations.

- **Security requirements on the master secret key**

  All private keys are generated by the master secret key. In particular, if the master secret key is compromised, the adversary has the ability to recreate any entity's private key and therefore can decrypt all messages protected with the corresponding public key or impersonate any entity. Any illegal access to the master secret key would compromise the security of the IBC system. Hence, the master secret key shall be stored in a hardened environment such as a HSM. Any access to the key shall be authenticated with strong security mechanisms.

- **Security requirement on the public parameters**

  The public key is computed from the public parameters and an identifier with the IBDerivate operation. Hence using a false set of public parameters generated by an adversary to encrypt message or verify signature will lead to the compromise of the secrecy of the encrypted message or reach a false conclusion of the originator of a signature. Hence, the public parameters shall be transmitted through a secure channel or with a valid signature. An entity shall verify the peer entity
of the secure channel or verify the validity of the signature with regarding to a trusted public key before accepting the public parameters.

- **Security requirement on the identifier**

  In the IoT, each entity owns an identifier. If the same identifier is assigned to more than one entity and the corresponding private key is provisioned to each entity, this could result in leakage of sensitive information or impersonation attacks. Hence, each device shall be assigned a unique identifier.

- **Security requirement on the private key**

  The private key could be leaked if the security environment of an IoT device is compromised. Hence, the private key shall be distributed through a secure channel and stored in a secure environment.

- **Security requirement on the ephemeral secrets**

  The ephemeral secrets such as the random secret used in the encryption or signature processes could be leaked if the security environment of an IoT device is compromised. Hence, the randomness of the ephemeral secrets shall be guaranteed.
Annex A

IBC generic formulation and supported IBC algorithms

This Annex gives a generic formulation of IBC and provides a list of IBC algorithms that are supported in this Recommendation. For those algorithms that follow this generic formulation but are not listed in current Recommendation may be easily included in the future as extensions to the framework defined in this Recommendation. The generic formulation defined here also guides the descriptions of related key data structures, key management operations and authentication and key establishment protocols in the following Annexes.

An IBC cryptosystem involves following types of key data. The categorization of these keys follows ISO 18033-5 [b-ISO 18033-5].

- $ib.msk$: the master secret key is the secret value used by the KMS to compute identity-based private key. $ib.msk$ is generated during the system initialization process and is known by KMS only.

- $ib.mpk$: the master public key that is uniquely determined by the corresponding master secret key. $ib.mpk$ is computed by KMS during the system initialization process.

- $ib.sysparam$: the system parameters for cryptographic computation including a selection of a particular cryptographic scheme or function from a family of cryptographic schemes or functions, or from a family of mathematical spaces. $ib.sysparam$ is chosen by KMS during the system initialization process.

- $ib.pubparam$: the public parameters are the combination of the system parameters $ib.sysparam$ with the master public key $ib.mpk$. This type of key is defined to provide a unified view among ISO standard documents such as ISO 18033-5 and RFCs related to IBC such as RFC 5091 [RFC 5091].

- $ib.prk$: the identity-base private key, which is generated by KMS with $ib.msk$ and $ib.pubparam$, corresponding to an identifier $ID$.

- $ib.pub$: the identity-based public key, which is computed from an identifier $ID$ and $ib.pubparam$ through a function defined by an identity-based cryptographic scheme.

An IBC cryptosystem may include following functions which are specified with inputs and outputs:

**IBSetup**

Input: security parameter

Output: $ib.pubparam, ib.msk$

**IBExtract**

Input: $ib.pubparam, ib.msk, ID$

Output: $ib.prk$

**IBDerivate**

Input: $ib.pubparam, ID$
Output: `ib.puk`

**IBEnc**

Input: `ib.pubparam, ID, message M`

Output: ciphertext `C`

**IBDec**

Input: `ib.pubparam, ID, ib.prk, ciphertext C`

Output: plaintext `M` or error

**IBSign**

Input: `ib.pubparam, ID, ib.prk, message M`

Output: signature `S`

**IBVerify**

Input: `ib.pubparam, ID, message M, signature S`

Output: valid or invalid

This Recommendation shall support the use of following identity-based algorithms including

- BB1-KEM [b-RFC 5091]
- BF-IBE [b-RFC 5091]
- SK-KEM [b-RFC 6508]
- SM9-IBE [b-SM9]
- Cha-Cheon-IBS (IBS2) [b-ISO 14888-3]
- ECCSI [b-RFC 6507]
- Hess-IBS (IBS1) [b-ISO 14888-3]
- SM9-IBS (Chinese IBS) [b-ISO 14888-3]
- Fujioka-Suzuki-Ustaoglu-AKA [b-ISO 11770-3]
- Smart-Chen-Cheng-AKA [b-ISO 11770-3]
- SM9-AKA [b-SM9]
- Wang-AKA [b-IEEE P1363.3]

All these algorithms are based on the discrete logarithm assumption and typically are implemented on the point group over an elliptic curve. Many of these algorithms further make use of a "cryptographic pairing" over an elliptic curve [b-GPS08]. A cryptographic pairing $e$ is an efficiently computable bilinear map $e: G_1 \times G_2 \rightarrow G_3$, satisfying equation: $e([a]P_1, [b]P_2) = e(P_1, P_2)^{ab}$, where $P_1$ and $P_2$ are the generator of cyclic group $G_1$ and $G_2$ respectively. $[a]P_1$ denotes $a$ times group operations with $P_1$, similarly $[b]P_2$ is the group operation with $P_2$.
A cryptographic pairing can be instantiated by the Weil pairing, Tate pairing, optimal Ate pairing, etc., over pairing-friendly elliptic curves [b-FST10]. The commonly used pairing-friendly elliptic curves include supersingular elliptic curves, BN curves, BLS-12 curves, KSS-16 curves, KSS-18 curves, and BLS-24 curves[b-FST10]. All these curves $E$ are based on a prime field $\mathbb{F}_p$, where $p$ is a prime integer. $G_1$ is the point subgroup on curve $E$. $G_2$ is either same as $G_1$ if the supersingular curves are used or is a point subgroup on the twist curve $E'$. $E'$ is constructed from some extension field of the based field $\mathbb{F}_p$. $G_3$ is the extension field $\mathbb{F}_p^k$ of the based field $\mathbb{F}_p$, where $k$ is the embedding degree.

There are IBC algorithms that are constructed with other mathematical mechanisms such as lattices, e.g. [b-DLP14]. This type of algorithms is efficient regarding computation while having larger key and output size than those based on the discrete logarithm over elliptic curves. The algorithms are generally believed to be resistant to attacks running on quantum computers. However, algorithms of this category are still under development. Hence it seems premature to consider them for standardization at the current stage. This Recommendation may consider those lattice-based IBC algorithms in the future.
Annex B
IBC key data definition

Using the standard ASN.1 method, [b-RFC5408] has defined a generic structure for the system parameter including \textit{ib.pubparam} and other auxiliary information, and [b-RFC5091] has defined two sets of key data structures including \textit{ib.msk} and \textit{ib.prk} for two IBE algorithms, i.e., BF-IBE and BB1-IBE. While maintaining compatibility with the existing definitions, this Recommendation extends the system parameter definition and defines new key data structures to support more algorithms and various efficient implementations with different curves and pairings.

A generic system parameter structure is defined as

\begin{verbatim}
IBSysParams ::= SEQUENCE {
  version    INTEGER { v3(3) },
  domainName   IA5String,
  domainSerial   INTEGER,
  validity   ValidityPeriod,
  ibPublicParameters  IBPublicParameters,
  ibIdentityType   OBJECT IDENTIFIER,
  ibParamExtensions  IBParamExtensions OPTIONAL,
  signatureAlgorithm  AlgorithmIdentifier OPTIONAL,
  signature    BIT STRING OPTIONAL
}
\end{verbatim}

\text{IBSysParams} corresponds to the IBESysParams definition in [b-RFC5408], but version is changed to v3 (3) and two extra fields are added. \text{districtName} and \text{districtSerial} have been renamed as \text{domainName} and \text{domainSerial} respectively. The definition of IBPublicParameter has been modified from the OCTET STRING type to the newly defined IBParameterData type, which is a CHOICE determined by the value of pkgAlgorithm. This definition removes the unnecessary of double-encoding caused by the previous definition, namely, encoding publicParameterData as a SEQUENCE of, for example, BFPublicParameters and further encoding the result as an OCTET STRING. Except the two new fields, the meaning of other fields remains unchanged as in [b-RFC5408]. The meaning of two new fields are as follows.

\begin{itemize}
  \item Signature Algorithm defines the used signature algorithm to generate the signature value. This field is optional as the signature field is not mandatory.
  \item Signature field contains the digital signature computed upon the ASN.1 DER encoded result from field version to \textit{ibParamExtensions}. This field is encoded as BIT STRING and is optional.
\end{itemize}

If presented, the signature field is used to help an entity to check the authenticity of the system public parameters without resorting other methods. For example, if an IoT device doesn't have the capability to establish a TLS-based secure channel as required in [b-RFC5408] to retrieve the public parameters of another IBC system, it may query its PPS with HTTP. In this case, the serving PPS shall sign the requested public parameters with its private signing key. The IoT device can verify the signature to
check the authenticity of the response. If a PPS is publishing the public parameters of another IBC system to its serving entities, it is recommended to treat the signing message as an identity and use the IBExtract algorithm as the signature algorithm to generate the private key as the corresponding signature value. In this way, the IoT devices verify if the signature value is a valid private key corresponding to the ASN.1 DER encoded result from field version to `ibParamExtensions` and do not need an extra verification public key to verify the signature.

\[
\text{IBPublicParameters ::= SEQUENCE (1..MAX) OF IBPublicParameter}
\]

\[
\text{IBPublicParameter ::= SEQUENCE } \{ \\
\quad \text{pkgAlgorithm OBJECT IDENTIFIER,} \\
\quad \text{publicParameterData IBParameterData} \\
\}\n\]

The value of `publicParameterData` is defined by `pkgAlgorithm`. It can be one of following choices.

\[
\text{IBParameterData ::= CHOICE } \{ \\
\quad \text{bb1ParameterData BB1PublicParameters,} \\
\quad \text{bfParameterData BFPublicParameters,} \\
\quad \text{eccsiParameterData ECCSIPublicParameters,} \\
\quad \text{skParameterData SKPublicParameters,} \\
\quad \text{sm9ParameterData SM9PublicParameters} \\
\}\n\]

\[
\text{IBParamExtensions ::= SEQUENCE OF IBParamExtension}
\]

\[
\text{IBParamExtension ::= SEQUENCE } \{ \\
\quad \text{ibParamExtensionOID OBJECT IDENTIFIER,} \\
\quad \text{ibParamExtensionValue OCTET STRING} \\
\}\n\]

In [b-RFC 5091], two sets of master secret key, public parameters and private key block, i.e.,

- BB1MasterSecret, BB1PublicParameters, BB1PrivateKeyBlock,
BFMasterSecret, BFPublicParameters, BFPPrivateKeyBlock are defined for BF and BB1 key
generation function. These definitions only fit with implementations of the functions with
symmetric pairings on supersingular elliptic curves defined over prime fields. This
Recommendation defines new structures with version changed to v3 to support implementations
of these algorithms with asymmetric pairings. For symmetric pairings over supersingular elliptic
curves, the corresponding field in BB1 and BF key data structures remains unchanged as in [b-
RFC 5091]. Three more sets of key data structures are defined for ECCSI, SM9, and SK-KEM.
respectively.

BB1MasterSecret ::= SEQUENCE {
  version INTEGER { v3(3) },
  alpha INTEGER,
  beta INTEGER,
  gamma INTEGER
}

- For implementations with asymmetric pairings, alpha shall be s1, beta shall be s2 and gamma
  shall be s3 in Section 9.3 "The BB1 key encapsulation mechanism" in [b-ISO18033-5].

BB1PublicParameters ::= SEQUENCE {
  version INTEGER { v3(3) },
  curve OBJECT IDENTIFIER,
  hashfcn OBJECT IDENTIFIER,
  pairing PAIRING OPTIONAL,
  p INTEGER OPTIONAL,
  q INTEGER OPTIONAL,
  pointP FpPoint,
  pointQ FpxPoint OPTIONAL
  pointP1 FpPoint,
  pointP2 FpxPoint OPTIONAL,
  pointP3 FpPoint,
  v FpxElement
}

- pairing specifies which type of bilinear map shall be used with generated parameters. Three types
  of pairing are supported including Weill pairing, Tate pairing, and optimal Ate pairing.
- p and q become optional. For some types of curves such as BN, BLS-12, etc., p and q are pre-determined by curve OID and hence unnecessary to specify them again.

- pointP and pointQ, for implementation with asymmetric pairings, shall be Q1 in G1 and Q2 in G2 in Section 9.3 The BB1 key encapsulation mechanism in [b-ISO18033-5]. For symmetric pairings, pointP equals to pointQ, so pointQ is OPTIONAL.

- PointP1 and pointP3, for implementation with asymmetric pairings, shall be R and T in Section 9.3 “The BB1 key encapsulation mechanism” in [b-ISO 18033-5].

- pointP2, for implementation with asymmetric pairings such as the optimal Ate pairing over BN curves, takes value from an extension field of F_p. pointP2 is optional because if v is given, pointP2 is unnecessary for the BB1-KEM algorithm to carry through.

- v is the pairing result, which is an element from the extension field of F_p. For implementation with asymmetric pairings such as the optimal Ate pairing over BN curves, the extension field is F_p^k, where k is the embedding degree. In this case, v shall be J in Section 9.3 The BB1 key encapsulation mechanism in [b-ISO18033-5].

- The meaning of other fields remains unchanged as in [b-RFC 5091].

```plaintext
PAIRING ::= ENUMERATED{
   weil   (1)  --Weil pairing
   tate   (2)  --Tate pairing
   optimalAte (3)  --Optimal Ate pairing
}

FpPoint ::= SEQUENCE{
   x      INTEGER,
   y      INTEGER
}
```

FpPoint defines a point on an elliptic curve over a prime field. A point has two coordinates which are named as x-coordinate and y-coordinate. Both coordinates take big integer values.

```plaintext
FpxPoint ::= CHOICE{
   fpPoint   FpPoint,
   fp2Point  [2] EXPLICIT Fp2Point,
   fp3Point  [3] EXPLICIT Fp3Point,
   fp4Point  [4] EXPLICIT Fp4Point
}
```
- Fp2Point defines a point on an elliptic curve over a field $F_p^2$. Each coordinate of a point takes value from an element of $F_p^2$.

- Fp3Point defines a point on an elliptic curve over a field $F_p^3$. Each coordinate of a point takes value from an element of $F_p^3$.

- Fp4Point defines a point on an elliptic curve over a field $F_p^4$. Each coordinate of a point takes value from an element of $F_p^4$.

Fp2Point ::= SEQUENCE{
  x  Fp2Element,
  y  Fp2Element
}

- Fp2Point defines a point on an elliptic curve over a field $F_p^2$. A point has two coordinates which are named as x-coordinate and y-coordinate. Both coordinates take values from $F_p^2$.

Fp3Point ::= SEQUENCE{
  x  Fp3Element,
  y  Fp3Element
}

- Fp3Point defines a point on an elliptic curve over a field $F_p^3$. Both coordinates of a point take values from $F_p^3$.

Fp4Point ::= SEQUENCE{
  x  Fp4Element,
  y  Fp4Element
}

- Fp4Point defines a point on an elliptic curve over a field $F_p^4$. Both coordinates of a point take values from $F_p^4$.

Fp2Element ::= SEQUENCE{
  a  INTEGER,
  b  INTEGER
}

- Fp2Element defines an element from a field $F_p^2$, which is represented as $a+b\alpha$ where $\alpha$ is
Fp3Element ::= SEQUENCE {
   a       INTEGER,
   b       INTEGER,
   c       INTEGER
}

- Fp3Element defines an element from a field \( F_p^3 \), which is represented as \( a + b \cdot \beta + c \cdot \beta^2 \) where \( \beta \) is non-cubic root in \( F_p \).

Fp4Element ::= SEQUENCE {
   a       Fp2Element,
   b       Fp2Element
}

- Fp4Element defines an element from a field \( F_p^4 \), which is represented as tower of two elements from \( F_p^2 \).

FpxElement ::= CHOICE {
   fp2Elemt    Fp2Element,
      --for super singular elliptic curve implementation
   fp12Elemt   Fp12Element,
      --using \( F_p \rightarrow F_p^2 \rightarrow F_p^4 \rightarrow F_p^6 \rightarrow F_p^12 \) tower representation
   fp16Elemt   Fp16Element,
      --using \( F_p \rightarrow F_p^2 \rightarrow F_p^4 \rightarrow F_p^8 \rightarrow F_p^16 \) tower representation
   fp18Elemt   Fp18Element,
      --using \( F_p \rightarrow F_p^3 \rightarrow F_p^6 \rightarrow F_p^18 \) tower representation
   fp24Elemt   Fp24Element
      --using \( F_p \rightarrow F_p^2 \rightarrow F_p^6 \rightarrow F_p^12 \rightarrow F_p^24 \) tower representation
}

- FpxElement defines the tower representation of an element in G3. The pairing \( e \) maps two inputs from G1 and G2 respectively to an element in G3. For the commonly used pairing-friendly curves, elements in G3 are normally represented in a tower method. For different embedding degrees, there could be different tower representations. This Recommendation defines a
commonly used tower representation of elements in field with embedding degree 12, 16, 18 and 24.

Fp12Element ::= SEQUENCE {
  a  Fp6Element,
  b  Fp6Element
}
- Fp12Element defines an element of $F_{p^12}$ with a 2x3x2 tower representation and shall be used in implementation with BN curves or BLS-12 curves or BLS-24 curves.

Fp6Element ::= SEQUENCE {
  a  Fp2Element,
  b  Fp2Element,
  c  Fp2Element
}
- Fp6Element defines an element of $F_{p^6}$ with a 3x2 tower representation and shall be used in implementation with BN curves or BLS-12 curves or BLS-24 curves.

Fp16Element ::= SEQUENCE {
  a  Fp8Element,
  b  Fp8Element
}
- Fp16Element defines an element of $F_{p^{16}}$ with a 2x2x2x2 tower representation and shall be used in implementation with KSS-16 curves.

Fp8Element ::= SEQUENCE {
  a  Fp4Element,
  b  Fp4Element
}
- Fp8Element defines an element of $F_{p^{8}}$ with a 2x2x2 tower representation and shall be used in implementation with KSS-16 curves.

Fp18Element ::= SEQUENCE {

- Fp18Element defines an element of $F_p^{18}$ with a 3x2x3 tower representation and shall be used in implementation with KSS-18 curves.

Fp6bElement ::= SEQUENCE{
    a Fp3Element,
    b Fp3Element
}
- Fp6bElement defines an element of $F_p^{6}$ with a 2x3 tower representation and shall be used in implementation with KSS-18 curves.

Fp24Element ::= SEQUENCE{
    a Fp12Element,
    b Fp12Element
}
- Fp24Element defines an element of $F_p^{24}$ with a 2x2x3x2 tower representation and shall be used in implementation with BLS-24 curves.

BB1PrivateKeyBlock ::= SEQUENCE {
    version INTEGER { v3(3) },
    pointD0 FpxPoint,
    pointD1 FpxPoint
}
- The meaning of pointD0 and pointD1 remains unchanged as in [b-RFC5091], but they are taken from G2 if BB1-KEM is implemented with asymmetric pairings. In this case, pointD0 and pointD1 shall be dID0 and dID1 respectively in Section 9.3 The BB1 key encapsulation mechanism in [b-ISO18033-5].

BFMasterSecret ::= SEQUENCE {
    version INTEGER {v3(3) },

masterSecret INTEGER

} - The meaning of each field remain unchanged as in [b-RFC5091].

BFPublicParameters ::= SEQUENCE {
  version INTEGER { v3(3) },
  curve OBJECT IDENTIFIER,
  hashfcn OBJECT IDENTIFIER,
  pairing PAIRING OPTIONAL,
  p INTEGER OPTIONAL,
  q INTEGER OPTIONAL,
  pointP FpxPoint,
  pointPpub FpxPoint
} - The meaning of each field remain unchanged as in [b-RFC5091], but pointP and pointPpub is taken from G2 if BF-IBE is implemented with asymmetric pairings. In this case, pointP and pointPpub shall be Q and R respectively in Section 8.2 The BF mechanism in [b-ISO18033-5].

BFPrivateKeyBlock ::= SEQUENCE {
  version INTEGER { v3(3) },
  privateKey FpPoint
} - The meaning of each field remain unchanged as in [b-RFC5091]. For implementations with asymmetric pairings, privateKey shall be skID in in Section 8.2 The BF mechanism in [b-ISO18033-5].

ECCSIMasterSecret ::= SEQUENCE {
  version INTEGER {v3(3) },
  masterSecret INTEGER
} - masterSecret shall be KSAK in [b-RFC6507].

ECCSIPublicParameters ::= SEQUENCE {
  version INTEGER { v3(3) },
  curve OBJECT IDENTIFIER,
  hashfcn OBJECT IDENTIFIER,
  pairing PAIRING OPTIONAL,
  p INTEGER OPTIONAL,
  q INTEGER OPTIONAL,
  pointP FpxPoint,
  pointPpub FpxPoint
} - The meaning of each field remain unchanged as in [b-RFC5091], but pointP and pointPpub is taken from G2 if BF-IBE is implemented with asymmetric pairings. In this case, pointP and pointPpub shall be Q and R respectively in Section 8.2 The BF mechanism in [b-ISO18033-5].

BFPrivateKeyBlock ::= SEQUENCE {
  version INTEGER { v3(3) },
  privateKey FpPoint
} - The meaning of each field remain unchanged as in [b-RFC5091]. For implementations with asymmetric pairings, privateKey shall be skID in in Section 8.2 The BF mechanism in [b-ISO18033-5].

ECCSIMasterSecret ::= SEQUENCE {
  version INTEGER {v3(3) },
  masterSecret INTEGER
} - masterSecret shall be KSAK in [b-RFC6507].

ECCSIPublicParameters ::= SEQUENCE {
  version INTEGER { v3(3) },
  curve OBJECT IDENTIFIER,
  hashfcn OBJECT IDENTIFIER,
  pairing PAIRING OPTIONAL,
  p INTEGER OPTIONAL,
  q INTEGER OPTIONAL,
  pointP FpxPoint,
  pointPpub FpxPoint
} - The meaning of each field remain unchanged as in [b-RFC5091], but pointP and pointPpub is taken from G2 if BF-IBE is implemented with asymmetric pairings. In this case, pointP and pointPpub shall be Q and R respectively in Section 8.2 The BF mechanism in [b-ISO18033-5].
version INTEGER { v2(2) },
curve OBJECT IDENTIFIER,
hashfcn OBJECT IDENTIFIER,
pointP FpPoint,
pointPpub FpPoint
}
- pointP shall be G in [b-RFC6507].
- pointPpub shall be KPAK in [b-RFC6507].

ECCSIPrivateKeyBlock ::= SEQUENCE {
  version INTEGER { v2(2) },
  ssk INTEGER ,
pvt OCTET STRING
}
- ssk and pvt shall be SSK and PVT in [b-RFC6507] respectively.

SKMasterSecret ::= SEQUENCE {
  version INTEGER {v3(3) },
  masterSecret INTEGER
}
- masterSecret shall be z_T in [b-RFC 6508] and s in Section 9.2 The SK key encapsulation mechanism [b-ISO18033-5].

SKPublicParameters ::= SEQUENCE {
  version INTEGER { v3(3) },
curve OBJECT IDENTIFIER,
hashfcn OBJECT IDENTIFIER,
pairing PAIRING OPTIONAL,
p INTEGER OPTIONAL,
q INTEGER OPTIONAL,
pointP1 FpPoint,
pointP1pub FpPoint OPTIONAL,
pointP2 FpxPoint OPTIONAL,
pointP2pub FpxPoint OPTIONAL,
v FpxElement
}

- For implementations with symmetric pairings over supersingular curves, p and q are defined in [b-RFC 5091]. For implementations with asymmetric pairings, p and q are pre-determined by the used curve and become optional.

- pointP1 shall be P in [b-RFC 6508] and Q1 in G1 in Section 9.2 The SK key encapsulation mechanism [b-ISO18033-5].

- pointP1pub shall be Z_T in [b-RFC 6508] and R in Section 9.2 The SK key encapsulation mechanism [b-ISO18033-5]. pointP1pub may be unnecessary for other algorithms such as signature algorithms based on the SK key generation function, so it is optional.

- pointP2 shall be Q2 in G2 in Section 9.2 The SK key encapsulation mechanism [b-ISO18033-5] if the SK-KEM is implemented with asymmetric pairings. pointP2 is not necessary for SK-KEM to carry through, so it is optional.

- pointP2pub shall be \[i_b.msk\]Q2 is unnecessary for SK-KEM but may be necessary for other algorithms such as signature algorithms based on the SK key generation function, so it is optional.

SKPrivateKeyBlock ::= SEQUENCE {
    version INTEGER { v3(3) },
    privateKey FpxPoint
}
- privateKey shall be RSK in [b-RFC 6508] and skID in Section 9.2 The SK key encapsulation mechanism [b-ISO 14888-3a].

SM9MasterSecret ::= SEQUENCE {
    version INTEGER {v3(3) },
    masterSecret INTEGER
}
- masterSecret shall be \[i_b.msk\] which is U defined in Section 7.4 Chinese-IBS in [b-ISO 14888-3a].

SM9PublicParameters ::= SEQUENCE {
    version INTEGER { v3(3) },
curve OBJECT IDENTIFIER,
hashfcn OBJECT IDENTIFIER,
pairing PAIRING OPTIONAL,
p INTEGER OPTIONAL,
q INTEGER OPTIONAL,
pointP1 FpPoint,
pointP1pub FpPoint OPTIONAL,
pointP2 FpxPoint OPTIONAL,
pointP2pub FpxPoint OPTIONAL,
v FpxElement

- For implementations with symmetric pairing over supersingular curves, p and q are as defined in [b-RFC 5091]. For implementations with asymmetric pairings, p and q are pre-determined by the used curve.
- pointP1 shall be P in Section 7.4 Chinese-IBS in [b-ISO 14888-3a].
- pointP1pub is unnecessary of SM9-IBS but necessary for SM9-IBE and in this case pointP2pub shall be \[ib.msk\]P.
- pointP2 shall be Q in Section 7.4 Chinese-IBS in [b-ISO 14888-3a]. pointP2 is not necessary for SM9-IBE to carry through, so it is optional.
- pointP2pub shall be V in Section 7.4 Chinese-IBS in [b-ISO 14888-3a]. pointP2pub is not necessary for SM9-IBE to carry through, so it is optional.

SM9PrivateKeyBlock ::= SEQUENCE {
  version INTEGER { v3(3) },
  privateKey FpxPoint
}
- privateKey shall be X in in Section 7.4 Chinese-IBS in [b-ISO 14888-3a] for signature and shall be the \[ib.prvk\] in G1 for SM9-IBE and SM9-AKA.

The BFMasterSecret, BFPublicParameters, and BFPrivateKeyBlock definition shall be used for those algorithms using the SOK key generation such as BF-IBE, Cha-Cheon-IBS, Hess-IBS, Fujioka-Suzuki-Ustaoglu-AKA, Smart-Chen-Cheng-AKA, and Wang-AKA. The BB1MasterSecret, BB1PublicParameters, and BB1PrivateKeyBlock definition shall be used for those algorithms using the BB1 key generation such as BB1-KEM. The SKMasterSecret, SKPublicParameters, and SKPrivateKeyBlock shall be used for SK-KEM and possibly other algorithms based on the SK key
generation function. The SM9MasterSecret, SM9PublicParameters and SM9PrivateKeyBlock shall be used for SM9 algorithms including SM9-IBE, SM9-IBS, and SM9-AKA. The ECCSIMasterSecret, ECCSIPublicParameters, and ECCSIPrivateKeyBlock shall be used for ECCSI.

If the private key needs to be protected, the following EncryptedPrivateKeyInfo structure shall be used.

```
EncryptedPrivateKeyInfo ::= SEQUENCE {
    encryptionAlgorithm EncryptionAlgorithmIdentifier,
    encryptedData EncryptedData
}
```

```
EncryptionAlgorithmIdentifier ::= AlgorithmIdentifier

EncryptedData ::= OCTET STRING
```

```
AlgorithmIdentifier ::= SEQUENCE {
    algorithm OBJECT IDENTIFIER,
    parameters ANY DEFINED BY algorithm OPTIONAL
}
```
Annex C
Key management operations

In an IBC system, key management operations include system initialization, identity/private key provision, identity/private key revocation, and system parameter publication. System initialization involves a step to invoke the IBSetup function, and private key provision involves a step to invoke the IBExtract function. These operations require interactions between a management entity and the KMS. This Recommendation uses KMIP to exchange messages between these two parties. The necessary extension is specified to satisfy the new requirements of the supported IBSetup and IBExtract algorithms. The protocols for interactions between the security modules and the management entities are defined based on HTTP for non-eUICC IoT devices. For embedded UICC, [b-eUICC] standards are used and extended if necessary.

C.1 System initialization

In each IBC system, a system initialization process should be completed before providing key management services to its users. In the process, the KMS executes one or more IBSetup functions to generate one or more sets of ib.msk and ib.pubparam key pairs. The method to harden the security of the KMS is out of the scope of this Recommendation. As a good practice, ib.msk shall be generated and stored in an HSM. If possible, a distributed key generation scheme which applies a secret sharing scheme to split ib.msk and spread the secret shares and the private key generation function across several KMSs may be deployed. In this case, only if more than a threshold number of KMS’ functioning properly, a private key corresponding to an identifier can be generated correctly.

![Figure C.1 -1: System initialization with KMIP](image)

**Start conditions:**

Assuming that IdP/SM-DP/MNO plays the role of a system initiator that is in charge of the system initialization process. Before IdP/SM-DP/MNO can invoke the IBSetup function in a KMS, the following conditions shall be satisfied.
a. There is a secure channel established between IdP/SM-DP/MNO and KMS.

b. IdP/SM-DP/MNO has completed an authentication process with KMS, and the authenticated IdP/SM-DP/MNO is authorized to perform the **IBSetup** request.

**Procedure:**

1. IdP/SM-DP/MNO shall prepare the Request Payload and invoke the Create Key Pair operation to send encoded request message to KMS.

2. KMS shall check the validity of the request and that IdP/SM-DP/MNO is authorized to invoke this operation. If any of these conditions is not satisfied, then KMS shall return a response indicating a failure. Otherwise, KMS shall execute **IBSetup** with parameters specified within the request.

3. KMS shall return to IdP/SM-DP/MNO the execution response. If the operation is a success, then KMS shall at least return a Private Key Unique Identifier to `ib.msk` and a Public Key Unique Identifier to `ib.pubparam` respectively.

4. Optionally, if the Create Key Pair operation is successful, IdP/SM-DP/MNO may invoke the Get operation with the Public Key Unique Identifier obtained from the last response to retrieve the public parameters `ib.pubparam`.

5. KMS shall return the key value of the newly generated public parameters.

The Request Payload for Create Key Pair is composed as follow:

<table>
<thead>
<tr>
<th>Request Payload</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Object</strong></td>
</tr>
<tr>
<td>Private Key Template-Attribute</td>
</tr>
</tbody>
</table>

Private Key Template-Attribute shall include following attributes:

<table>
<thead>
<tr>
<th><strong>Object</strong></th>
<th><strong>REQUIRED</strong></th>
<th><strong>Encode</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic Algorithm</td>
<td>YES</td>
<td>Enumeration, see Table Cryptographic Algorithm (Key Generation).</td>
<td>Specifies the <strong>IBSetup</strong> function.</td>
</tr>
<tr>
<td>Cryptographic Length</td>
<td>NO</td>
<td>Integer</td>
<td>Specifies the bit length of the characteristic of the prime field on</td>
</tr>
</tbody>
</table>
which the elliptic curve is based.

<table>
<thead>
<tr>
<th>Cryptographic Usage Mask</th>
<th>YES</th>
<th>Integer</th>
<th>Specifies the usage of the <em>ib.msk</em> which shall be Sign for key generation. <strong>IBExtract</strong> essentially is a signing process.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cryptographic Domain Parameters</td>
<td>YES</td>
<td>Object</td>
<td>Specifies more parameters for choosing system parameters such as the used elliptic curve, etc.</td>
</tr>
<tr>
<td>Cryptographic Parameters</td>
<td>YES</td>
<td>Object</td>
<td>Specifies other functions such as a hash function which shall be used with the <strong>IBExtract</strong> functions.</td>
</tr>
</tbody>
</table>

Cryptographic Algorithm shall be one of following values:

**Table C.1-3**

<table>
<thead>
<tr>
<th>Cryptographic Algorithm (Key Generation)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>IBC-KGA-BB1</td>
</tr>
<tr>
<td>IBC-KGA-BF</td>
</tr>
<tr>
<td>IBC-KGA-ECCSI</td>
</tr>
<tr>
<td>IBC-KGA-SK</td>
</tr>
<tr>
<td>IBC-KGA-SM9</td>
</tr>
</tbody>
</table>

Cryptographic Length shall be a value equal or greater than 110.

Cryptographic Usage shall be set as 00000001 (Sign).

Cryptographic Domain Parameters shall include following attributes:

**Table C.1-4**
<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Encode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>QLength</td>
<td>NO</td>
<td>Integer</td>
<td>Specifies the bit length of the order of the group from which ib.msk is chosen.</td>
</tr>
<tr>
<td>Recommended Curve</td>
<td>YES</td>
<td>Enumeration, see Table Recommended Curve.</td>
<td>Specifies the used curve.</td>
</tr>
<tr>
<td>Pairing Type</td>
<td>NO</td>
<td>Enumeration, see Table Pairing Type</td>
<td>Specifies if a pairing is used in an identity-based algorithm.</td>
</tr>
<tr>
<td>Domain Name</td>
<td>NO</td>
<td>TEXT STRING</td>
<td>Specifies a unique name for the generated system parameters ib.pubparam.</td>
</tr>
<tr>
<td>Domain Serial</td>
<td>NO</td>
<td>INTEGER</td>
<td>Specifies a version number for the generated system parameters ib.pubparam.</td>
</tr>
</tbody>
</table>

Recommended Curve shall be one of the following values:

**Table C.1-5**

<table>
<thead>
<tr>
<th>Recommended Curve</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>IBC-CURVE-SS1</td>
<td>00000070</td>
</tr>
<tr>
<td>IBC-CURVE-SS2</td>
<td>00000071</td>
</tr>
<tr>
<td>IBC-CURVE-BN-254-1</td>
<td>00000072</td>
</tr>
<tr>
<td>IBC-CURVE-BN-256-1</td>
<td>00000073</td>
</tr>
<tr>
<td>IBC-CURVE-BN-256-2</td>
<td>00000074</td>
</tr>
<tr>
<td>IBC-CURVE-BN-382-1</td>
<td>00000077</td>
</tr>
</tbody>
</table>
**Table C.1-6**

<table>
<thead>
<tr>
<th>Pairing Type</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weil-Pairing</td>
<td>00000001</td>
</tr>
<tr>
<td>Tate-Pairing</td>
<td>00000002</td>
</tr>
<tr>
<td>Optimal-Ate-Pairing</td>
<td>00000003</td>
</tr>
</tbody>
</table>

**Table C.1-7**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Encode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hashing Algorithm</td>
<td>YES</td>
<td>Enumeration, see Table Cryptographic Algorithm (Hash)</td>
<td>Specifies the hash function which shall be used with the key generation function.</td>
</tr>
<tr>
<td>Private Key Group</td>
<td>NO</td>
<td>Enumeration, see Table Private Key Group</td>
<td>Specifies in which group the private key is generated if a pairing is used.</td>
</tr>
</tbody>
</table>

**Table C.1-8**

<p>| Cryptographic Algorithm (Hash) |</p>
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHA224</td>
<td>00000040</td>
</tr>
<tr>
<td>SHA256</td>
<td>00000041</td>
</tr>
<tr>
<td>SHA384</td>
<td>00000042</td>
</tr>
<tr>
<td>SHA512</td>
<td>00000043</td>
</tr>
<tr>
<td>SHA3-224</td>
<td>00000044</td>
</tr>
<tr>
<td>SHA3-256</td>
<td>00000045</td>
</tr>
<tr>
<td>SHA3-384</td>
<td>00000046</td>
</tr>
<tr>
<td>SHA3-512</td>
<td>00000047</td>
</tr>
<tr>
<td>SM3</td>
<td>00000048</td>
</tr>
</tbody>
</table>

Private Key Group shall be one of the following values:

**Table C.1-9**

<table>
<thead>
<tr>
<th>Private Key Group</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>IBC-PRK-GROUP1</td>
<td>00000001</td>
</tr>
<tr>
<td></td>
<td>IBC-PRK-GROUP2</td>
<td>00000002</td>
</tr>
<tr>
<td></td>
<td>IBC-PRK-TWOGROUPS</td>
<td>00000003</td>
</tr>
</tbody>
</table>

The Create Key Pair response payload is composed as follow:

**Table C.1-10**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Key Unique Identifier</td>
<td>YES</td>
<td>The Unique Identifier of the newly created Private Key object which can be used to access <em>ib.msk</em>. The identifier is encoded as Text String.</td>
</tr>
</tbody>
</table>
The Unique Identifier of the newly created Public Key object which can be used to access ib.pubparam. The identifier is encoded as Text String.

The Request Payload for Get operation is composed as follow:

**Table C.1-11**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Key Unique Identifier</td>
<td>YES</td>
<td>The Unique Identifier of the Public Key object which can be used to access ib.pubparam. The identifier is encoded as Text String.</td>
</tr>
</tbody>
</table>

The Get response payload is composed as follow:

**Table C.1-12**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object Type</td>
<td>YES</td>
<td>Type of object</td>
</tr>
<tr>
<td>Unique Identifier</td>
<td>YES</td>
<td>The Unique Identifier of the object</td>
</tr>
<tr>
<td>Public Key</td>
<td>YES</td>
<td>A public key structure encapsulating the data of the IBC public parameters ib.pubparam</td>
</tr>
</tbody>
</table>

Unique Identifier shall be same as the Public Key Unique Identifier sent in Get request payload. Object Type shall be 00000003 (Public Key).

The Key Block in the Public Key field is composed as follows:

**Table C.1-13**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Encode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Format Type</td>
<td>YES</td>
<td>Enumerate, see Table Key Format Type.</td>
<td>Specifies the format of the key value.</td>
</tr>
<tr>
<td>Key Compression</td>
<td>NO</td>
<td>Enumerate.</td>
<td>Specifies if the key value should be compressed.</td>
</tr>
<tr>
<td>Key Value</td>
<td>YES</td>
<td>Transparent Key structure for IBC public parameters.</td>
<td>A newly defined Transparent Key structure for IBC public key.</td>
</tr>
<tr>
<td>-----------</td>
<td>-----</td>
<td>-----------------------------------------------------</td>
<td>---------------------------------------------------------------</td>
</tr>
<tr>
<td>Cryptographic Algorithm</td>
<td>YES</td>
<td>Enumerate, see Table Cryptographic Algorithm (Key Generation).</td>
<td>Same as Create Key Pair request payload</td>
</tr>
</tbody>
</table>

Key Format Type shall be following value:

**Table C.1-14**

<table>
<thead>
<tr>
<th>Key Format Type</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transparent IBC Public Parameters</td>
<td>Value</td>
<td>00000016</td>
</tr>
</tbody>
</table>

Key Compression shall be either 00000001 (uncompressed) or 00000002 (compressed prime).

Key Value shall have following attributes:

**Table C.1-15**

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Encode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>NO</td>
<td>Big Integer</td>
<td>For curves based on a prime field, P is the characteristic (p) of the prime field.</td>
</tr>
<tr>
<td>Q</td>
<td>NO</td>
<td>Big Integer</td>
<td>Q is the order of the point subgroup (G1) in which the cryptographic operations are computed.</td>
</tr>
<tr>
<td>J</td>
<td>NO</td>
<td>Big Integer</td>
<td>J is the cofactor such that J*Q = X-1, where X is the order of point group of the specified curve.</td>
</tr>
<tr>
<td>P1</td>
<td>STRING</td>
<td>BYTE STRING</td>
<td>For pairing-based algorithms, P1 is the generator of group G1.</td>
</tr>
<tr>
<td>Component</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>-----------</td>
<td>------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td>P2</td>
<td>STRING</td>
<td>NO</td>
<td>BYTE STRING</td>
</tr>
<tr>
<td>sP1</td>
<td>STRING</td>
<td>NO</td>
<td>BYTE STRING</td>
</tr>
<tr>
<td>sP2</td>
<td>STRING</td>
<td>NO</td>
<td>BYTE STRING</td>
</tr>
<tr>
<td>sP3</td>
<td>STRING</td>
<td>NO</td>
<td>BYTE STRING</td>
</tr>
<tr>
<td>Public</td>
<td>Pairing STRING</td>
<td>NO</td>
<td>BYTE STRING</td>
</tr>
</tbody>
</table>
[s2]P2) for BB1-KEM, where s1, s2 are integer components of ib.msk.

The new Tag definitions are listed in the following table.

<table>
<thead>
<tr>
<th>Tag Object</th>
<th>Tag Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pairing Type</td>
<td>420100</td>
</tr>
<tr>
<td>Private Key Group</td>
<td>420101</td>
</tr>
<tr>
<td>Domain Name</td>
<td>420102</td>
</tr>
<tr>
<td>Domain Serial</td>
<td>420103</td>
</tr>
<tr>
<td>P1 STRING</td>
<td>420104</td>
</tr>
<tr>
<td>sP1 STRING</td>
<td>420105</td>
</tr>
<tr>
<td>P2 STRING</td>
<td>420106</td>
</tr>
<tr>
<td>sP2 STRING</td>
<td>420107</td>
</tr>
<tr>
<td>sP3 STRING</td>
<td>420108</td>
</tr>
<tr>
<td>Public Pairing STRING</td>
<td>420109</td>
</tr>
</tbody>
</table>

**End condition:** The KMS is initialized successfully and IdP/SM-DP/MNO has the Private Key Unique Identifier and Public Key Unique Identifier to access the generated master secret key *ib.msk* and the public parameters *ib.pubparam* respectively. IdP/SM-DP/MNO shall use the Private Key Unique Identifier to call the Sign operation to generate identity private keys and shall use the Public Key Unique Identifier to call the Get operation to retrieve the public parameters.

**C.2 Device initialization**

Device initialization operation is to prepare the device for identity and key provision. For eUICC and other non-eUICC IoT devices, different device initialization procedures are followed.

**Case 1: Initialization for eUICC**
For eUICC, the identity and corresponding private key \textit{ib.prk} and public parameters \textit{ib.sysparam} are downloaded in an ISD Profile. Hence, after device initialization process, eUICC should be ready for ISD Profile creation. Following [b-eUICC], the registration operation shall be completed. The following is the reiteration of Section 3.5.1 in [b-eUICC].

- **eUICC Registration at SM-SR**

  **Start condition:**
  
a. eUICCs are produced and a Provisioning Profile is loaded and active in the Provisioning operator’s network. They are tested and ready for shipment. Each eUICC has a corresponding EIS.

  **Procedure:**
  
  (1). The EUM sends a eUICC registration request to the selected SM-SR, containing the EIS.
  
  (2). The SM-SR stores the EIS in its database, with EID as the key parameter.
  
  (3). The SM-SR confirms the successful registration towards the EUM. The confirmation message includes the EID.

  **End condition:** The eUICC is registered at the SM-SR and ready for Profile download. It can now be shipped to the machine to machine Device manufacturer.

**Case 2: Initialization for non-eUICC IoT devices**

For non-eUICC IoT devices, the following registration operation shall be completed.

- **SecM Registration at AuC**

  **Start condition:**
  
a. The SecM is produced and the IoT device shall be able to communicate with IdP in the operator’s network.

  **Procedure:**
  
  (1). The SecM sends a SecM provision data acquisition request to the AuC.
  
  (2). The AuC generates a provision ID (PROV.ID) and a related authentication credential (PROV.CRED) for the requesting SecM.
  
  (3). The AuC sends PROV.ID and PROV.CRED to the SecM. In the same message, the AuC also sends a key identity IdP.ID and a related public key IdP.PUK or \textit{ib.sysparam} to the SecM if the SecM has no capability to carry out the TLS protocol.
  
  (4). The SecM securely stores PROV.ID and PROV.CRED, and store IdP.ID and IdP.PUK or \textit{ib.sysparam} if provided. The SecM shall protect IdP.ID and IdP.PUK or \textit{ib.sysparam} from authorized change

  **End condition:** The SecM is registered at the AuC and ready for identity and key provision.
C.3 Public parameter lookup

An entity shall use the identity/key provision procedure to get the public parameters for the IBC system with which the entity has registered. An entity, which can be an IoT device or a management entity of an IBC system, shall follow the specification defined in Section 4 Public Parameter Lookup in [b-RFC5408] to retrieve the public parameters of another IBC system from the known PPS. The IBESysParams in the response in [b-RFC 5408] shall be replaced with IBSysParams defined in this Recommendation. [b-RFC5408] assumes that the querying IoT device can establish a TLS based secure channel with the requested PPS. If such requirement cannot be satisfied, the signatureAlgorithm and signature field in IBSysParams shall exist and be valid. Once the IBSysParams has been retrieved, a proper signature verification process shall be followed, and only if the signature in IBSysParams is valid and the signature verifying public key is authentic and valid, the retrieved public parameters shall be accepted.

C.4 Identity/key provisioning

Identity and key provision include the identity assignment, the private key extraction, and the key distribution procedure. Devices after the initialization process only have a provisional identity. The IdP or SM-DP or MNO shall determine which identity to be assigned to the requesting device and then communicate with the KMS to generate the corresponding private key and finally distribute the identity, the private key and the public parameters to the device securely.

![Figure C.4-1: Private key generation with KMIP](image)

- Private key generation with KMIP

**Start conditions:**

Assuming that the IdP/SM-DP/MNO plays the role of generating private key \(ib.prk\). Before IdP/SM-DP/MNO can invoke the IBExtract function in a KMS, the following conditions shall be satisfied.

a. There is a secure channel established between IdP/SM-DP/MNO and KMS.

b. The IdP/SM-DP/MNO has completed an authentication process to KMS and the authenticated IdP/SM-DP/MNO is authorized to perform the IBExtract request.
Procedure:

(1). The IdP/SM-DP/MNO shall prepare the Request Payload and invoke Sign operation to send the encoded request message to the KMS.

(2). The KMS shall check the validity of the request and that the IdP/SM-DP/MNO is authorized to invoke this operation. If any of these conditions is not satisfied, then the KMS shall return a response indicating a failure. Otherwise, the KMS shall execute `IBExtract` with `ib.msk`, `ib.pubparam`, and parameters specified within the request.

(3). The KMS shall return to the IdP/SM-DP/MNO the execution response. If the operation is a success, then the KMS shall return the generated private key `ib.prk` in the form of `IBPrivateKeyBlock` which is a CHOICE defined with ASN.1 as follows:

```
IBPrivateKeyBlock ::= CHOICE{
  bb1PrivateKeyBlock  BB1PrivateKeyBlock,
  bfPrivateKeyBlock  BFPrivateKeyBlock,
  eccsiPrivateKeyBlock  ECCSIPrivateKeyBlock,
  skPrivateKeyBlock  SKPrivateKeyBlock,
  sm9PrivateKeyBlock  SM9PrivateKeyBlock
}
```

The Request Payload for Sign is composed as follow:

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unique Identifier</td>
<td>NO</td>
<td>The Unique Identifier of the Managed Cryptographic Object that is the <code>ib.msk</code> key to use for the <code>IBExtract</code> operation. If omitted, then the ID Placeholder value shall be used by the server as the Unique Identifier.</td>
</tr>
<tr>
<td>Cryptographic Parameters</td>
<td>NO</td>
<td>Cryptographic Parameters may specify the group of which the private key shall be generated.</td>
</tr>
<tr>
<td>Data</td>
<td>YES</td>
<td>Data specifies the identity value from which the private key shall be extracted.</td>
</tr>
</tbody>
</table>

Cryptographic Parameters shall include following attributes:
Table C.4-2

<table>
<thead>
<tr>
<th>Object</th>
<th>REQUIRED</th>
<th>Encode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Key Group</td>
<td>NO</td>
<td>Enumeration, see</td>
<td>Specify the group (G1 or G2) of which the private key shall be generated.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Table Private Key</td>
<td>Group.</td>
</tr>
</tbody>
</table>

**End condition:** The IdP/SM-DP/MNO retrieved the private key corresponding to the request identity.

- **Identity/key provision for eUICC**

**Start conditions:**

a. The eUICC is registered at the SM-SR and ready for Profile download.

b. An Un-personalised Profile has been created by the SM-DP based on the Profile Description provided by the MNO.

c. The MNO has a demand for a quantity of eUICC Profiles.

d. The Un-personalised Profile has been validated on the target eUICC type using the Un-personalised Profile verification procedure.

**Procedure:**

1. The MNO provides the profile ordering to a selected SM-DP. The details of profile ordering process is shown in Section 3.5.3 in [b-eUICC].

2. Personalized profile is created by the SM-DP using the data received from the MNO. In particular, SM-DP shall use the chosen IMSI as the identity to complete the Sign operation with the KMS as defined in Private Key Generation with KMIP to generate private key for the chosen IMSI. The generated private key and ibPublicParameters in IBSysParams shall be included as keys in the profile.

3. The target profile is provisioned on the eUICC from the MNO. The details of profile Download and Installation process can be seen in Section 3.5.4 in [b-eUICC].

4. The target profile of eUICC is enabled via SM-SR or via SM-DP and SM-SR. The specific steps of profile enabling can refer to the Section 3.5.6 or Section 3.5.7 in [b-eUICC].

**End condition:** The target Profile is enabled on the eUICC. The previously Enabled Profile is disabled. The EIS is up to date.

- **Identity/key provision for non-eUICC IoT devices.**

**Case 1:** SecM has the capability to establish a TLS session with IdP

**Start conditions:**
a. SecM has been registered at AuC.

Procedure:

1. The SecM establishes a TLS session with the IdP and it shall successfully verify the validity of the IdP's TLS certificate.

2. The SecM carries through a web authentication procedure with the IdP using PROV.ID and PROV.CRED.

3. The IdP chooses an identity assigned to the requesting device and completes the Sign operation with the KMS as defined in Private Key Generation with KMIP to generate the private key for the chosen identity.

4. The IdP sends the assigned identity, the generated private key, and the public parameters to the SecM through the TLS session.

5. The SecM stores the private key securely, and the public parameters shall be protected from unauthorized change.

End condition: The target key is provisioned on the SecM.

The SecM and IdP shall follow the protocol defined in Section 5 and the Private Key Request Protocol in [b-RFC5408] to complete the identity/key provision procedure. In the response, the IBPrivateKeyReply structure defined in [b-RFC 5408] shall be replaced by IBPrivateKeyReply.

IBPrivateKeyReply ::= SEQUENCE (1..MAX) OF IBPrivateKey

IBPrivateKey ::= SEQUENCE {
    pkgIdentity  IBIdentityInfo OPTIONAL,
    pkgAlgorithm  OBJECT IDENTIFIER,
    pkgKeyData  IBPrivateKeyBlock, --defined by pkgAlgorithm
    pkgOptions  SEQUENCE SIZE (1..MAX) OF PkgOption,
    ibSysParams  IBSysParams OPTIONAL
}

Case 2: SecM has no TLS implementation

Start conditions:

a. SecM has been registered at AuC.

Procedure:

1. The SecM generates the key encryption key (kek) and encodes the key provisioning request (KeyProvRequest). The request includes kek, the provision ID (PROV.ID) and credential (PROV.CRED) which are encrypted by using the IdP’s public key identified by IdP.ID. The encryption result is encoded as EncryptedMsg. It sends the encrypted request as the body of a HTTP POST request to the IdP.

2. The IdP decrypts the ciphertext by using the privacy key identified by IdP.ID in the request,
and checks the freshness of the timestamp, or the correctness of the counter or both. If the request doesn't pass these checks, the IdP shall return a response indicating a failure. The IdP further checks the correctness of PROV.ID and PROV.CRED with the AuC. If such check fails, the IdP shall return a response indicating a failure. The IdP chooses an identity assigned to the requesting device and completes the Sign operation with the KMS as defined in Private Key Generation with KMIP to generate private key for the chosen identity.

(3). The IdP encrypts the generated private key, and if necessary the identity and the public parameters with the key encryption key (kek) using the specified algorithm (keyProtAlg) conveyed in the request. The ciphertext is encoded as EncryptedMsg. The IdP sends the encrypted response as the body of the HTTP response to the SecM through.

(4). The SecM decrypts the response and gets the assigned identity, the private key, and the public parameters. It stores the private key securely, and the public parameters shall be protected from unauthorized change.

End condition: The target key is provisioned on the SecM.

IBKeyProvisionRequest ::= SEQUENCE{
  version            INTEGER { v1(1) },
  timer              TIMESTAMP OPTIONAL,
  counter            INTEGER     OPTIONAL,
  identity           OCTET STRING,
  credential         OCTET STRING,
  keyProtAlg         OBJECT IDENTIFIER,
  kek                OCTET STRING
}

IBKeyProvisionResponse ::= SEQUENCE (1..MAX) OF IBKeyProvisionData

IBKeyProvisionData ::= SEQUENCE{
  identity           OCTET STRING OPTIONAL,
  ibSysParams        IBSysParams OPTIONAL,
  ibPrivateKey       IBPrivateKeyBlock
}

EncryptedMsg ::= SEQUENCE {
  encryptionAlgorithm EncryptionAlgorithmIdentifier,
  encryptedData        EncryptedData
}
C.5 Identity/key revocation

If an identity must be disallowed in the IBC system for various reasons such as the owner of the
identity has unsubscribed the service, or the corresponding private key has been compromised, etc.,
the identity should be revoked, and corresponding private key may need to be destroyed for security
reasons. If an identity is revoked, the identity shall be set in the revoked status. If an entry queries the
status of a revoked identity, IdP/SM-DP/MNO shall return the correct value as defined in the Online
Identity Status Protocol. To check identity status more efficiently, an entity can retrieve the Identity
Revocation List from IdP/SM-DP/MNO regularly and store it locally, and the entity can check with
the fresh Identity Revocation List to determine if an identity is revoked or not without querying status
online for each identity. For eUICC, the process to destroy the private key is possible by first disabling
and then deleting the profile from the eUICC.

**Identity/Key Revocation for eUICC**

**Start conditions:**

a. The target Profile is enabled on the eUICC.

**Procedure:**

1. MNO starts the profile disabling via SM-DP process. The details of profile disabling process
   is shown in Section 3.5.8 in [b-eUICC]. SM-DP shall set the identity in the revoked status.

2. MNO starts the profile deletion process. The specific steps of ISD-P Deletion can refer to
   the Section 3.5.10 in [b-eUICC]. SM-DP shall set the identity in the revoked, and if the ISD-
   P Deletion process is successful the identity is also set in the deleted status. When an entity
   queries the status of an identity, SM-DP shall respond properly according to the status record.
   Periodically, SM-DP shall publish the Identity Status List for those revoked identity during the
   period.

**End condition:** The target Profile is disabled and deleted from the eUICC.

**Identity/key revocation for non-eUICC IoT devices.**

If an identity is revoked, IdP shall set the identity in the revoked status. When an entity queries the
status of an identity, IdP shall respond properly according to the status record. Periodically, IdP shall
publish the Identity Status List for those revoked identity during the period.

How to trigger revocation process and how to keep the status of each identity is out of the scope of
this Recommendation.

**Online identity status protocol(OISP)**

With a large number of IoT devices connecting to a Telecom operator, it may be necessary for
SM-DP or IdP or an IoT device to obtain timely information regarding the revocation status of
IoT devices' identity. In this Recommendation, an Online Identity Status Protocol (OISP) is
specified to enable the SM-DP or IdP or an IoT device to determine the current status of an
identity with online queries. An OISP client issues a status request to an OISP responder and
suspends acceptance of the identity in question until the responder responds. OISP shares
similarity with Online Certificate Status Protocol (OCSP)[b-RFC 6960].
An OISP request contains the following data:

OISPRequest ::= SEQUENCE {
    version INTEGER { v1(1) },
    identity IBIdentityInfoSet
}
- version indicates the version of the protocol, which for this document is v1(1).
- identity is the OISP request.

IBIdentityInfoSet ::= SEQUENCE (1..MAX) OF IBIdentityInfo

IBIdentityInfo ::= SEQUENCE {
    domainName IA5String OPTIONAL,
    domainSerial INTEGER OPTIONAL,
    identityType OBJECT IDENTIFIER OPTIONAL,
    identityData OCTET STRING
}
- domainName is OPTIONAL and IA5String represents the URI [b-URI] or IRI [b-IRI].
- domainSerial is OPTIONAL and includes an INTEGER that defines a unique set of IBC public parameters in the event that more than one set of parameters is used by a single domain.
- identityType is OPTIONAL and contains an OBJECT IDENTIFIER that defines the format that the identityData field is encoded with. If this field is missing, default identity type is used.
- identityData is the data of the target identity.

Upon receipt of a request, an OISP responder checks if the message is well formed and the request contains the information needed by the responder. If such check fails, the OISP responder produces an error message; otherwise, it returns a definitive response according to the status of queried identities in the request.

OISPResponse ::= SEQUENCE {
    responseStatus OISPResponseStatus,
    responseData OISPResponseData OPTIONAL
}
- responseStatus indicates the processing status of the prior request.
- responseData is OPTIONAL and includes the response data of the request. If the value of responseStatus is one of the error conditions, the responseData field is not set.

OISPResponseStatus ::= ENUMERATED {
    successful          (0),   -- Response has valid confirmations
    malformedRequest   (1),   -- Illegal confirmation request
    internalError        (2),   -- Internal error in issuer
    tryLater             (3),   -- Try again later
    -- (4) is not used
    unauthorized        (5)   -- Request unauthorized
}

OISPResponseData ::= SEQUENCE {
    version INTEGER { v1(1) },
    producedAt GeneralizedTime,
    hashAlgorithm AlgorithmIdentifier OPTIONAL,
    tbsIdStatus SEQUENCE OF SingleIdStatus,
    signatureAlgorithm AlgorithmIdentifier OPTIONAL,
    signature BIT STRING OPTIONAL,
    certs [0] EXPLICIT SEQUENCE OF Certificate OPTIONAL
}

- version MUST be v1(1) for this version of the basic response syntax.
- producedAt is the time at which the OISP responder signed this response.
- hashAlgorithm defines a hash algorithm to generate idHash in tbsIdStatus if such field exists. The field is optional and default value is OBJECT IDENTIFIER for SHA256 without parameters.
- tbsIdStatus indicates the responses for each of the identity in a request.
- signatureAlgorithm is OPTIONAL and includes the algorithm that was used to sign the response.
- signature is computed upon the ASN.1 DER encode result from field producedAt to tbsIdStatus with the specified signature algorithm. This field is OPTIONAL and may not be set if the OISP client has other method to guarantee the authenticity of the response. For example, the response is transmitted through a TLS secure channel between the client and responder.
- certs is OPTIONAL and indicates the certificate that helps the OISP client verify the responder’s signature.
SingleIdStatus ::= SEQUENCE {
  idHash OCTET STRING OPTIONAL, identityID IBIdentityInfo OPTIONAL, identityStatus IdentityStatus,
}

- idHash is OPTIONAL and includes the hash of the request identity. If identityID is too long, idHash can be used to represent the queried identity. identityID is OPTIONAL and contains the target identity’s IBIdentityInfo field in the request.
- identityStatus indicates the status of the identity in the prior request.

IdentityStatus ::= CHOICE {
  good [0] IMPLICIT NULL,
  revoked [1] IMPLICIT RevokedInfo,
  unknown [2] IMPLICIT UnknownInfo,
  updated [3] IMPLICIT IBIdentityInfo,
  revokedAndDeleted [4] IMPLICIT RevokedInfo
}

- The “good” state indicates a positive response to the status inquiry.
- The “revoked” state indicates that the identity has been revoked, either temporarily or permanently and value is the revocation information.
- The “unknown” state indicates that the responder doesn’t know about the certificate being requested.
- The “updated” state indicates that the identity has been updated and the value is newly assigned identity for the queried identity.
- revokedAndDeleted state indicates the identity has been revoked and the private key has been destroyed from the remote device.

RevokedInfo ::= SEQUENCE {
  revocationTime GeneralizedTime,
  revocationReason [0] EXPLICIT IRLReason OPTIONAL
}

IRLReason ::= ENUMERATED {
  unspecified (0),
  keyCompromise (1),
  pkgCompromise (2),
  affiliationChanged (3),

superseded (4),
cessationOfOperation (5),
identityHold (6),
-- value 7 is not used
removeFromIRL (8),
privilegeWithdrawn (9)

● **Identity revocation list**

Apart from using OSIP to respond the identity status queries, an entity such as IdP or SM-DP may publish a complete list of the revoked identities in a regular period. This list is called Identity Revocation List. To speed up the identity status checking process, a status checking entity with large storage may query the IRL and store it locally. The checking entity can determine if an identity is acceptable for certain operations such as network access authorization based on the IRL. If such identity is not in the IRL, then it is assumed that the identity is valid. To improve system efficiency, IdP/SM-DP/MNO may just publish newly revoked identities since a specific time. It is called a delta IRL. A delta IRL contains the information of identities revoked since a complete IRL publication. The use of delta IRLs can significantly reduce the communication overhead and the processing time of IRLs. IRL shares similarity with Certificate Revocation List (CRL)[b-RFC5280].

The identity revocation list is defined as

\[
\text{IdentityRevocationList ::= SEQUENCE } \{ \\
\quad \text{tbsIdentityList TBSIdentityRevocationList , } \\
\quad \text{signatureAlgorithm AlgorithmIdentifier OPTIONAL, } \\
\quad \text{signatureValue BIT STRING OPTIONAL} \\
\}\n\]

- \text{tbsIdentityList} is the list of revoked identities with extra information such as the revocation time.
- \text{signatureAlgorithm} defines the algorithm the issuer of IRL used to sign the list. This field is optional and is not present if \text{signatureValue} does not exist.
- \text{signatureValue} defines the value of signature generated by issuer on \text{tbsIdentityList}. This field is optional and not present if the request client has other means to guarantee that the retrieved list is authentic.

\[
\text{TBSIdentityRevocationList ::= SEQUENCE } \{ \\
\quad \text{version INTEGER } \{ \text{v1(1)} \}, \\
\quad \text{issuer Name, } \\
\}\n\]
irlNumber INTEGER OPTIONAL,
deltaList BOOL OPTIONAL,
thisUpdate Time,
nextUpdate Time OPTIONAL,
domainName IA5String OPTIONAL,
domainSerial INTEGER OPTIONAL,
revokedIdentities SEQUENCE OF SEQUENCE {
    identity IBIdentityInfo,
    revocationDate Time,
    irlEntryExtensions Extensions OPTIONAL
        -- if present, version MUST be v2
} OPTIONAL,
irlExtensions [0] EXPLICIT Extensions OPTIONAL
        -- if present, version MUST be v2
}

- version indicates the version of Identity Revocation List structure.
- issuer is the name of the entity that issues the IRL.
- irlNumber is the issuer number of current IRL. It starts from 0. For each complete IRL publication, the number increases by 1. It is optional.
- deltaList indicates if the current IRL is a delta IRL. The list contains only the information of identities revoked since a complete IRL publication indexed by irlNumber.
- thisUpdate specifies the time of generating this IRL.
- nextUpdate defines the time to generate next IRL. It is optional.
- domainName defines the IBC identity domain.
- domainSerial defines the IBC identity domain number.
- revokedIdentities is the revoked identity set.
    - identity is the data of revoked identity.
    - revocationDate is the time when the identity is revoked.
    - irlEntryExtensions defines possible extensions of revokedIdentity. Current no extension is defined.
- irlExtensions defines possible extensions for IRL. Currently no extension is defined.
Annex D
Authentication

In this Annex, four existing authentication protocols are extended to support IBC.

D.1 One-pass secret transport protocol

This protocol corresponds to the Secret key transport mechanism 2 in [b-ISO 11770-3]. It transfers a secret key, generated and encrypted and signed by entity A, from entity A to entity B with explicit key authentication from entity A to entity B and implicit key authentication from entity B to entity A. The explicit key authentication from entity A to entity B is achieved by entity A signing the encrypted secret and a Time-Variant Parameter (TVP). The implicit key authentication from entity B to entity A is achieved by encrypting the secret with B's identifier which implies only B can recover the secret.

To conduct the protocol, the following requirements shall be satisfied:

- Entity A has a signature private key $A_{ib}.prk$ corresponding to its identifier and related public parameters $A_{ib}.pubparam$.
- Entity B has a decryption private key $B_{ib}.prk$ corresponding to its identifier and related public parameters $B_{ib}.pubparam$.
- Entity A has access to an authenticated copy of entity B’s public parameter for encryption $B_{ib}.pubparam$ and B's identifier.
- Entity B has access to an authenticated copy of entity A’s public parameter for signature $A_{ib}.pubparam$ and A's identifier.
- The optional TVP shall be either a timestamp or sequence number. If timestamps are used, then entity A and B need to maintain synchronous clocks or use a Trusted Third Party Time Stamp.
- A and B may share the same public parameters, i.e., $A_{ib}.pubparam = B_{ib}.pubparam$.

\[\text{Figure C.1-1: One-pass secret transport protocol}\]

1. Entity A generates random secret $K$ of the required length.
2. Entity A generates $BE = \text{IBE}nc(B_{ib}.pubparam, ID_B, [ID_A]||K||Text1)$. Text1 can be empty, and $ID_A$ is optional if entity B has other means to get entity A's identifier.
3. Entity A generates $S = \text{IB}Sign(A_{ib}.pubparam, ID_A, A_{ib}.prk, [ID_B]||TVP||BE||Text2)$. Text2 can be empty, and $ID_B$ is optional if entity B knows the used identifier $ID_B$ for
encryption.


5. When the TVP is a timestamp, entity B checks the TVP is within the allowed time difference. Otherwise, entity B rejects the token.

6. If entity B can get ID_A by other means and TVP is a sequence number, entity B first check the sequence number is larger than the one maintained for entity B. Otherwise, entity B rejects the token.

7. If entity B can get ID_A by other means, entity B verifies the signature S in KTA by IBVerify(A.ib.pubparam, ID_A, [ID_B]||TVP||BE||Text2, S). If the signature is invalid, entity B rejects the token.


9. If entity B can only get ID_A after step 8, entity B checks the freshness of TVP if TVP is a sequence number. If TVP is not fresh, entity B rejects the token. Entity B further verifies the signature S. If the signature is invalid, entity B rejects the token.

10. If all the checks and verifications are passed, entity A and entity B use K to protect the following messages. Both entities can use Key Derivation Function (KDF) [b-IEEE P1363] to generate keys for encryption and message authentication.

NOTE 1: The protocol can be converted to a unilateral entity authentication protocol by removing BE from the message signed by entity A and KTA. This modification becomes the one-pass entity authentication scheme defined in [b-ISO 9798-3].

NOTE 2: The protocol can be converted to a bilateral entity authentication protocol by requiring that entity B returns K to entity A. Entity B is authenticated by entity A by showing it can recover K which requires it owns the private key B.ib.prk.

NOTE 3: The identity-based signcryption algorithms such as BLMQ signcryption algorithm [b-BLMQ05], Chen- Malone-Lee signcryption algorithm [b-CL05] may be used to improve efficiency.

D.2 TLS-IBS

In this section, another authentication protocol named TLS-IBS is specified. It is assumed that both the server side and the IoT device side are provisioned with identity-based credentials, which include an identity, a private key for signature and KMS public parameters (e.g., KMS Public Authentication Key (KPAK) as defined in RFC 6507 [b-RFC 6507] as a computing parameter). The KMS public parameter structure definitions for the supported algorithms can be found in Annex B.

The TLS-IBS is developed based on RFC 7250 [b-RFC 7250]. Traditionally, TLS client and server exchange public keys endorsed by PKIX certificates. It is considered complicate and may cause security weaknesses with the use of PKIX certificates. To simplify certificates exchange, using RAW public key in TLS has been specified in RFC 7250. That is, instead of transmitting a full certificate in the TLS messages, only public keys are exchanged between client and server. However, an out-of-band mechanism for public key and identity binding is assumed. For IoT networks, TLS with RAW public key is particularly attractive, but binding identities with public keys might be challenging. The cost to maintain a large table for identity and public key mapping at the server side incurs additional maintenance cost, e.g. devices have to pre-register to the server. To simplify the binding between the
public key and the entity presenting the public key, a better way could be using Identity-Based Cryptography (IBC), such as ECCSI public key specified in RFC 6507, for authentication. Different from X.509 certificates and raw public keys, a public key in IBC takes the form of the entity’s identity. This helps eliminate the necessity of binding between a public key and the entity presenting the public key.

When IBS is used as RAW public for TLS, signature and hash algorithms are negotiated during the handshake. The handshake between the TLS client and the server follows the procedures defined in RFC 7250 [b-RFC7250] and TLS 1.3 [b-RFC 8446], but with the support of the IBS algorithms as the signature schemes.

In the following, the TLS-IBS protocol developed based on RFC 7250 and TLS 1.3 with ECCSI [b-RFC 6507], IBS1 (Hess-IBS), IBS1 (Cha-Cheon-IBS) and SM9-IBS [b-ISO 14888-3] as the signature algorithms is specified as follows:

1. The IoT device sends ClientHello including extension key_share, signature_algorithms, server_certificate_type, and client_certificate_type to the server, indicating that it supports Raw Public Key and IBS algorithms.

2. The server sends ServerHello including extension key_share, server_certificate_type, client_certificate_type, Certificate, CertificateRequest, CertificateVerify and Finished to the IoT device, indicating Raw Public Key is supported, and includes its identity (ServerID) in the certificate part. A signature generated with the private key owned by the server is included in CertificateVerify message.

3. After verifying the identity and signature of the server, the IoT device sends its raw public key as Certificate, CertificateVerify, and Finished to the server. The IoT device includes its identity (ClientID) in the certificate area, which is the raw public key of client. A signature generated with the private key of client is included.

4. The remaining steps are the same as TLS 1.3 in RFC 8446.
D.2.1 ClientHello

The ClientHello message format is the same as that specified in TLS 1.3 [b-RFC 8446], but the values for signature algorithm need be extended for IBS.

The ClientHello message tells the server the types of certificate or raw public key supported by the client, and also the certificate types that the client expects to receive from the server. The ClientHello message includes desired IBS algorithms based on the order of client preference. In TLS 1.3, a data structure named SignatureScheme is defined for signature algorithms. To support IBS algorithm, it has to be extended as follows:

```c
enum {
    ...
    /* IBS signature algorithm */
    eccsi_sha256 (TBD),
    ibs1_sha256(TBD)
    ibs2_sha256(TBD)
    sm9_ibs_sm3(TBD)

    /* Reserved Code Points */
    private_use (0xFE00..0xFFFF),
    (0xFFFF)
} SignatureScheme;
```

D.2.2 ServerHello
The ServerHello message format is the same as that specified in TLS 1.3 [b-RFC 8446]. The SignatureScheme is extended in the same way as in Client_Hello.

D.2.3 Server certificate

For the server certificate, a certificate structure is defined as RawPublicKey in [b-RFC 7250]. As in [b-RFC7250], a data structure subjectPublicKeyInfo is used to specify the raw public key and its cryptographic algorithm. Within the subjectPublicKeyInfo structure, two fields, algorithm and parameters, are defined. The algorithm specifies the cryptographic algorithm used with raw public key, which is represented by an object Identifiers (OID); and the parameters field provides necessary parameters associated with the algorithm. The identity of the server should be in the subjectPublicKey part.

Note: The identity should follow the format defined in the Appendix.

subjectPublicKeyInfo ::= SEQUENCE {
  algorithm                  AlgorithmIdentifier,
  subjectPublicKey                                 BIT STRING
}

AlgorithmIdentifier ::= SEQUENCE {
  algorithm    OBJECT IDENTIFIER,
  parameters    ANY DEFINED BY algorithm OPTIONAL
}

When using an IBS algorithm, an identity is used as raw public key, which can be converted to an OCTET string. Therefore, the Certificate and subjectPublicKey structure can be reused without changes.

The algorithm field in AlgorithmIdentifier structure is the object identifier of the IBS algorithm used. Besides that, it is necessary to tell the peer the set of public parameters used by the signer. The information can be carried in the payload of the parameters field in AlgorithmIdentifier. Corresponding to the algorithms above, the public parameter structures are ECCSIPublicParameters, BFPublicParameters, BFPublicParameters and SM9PublicParameters respectively as defined in Annex B.

To support IBS algorithms over TLS protocol to generate message CertificateVerify, a data structure for signature value need to be defined.

- A data structure for ECCSI is defined as follows(based RFC 6507):
  ECCSI-Sig-Value ::= SEQUENCE {
    r INTEGER,
    s INTEGER,
    pvt OCTET STRING
  }
  where pvt (as PVT defined in RFC 6507) is encoded as 0x04 || x-coordinate of [v]G || y-coordinate of [v]G.

- A data structure for IBS1 is defined as follows:
IBS1-Sig-Value ::= SEQUENCE {
  r INTEGER,
  s ECPoint
}
ECPoint ::= OCTET STRING as defined in [b-RFC 5480]

- A data structure for IBS2 is defined as follows:
  IBS2-Sig-Value ::= SEQUENCE {
    r ECPoint,
    s ECPoint
  }

- A data structure for SM9-IBS is defined as follows:
  SM9-Sig-Value ::= SEQUENCE {
    r INTEGER,
    s ECPoint
  }

To use a signature algorithm with TLS, an OID for the signature algorithm needs to be provided. The following table shows the basic information needed for the IBS signature algorithms to be used for TLS.

<table>
<thead>
<tr>
<th>Key Type</th>
<th>Document</th>
<th>OID</th>
</tr>
</thead>
<tbody>
<tr>
<td>ISO/IEC 14888-3 ibs-1</td>
<td>ISO/IEC 14888-3: IBS-1 mechanism</td>
<td>1.0.14888.3.0.7</td>
</tr>
<tr>
<td>ISO/IEC 14888-3 ibs-2</td>
<td>ISO/IEC 14888-3: IBS-2 mechanism</td>
<td>1.0.14888.3.0.8</td>
</tr>
<tr>
<td>SM9-IBS</td>
<td>ISO/IEC 14888-3: Chinese IBS mechanism</td>
<td>1.2.156.10197.1.302.1</td>
</tr>
<tr>
<td>Elliptic Curve-Based Signatureless For Identity-based Encryption (ECCSI)</td>
<td>Section 5.2 in [b-RFC 6507]</td>
<td>1.3.6.1.5.5.7.6.29</td>
</tr>
</tbody>
</table>

**D.2.4 Client certificate**

To support IBS, the client certificate is extended in the same way as the server certificate.

**D.3 EAP-TLS-IBS**

In this section, EAP-TLS authentication protocol is extended to support IBS. Both the network side and the UE side are provisioned with Identity-based credentials, which include an identity, a private
key for signature and KMS Public parameters (e.g., KMS Public Authentication Key (KPAK) as defined in RFC 6507).

The EAP-TLS is modified as follows:

1. The same as EAP-TLS
2. After receiving the EAP-Response with UE’s identity, ID_UE.
3. AU send ID_UE to RSF for validation
4. RSF validate the ID_UE based on the revocation list stored.
5. RSF sends back the validation result to the AU
6. If ID_UE is valid, and then AU sends the EAP-TLS start message to UE.
7. Step 7-9 are the same as those described in aforementioned TLS-IBS.
8. Step 10 is EAP-Success.

Figure D.3-1: EAP-TLS-IBS

D.3.1 EAP-Request

The EAP-Request message format is the same as that specified in RFC5216.

D.3.2 EAP-Response

The EAP-Response message format is the same as that specified in RFC5216.
D.3.3 ClientHello

The ClientHello message format is the same as D.2.1.

D.3.4 ServerHello

The ServerHello message format is the same as D.2.2.

D.3.5 Server Certificate

The Server Certificate format is the same as D.2.3.

D.3.7 Client certificate

The Server Certificate format is the same as D.2.4.

D.4 EAP-PSK-ECCSI

In this section, EAP-PSK is extended to support one of the IBS algorithm, ECCSI, for authentication. Both UE and AU are provisioned with Identity-based credentials, which include an identity, a private key for signature (SSK), a Public Verification Token (PVT), and a KMS Public Authentication Key (KPAK) as defined in RFC 6507 [b-RFC 6507] as a computing parameter.

With the provisioned credentials, the UE and AU can derive symmetric keys based on static Diffie-Hellman by exchanging the identity information and the PVT, and then use the SSK owned by each entity. For example, UE can derive a key after it receives the identity of AU and its PVT, denoted as ID_AU and PVT_AU respectively, as follows:

\[ K_{UE} = [SSK_{UE}](KPAK+[hash(G \ || \ KPAK \ || \ ID_{AU} \ || \ PVT_{AU})]PVT_{AU}) \]

where G is a generation point on the elliptic curve used by the KMS to generate keys for UE and networks. It is provisioned to UE and AU by the KMS together with SSK, PVT, and KPAK etc. The use of hash function can follow the Annex A of RFC 6507 [b-RFC 6507].

Similarly, AU can derive also derive K_AU as follows after it receives the identity and PVT from UE as follows:

\[ K_{AU} = [SSK_{AU}](KPAK+[hash(G \ || \ KPAK \ || \ ID_{UE} \ || \ PVT_{UE})]PVT_{UE}) \]

It can be proved that K_{UE} actually equals to K_{AU}. 
With above properties, we can use EAP-PSK for mutual authentication as follows:

1. UE sends an attach request to the AU and indicates that EAP-PSK shall be used for mutual authentication.
2. AU verifies the authentication type and decides authentication method.
3. AU sends the first message of EAP-PSK to UE with identity field contains the ID_AU and PVT_AU, and also a random number RAND_S as required by EAP-PSK.
4. UE derives a symmetric key as \( K = [SSK_{UE}](KPAK+[\text{hash}(G \ || \ KPAK \ || \ ID_AU \ || \ PVT_AU)]PVT_AU) \). UE generates a random number RAND_P and further derives \( K' = KDF(K, RAND_P, RAND_S) \). UE derives Authentication Key AK and Key Derivation Key (KDK) based on RFC 4764 [b-RFC4764] for EAP-PSK.
5. UE sends the second massage of EAP-PSK to AU, which contains RAND_S, RAND_P, a MAC_P (MAC_P=CMAC-AES-128(AK, ID_P||ID_S||RAND_S||RAND_P)) for authentication, and identity field that consists of ID_UE and PVT_UE.
6. AU sends ID_UE to RSF for validation.
7. RSF validates the ID_UE according to its revocation list.
8. RSF sends back the validation results to the AU.
9. If the ID is valid, then AU derives a symmetric key as \( K = [SSK_{AU}](KPAK+[\text{hash}(G \ || \ KPAK \ || \ ID_AU \ || \ PVT_AU)]PVT_AU) \). AU further derives \( K' = KDF(K, RAND_P, RAND_S) \). AU derives Authentication Key AK and Key Derivation Key (KDK) based on RFC 4764 for EAP-PSK [b-RFC4764]. AU authenticates the UE based on the MAC_P received from the message. AU further derives session key base on RAND_P and KDK.
10. AU sends the third message of EAP-PSK to UE with a MAC_S (MAC_S=CMAC-AES-128(AK, ID_S||RAND_P)) for authentication and other fields required by EAP-PSK.
11. UE authenticates the AU with MAC_S received and derives session key with RAND_P and KDK derived previously.
12. UE sends the last message of EAP-PSK to AU to finish the EAP-PSK authentication procedure.
D.4.1 Attach

This message is for the purpose of imitating the authentication procedure.

D.4.2 EAP-PSK--ECCSI First Message (Message 3 in Figure D.4-1)

The first EAP-PSK--ECCSI message is sent by the server to the peer. The format is as follows.

The first EAP-PSK--ECCSI message consists of:

A 1-byte Flags field

A 16-byte random number: RAND_S

A variable length field that conveys the server's NAI: ID_S. The length of this field is deduced from the EAP length field. The length of this NAI must not exceed 966 bytes. This restriction aims at avoiding fragmentation issues.

Figure D.4-1 shows an example format of the first message of EAP-PSK.
Figure D.4-2: Format of EAP-PSK

To support IBC-based EAP-PSK authentication, the ID_S for the protocol of EAP-PSK is used to carry the ID_AU and PVT_AU. ID_S and PVT_AU is carried in the Tag, Length, and Vector (TLV) data structure, wherein the first octet carried a tag indicator, the second octet carried a length field, indicate the length of the field followed. The vector field carried the value.

Table D.4-1 shows the table that defines the TLV for ID and PVT used with the EAP-PSK.

Table D.4-1: TLV definition for Identity and PVT

<table>
<thead>
<tr>
<th></th>
<th>Tag</th>
<th>Length</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>1</td>
<td>Variable (&lt;= 255)</td>
<td>defined by service provider</td>
</tr>
<tr>
<td>PVT</td>
<td>2</td>
<td>65</td>
<td>Number in hexadecimal</td>
</tr>
</tbody>
</table>

Figure D.4.2-2 shows the format of EAP-PSK--ECCSImessage carrying the Identity and PVT within the ID_S field.
### D.4.3 EAP-PSK--ECCSI Second Message (Message 5 in Figure D.4-1)

The second EAP-PSK-ECCSI message is sent by the peer to the server. The format is as follows.

The second EAP-PSK-ECCSI message consists of:

- A 1-byte Flags field
- The 16-byte random number sent by the server in the first EAP-PSK--ECCSI message (RAND_S) that serves as a session identifier
- A 16-byte random number: RAND_P
- A 16-byte MAC: MAC_P
- A variable length field that conveys the peer's NAI: ID_P. The length of this field is deduced from the EAP length field. The length of this NAI must not exceed 966 bytes.

Similarly, the ID_S field of EAP-PSK is used to carry the ID_UE and PVT_UE field. Figure D.4-3 shows the format of the second EAP-PSK message.
D.4.4 EAP-PSK--ECCSI Third Message (Message 10 in Figure D.4-1)

The third EAP-PSK--ECCSI message is sent by the server to the peer. The format is the same as that presented in RFC 4764.

D.4.5 EAP-PSK--ECCSI Fourth Message (Message 12 in Figure D.4-1)

The third EAP-PSK-ECCSI message is sent by the peer to the server. The format is the same as that presented in RFC 4764.
Appendix I

Identity naming

(This appendix does not form an integral part of this Recommendation.)

The identifier in an IoT application can be the identifier of a terminal or the identifier of an IoT platform. The identifier is a name that serves the purpose of identification. Identifier is a handy representation of the object and allow to reference or address the object for example in database or in communication protocols. In order to fulfill this purpose, identifiers must be unique, or the identifier is unique in an independent system. For example, the postal code is unique in a country, the uniqueness of the identifier is given inside a certain scope. In addition, an identifier is not only for a single object, but also for a group of objects, which enables uniform management and operation for this group.

Object identifiers (OIDs) [b-ITU-T X.660, ITU-T-REC-X.Sup31] are jointly developed by ISO/IEC and ITU-T, and have many characteristics. An OID has a hierarchical tree structure, which can flexibly extend its layers and the length of the identifiers. An OID corresponds to a node in the "OID tree", which is able to identify anything (physical or virtual, devices or non-devices), and is able to connect them with global information and communication infrastructures. The root of the tree contains the following three arcs: 0 (ITU-T), 1 (ISO) and 2 (joint-iso-itu-t). Each node in the tree is represented by a series of integers separated by periods, corresponding to the path from the root through the series of ancestor nodes, to the node. Each level of registration authority ID should be allocated by the upper level registration authority. For example, the OID denoting China National IC Card Registration Center, 1.2.156.20005 is allocated by 1.2.156 (ISO.member.china), the OID of China National OID Registration Center.

A complete OID would be a combination of registration authority ID and entity ID, and these two constituent parts are separated by a period, as shown in Figure I-1. In the case that the company has registered an OID by the upper level registration authority, only the entity ID needs to be designed.

![Figure I-1 Structure of complete OID for objects](image)

For example, the entity ID shall have the following structure:

### Table I-1: Detailed information of the entity ID

<table>
<thead>
<tr>
<th>byte</th>
<th>constituent part</th>
<th>interpretation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>version &amp; reserved</td>
<td>4 bits for version of the entity ID, and 4 bits for reserved digits for the future</td>
</tr>
<tr>
<td>2</td>
<td>business</td>
<td>business type</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>3~11</td>
<td>expired time</td>
<td>invalid time of the identity, 5 bytes for issuing time in Unix time, and 4 bytes for validity period in seconds</td>
</tr>
<tr>
<td>12</td>
<td>type</td>
<td>value 0 for meaningless number, 1 for MAC, and 2 for IMSI</td>
</tr>
<tr>
<td>13</td>
<td>length (value l)</td>
<td>the size of the value part (in bytes), 6 for MAC and 8 for IMSI</td>
</tr>
<tr>
<td>14~13+l</td>
<td>value</td>
<td>individual identification number</td>
</tr>
</tbody>
</table>

The entity ID is 19 bytes long for using MAC as the individual identification number and 21 bytes long for using IMSI. An IMSI is usually presented as a 15 digit number or shorter, and the first digit is not zero except the Test network [b-ITU-T E.212]. Padding zeroes before the IMSI to 16 digit and using 4 bits for one digit, 8 bytes is enough for an IMSI.

The IoT platform maintains a list for addressing. When a terminal device register for the first time, the platform will add a row, which contains the identifier of the device, and the IP address of the device. By searching the identifier of a device in the list one can obtain the IP address corresponding to the device.

**Table I-2: An example for the list for addressing**

<table>
<thead>
<tr>
<th>Identifier</th>
<th>IP address</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.9e.4e25.10.1.5b3e408003c26700.1.6.38B1DBC3156F</td>
<td>180.168.174.129</td>
</tr>
</tbody>
</table>
**Bibliography**

[ITU-T Y.4100] Common requirements of the Internet of things


[RFC 6507] M. Groves, Elliptic Curve-Based Certificateless Signatures for Identity-Based Encryption (ECCSI)


<table>
<thead>
<tr>
<th>Reference</th>
<th>Description</th>
</tr>
</thead>
</table>