Draft ETSI GS QKD 014 V0.0.5 (2019-01)

Quantum Key Distribution (QKD);
Protocol and data format of REST-based key delivery API

Disclaimer

This DRAFT is a working document of ETSI ISG QKD. It is provided for information only and is still under development within ETSI ISG QKD. DRAFTS may be updated, deleted, replaced, or obsoleted by other documents at any time.

ETSI and its Members accept no liability for any further use / implementation of the present DRAFT.

Do not use as reference material.

Do not cite this document other than as "work in progress".

Approved and PUBLISHED deliverables shall be obtained via the ETSI Standards search page at: http://www.etsi.org/standards-search

This draft is being developed by the Quantum Key Distribution (QKD) ETSI Industry Specification Group (ISG) but it does not yet represent the considered views of all members who participated in this ISG.

It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/QKD-014KeyDeliv

Keywords
quantum cryptography, Quantum Key Distribution, API, protocol

ETSI

650 Route des Lucioles
F-06921 Sophia Antipolis Cedex - FRANCE

Tel.: +33 4 92 94 42 00 Fax: +33 4 93 65 47 16

Siret N° 348 623 562 00017 - NAF 742 C
Association à but non lucratif enregistrée à la
Sous-préfecture de Grasse (06) N° 7803/88

Important notice

If a published Group Specification results from this draft it will be made available for download from:
http://www.etsi.org/standards-search

The present document may be made available in electronic versions and/or in print. The content of any electronic and/or print versions of the present document shall not be modified without the prior written authorization of ETSI. In case of any existing or perceived difference in contents between such versions and/or in print, the only prevailing document is the print of the Portable Document Format (PDF) version kept on a specific network drive within ETSI Secretariat.

Users of the present document should be aware that the document may be subject to revision or change of status.
Information on the current status of this and other ETSI documents is available at
https://portal.etsi.org/TB/ETSIDeliverableStatus.aspx

If you find errors in the present document, please send your comment to one of the following services:
https://portal.etsi.org/People/CommiteeSupportStaff.aspx

Copyright Notification

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The information pertaining to these essential IPRs, if any, is publicly available for **ETSI members and non-members**, and can be found in ETSI SR 000 314: *"Intellectual Property Rights (IPRs); Essential, or potentially Essential, IPRs notified to ETSI in respect of ETSI standards"*, which is available from the ETSI Secretariat. Latest updates are available on the ETSI Web server (https://ipr.etsi.org).

Pursuant to the ETSI IPR Policy, no investigation, including IPR searches, has been carried out by ETSI. No guarantee can be given as to the existence of other IPRs not referenced in ETSI SR 000 314 (or the updates on the ETSI Web server) which are, or may be, or may become, essential to the present document.

Trademarks

The present document may include trademarks and/or tradenames which are asserted and/or registered by their owners. ETSI claims no ownership of these except for any which are indicated as being the property of ETSI, and conveys no right to use or reproduce any trademark and/or tradename. Mention of those trademarks in the present document does not constitute an endorsement by ETSI of products, services or organizations associated with those trademarks.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group Quantum Key Distribution (QKD).

Modal verbs terminology

In the present document "**shall**", "**shall not**", "**should**", "**should not**", "**may**", "**need not**", "**will**", "**will not**", "**can**" and "**cannot**" are to be interpreted as described in clause 3.2 of the ETSI Drafting Rules (Verbal forms for the expression of provisions).

"**must**" and "**must not**" are **NOT** allowed in ETSI deliverables except when used in direct citation.

Executive summary

The present document describes a communication protocol and data format for a quantum key distribution (QKD) network to supply cryptographic keys to an application. It is intended to allow interoperability of equipment from different vendors. A REST (REpresentational State Transfer) API is specified as a simple, scalable, widely deployed approach that is familiar to a large developer community. The REST-based API specifies the format of the URIs, the communication protocols (HTTPS), and the JSON (JavaScript Object Notation) data format encoding of posted parameters and responses, including key material.

Introduction

QKD networks deliver cryptographic keys to applications. In order to ensure the interoperability of QKD networks, QKD equipment, and applications from different vendors, a specification for a key delivery API from QKD networks to applications is important.

Another Group Specification ETSI GS QKD 004 [i.1] defines an object-based remote function call-style API between applications and QKD key management layer and provides key data streams with QoS functionalities for applications. On the other hand, the present document defines a simpler key delivery API, which is a REST-based API using the HTTPS protocol and data encoded in the JSON format to deliver block keys with key IDs to applications.

REST-based APIs are simple and easy for developers to understand and are popular in many application domains. They have a large developer community and many libraries, implementations, and guidance documents are available to the

community. REST-based APIs are lightweight and scale to the "Internet" level regarding both the number of nodes and the number of applications.

It is hoped that this REST-based API specification for key delivery can encourage new entrants/developers into the QKD market, to promote new applications of QKD, and to develop a business ecosystem for QKD.

1 Scope

The present document specifies a communication protocol and data format for a quantum key distribution (QKD) network to supply cryptographic keys to an application.

It is in the form of an API (Application Programming Interface) that allows application developers to make simple method calls to a QKD network and to be delivered key material. It is intended to allow interoperability of equipment from different vendors.

The QKD network can consist of a single link between a single QKD transmitter and a single QKD receiver, or it can be an extended network involving many such QKD links. The API defines a single interface for the delivery of key material to applications in both scenarios. It is beyond the scope of the present document to describe how a QKD network generates key material shared between distant nodes.

A REST (REpresentational State Transfer) API is specified as a simple, scalable, widely deployed approach that is familiar to a large developer community. The REST API specifies the format of the URIs, the communication protocols (HTTPS), and the JSON (JavaScript Object Notation) data format encoding of posted parameters and responses, including key material.

2 References

2.1 Normative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

Referenced documents which are not found to be publicly available in the expected location might be found at https://docbox.etsi.org/Reference.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are necessary for the application of the present document.

[1] IETF RFC 7230: "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing"

NOTE: RFC 7230, DOI 10.17487/RFC7230, June 2014, <https://www.rfc-editor.org/info/rfc7230>.

[2] IETF RFC 7231: "Hypertext Transfer Protocol (HTTP/1.1): Semantics and Content"

NOTE: RFC 7231, DOI 10.17487/RFC7231, June 2014, <https://www.rfc-editor.org/info/rfc7231>.

[3] IETF RFC 7235: "Hypertext Transfer Protocol (HTTP/1.1): Authentication"

NOTE: RFC 7235, DOI 10.17487/RFC7235, June 2014, <https://www.rfc-editor.org/info/rfc7235>.

[4] IETF RFC 5246: "The Transport Layer Security (TLS) Protocol Version 1.2"

NOTE: RFC 5246, DOI 10.17487/RFC5246, August 2008, <https://www.rfc-editor.org/info/rfc5246>.

[5] IETF RFC 8259: "The JavaScript Object Notation (JSON) Data Interchange Format"

NOTE: RFC 8259, DOI 10.17487/RFC8259, December 2017, <https://www.rfc-editor.org/info/rfc8259>.

[6] IETF RFC 4648: "The Base16, Base32, and Base64 Data Encodings"

NOTE: RFC 4648, DOI 10.17487/RFC4648, October 2006, <https://www.rfc-editor.org/info/rfc4648>.

2.2 Informative references

References are either specific (identified by date of publication and/or edition number or version number) or non-specific. For specific references, only the cited version applies. For non-specific references, the latest version of the referenced document (including any amendments) applies.

NOTE: While any hyperlinks included in this clause were valid at the time of publication, ETSI cannot guarantee their long term validity.

The following referenced documents are not necessary for the application of the present document but they assist the user with regard to a particular subject area.

[i.1] ETSI ISG-QKD ETSI GS QKD 004 v1.1.1 (2010-12): "Quantum Key Distribution (QKD); Application Interface".

3 Definitions, symbols and abbreviations

3.1 Definitions

For the purposes of the present document, the following terms and definitions apply:

Application Programming Interface (API): interface implemented by a software program to be able to interact with other software programs

key: random digital data with an associated universally unique ID

key container: JSON data format containing key data

NOTE: As specified in Clause 6.3 below.

Key Management Entity (KME): entity that manages keys in a network in cooperation with one or more other Key Management Entities

Master Secure Application Entity: Secure Application Entity that initiates a request to a Key Management Entity for one or more new keys that can subsequently be requested by a Slave Secure Application Entity specified in the request

QKD Entity (QKDE): entity providing key distribution functionality including acting as an endpoint for the distribution of keys to at least one other QKD Entity using QKD protocols

QKD link: link connecting a pair of QKD Entities

QKD network: network comprised of two or more Trusted Nodes

quantum key distribution (QKD): procedure or method for generating and distributing symmetrical cryptographic keys with information theoretical security based on quantum information theory

Secure Application Entity (SAE): entity that requests one or more keys from a Key Management Entity for one or more applications running in cooperation with one or more other Secure Application Entities

Slave Secure Application Entity: Secure Application Entity that initiates a request to a Key Management Entity for one or more keys based on one or more key IDs that were previously delivered to a Master Secure Application Entity

Trusted Node (TN): node containing trusted equipment including one or more Key Management Entities and one or more QKD Entities situated within a security boundary

Web API: Application Programming Interface that can be accessed using HTTP or HTTPS protocols

3.2 Symbols

3.3 Abbreviations

For the purposes of the present document, the following abbreviations apply:

API	Application Programming Interface
JSON	JavaScript Object Notation
KME	Key Management Entity
QKD	Quantum Key Distribution
QKDE	QKD Entity
QoS	Quality of Service
REST	REpresentational State Transfer
SAE	Secure Application Entity
TN	Trusted Node
URI	Uniform Resource Identifier
UUID	Universally Unique Identifier

4 Key delivery API Specification Overview

This key delivery API is a REST-based API, a simple request and response style API between a SAE and a KME. SAEs request KMEs to deliver keys and KMEs deliver the keys. Calls to the API on a KME are intended to be made by SAEs with a point of presence within the same secure site as the KME they are connecting to. The QKD network may deliver common shared keys to SAEs in different sites.

Optical switches, encryption modules, and security management systems are examples of SAEs.

Keys are generated and shared securely with QKD technology by KMEs. Key management methods used by KMEs and how KMEs relay keys securely in a QKD network is outside the scope of the present document.

A QKD network can consist of a single QKD link, or it can be an extended network involving many such QKD links. An example of a QKD network is shown in Figure 1. Installing and configuring a QKD network, registering a new trusted node or QKD link into a QKD network and removing them from a QKD network are QKD network management issues and outside the scope of the present document.

Each KME shall have one or multiple QKDEs to connect with other KMEs via QKD links. KMEs shall be able to distribute keys to other KMEs. In each Trusted Node, there shall be at least one KME. One or multiple SAEs may connect with a KME within a Trusted Node. It is assumed that each Trusted Node is securely operated and managed. Each trusted node shall be located in its site. SAEs shall be located with its connected KMEs in its site. The API between SAE and KME shall be used within a security boundary in each site.

KMEs shall provide Web API server functionality to deliver keys to SAEs via HTTPS protocols.

Each KME shall have a unique ID (KME ID). A KME ID shall be unique in a QKD network. The format and the assignment of KME IDs is outside the scope of the present document.

SAEs make HTTPS requests to KMEs to get keys and status information.

Each SAE shall have a unique ID (SAE ID). A SAE ID shall be unique in a QKD network. The format and the assignment of SAE IDs is outside the scope of the present document.

All communications between SAE and KME shall use the HTTPS protocols (IETF RFC 7230 [1], IETF RFC 7231 [2], IETF RFC 7235 [3], IETF RFC 5246 [4]).

KMEs shall authenticate each request and identify the unique SAE ID of the calling SAE.

Data in the message body of HTTPS requests from SAE to KME and HTTPS responses from KME to SAE shall be encoded in JSON format as per IETF RFC 8259 [5].

The SAE making an initial "Get key" request is referred to as the Master SAE for the key(s) returned. An SAE making a subsequent “Get key with key IDs” request is called the Slave SAE for the key(s) returned.

Applications shall communicate Key IDs between SAEs as necessary for their operations but how they do so is outside the scope of the present document.

This document makes the following security assumptions about the use of this API with a QKD network:

- each Trusted Node is securely operated and managed;

- this API is used between SAEs and KMEs within a secure site;
- each SAE is secure;
- each KME is secure.

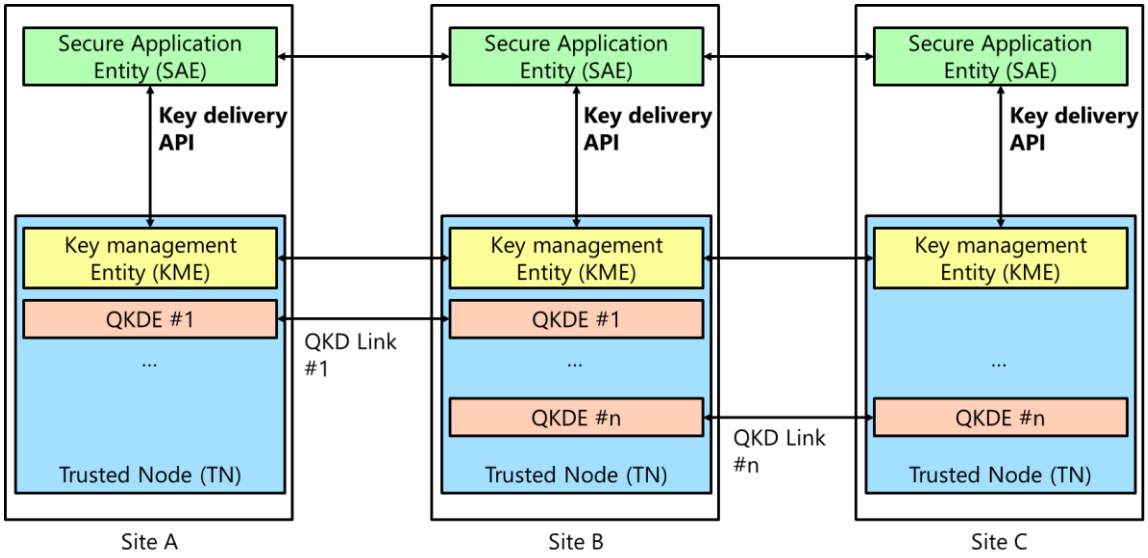

Figure 1: Example of QKD network

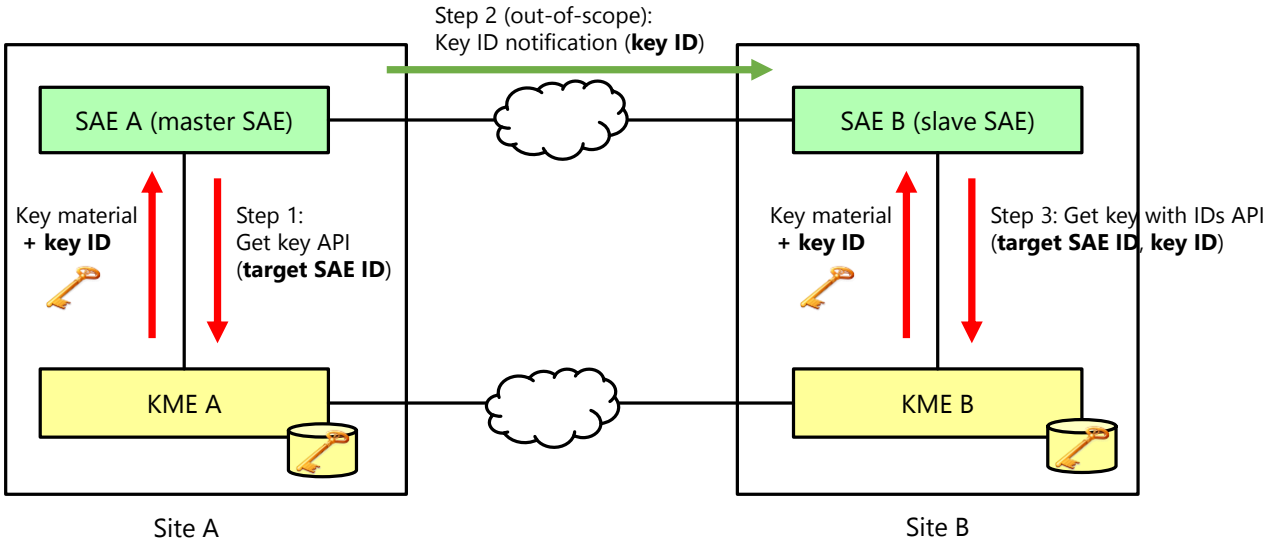

Figure 2: Use-case of key delivery API

Figure 2 shows a use-case illustrating a way the key delivery API can be used. KME A and KME B are either connected by a direct QKD link or a QKD network comprising multiple QKD links. SAE A is connected to KME A. SAE B is connected to KME B.

KME A and KME B exchange and store keys and each key delivered is assigned a universally unique ID. SAE A (master SAE) can initiate secure communication with SAE B (slave SAE) according to the following steps:

- Step 1: SAE A calls the key delivery API method "Get key" with the SAE B ID of the slave SAE to get keys from KME A. KME A delivers to SAE A key materials with the associated key IDs that are (to be) shared with KME B.
- Step 2: SAE A notifies SAE B of the key IDs. This communication between master SAE and slave SAE is outside the scope of the present document.

- Step 3: SAE B calls the key delivery API method “Get key with key IDs” with the SAE A ID of the master SAE and the notified key IDs information to get the identical keys from KME B. KME B delivers to SAE B the identical key materials with the identical associated key IDs that are shared with KME A.

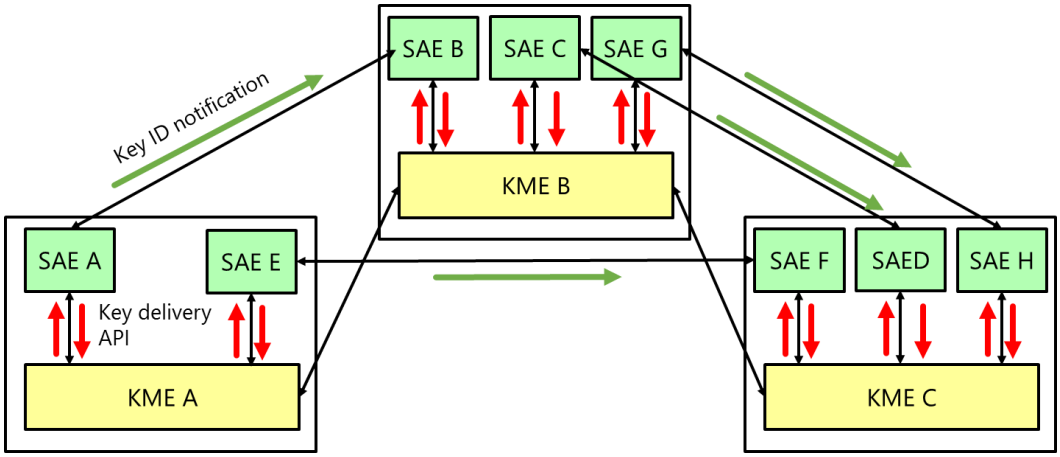

Figure 3: Use-case illustrating multiple SAEs connecting to each KME

Figure 3 shows another use-case illustrating how the key delivery API can be used. Multiple SAEs are connected to a single KME.

5 Protocol Specifications

5.1 Common Specification

The common specification is as follows.

Name	Description
Communication Protocol	HTTPS
Character code	UTF-8
HTTP Content-type	application/json

SAE shall connect to KME using HTTPS protocols (IETF RFC 7230 [1], IETF RFC 7231 [2], IETF RFC 7235 [3], IETF RFC 5246 [4]). At the connection establishment, mutual authentication between SAE and KME shall be performed. SAE shall verify the validity of a certificate the KME possesses and shall confirm the KME ID of the KME it is connecting to based on the certificate. Unless the KME ID has been verified in this manner the SAE shall not proceed to use this API with the KME. KME shall verify the validity of a certificate the SAE possesses and shall confirm the SAE ID of the connecting SAE based on the certificate. Unless the KME ID has been verified in this manner the KME shall reject the connection from the SAE. After the mutual authentication, the SAE may call API methods on the KME. The list of API methods shall be as follows.

No.	Method name	URL	Access Method
1	Get status	https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/status	GET
2	Get key	https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/enc_keys	POST (or GET)
3	Get key with key IDs	https://{KME_hostname}/api/v1/keys/{master_SAE_ID}/dec_keys	POST (or GET)

How to install or upgrade the certificate on each KME and on each SAE is outside the scope of the present document.

5.2 Get status

The specification of the "Get status" method shall be as follows.

Name	Description		
Overview	Returns **Status** from a KME to the calling SAE. **Status** contains information on keys available to be requested by a master SAE for a specified slave SAE.		
Access method	GET		
Access URL	https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/status		
Parameters	Name	Data type	Description
	{KME_hostname}	String (in URL)	Hostname or IP address of the KME. A port number may be specified separated from the hostname or IP address by a colon
	{slave_SAE_ID}	String (in URL)	URL-encoded SAE ID of slave SAE
Request data model (from SAE to KME)	None.		
Response data model (from KME to SAE)	**Status** (see clause 6)		
Pre-condition	None.		
Post-condition	None.		

Get status may return error responses as follows.

HTTP status code	Response data model	Description
400	**Error**	Bad request format.
401	-	Unauthorized.
503	**Error**	Error on server side.

5.3 Get key

The specification of the "Get key" method shall be as follows.

Name	Description		
Overview	Returns **Key container** data from the KME to the calling master SAE. **Key container** data contains one or more keys. The calling master SAE may supply **Key request** data to specify the requirement on **Key container** data. The slave SAE specified by the slave_SAE_ID parameter may subsequently request matching keys from a remote KME using key_ID identifiers from the returned **Key container**.		
Access method	POST (or GET for specified simple requests only (see clause 6))		
Access URL	https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/enc_keys		
Parameters	Name	Data type	Description
	{KME_hostname}	String (in URL)	Hostname or IP address of the KME. A port number may be specified separated from the hostname or IP address by a colon
	{slave_SAE_ID}	String (in URL)	URL-encoded SAE ID of slave SAE
Request data model (from SAE to KME)	**Key request** (POST only; see clause 6)		
Response data model (from KME to SAE)	**Key container** (see clause 6)		
Pre-condition	None.		
Post-condition	Requested number of keys provided to SAE are removed from key pool stored in KME.		

The "Get key" method may return error responses as follows.

HTTP status code	Response data model	Description
400	**Error**	Bad request format.
401	-	Unauthorized.
503	**Error**	Error on server side.

5.4 Get key with key IDs

The specification of the "Get key with key IDs" method shall be as follows.

Name	Description		
Overview	Returns **Key container** from the KME to the calling slave SAE. **Key container** contains keys matching those previously delivered to a remote master SAE based on the **Key IDs** supplied from the remote master SAE in response to its call to Get key. The KME shall reject the request with a 401 HTTP status code if the SAE ID of the requestor was not an SAE ID supplied to the "Get key" method each time it was called resulting in the return of any of the **Key IDs** being requested.		
Access method	POST (or GET for specified simple requests only (see clause 6))		
Access URL	https://{KME_hostname}/api/v1/keys/{master_SAE_ID}/dec_keys		
Parameters	Name	Data type	Description
	{KME_hostname}	String (in URL)	Hostname or IP address of the KME. A port number may be specified separated from the hostname or IP address by a colon
	{master_SAE_ID}	String (in URL)	URL-encoded SAE ID of master SAE
Request data model (from SAE to KME)	**Key IDs** (POST only; see clause 6)		
Response data model (from KME to SAE)	**Key container** (see clause 6)		
Pre-condition	None.		
Post-condition	Specified keys by Key IDs provided to SAE are removed from key pool stored in KME.		

The "Get key with key IDs" method may return error responses as follows.

HTTP status code	Response data model	Description
400	**Error**	Bad request format.
401	-	Unauthorized.
503	**Error**	Error on server side.

6 Data Format Specifications

6.1 Status data format

Status data format is used for a response data model of API "Get status" method. JSON data format of Status shall be as follows.

Items	Data type	Description
source_KME_ID	string	KME ID of the KME
target_KME_ID	string	KME ID of the target KME
master_SAE_ID	string	SAE ID of the calling master SAE
slave_SAE_ID	string	SAE ID of the specified slave SAE
key_size	integer	Default size of key the KME can deliver to the SAE (in bit)
stored_key_count	integer	Number of stored keys KME can deliver to the SAE
max_key_count	integer	Maximum number of stored_key_count
max_key_per_request	integer	Maximum number of keys per request
max_key_size	integer	Maximum size of key the KME can deliver to the SAE (in bit)
min_key_size	integer	Minimum size of key the KME can deliver to the SAE (in bit)
max_SAE_ID_count	integer	Maximum number of additional_slave_SAE_IDs the KME allows. "0" when the KME does not support key multicast.
status_extension	object	(Option) for future use

An example of Status data format is as follows.

```
{
    "source_KME_ID": "AAAABBBBCCCCDDDD",
    "target_KME_ID": "EEEEFFFFGGGGHHHH",
    "master_SAE_ID": "IIIIJJJJKKKKLLLL",
    "slave_SAE_ID": "MMMMNNNNOOOOPPPP",
    "key_size": 352,
    "stored_key_count": 25000,
    "max_key_count": 100000,
    "max_key_per_request": 128,
    "max_key_size": 1024,
    "min_key_size": 64,
    "max_SAE_ID_count": 0
}
```

6.2 Key request data format

Key request data format is used for a request data model of API "Get key" method. JSON data format of Key request shall be as follows.

Items	Data type	Description
number	integer	(Option) Number of keys requested, default value is 1.
size	integer	(Option) Size of each key in bits, default value is defined as key_size in Status data format.
additional_slave_SAE_IDs	array of strings	(Option) Array of IDs of slave SAEs. It is used for specifying two or more slave SAEs to share identical keys. The maximum number of IDs is defined as max_SAE_ID_count in Status data format.
extension_mandatory	array of objects	(Option) Array of extension parameters specified as name/value pairs that KME shall handle or return an error. Parameter values may be of any type, including objects.
extension_optional	array of objects	(Option) Array of extension parameters specified as name/value pairs that KME may ignore. Parameter values may be of any type, including objects.

Examples of Key request data format follow:

```
{
    "number": 3,
    "size": 1024
}


{
    "number": 1,
    "size": 4096,
    "additional_slave_SAE_IDs": [
        "ABCDEFG",
        "HIJKLMN"
    ]
}


{
    "number": 20,
    "size": 512,
    "extension_mandatory": [
        {
            "abc_route_type": "direct"
        },
        {
            "abc_transfer_method": "qkd"
        }
    ],
    "extension_optional": [
        {
            "abc_max_age": 30000
        }
    ]
}
```

All items in the Key request data format are optional and the JSON data format of Key request may be empty.

When Key request would be empty, an SAE may submit the request using the GET access method with no message body.

Where "number" and/or "size" are the only items in the Key request data format, the SAE may submit the request using the GET access method by specifying "number" and/or "size" as request URI parameters as string data.

EXAMPLE 1: An example URL for such a request using the GET access method is as follows:

https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/enc_keys?number=3&size=1024

In all other cases, the SAE shall submit the request using the POST access method.

A KME may return keys in response to requests where key size is not a multiple of 8. Where a KME only supports key size being a multiple of 8, it shall respond to requests for key request data where the size parameter is not such a multiple with a 400 error response with the message "size shall be a multiple of 8".

"additional_slave_SAE_IDs" is used for specifying multiple SAEs. In the case where "additional_slave_SAE_IDs" lists at least one SAE ID, the identical key material shall be shared not only between master SAE and slave SAE, but also shared with other slave SAE(s) specified in "additional_slave_SAE_IDs". How to handle the additional_slave_SAE_IDs is up to KME, but Annex B shows an example.

Key request data format defines two types of extension fields: "extension_mandatory" and "extension_optional". Both may be used for introducing new parameters / vendor specific parameters to Key request data format for future use.

"extension_mandatory" is used for defining a list of extension parameters that a KME shall handle to be permitted to return keys in response to the request. If a KME does not support one or more of the extension parameters supplied in "extension_mandatory" the KME shall return a 400 error response with the message "not all extension_mandatory parameters are supported". If a KME supports all the extension parameters within "extension_mandatory" but is unable to deliver the requested keys meeting all of the requirements they specify the KME shall return a 400 error response with the message "not all extension_mandatory request options could be met".

"extension_optional" is used for defining extension parameters that may be ignored by KME. KME may handle all optional parameters defined in "extension_optional" or it may ignore one or more optional extension parameters. If a KME ignores one or more of the optional extensions parameters within "extension_optional" KME may return any response it would otherwise have given (including a 200 OK response including Key container data format if appropriate) or it may return a 400 error response with message "not all extension_optional request options handled".

"extension_mandatory" and "extension_optional" may convey any kind of information from SAE to KME.

EXAMPLE 2: The type of key required or information about the future desired quality of service.

"extension_mandatory" and "extension_optional" may be used to pass named parameters of any type with any name as long as it is specified as valid JSON.

Future versions of the present document may specify reserved parameter names for parameters within "extension_mandatory" and "extension_optional" and their usage. To reduce the risk of conflict between extension parameter names vendor specific prefixes ending with an underscore should be used for extension parameter names until they have been added to the present document. A group of vendors may choose to adopt a common prefix ending with an underscore where they have agreed to share one or more common extension parameters.

EXAMPLE 3: Company "Abcdefg" can choose to introduce vendor specific extension parameters named like "abc_xxxxx" and "abc_yyyy".

EXAMPLE 4: SAE can request a specific type of key to KME. In some use-cases the type of route used can be important. A abc_route_type of "direct" could be defined by a vendor using the prefix "abc_" to request keys shared between adjacent Trusted Nodes connected by a single QKD link and a abc_route_type of "indirect" defined to request keys that are relayed using more than one QKD link via at least one Trusted Node other than those to which the SAEs connect using this API. SAE can request "direct" abc_route_type keys as an "extension_mandatory" parameter if "indirect" abc_route_type keys are not acceptable for the application. If SAE prefers "direct" abc_route_type keys but "indirect" abc_route_type keys are also acceptable, SAE can call specify "direct" abc_route_type within "extension_optional".

6.3 Key container data format

Key container data format is used for a response data model of API "Get key" method and "Get key with key IDs" method. JSON data format of Key container shall be as follows.

Items		Data type	Description
Keys		array of objects	Array of keys. The number of keys is specified by the “number” parameter in “Get key”. If not specified, the default number of keys is 1.
	key_ID	string	ID of the key: UUID format (example: “550e8400-e29b-41d4-a716-446655440000”)
	key_ID_extension	object	(Option) for future use
	key	string	Key data encoded by base64 [6]. The key size is specified by the “size” parameter in “Get key”. If not specified, the “key_size” value in Status data model is used as the default size.
	key_extension	object	(Option) for future use
key_container_extension		object	(Option) for future use

An example of Key container data format is as follows.

```
{
    "keys": [
        {
            "key_ID": "bc490419-7d60-487f-adc1-4ddcc177c139",
            "key": "wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly+s="
        },
        {
            "key_ID": "0a782fb5-3434-48fe-aa4d-14f41d46cf92",
            "key": "OeGMPxh1+2RpJpNCYixWHFLYRubpOKCw94FcCI7VdJA="
        },
        {
            "key_ID": "64a7e9a2-269c-4b2c-832c-5351f3ac5adb",
            "key": "479G1Osfljpmfa5vn24tdzE5zqv5CafkGxYrLCk8384="
        },
        {
            "key_ID": "550e8400-e29b-41d4-a716-446655440000",
            "key": "csEMV9KkmjgOPF90uc54+hykhg6iI5GTPHlP9PjgLVU="
        }
    ]
}
```

"key_ID_extension", "key_extension", and "key_container_extension" are used to return additional information about the key ID, key, and key container respectively. Sometimes, these extension values may be used to convey information relating to extension parameters supplied within "extension_mandatory" and/or "extension_optional" in the Key request data supplied in a call to the "Get key" method.

6.4 Key IDs data format

Key IDs data format is used for a request data model of API "Get key with key IDs" method. JSON data format of Key IDs shall be as follows.

Items		Data type	Description
key_IDs		array of objects	Array of key IDs
	key_ID	string	ID of the key: UUID format (example: “550e8400-e29b-41d4-a716-446655440000”)
	key_ID_extension	object	(Option) for future use
key_IDs_extension		object	(Option) for future use

An example of Key IDs data format is as follows.

```
{
    "key_IDs": [
        {
            "key_ID": "bc490419-7d60-487f-adc1-4ddcc177c139"
        },
        {
            "key_ID": "0a782fb5-3434-48fe-aa4d-14f41d46cf92"
        },
        {
            "key_ID": "64a7e9a2-269c-4b2c-832c-5351f3ac5adb"
        },
        {
            "key_ID": "550e8400-e29b-41d4-a716-446655440000"
        }
    ]
}
```

In the case where a single key ID is specified and no extension is specified, SAE may make the request using a GET access method with no body message. Otherwise the SAE shall use the POST access method with Key IDs data format for Get key with key IDs API.

EXAMPLE 1: An example of an access URL using the GET access method for Get key with key IDs API is as follows:

https://{KME_hostname}/api/v1/keys/{slave_SAE_ID}/dec_keys?key_ID=bc490419-7d60-487f-adc1-4ddcc177c139

In the case where two or more key IDs are specified the POST access method shall be used. The POST access method is used to support multiple key IDs or extension field defined in Key ID data format in Get key with IDs method.

If a KME is not able to return one or more keys specified in key IDs data format, the KME shall return a 400 error response with the message "one or more keys specified are not found on KME".

6.5 Error data format

Error data format is used for an error response data model of API "Get status" method, "Get key" method, and "Get key with key IDs" method. JSON data format of Error shall be as follows.

Items	Data type	Description
message	string	Error message
details	array of objects	(Option) Array to supply additional detailed error information specified as name/value pairs. Values may be of any type, including objects.

Examples of Error data format follow.

```
{
    "message": "key data access error"
}


{
    "message": "not all extension_mandatory parameters are supported",
    "details": [
        {
            "extension_mandatory_unsupported": "abc_route_type is not supported"
        }
    ]
}
```

Annex A (informative): API function mapping to GS QKD 004

The present document is a REST-based key delivery API. It is a request-response style API comprising of HTTPS protocols and JSON data format. It provides three methods: Get status, Get key, and Get key with IDs. By calling Get key, an application (master SAE) gets block key materials with key IDs. The key IDs are sent to an application (slave SAE) that can be connected to a different KME. The application (slave SAE) calls Get key with key IDs and obtains identical block key materials.

On the other hand, GS QKD 004 [i.1] defines an OMG IDL remote function call-based application interface. It provides three primitives for applications: QKD_OPEN, QKD_GET_KEY, and QKD_CLOSE. By calling QKD_OPEN API, an application creates a session and gets a key stream ID. Then by calling QKD_GET_KEY with the key stream ID, the application gets a key from the data stream. The key stream ID is sent to an application on the other side. The key stream ID is used to get identical key from the identical key stream by the application on the other side. By calling QKD_CLOSE API, the application destroys the session.

One can consider that the present document is an easy implementation of GS QKD 004 [i.1]. It is possible to map API defined in the present document into the GS QKD 004 [i.1] application interface model, as follows:

GS QKD 014			GS QKD 004		
	API	Description		Method	Description
1	Get key API	To get block key materials with associated key IDs	1	QKD_OPEN	To create a session and to get key stream ID
			2	QKD_GET_KEY	To get key material from the specified key stream ID
			3	QKD_CLOSE	To destroy the session
2	Key ID notification	Outside the scope	4	Key stream ID notification	Outside the scope
3	Get key with key IDs API	To get block key materials corresponding to the specified key IDs	5	QKD_OPEN	To open the session corresponding to the specified key stream ID
			6	QKD_GET_KEY	To get key material from the specified key stream ID
			7	QKD_CLOSE	To destroy the session

In this mapping, the key stream ID (in GS QKD 004) is used as the key ID (in the present document).

Annex B (informative): An example of how to deliver keys to multiple SAEs

The present document provides an optional parameter for Get key API to specify multiple SAEs (see clause 6.2). With the optional parameter, master SAE can specify one or more additional slave SAEs that are also authorised to retrieve identical copies of the requested key(s). The implementation of this functionality is optional for a KME. The way in which the underlying QKD network and KME deliver keys to multiple SAEs is outside the scope of the present document. However, the following is a simple example of how keys might be delivered to multiple SAEs.

SAE A (master SAE) can initiate secure communication with SAE B (slave SAE) and SAE C (slave SAE) according to the following steps:

- Step 1: SAE A calls the key delivery API "Get key" with the SAE B ID and the SAE C ID of the slave SAEs using additional_slave_SAE_IDs option field to get keys from KME A. KME A generates key materials with the associated key IDs and delivers the key materials to SAE A that are (to be) shared with KME B and KME C.
- Step 2: KME A securely transfers the generated key materials to KME B using QKD keys shared via QKD link #1 between KME A and KME B. Then, KME B securely transfers the received key materials to KME C using QKD keys shared via QKD link #2 between KME B and KME C.
- Step 3: SAE A notifies SAE B and SAE C of the key IDs.
- Step 4: SAE B calls the key delivery API "Get key with key IDs" with the SAE A ID of the master SAE and the notified key IDs information to get the identical keys from KME B. KME B delivers to SAE B the identical key materials when requested with key IDs matching those notified by KME A.
- Step 5: SAE C calls the key delivery API "Get key with key IDs" with the SAE A ID of the master SAE and the notified key IDs information to get the identical keys from KME C. KME C delivers to SAE C the identical key materials when requested with key IDs matching those notified by KME A.

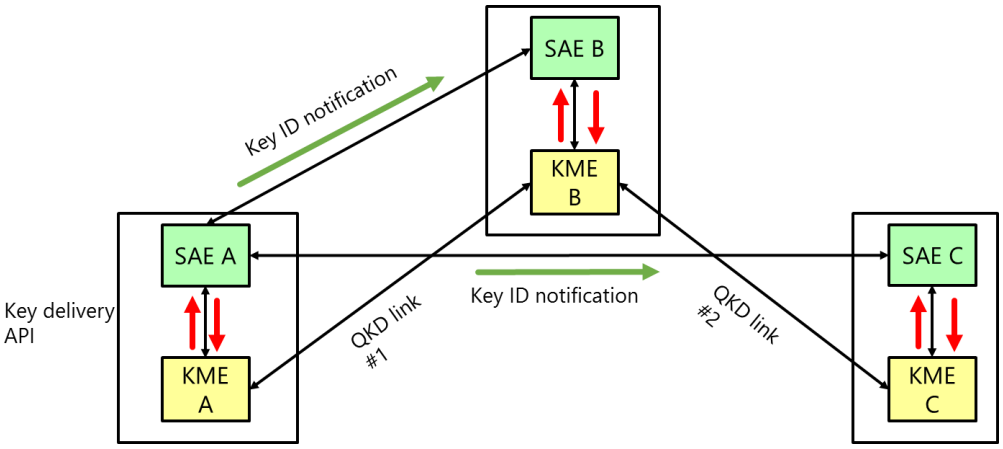

Figure 4: Example of how to deliver keys to multiple SAEs

Annex (informative): Authors & contributors

The following people have contributed to the present document:

Rapporteur:

Yoshimichi Tanizawa, Toshiba

Other contributors:

Martin Ward, Toshiba

Hideaki Sato, Toshiba

Vicente Martin Ayuso, Facultad de Informatica, Universidad Politécnica de Madrid

Alejandro Aguado, Facultad de Informatica, Universidad Politécnica de Madrid

Oliver Maurhart, Austrian Institute of Technology GmbH

Alan Mink, Applied Communication Sciences - Vencore Labs, Inc.

Norbert Lutkenhaus, University of Waterloo

Momtchil Peev, Huawei Technologies Duesseldorf GmbH

Bruno Huttner, ID Quantique SA

Catherine White, British Telecommunications plc

Graham Wallace, Senetas Europe Limited

Joo Cho, ADVA Optical Networking SE

History

Document history		
<Version>	<Date>	<Milestone>