ETSI GS QKD 020 V1.1.1 (2026-06)

GROUP SPECIFICATION

Quantum Key Distribution (QKD); Protocol and data format of REST-based Interoperable Key Management System API

Disclaimer

The present document has been produced and approved by the Quantum Key Distribution (QKD) ETSI Industry Specification Group (ISG) and represents the views of those members who participated in this ISG.
It does not necessarily represent the views of the entire ETSI membership.

Reference
DGS/QKD-020_InteropKMS

Keywords
API, protocol, quantum cryptography, quantum key distribution

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 - APE 7112B
Association à but non lucratif enregistrée à la
Sous-Préfecture de Grasse (06) N° w061004871

Contents

Intellectual Property Rights

Essential patents

IPRs essential or potentially essential to normative deliverables may have been declared to ETSI. The declarations pertaining to these essential IPRs, if any, are 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 IPR online database.

Pursuant to the ETSI Directives including the ETSI IPR Policy, no investigation regarding the essentiality of IPRs, 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.

DECT™, **PLUGTESTS™**, **UMTS™** and the ETSI logo are trademarks of ETSI registered for the benefit of its Members. **3GPP™**, **LTE™** and **5G™** logo are trademarks of ETSI registered for the benefit of its Members and of the 3GPP Organizational Partners. **oneM2M™** logo is a trademark of ETSI registered for the benefit of its Members and of the oneM2M Partners. **GSM**® and the GSM logo are trademarks registered and owned by the GSM Association.

Foreword

This Group Specification (GS) has been produced by ETSI Industry Specification Group (ISG) 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.

1 Scope

The present document specifies a REST API that allows key management systems to interoperate to pass keys horizontally between two systems located in a common trusted node. The API enables QKD networks to serve applications that request shared secret keys from key management systems that are not linked by a contiguous chain of systems from the same vendor. It is beyond the scope of the present document to describe how the underlying QKD network agrees key material between nodes. URI formats, communication protocols (HTTPS), and the JSON data format encoding of posted parameters and responses (including key material) are described. An OpenAPI description of the API is available.

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 in the ETSI docbox.

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 9110: STD 97 (June 2022): "HTTP Semantics".

[2] IETF RFC 8259: STD 90 (December 2017): "The JavaScript Object Notation (JSON) Data Interchange Format".

[3] IETF RFC 9112: STD 99 (June 2022): "HTTP/1.1".

[4] IETF RFC 8446 (August 2018): "The Transport Layer Security (TLS) Protocol Version 1.3".

[5] IETF RFC 3986: STD 66 (January 2005): "Uniform Resource Identifier (URI): Generic Syntax".

[6] IETF RFC 5234: STD 68 (October 2005): "Augmented BNF for Syntax Specifications: ABNF".

[7] IETF RFC 4648 (October 2006): "The Base16, Base32, and Base64 Data Encodings".

[8] IETF RFC 9562 (May 2024): "Universally Unique IDentifiers (UUIDs)".

[9] IETF RFC 9457 (July 2023): "Problem Details for HTTP APIs".

[10] OpenAPI Initiative: "OpenAPI Specification v3.1.2 (September 2025)".

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 may be useful in implementing an ETSI deliverable or add to the reader's understanding, but are not required for conformance to the present document.

[i.1] ETSI GR QKD 007: "Quantum Key Distribution (QKD); Vocabulary".

[i.2] ETSI GS QKD 014: "Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API".

[i.3] ETSI GS QKD 004: "Quantum Key Distribution (QKD); Application Interface".

3 Definition of terms, symbols and abbreviations

3.1 Terms

For the purposes of the present document, the terms given in ETSI GR QKD 007 [i.1] and the following apply:

NOTE: A term defined in the present document takes precedence over the definition of the same term, if any, in ETSI GR QKD 007 [i.1].

interworking node: node of more than one QKD network that contains key management entities from each QKD network interworking within the node

key management entity: entity that manages keys in a network in cooperation with one or more other key management entities

secure application entity: 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

3.2 Symbols

Void.

3.3 Abbreviations

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

API	Application Programming Interface
HEXDIG	HEXadecimal DIGits
HTTP	HyperText Transfer Protocol
HTTPS	HyperText Transfer Protocol Secure
IANA	Internet Assigned Numbers Authority
IETF	Internet Engineering Task Force
IP	Internet Protocol
JSON	JavaScript Object Notation
KME	Key Management Entity
KMS	Key Management System
PEN	Private Enterprise Number
QKD	Quantum Key Distribution
REST	REpresentational State Transfer
SAE	Secure Application Entity
TLS	Transport Layer Security
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTF-8	UCS Transformation Format 8 bits
UUID	Universally Unique IDentifier

4 Description

4.1 Overview

The present document specifies a communication protocol and data format for Key Management Entities (KMEs) for use by Quantum Key Distribution (QKD) networks to exchange keys when located together within a protected operational environment. Use-cases include where:

- KMEs belong to different QKD networks and are located within an interworking node.
- KMEs belong to different policy domains of a QKD network.
- KMEs are otherwise incompatible, e.g. from different vendors.

Since use-cases include interworking between different network operators and policy domains, the information that KMEs are required to exchange is intentionally limited.

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 protocol "HTTPS" i.e. HTTP with TLS, as well as the data format and encoding of request and response content, including key material.

4.2 Operation of the API

The primary purpose of QKD networks is to deliver shared secure random keys to authorized cryptographic applications called Secure Application Entities (SAEs). APIs have been specified in ETSI GS QKD 014 [i.2] and ETSI GS QKD 004 [i.3] for the delivery of keys from a KME to an SAE.

Where there is no direct QKD link between the KMEs from which SAEs are requesting to share keys, the KMEs within QKD networks typically relay keys. They can consume QKD keys to help achieve this securely. Different sets of KMEs might not be fully compatible between sets of KMEs. Sets of KMEs could also be managed separately, e.g. by different network operators that wish to limit unnecessary sharing of internal network information.

In such cases a standardized interface is required for KMEs to pass keys horizontally between different sets of KMEs that cannot otherwise interoperate due to incompatibility or policy, etc. The API described in the present document enables key transfers between KMEs within a secure operational environment. This can enable shared keys to be delivered to two (or more) SAEs connected to KMEs in different QKD networks, policy domains, etc.

There are two primary modes of operation for the API methods described in the present document: asynchronous mode and synchronous mode. In the asynchronous mode, requests are processed in a non-blocking manner, allowing a KME to accept a request without undue delay and close the request connection by issuing a "202" ('Request accepted') response. Actual key processing can then be performed in the background before the KME acknowledges the result of that key processing by issuing one or more requests back to the KME that made the initial request to provide details of the outcome. This mode is particularly advantageous in high-traffic environments, or where request processing can be time consuming. It avoids blocking the requestor and minimizes the number of simultaneous open connections between KMEs. As a result, this mode can help achieve scalability as well as predictable performance under anomalous conditions. An overview message sequence diagram for the asynchronous mode is provided in Figure 4.2-1.

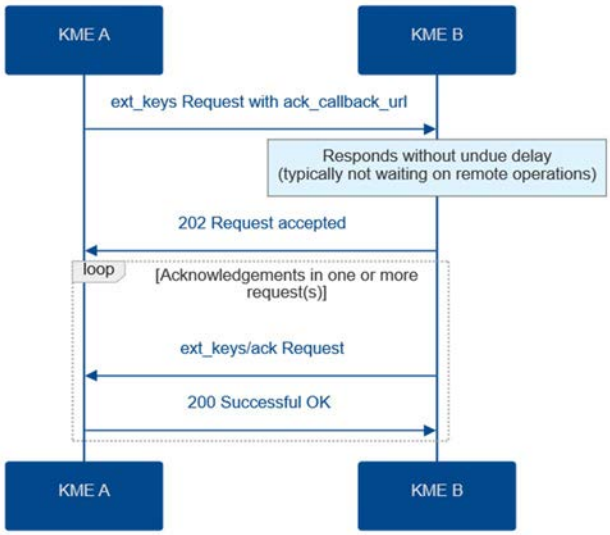

Figure 4.2-1: Overview message sequence diagram for successful required asynchronous mode

Conversely, in the synchronous mode the requestor has to wait for all parts of a request to either complete or fail before responding since all results are returned within a single response. This mode provides a more straightforward and direct interaction between KMEs and can be suitable where results can be returned almost immediately, but it can limit scalability in large or congested networks. A message sequence diagram for the optional synchronous mode is provided in Figure 4.2-2.

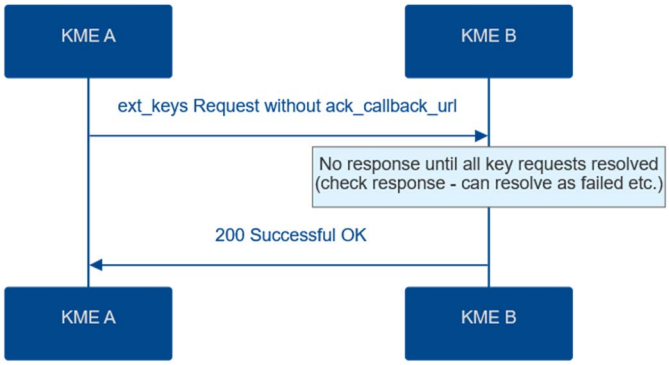

Figure 4.2-2: Overview message sequence diagram for successful optional synchronous mode

To ensure interoperability, all KMEs shall support the asynchronous mode of operation for the API methods defined in the present document.

KMEs may support the synchronous mode of operation as an optional mode in addition to the required asynchronous mode for specific implementation(s).

A KME shall not retain an association for any key in a request that arose from a call it received to "`ext_keys`" and for which it returned either a "`400`" ('Bad request format') or a "`401`" ('Unauthorised') response.

KMEs shall assume keys could have been associated with the "`initiator_sae_id`" and "`target_sae_id`" as a result of an "`ext_keys`" request unless either a "`400`" ('Bad request format') or "`401`" ('Unauthorised') response is returned.

Unless one of these two responses is successfully received, a KME shall assume that assignment(s) of the key(s) in the request could potentially have been created.

NOTE 1: Potential associations could reside within the KME or have been created by other entities it interacted with. Potential associations could have been created where a timeout occurs.

KMEs should not reassign any key that was in a "`ext_keys`" call without considering the potential assignments that could have been created.

NOTE 2: KMEs will typically not release a key to SAEs until they know that the key was successfully relayed to all intended "target_sae_id".

Where errors or timeouts other than a "400" ('Bad request format') or "401" ('Unauthorised') occur in an "ext_keys" request, KMEs shall either retry the failed requests for all impacted keys in subsequent "ext_keys" requests or void the impacted keys within their network(s) and by issuing "ext_keys/void" requests.

For "400" ('Bad request format') or a "401" ('Unauthorised') response KMEs may optionally retry the failed requests.

Where errors or timeouts occur in an "ext_keys/ack" request, KMEs shall either retry the failed requests for all impacted keys in subsequent "ext_keys/ack" requests or void the impacted keys within their network(s) and issue "ext_keys/void" requests.

NOTE 3: For "ext_keys/ack" the call to "ext_keys/void" is issued in the opposite direction to the KME that issued the related "ext_keys" request.

More comprehensive sequence diagrams are provided for the required asynchronous mode in Figure 4.2-3 and for the optional synchronous mode in Figure 4.2-4.

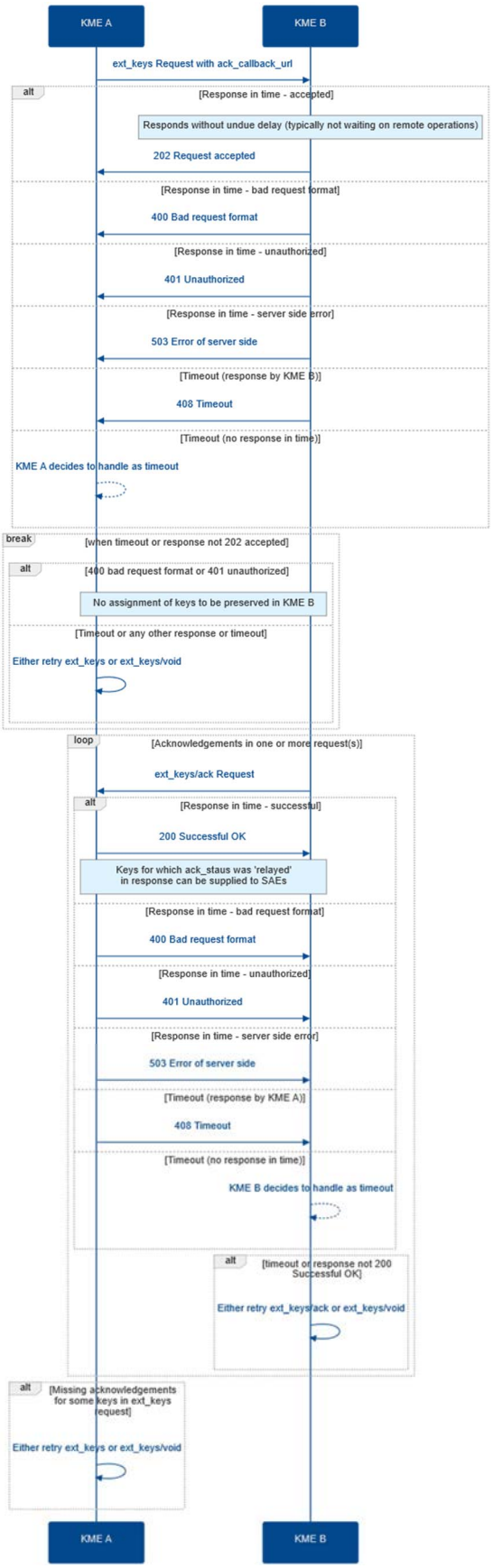

Figure 4.2-3: Message sequence diagram for required asynchronous mode

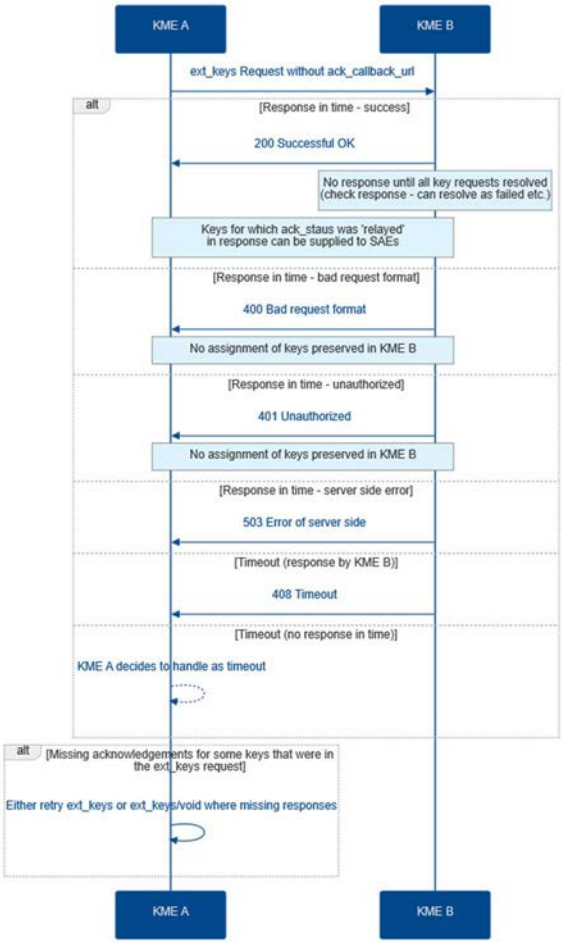

Figure 4.2-4: Message sequence diagram for optional synchronous mode

4.3 Connection specification and security scheme

The common connection specifications for the API are shown in Table 4.3-1.

Table 4.3-1: Common connection specifications

Name	Description
Communication Protocol	"HTTPS": HTTP/1.1 and optionally higher versions in addition; TLS v1.3 or higher versions
Content-Type	application/json; charset=UTF-8

All communication functions specified in this API shall be implemented using the HTTP/1.1 protocol as specified in IETF RFC 9110 [1] and IETF RFC 9112 [3] with Transport Layer Security (TLS).

NOTE 1: This combination is often referred to as an HTTPS protocol.

Higher versions of the HTTP protocol may be implemented in addition to HTTP/1.1.

TLS v1.3 as specified in IETF RFC 8446 [4] or higher versions shall be used.

At the connection establishment, mutual authentication between the KMEs shall be performed using certificates.

Each of the two KMEs shall verify the validity of the certificate of the other KME and shall confirm its identity from the certificate.

Where requests and responses include body content these shall be encoded using the "`application/json`".

NOTE 2: OpenAPI introduced support for specifying mutual TLS security in version 3.1. While other parts of this OpenAPI description can be easily converted to OpenAPI version 3.0 for use with older tools, it would be necessary to ensure that whatever security is specified in a Security Scheme Object is manually overridden according to this clause 4.3 in any implementation.

4.4 Security and operational environment

This API is for use between KMEs that are both located within the same site (an interworking node when used between two QKD networks). The KMEs and the communications the present document specifies between them should remain within a secure operational environment. Appropriate security measures should be considered for the operational environment.

General security requirements for key processing and management are outside the scope of the present document.

4.5 Extensions

Some methods allow for extensions to be provided. Extensions can be specified by vendors and KMEs shall follow the specification of any extension it claims to have implemented. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension.

Properties may be of any type, but their names shall be unique and start with a vendor prefix that shall be the capital letter "`E`" followed by an IANA Private Enterprise Number (PEN) followed by an underscore. The organization to which the PEN is assigned can use the extension for their own purposes, or can provide a specification for the extension to other organizations so they can also implement the extension. No organization shall use an extension starting with a vendor prefix with the PEN of any other organization without following a specification from the organization associated with the PEN.

ETSI might specify extensions in future but unless it does ETSI's IANA Private Enterprise Number "`13019`" is reserved and shall not be used.

NOTE: Extensions can originate from SAE requests that triggered key relaying, or can be added by a KME, e.g. to add metadata etc. Requirements in the specification of an extension can prohibit addition of an extension by a KME in subsequent calls. Extensions could be deserialized to a dictionary within a KME. Both property names and their values can be significant.

Relevant schema include "`extension`", "`extension_mandatory`", and "`extension_optional`".

4.6 SAE IDs

SAE IDs shall be unique across any set of connected QKD networks. In some circumstances, duplicate SAE IDs could result in keys being made available to an SAE other than the intended SAE(s). To obtain a key intended for another SAE in the presence of duplicate SAE IDs using the API in ETSI GS QKD 014 [i.2] from an otherwise securely-implemented KME, a malicious target SAE would need to specify a valid "`key_id`" associated with the other target SAE in an appropriately-timed request. However, even where potential mitigations apply in some circumstances (such as maintaining the "`key_id`" as confidential) reliable delivery can be compromised by duplicate SAE IDs.

How connected QKD networks ensure the uniqueness of SAE ID is beyond the scope of the present document.

SAE IDs ("`initiator_sae_id`" and "`target_sae_id`") shall not exceed 64 characters in length. Where SAEs are likely to want to communicate SAE IDs in restricted-length-parameters of other protocols connected QKD networks can consider further limiting SAE ID lengths.

SAE IDs shall not include characters that would not be allowed within a URI as specified in IETF RFC 3986 [5] with reference to IETF RFC 5234 [6]. SAE IDs are not required to be URIs.

NOTE: IETF RFC 3986 [5] allows unreserved characters "`unreserved = (ALPHA / DIGIT / "-" / "." / "_" / "~")`", reserved general delimiter characters "`gen-delims = (":" / "/" / "?" / "#" / "[" / "]" / "@")`", and sub-delimiter characters "`sub-delims = ("!" / "$" / "&" / "'" / "(" / ")" / "*" / "+" / "," / ";" / "=")`" as well as percent encoded characters "`pct-encoded = "%" HEXDIG HEXDIG`". See IETF RFC 5234 [6] for definitions of "`ALPHA = (%x41-5A / %x61-7A; A-Z / a-z)`", "`DIGIT = %x30-39; 0-9`", and "`HEXDIG = (DIGIT / "A" / "B" / "C" / "D" / "E" / "F")`".

4.7 Example use case for the API

Figure 4.7-1 shows a use-case in which the "ext_keys" operation is used to pass keys between KMEs in an interworking node of a three-node network. It is comprised of two networks X and Y that both have a presence in an interworking node. Key delivery to applications using ETSI GS QKD 014 [i.2] is considered in this use case.

The asynchronous key delivery between two QKD networks is uncoupled from any SAE protocol or workflow. This use case considers a reactive pattern in which keys are requested by an initiator SAE. Use cases following a proactive pattern are also possible in which a KME node acts as an initiator SAE to request keys to fill one or more shared buffers of keys that such KMEs can then use to delivery keys to client SAEs.

1) SAE A makes a local request within Site A to KME X1 of QKD network X for key(s) shared with SAE C, which is in a separate Site C served by another QKD network Y.

2) QKD network X performs operations using QKD keys to establish shared key(s) between KME X1 in SAE A's site and KME X2, which is in an interworking node at Site B. Communications can be proprietary to systems within QKD network X and details are not indicated in Figure 4.7-1.

3) Within Site B KME X2 issues a request to "ext_keys" on KME Y2 using the API specified in the present document.

4) KME Y2 responds immediately with HTTP status code "202" ('Request accepted').

5) QKD network Y performs operations using QKD keys to transfer the key(s) to KME Y1 in Site C. Communications can be proprietary to systems within QKD network Y, hence details are not indicated in Figure 4.7-1.

6) Once QKD network Y has ensured the key(s) are available in KME Y1, KME Y2 issues a request to "ext_keys/ack" on KME X2 to acknowledge that the request was successful.

7) KME X2 responds immediately with HTTP status code "200" ('Successful OK') to confirm that KME X2 has received the acknowledgement.

8) QKD network X communicates the successful result back to KME X1. Communications can be proprietary to systems within QKD network X and details are not indicated in Figure 4.7-1.

9) KME X1 returns the key(s) to SAE A by responding to the original request from SAE A.

10) SAE A communicated the key ID(s) to SAE C. Communications can be proprietary to systems of the application, hence details are not indicated in Figure 4.7-1.

11) SAE C uses the key ID(s) to request the key(s) from KME Y1 locally within Site C.

12) KME Y1 responds to SAE C with the key(s). Both SAE A and SAE C now share identical key(s) and can proceed to consume them for cryptographic purposes.

In this manner keys are shared between SAEs A and C via the use of "ext_keys" within the interworking node.

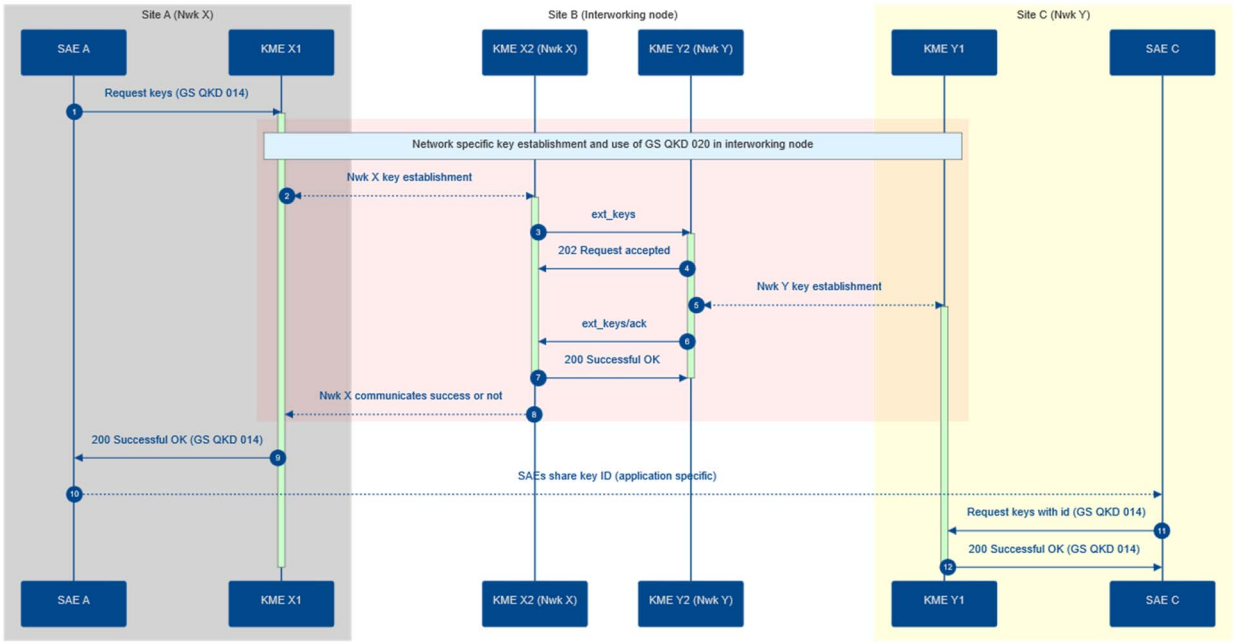

Figure 4.7-1: Sequence diagram for use case of real-time ETSI GS QKD 014 [i.2] key request established via interwork node

5 versions: API versions available

5.1 Tag description

Versionless path that can be used to query the available versions of this API on a KME.

5.2 GET /kmapi/versions: Get supported API versions

5.2.1 Description

Return list of versions of the API that are supported by the KME.

When an KME makes a request to another KME, it should use the most recent version of the API that the KMEs both support.

The SAE can use this end-point to determine which API versions the KME supports.

5.2.2 Responses

5.2.2.1 200 Successful OK response application/json

The response body shall match the "`version_container`" schema as shown in Table 5.2.2.1-1 encoded as media type "`application/json`" in JSON format as specified in IETF RFC 8259 [2].

Table 5.2.2.1-1: Data type "version_container" for 200 response to GET /kmapi/versions as application/json

Field Name	Type	Description
versions	array of string schema: versions	***REQUIRED*** Versions "versions" shall be an array of supported API versions "version". *Maximum items:* 1 000
capabilities	array of string schema: capabilities	Capabilities "capabilities" shall be an array of strings each indicating a "capability" supported by the host. Where a host supports the optional synchronous mode it should report this capability of "synchronous_mode". Clients should confirm this capability before sending requests for synchronous mode. *Maximum items:* 1 000
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "200" status code "version_container" response body for "/kmapi/versions" as media type "application/json":

```
{
  "versions": [
    "v1"
  ],
  "capabilities": [
    "synchronous_mode"
  ],
  "extension": {
    "E32473_extension1": "Some string",
    "E32473_extension2": {
      "property1": 10111,
      "property2": "Some text",
      "property3": {
        "subprop1": 21,
        "subprop2": true
      }
    }
  }
}
```

5.2.2.2 401 Unauthorized response application/json

The response body shall match the "problem_details" schema as shown in Table 5.2.2.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 5.2.2.2-1: Data type "problem_details" for 401 response to GET /kmapi/versions as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024

Field Name	Type	Description
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "`401`" status code "`problem_details`" response body for "`/kmapi/versions`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/unauthorized",
  "status": 401,
  "title": "unauthorized",
  "details": {
    "unauthorized": "User-supplied certificate is invalid."
  }
}
```

5.2.2.3 503 Error of server side response application/json

The response body shall match the "`problem_details`" schema as shown in Table 5.2.2.3-1 encoded as media type "`application/json`" in JSON format as specified in IETF RFC 8259 [2].

Table 5.2.2.3-1: Data type "problem_details" for 503 response to GET /kmapi/versions as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "`503`" status code "`problem_details`" response body for "`/kmapi/versions`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/server-side-general-error",
  "status": 503,
  "title": "server side general error",
  "details": {
    "server_side_general_error": "The server encountered a general failure and cannot respond."
  }
}
```

EXAMPLE 2: An example of "503" status code "problem_details" response body for "/kmapi/versions" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 503,
  "title": "key routing error",
  "details": {
    "no_route_multiple_targets": "Routes not available to support these multiple target SAEs."
  }
}
```

6 ext_keys: Interoperation (external keys)

6.1 Tag description

Collection of versioned paths to pass keys to another KME, acknowledge the status of keys, or void keys.

6.2 POST /kmapi/v1/ext_keys: Transfer keys to external KMS

6.2.1 Description

Pass an "ext_key_container" extended key request container comprising key material and associated data to another KME, for the key(s) to be delivered (by relay where necessary) to the target SAE(s) specified. The extended key request container contains keys matching those to be delivered to the initiator SAE.

Upon a valid request, for the required asynchronous mode, the KME shall aim to respond without undue delay with a "202" ('Accepted'), then it shall issue a separate call (or multiple calls) to the specified "ack_callback_url" endpoint once the keys are actually delivered (or fail to be delivered).

If synchronous mode is implemented in addition to the asynchronous mode, the KME shall not use "ext_keys/ack" to acknowledge outcomes for keys from synchronous mode requests ("ack_callback_url" would have been omitted from these requests). Instead, the KME shall respond with a "200" ('Successful OK').

A "400" error may be returned if the request is known to be invalid without further investigation (e.g. if the container format is invalid or it includes initiator/target SAE IDs for which a valid route is not known to the KME).

6.2.2 Request Body: application/json

The request body shall match the "ext_key_container" schema as shown in Table 6.2.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.2-1: Data type "ext_key_container" for request to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
keys	array of object schema: keys	***REQUIRED*** Keys "keys" shall be an array of "key".
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
ack_callback_url	string schema: ack_callback_url	Acknowledgements callback URL "ack_callback_url" to which acknowledgement(s) should be sent after all or part of the request completes or fails. For the required asynchronous mode, this parameter is required and shall be specified. If this parameter is omitted the request shall be interpreted as a request to invoke the endpoint in the optional synchronous mode. Otherwise, it shall be interpreted as a request to invoke the endpoint using the required asynchronous mode. Refer to clause 4.2 for the required asynchronous and optional synchronous modes of operation. Before submitting synchronous mode requests without an "ack_callback_url" to a new KME, the "/kmapi/versions" endpoint should be called to confirm that the new KME supports the optional synchronous mode "capability". Where provided, "ack_callback_url" shall be a complete URL starting with the protocol, i.e. starting "https://". The authority may optionally include a port number, in addition to a domain or IP address. *Format:* uri *Maximum length:* 1 024
extension_mandatory	object schema: extension_mandatory	Extension "extension_mandatory" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs shall handle or reject the request with a "503" ('Error of server side response') adding a "details" entry named "unsupported_mandatory_extension". Unless the receiving KME recognizes and implements all requirements within the specification of an "extension_mandatory" extension it shall reject the request. KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

Field Name	Type	Description
extension_optional	object schema: extension_optional	Extension "extension_optional" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs may pass on even when the KME does not support the extension. If a key management entity does not recognize and implement all requirements within the specification of an "extension_optional" extension it shall pass on the "extension_optional" extension without change. KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "ext_key_container" request body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "keys": [
    {
      "key_id": "550e8400-e29b-41d4-a716-446655440000",
      "value": "wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly\u002Bs=",
      "extension": {
        "E32473_extension1": "Some string applicable to a particular key",
        "E32473_extension2": {
          "property1": 10111,
          "property2": "Some text",
          "property3": {
            "subprop1": 21,
            "subprop2": true
          }
        }
      }
    }
  ],
  "initiator_sae_id": "encryptor1",
  "target_sae_ids": [
    "encryptor2"
  ],
  "ack_callback_url": "https://kme1/kmapi/v1/ext_keys/ack",
  "extension_mandatory": {
    "E32473_route_type": "direct",
    "E32473_method": {
      "hybrid_keys": true,
      "primary": "qkd",
      "secondary": "pqc"
    }
  },
  "extension_optional": {
    "E32473_extension1": "Some string applicable to all keys in the container",
    "E32473_module_type": {
      "vendor": "Company ABC",
      "protocol": "BB84",
      "min_version": 2.5
    },
    "E32473_qos_session": "e73d9abe"
  }
}
```

6.2.3 Responses

6.2.3.1 200 Successful response (synchronous mode) application/json

Successful OK response to synchronous mode request. Hosts that do not support the optional synchronous mode will never return this response.

The response body shall match the "ack_containers" schema as an array of objects matching the "ack_container" schema as shown in Table 6.2.3.1-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.3.1-1: Data type "ack_container" used as element of array in 200 response to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
key_id_container	array of object schema: key_id_container	***REQUIRED*** Key ID container "key_id_container" shall be an array of objects, each providing "key_id" and optional "extension" information. *Maximum items:* 1 024
ack_status	string schema: ack_status	***REQUIRED*** Acknowledgement status "ack_status" shall be an enum that describes the status of the acknowledged keys. See "ext_keys/ack" for details of the usage of "ack_status". *Maximum length:* 16
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
message	string schema: ack_message	Acknowledgement message "ack_message" optionally provides further details to expand upon the "ack_status". *Maximum length:* 4 096
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "200" status code "ack_containers" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
[
  {
    "key_id_container": [
      {
        "key_id": "550e8400-e29b-41d4-a716-446655440000",
        "extension": {
          "E32473_extension1": "Some string applicable to a particular key",
          "E32473_extension2": {
            "property1": 10111,
            "property2": {
              "subprop1": 21,
              "subprop2": true
            }
          }
        }
      }
    ],
    "ack_status": "relayed",
    "initiator_sae_id": "encryptor1",
    "target_sae_ids": [
      "encryptor2"
    ],
    "message": "string",
    "extension": {
      "E32473_extension1": "Some string applicable to all keys in the container",
      "E32473_extension2": {
        "property1": 10111,
        "property2": {
          "subprop1": 21,
          "subprop2": true
        }
      }
    }
  }
]
```

6.2.3.2 202 Request accepted response

The response can be given to indicate that the server accepted the request. It is for requests using the asynchronous mode for which it is expected that a response will be given without undue delay after receipt of the request, e.g. without waiting for communications with other KMEs etc.

The response shall not include a body.

6.2.3.3 400 Bad request format response application/json

Used to reflect an error in the format of a request. A KME is not allowed to return response code 400 if there is a possibility any association with a key in the request has been retained by the KME. See clause 4.2 for requirements.

The response body shall match the "problem_details" schema as shown in Table 6.2.3.3-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.3.3-1: Data type "problem_details" for 400 response to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024

Field Name	Type	Description
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: Example response where parameters were missing from the request. An example of "`400`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/missing-parameters",
  "status": 400,
  "title": "missing parameters",
  "details": {
    "missing_parameters": "Required parameters are missing from the API call."
  }
}
```

EXAMPLE 2: An example of "`400`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 400,
  "title": "key routing error",
  "details": {
    "target_sae_id_not_recognized": "KME associated with a target SAE ID is not known."
  }
}
```

EXAMPLE 3: An example of "`400`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 400,
  "title": "key routing error",
  "details": {
    "invalid_routing": "Routing this key to target SAE ID requires directly passing it back to the
calling KME, which is invalid."
  }
}
```

6.2.3.4 401 Unauthorized response application/json

Used to reflect that the request was not successfully authorized. A KME is not allowed to return response code 401 if there is a possibility any association with a key in the request has been retained by the KME. See clause 4.2 for requirements.

The response body shall match the "`problem_details`" schema as shown in Table 6.2.3.4-1 encoded as media type "`application/json`" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.3.4-1: Data type "problem_details" for 401 response to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599

Field Name	Type	Description
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "`401`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/unauthorized",
  "status": 401,
  "title": "unauthorized",
  "details": {
    "unauthorized": "User-supplied certificate is invalid."
  }
}
```

6.2.3.5 408 Request timeout response application/json

Request timeout response. This is most likely to be issued where a complete request is not received and the server terminates the connection after some period of inactivity.

The response body shall match the "`problem_details`" schema as shown in Table 6.2.3.5-1 encoded as media type "`application/json`" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.3.5-1: Data type "problem_details" for 408 response to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties*: 1 *Maximum properties*: 1 024

EXAMPLE: An example of "`408`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/timeout",
  "status": 408,
  "title": "timeout",
  "details": {
    "timeout": "A complete request was not received in time."
  }
}
```

6.2.3.6 503 Error of server side response application/json

The response body shall match the "problem_details" schema as shown in Table 6.2.3.6-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.2.3.6-1: Data type "problem_details" for 503 response to POST /kmapi/v1/ext_keys as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
details	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "details" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "503" status code "problem_details" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/server-side-general-error",
  "status": 503,
  "title": "server side general error",
  "details": {
    "server_side_general_error": "The server encountered a general failure and cannot respond."
  }
}
```

EXAMPLE 2: An example of "503" status code "problem_details" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 503,
  "title": "key routing error",
  "details": {
    "no_route_multiple_targets": "Routes not available to support these multiple target SAEs."
  }
}
```

EXAMPLE 3: An example of "503" status code "problem_details" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 503,
  "title": "key routing error",
  "details": {
    "no_known_route": "KME does not know a route to deliver to a target SAE ID."
  }
}
```

EXAMPLE 4: An example of "503" status code "problem_details" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/insufficient-key-material",
  "status": 503,
  "title": "insufficient key material",
  "details": {
    "insufficient_key_material": "Insufficient key material available to deliver key."
  }
}
```

EXAMPLE 5: Example response from a KME that cannot handle two "extension_mandatory" extensions that were supplied with a request. An example of "503" status code "problem_details" response body for "/kmapi/v1/ext_keys" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/unsupported-mandatory-extension",
  "status": 503,
  "title": "unsupported mandatory extension",
  "details": {
    "unsupported_mandatory_extension": "Mandatory extensions E32473_ext1 and E32473_ext2 cannot be
supported for this request."
  }
}
```

6.3 POST /kmapi/v1/ext_keys/ack: Acknowledge status of keys after API request(s)

6.3.1 Description

Pass an array of one or more key acknowledgement container "ack_container" comprising key IDs associated with a previous call to the "ext_keys" or "ext_keys/void" method by an external KMS. The status of all keys in each container is indicated by the status field. One or more additional "ack_container" can be added to the array if one or more additional status needs to be reported for different sets of keys.

The "ack_status" field shall not take a value other than one of the following: "relayed", "voided", "failed", "failed to void", or "key not present".

- "relayed" indicates the KME has successfully relayed the specified keys to the target(s) in "target_sae_ids".
- "voided" indicates the KME has successfully voided the specified keys so they will not be delivered to applications.
- "failed" indicates that "ext_keys" relay request on the specified keys could not be completed. For example, an "ext_keys" request may not complete if there is insufficient key material for relaying.
- "failed to void" indicates that "ext_keys/void" request on the specified keys could not be completed. For example, an "ext_keys/void" request may not complete if a KME is offline. Note that a "ext_keys/void" request will fail if the specified keys have already been delivered to a requesting SAE.
- "key not present" indicates that the specified keys could not be found by the KME.

In addition to the "status" field, additional information (e.g. explaining a failure) can be included in the optional "message" field.

It is possible that a single "ext_keys" request included multiple keys, where some are successfully delivered and others fail. This can be indicated by a KME including more than one acknowledgement request "ack_container" in the array to specify different statuses for different sets of keys. Multiple calls to "ext_keys/ack" can also be issued, for example to enable faster acknowledgements for some keys than for others.

Keys acknowledged in a call to "ext_keys/ack" do not necessarily arise from a single request to the "ext_keys" or "ext_keys/void" methods. They can be combined in any manner but each key acknowledgement container "ack_container" within the array groups acknowledgements sharing the same "ack_status", "initiator_sae_id" and "target_sae_ids".

Extension fields may also be optionally included (which could be, e.g. metadata for each key to indicate its routing information from the remote KMS). If such extensions are included, the extensions should be handled accordingly (e.g. to keep the provided metadata with each key so it can be relayed to the initiator SAE).

A KME receiving a valid acknowledgement shall return a "200" code, otherwise a status code indicating an error shall be returned including details of the error.

6.3.2 Request Body: application/json

The request body shall match the "ack_containers" schema as an array of objects matching the "ack_container" schema as shown in Table 6.3.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.3.2-1: Data type "ack_container" used as element of array in request to POST /kmapi/v1/ext_keys/ack as application/json

Field Name	Type	Description
key_id_container	array of object schema: key_id_container	***REQUIRED*** Key ID container "key_id_container" shall be an array of objects, each providing "key_id" and optional "extension" information. *Maximum items:* 1 024
ack_status	string schema: ack_status	***REQUIRED*** Acknowledgement status "ack_status" shall be an enum that describes the status of the acknowledged keys. See "ext_keys/ack" for details of the usage of "ack_status". *Maximum length:* 16
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
message	string schema: ack_message	Acknowledgement message "ack_message" optionally provides further details to expand upon the "ack_status". *Maximum length:* 4 096
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "ack_containers" request body for "/kmapi/v1/ext_keys/ack" as media type "application/json":

```
[
  {
    "key_id_container": [
      {
        "key_id": "550e8400-e29b-41d4-a716-446655440000",
        "extension": {
          "E32473_extension1": "Some string applicable to a particular key",
          "E32473_extension2": {
            "property1": 10111,
            "property2": {
              "subprop1": 21,
              "subprop2": true
            }
          }
        }
      }
    ],
    "ack_status": "relayed",
    "initiator_sae_id": "encryptor1",
    "target_sae_ids": [
      "encryptor2"
    ],
    "message": "string",
    "extension": {
      "E32473_extension1": "Some string applicable to all keys in the container",
      "E32473_extension2": {
        "property1": 10111,
        "property2": {
          "subprop1": 21,
          "subprop2": true
        }
      }
    }
  }
]
```

6.3.3 Responses

6.3.3.1 200 Successful OK response

The response shall not include a body.

6.3.3.2 400 Bad request format response application/json

The response body shall match the "problem_details" schema as shown in Table 6.3.3.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.3.3.2-1: Data type "problem_details" for 400 response to POST /kmapi/v1/ext_keys/ack as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
details	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "details" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "400" status code "problem_details" response body for "/kmapi/v1/ext_keys/ack" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/ack-not-understood",
  "status": 400,
  "title": "ack not understood",
  "details": {
    "ack_not_understood": "KME cannot parse the submitted acknowledgement."
  }
}
```

EXAMPLE 2: An example of "400" status code "problem_details" response body for "/kmapi/v1/ext_keys/ack" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-not-present",
  "title": "key not present",
  "details": {
    "key_not_present": "The acknowledged key ID(s) are not known to this KME."
  }
}
```

6.3.3.3 401 Unauthorized response application/json

The response body shall match the "problem_details" schema as shown in Table 6.3.3.3-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.3.3.3-1: Data type "problem_details" for 401 response to POST /kmapi/v1/ext_keys/ack as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
details	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "details" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "401" status code "problem_details" response body for "/kmapi/v1/ext_keys/ack" as media type "application/json":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/unauthorized",
  "status": 401,
  "title": "unauthorized",
  "details": {
    "unauthorized": "User-supplied certificate is invalid."
  }
}
```

6.3.3.4 503 Error of server side response application/json

The response body shall match the "problem_details" schema as shown in Table 6.3.3.4-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.3.3.4-1: Data type "problem_details" for 503 response to POST /kmapi/v1/ext_keys/ack as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "`503`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/ack`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/server-side-general-error",
  "status": 503,
  "title": "server side general error",
  "details": {
    "server_side_general_error": "The server encountered a general failure and cannot respond."
  }
}
```

EXAMPLE 2: An example of "`503`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/ack`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
  "status": 503,
  "title": "key routing error",
  "details": {
    "no_route_multiple_targets": "Routes not available to support these multiple target SAEs."
  }
}
```

6.4 POST /kmapi/v1/ext_keys/void: Signal keys as void to external KMS (i.e. discard keys)

6.4.1 Description

Pass a "`void_container`" void request container comprising key IDs to another KME, for the key(s) to be marked as void (i.e. discarded and not delivered to SAEs). The "`void_container`" void request container contains keys matching those already passed to the KME.

Upon a valid request, for the required asynchronous mode, a KME shall discard keys relating to the provided key IDs and post a call to the specified "`ack_callback_url`" describing the completed operation. After this, the KMEs shall not use the impacted key(s) including rejecting any application requests to retrieve them.

If synchronous mode is implemented in addition to the asynchronous mode, the KME shall not use "`ext_keys/ack`" to acknowledge outcomes for keys from synchronous mode requests ("`ack_callback_url`" would have been omitted from these requests). Instead, the KME shall respond with a "`200`" ('Successful OK').

A "`400`" code shall be returned if the request is known to be invalid without further investigation. Otherwise, failures to void keys can be reported subsequently via "`ack_callback_url`".

If this operation is requested with an empty "key_ids" array, all keys shared between the pre-specified SAEs shall be voided. To reduce the risk of accidental key loss, the KME shall reject any void request that does not specify one or more "key_id" unless the "all_confirmation" parameter is included in the "void_container" with a value of true. In the case of such a rejection the KME shall return a "400" code.

6.4.2 Request Body: application/json

The request body shall match the "void_container" schema as shown in Table 6.4.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.2-1: Data type "void_container" for request to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
key_ids	array of string schema: key_ids	***REQUIRED*** Key IDs "key_ids" shall be an array of "key_id".
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
ack_callback_url	string schema: ack_callback_url	Acknowledgements callback URL "ack_callback_url" to which acknowledgement(s) should be sent after all or part of the request completes or fails. For the required asynchronous mode, this parameter is required and shall be specified. If this parameter is omitted the request shall be interpreted as a request to invoke the endpoint in the optional synchronous mode. Otherwise, it shall be interpreted as a request to invoke the endpoint using the required asynchronous mode. Refer to clause 4.2 for the required asynchronous and optional synchronous modes of operation. Before submitting synchronous mode requests without an "ack_callback_url" to a new KME, the "/kmapi/versions" endpoint should be called to confirm that the new KME supports the optional synchronous mode "capability". Where provided, "ack_callback_url" shall be a complete URL starting with the protocol, i.e. starting "https://". The authority may optionally include a port number, in addition to a domain or IP address. *Format:* uri *Maximum length:* 1 024
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

Field Name	Type	Description
all_confirmation	boolean schema: all_confirmation	"all_confirmation" acts as a confirmation flag that shall be included with a value of "true" to confirm that voiding of all keys between the specified SAEs is intended when no key IDs are specified in a call to "ext_keys/void". If "all_confirmation" is omitted or its value is not "true" when no key IDs are specified in a call to "ext_keys/void" the response shall be "400" ('Bad request format') with a string named "no_all_confirmation" within "details" in the response.

EXAMPLE: An example of "void_container" request body for "/kmapi/v1/ext_keys/void" as media type "application/json":

```
{
  "key_ids": [
    "550e8400-e29b-41d4-a716-446655440000",
    "373b0b2c-d841-4765-af6f-c6232cda6531",
    "65380d2d-7de4-4445-9a1e-23f77218e61d"
  ],
  "initiator_sae_id": "encryptor1",
  "target_sae_ids": [
    "encryptor2"
  ],
  "ack_callback_url": "https://kme1/kmapi/v1/ext_keys/ack",
  "extension": {
    "E32473_extension1": "Some string",
    "E32473_extension2": {
      "property1": 10111,
      "property2": "Some text",
      "property3": {
        "subprop1": 21,
        "subprop2": true
      }
    }
  }
}
```

6.4.3 Responses

6.4.3.1 200 Successful response (synchronous mode) application/json

Successful OK response to synchronous mode request. Hosts that do not support the optional synchronous mode will never return this response.

The response body shall match the "ack_containers" schema as an array of objects matching the "ack_container" schema as shown in Table 6.4.3.1-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.3.1-1: Data type "ack_container" used as element of array in 200 response to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
key_id_container	array of object schema: key_id_container	***REQUIRED*** Key ID container "key_id_container" shall be an array of objects, each providing "key_id" and optional "extension" information. *Maximum items:* 1 024
ack_status	string schema: ack_status	***REQUIRED*** Acknowledgement status "ack_status" shall be an enum that describes the status of the acknowledged keys. See "ext_keys/ack" for details of the usage of "ack_status". *Maximum length:* 16

Field Name	Type	Description
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
message	string schema: ack_message	Acknowledgement message "ack_message" optionally provides further details to expand upon the "ack_status". *Maximum length:* 4 096
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "200" status code "ack_containers" response body for "/kmapi/v1/ext_keys/void" as media type "application/json":

```
[
  {
    "key_id_container": [
      {
        "key_id": "550e8400-e29b-41d4-a716-446655440000",
        "extension": {
          "E32473_extension1": "Some string applicable to a particular key",
          "E32473_extension2": {
            "property1": 10111,
            "property2": {
              "subprop1": 21,
              "subprop2": true
            }
          }
        }
      }
    ],
    "ack_status": "voided",
    "initiator_sae_id": "encryptor1",
    "target_sae_ids": [
      "encryptor2"
    ],
    "message": "string",
    "extension": {
      "E32473_extension1": "Some string applicable to all keys in the container",
      "E32473_extension2": {
        "property1": 10111,
        "property2": {
          "subprop1": 21,
          "subprop2": true
        }
      }
    }
  }
]
```

6.4.3.2 202 Request accepted response

The response can be given to indicate that the server accepted the request. It is for requests using the asynchronous mode for which it is expected that a response will be given without undue delay after receipt of the request, e.g. without waiting for communications with other KMEs etc.

The response shall not include a body.

6.4.3.3 400 Bad request format response application/json

The response body shall match the "problem_details" schema as shown in Table 6.4.3.3-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.3.3-1: Data type "problem_details" for 400 response to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024

Field Name	Type	Description
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "`400`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/void`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/void-request-not-understood",
  "status": 400,
  "title": "void request not understood",
  "details": {
    "key_voiding_error": "KME cannot parse the submitted void request."
  }
}
```

EXAMPLE 2: An example of "`400`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/void`" as media type "`application/json`":

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/no-key-ids-or-confirmation",
  "title": "no key ids or confirmation",
  "details": {
    "no_key_ids_or_confirmation": "When no key_ids are passed, all keys shared between the SAEs
will be voided. If this is the intended action, the all_confirmation field also needs to be set with
a value of true. Otherwise, please specify key_ids to be voided."
  }
}
```

6.4.3.4 401 Unauthorized response application/json

The response body shall match the "`problem_details`" schema as shown in Table 6.4.3.4-1 encoded as media type "`application/json`" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.3.4-1: Data type "problem_details" for 401 response to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "401" status code "problem_details" response body for "/kmapi/v1/ext_keys/void" as media type "application/json":

```
{
   "type": "https://qkd.etsi.org/gs020-interop-kms/unauthorized",
   "status": 401,
   "title": "unauthorized",
   "details": {
     "unauthorized": "User-supplied certificate is invalid."
   }
 }
```

6.4.3.5 408 Request timeout response application/json

Request timeout response. This is most likely to be issued where a complete request is not received and the server terminates the connection after some period of inactivity.

The response body shall match the "problem_details" schema as shown in Table 6.4.3.5-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.3.5-1: Data type "problem_details" for 408 response to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
details	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "details" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "408" status code "problem_details" response body for "/kmapi/v1/ext_keys/void" as media type "application/json":

```
{
   "type": "https://qkd.etsi.org/gs020-interop-kms/timeout",
   "status": 408,
   "title": "timeout",
   "details": {
     "timeout": "A complete request was not received in time."
   }
 }
```

6.4.3.6 503 Error of server side response application/json

The response body shall match the "problem_details" schema as shown in Table 6.4.3.6-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 6.4.3.6-1: Data type "problem_details" for 503 response to POST /kmapi/v1/ext_keys/void as application/json

Field Name	Type	Description
`type`	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
`status`	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
`title`	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
`details`	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "`details`" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE 1: An example of "`503`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/void`" as media type "`application/json`":

```
{
   "type": "https://qkd.etsi.org/gs020-interop-kms/server-side-general-error",
   "status": 503,
   "title": "server side general error",
   "details": {
     "server_side_general_error": "The server encountered a general failure and cannot respond."
   }
 }
```

EXAMPLE 2: An example of "`503`" status code "`problem_details`" response body for "`/kmapi/v1/ext_keys/void`" as media type "`application/json`":

```
{
   "type": "https://qkd.etsi.org/gs020-interop-kms/key-routing-error",
   "status": 503,
   "title": "key routing error",
   "details": {
     "no_route_multiple_targets": "Routes not available to support these multiple target SAEs."
   }
 }
```

7 Schemas

7.1 ack_callback_url

Acknowledgements callback URL "`ack_callback_url`" to which acknowledgement(s) should be sent after all or part of the request completes or fails.

For the required asynchronous mode, this parameter is required and shall be specified.

If this parameter is omitted the request shall be interpreted as a request to invoke the endpoint in the optional synchronous mode. Otherwise, it shall be interpreted as a request to invoke the endpoint using the required asynchronous mode. Refer to clause 4.2 for the required asynchronous and optional synchronous modes of operation.

Before submitting synchronous mode requests without an "`ack_callback_url`" to a new KME, the "`/kmapi/versions`" endpoint should be called to confirm that the new KME supports the optional synchronous mode "`capability`".

Where provided, "`ack_callback_url`" shall be a complete URL starting with the protocol, i.e. starting "`https://`". The authority may optionally include a port number, in addition to a domain or IP address.

The data format shall be a string.

Format: uri

Maximum length: 1 024

EXAMPLE: An example of "ack_callback_url" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"https://kme1/kmapi/v1/ext_keys/ack"
```

7.2 ack_container

Acknowledgements container "ack_container" through which results shall be reported once the requested operations have completed.

The data format shall be as specified in Table 7.2-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.2-1: Data type of "ack_container"

Field Name	Type	Description
key_id_container	array of object schema: key_id_container	***REQUIRED*** Key ID container "key_id_container" shall be an array of objects, each providing "key_id" and optional "extension" information. *Maximum items:* 1 024
ack_status	string schema: ack_status	***REQUIRED*** Acknowledgement status "ack_status" shall be an enum that describes the status of the acknowledged keys. See "ext_keys/ack" for details of the usage of "ack_status". *Maximum length:* 16
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
message	string schema: ack_message	Acknowledgement message "ack_message" optionally provides further details to expand upon the "ack_status". *Maximum length:* 4 096
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "ack_container" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "key_id_container": [
    {
      "key_id": "550e8400-e29b-41d4-a716-446655440000",
      "extension": {
        "E32473_extension1": "Some string applicable to a particular key",
        "E32473_extension2": {
          "property1": 10111,
          "property2": {
            "subprop1": 21,
            "subprop2": true
          }
        }
      }
    }
  ],
  "ack_status": "relayed",
  "initiator_sae_id": "encryptor1",
  "target_sae_ids": [
    "encryptor2"
  ],
  "message": "string",
  "extension": {
    "E32473_extension1": "Some string applicable to all keys in the container",
    "E32473_extension2": {
      "property1": 10111,
      "property2": {
        "subprop1": 21,
        "subprop2": true
      }
    }
  }
}
```

7.3 ack_containers

Acknowledgements container "ack_containers" shall be an array of acknowledgements containers "ack_container". An "ack_containers" can be used to acknowledge keys with different statuses, SAE IDs etc., whereas keys in a single "ack_container" share some common properties.

The data format shall be an array.

Maximum items: 1 024

EXAMPLE: An example of "ack_containers" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:


```
[
  {
    "key_id_container": [
      {
        "key_id": "550e8400-e29b-41d4-a716-446655440000",
        "extension": {
          "E32473_extension1": "Some string applicable to a particular key",
          "E32473_extension2": {
            "property1": 10111,
            "property2": {
              "subprop1": 21,
              "subprop2": true
            }
          }
        }
      }
    ],
    "ack_status": "relayed",
    "initiator_sae_id": "encryptor1",
    "target_sae_ids": [
      "encryptor2"
    ],
    "message": "string",
    "extension": {
      "E32473_extension1": "Some string applicable to all keys in the container",
      "E32473_extension2": {
        "property1": 10111,
        "property2": {
          "subprop1": 21,
          "subprop2": true
        }
      }
    }
  }
]
```


7.4 ack_message

Acknowledgement message "ack_message" optionally provides further details to expand upon the "ack_status".

The data format shall be a string.

Maximum length: 4 096

7.5 ack_status

Acknowledgement status "ack_status" shall be an enum that describes the status of the acknowledged keys. See "ext_keys/ack" for details of the usage of "ack_status".

"ack_status" shall not take a value other than one of the following: "relayed", "failed", "voided", "failed to void", "key not present".

The data format shall be a string.

Maximum length: 16

EXAMPLE: An example of "ack_status" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:


```
"relayed"
```

7.6 all_confirmation

"all_confirmation" acts as a confirmation flag that shall be included with a value of "true" to confirm that voiding of all keys between the specified SAEs is intended when no key IDs are specified in a call to "ext_keys/void".

If "all_confirmation" is omitted or its value is not "true" when no key IDs are specified in a call to "ext_keys/void" the response shall be "400" ('Bad request format') with a string named "no_all_confirmation" within "details" in the response.

The data format shall be a boolean.

EXAMPLE: An example of "all_confirmation" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
true
```

7.7 capabilities

Capabilities "capabilities" shall be an array of strings each indicating a "capability" supported by the host.

Where a host supports the optional synchronous mode it should report this capability of "synchronous_mode". Clients should confirm this capability before sending requests for synchronous mode.

The data format shall be an array.

Maximum items: 1 000

EXAMPLE: An example of "capabilities" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  "synchronous_mode"
]
```

7.8 capability

Capability "capability" shall be a string that indicates a capability supported by the host.

The data format shall be a string.

Maximum length: 128

EXAMPLE: An example of "capability" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"synchronous_mode"
```

7.9 ext_key_container

Extended key request container "ext_key_container" is a container used in calls to the "/kmapi/v1/ext_keys" endpoint.

The data format shall be as specified in Table 7.9-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.9-1: Data type of "ext_key_container"

Field Name	Type	Description
keys	array of object schema: keys	***REQUIRED*** Keys "keys" shall be an array of "key".
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
ack_callback_url	string schema: ack_callback_url	Acknowledgements callback URL "ack_callback_url" to which acknowledgement(s) should be sent after all or part of the request completes or fails. For the required asynchronous mode, this parameter is required and shall be specified. If this parameter is omitted the request shall be interpreted as a request to invoke the endpoint in the optional synchronous mode. Otherwise, it shall be interpreted as a request to invoke the endpoint using the required asynchronous mode. Refer to clause 4.2 for the required asynchronous and optional synchronous modes of operation. Before submitting synchronous mode requests without an "ack_callback_url" to a new KME, the "/kmapi/versions" endpoint should be called to confirm that the new KME supports the optional synchronous mode "capability". Where provided, "ack_callback_url" shall be a complete URL starting with the protocol, i.e. starting "https://". The authority may optionally include a port number, in addition to a domain or IP address. *Format:* uri *Maximum length:* 1 024
extension_mandatory	object schema: extension_mandatory	Extension "extension_mandatory" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs shall handle or reject the request with a "503" ('Error of server side response') adding a "details" entry named "unsupported_mandatory_extension". Unless the receiving KME recognizes and implements all requirements within the specification of an "extension_mandatory" extension it shall reject the request. KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

Field Name	Type	Description
extension_optional	object schema: extension_optional	Extension "extension_optional" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs may pass on even when the KME does not support the extension. If a key management entity does not recognize and implement all requirements within the specification of an "extension_optional" extension it shall pass on the "extension_optional" extension without change. KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "ext_key_container" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "keys": [
    {
      "key_id": "550e8400-e29b-41d4-a716-446655440000",
      "value": "wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly\u002Bs=",
      "extension": {
        "E32473_extension1": "Some string applicable to a particular key",
        "E32473_extension2": {
          "property1": 10111,
          "property2": "Some text",
          "property3": {
            "subprop1": 21,
            "subprop2": true
          }
        }
      }
    }
  ],
  "initiator_sae_id": "encryptor1",
  "target_sae_ids": [
    "encryptor2"
  ],
  "ack_callback_url": "https://kme1/kmapi/v1/ext_keys/ack",
  "extension_mandatory": {
    "E32473_route_type": "direct",
    "E32473_method": {
      "hybrid_keys": true,
      "primary": "qkd",
      "secondary": "pqc"
    }
  },
  "extension_optional": {
    "E32473_extension1": "Some string applicable to all keys in the container",
    "E32473_module_type": {
      "vendor": "Company ABC",
      "protocol": "BB84",
      "min_version": 2.5
    },
    "E32473_qos_session": "e73d9abe"
  }
}
```

7.10 extension

Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements.

Further requirements are given in clause 4.5.

The data format shall be a object.

Minimum properties: 1

Maximum properties: 1 024

EXAMPLE: An example of "extension" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "E32473_extension1": "Some string",
  "E32473_extension2": {
    "property1": 10111,
    "property2": "Some text",
    "property3": {
      "subprop1": 21,
      "subprop2": true
    }
  }
}
```

7.11 extension_mandatory

Extension "extension_mandatory" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs shall handle or reject the request with a "503" ('Error of server side response') adding a "details" entry named "unsupported_mandatory_extension".

Unless the receiving KME recognizes and implements all requirements within the specification of an "extension_mandatory" extension it shall reject the request.

KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension.

Further requirements are given in clause 4.5.

The data format shall be a object.

Minimum properties: 1

Maximum properties: 1 024

EXAMPLE: An example of "extension_mandatory" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "E32473_module_type": {
    "vendor": "Company ABC",
    "protocol": "BB84",
    "min_version": 2.5
  },
  "E32473_qos_session": "e73d9abe"
}
```

7.12 extension_optional

Extension "extension_optional" shall be an object containing any number of properties that each represent an extension specified by ETSI or another organization (such as a vendor) that KMEs may pass on even when the KME does not support the extension.

If a key management entity does not recognize and implement all requirements within the specification of an "extension_optional" extension it shall pass on the "extension_optional" extension without change.

KMEs shall follow the specification of any extension it claims to have implemented for a request. Where relevant, this includes passing on without change any information indicated to be immutable in the specification of the extension.

Further requirements are given in clause 4.5.

The data format shall be a object.

Minimum properties: 1

Maximum properties: 1 024

EXAMPLE: An example of "extension_optional" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "E32473_module_type": {
    "vendor": "Company ABC",
    "protocol": "BB84",
    "min_version": 2.5
  },
  "E32473_qos_session": "e73d9abe"
}
```

7.13 initiator_sae_id

Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys".

A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id".

SAE IDs shall follow the requirements under clause 4.6 in the main description.

The data format shall be a string.

Minimum length: 1

Maximum length: 64

EXAMPLE: An example of "initiator_sae_id" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"encryptor1"
```

7.14 key_id

Key ID "key_id" is the ID of the key and shall be a UUID as specified in IETF RFC 9562 [8], clause 4.

The data format shall be a string.

Format: uuid

EXAMPLE: An example of "key_id" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"550e8400-e29b-41d4-a716-446655440000"
```

7.15 key_id_container

Key ID container "key_id_container" shall be an array of objects, each providing "key_id" and optional "extension" information.

The data format shall be an array of objects as specified in Table 7.15-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Maximum items: 1 024

Table 7.15-1: Data type for an element of "key_id_container" array

Field Name	Type	Description
key_id	string schema: key_id	***REQUIRED*** Key ID "key_id" is the ID of the key and shall be a UUID as specified in IETF RFC 9562 [8], clause 4. *Format:* uuid
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "key_id_container" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  {
    "key_id": "550e8400-e29b-41d4-a716-446655440000",
    "extension": {
      "E32473_vendor": "Company ABC",
      "E32473_protocol": "BB84"
    }
  }
]
```

7.16 key_ids

Key IDs "key_ids" shall be an array of "key_id".

The data format shall be an array.

EXAMPLE: An example of "key_ids" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  "550e8400-e29b-41d4-a716-446655440000",
  "373b0b2c-d841-4765-af6f-c6232cda6531",
  "65380d2d-7de4-4445-9a1e-23f77218e61d"
]
```

7.17 key

Key "key" shall be an object containing a "key_id" and its "value" with an optional "extension" dictionary.

The data format shall be as specified in Table 7.17-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.17-1: Data type of "key"

Field Name	Type	Description
key_id	string schema: key_id	***REQUIRED*** Key ID "key_id" is the ID of the key and shall be a UUID as specified in IETF RFC 9562 [8], clause 4. *Format:* uuid
value	string schema: value	***REQUIRED*** Value "value" shall be key data encoded by the base64 data encoding scheme specified in IETF RFC 4648 [7], using the alphabet in table 1. Implementations shall ensure that padding used in the base64 data encoding scheme is never used as key material. This includes the zero, two, or one "=" padding characters at the end of the final encoded unit of output where the final quantum of encoding input is exactly 24 bits, 8 bits, or 16 bits, respectively. When non-integer-byte-size keys are used, it is essential to strip any padding bits with value zero that were added (on the right) when decoding. It is not safe to strip all bits with value zero from the end of the decoded key since this can bias keys. Decoding needs to make use of independent knowledge of the requested key size to correctly strip such padding to recover a valid key. (The base64 data encoding scheme and the "=" padding character rules it includes can only indicate the size of the encoding input in integer byte sizes. The final character of the encoded output or the final character before the first "=" padding character can include information from padding bits with value zero that were added during encoding in the case of non-integer-byte-size keys.) Support for non-integer-byte-size keys is optional and vendors may choose to support only integer byte sizes.
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "key" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "key_id": "550e8400-e29b-41d4-a716-446655440000",
  "value": "wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly\\u002Bs=",
  "extension": {
    "E32473_protocol": "BB84"
  }
}
```

7.18 keys

Keys "keys" shall be an array of "key".

The data format shall be an array.

EXAMPLE: An example of "keys" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  {
    "key_id": "550e8400-e29b-41d4-a716-446655440000",
    "value": "wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly\\u002Bs=",
    "extension": {
      "E32473_protocol": "BB84"
    }
  }
]
```

7.19 problem_details

Problem Details "problem_details" shall be an object satisfying the requirements in IETF RFC 9457 [9] "Problem Details for HTTP APIs" [9].

The data format shall be as specified in Table 7.19-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.19-1: Data type of "problem_details"

Field Name	Type	Description
type	string	***REQUIRED*** A URI reference that identifies the problem type. *Format:* uri *Maximum length:* 1 024
status	integer	The HTTP status code generated by the origin server for this occurrence of the problem. *Format:* int32 *Minimum:* 100 *Maximum:* 599
title	string	A short, human-readable summary of the problem type. *Maximum length:* 1 024
details	object	Object containing one or a number of string properties to convey details about the problem. Typically, the name of each string property will identify the topic with the string value providing more detailed information. "details" can be deserialized to a dictionary of strings. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "problem_details" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "type": "https://qkd.etsi.org/gs020-interop-kms/invalid-parameter",
  "status": 400,
  "title": "Invalid parameter format",
  "details": {
    "malformed_property": "Property xxxx_xxxx does not match the expected format",
    "invalid_character": "Property xxxx_xxxx contains an invalid character"
  }
}
```

7.20 target_sae_id

Target SAE ID "target_sae_id" shall be the ID of an SAE with which an initiator SAE originally requested to share key(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys".

A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id".

SAE IDs shall follow the requirements under clause 4.6 in the main description.

The data format shall be a string.

Minimum length: 1

Maximum length: 64

EXAMPLE: An example of "target_sae_id" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"encryptor2"
```

7.21 target_sae_ids

"target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key.

KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.

The data format shall be an array.

EXAMPLE: An example of "target_sae_ids" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  "encryptor2",
  "encryptor3"
]
```

7.22 value

Value "value" shall be key data encoded by the base64 data encoding scheme specified in IETF RFC 4648 [7], using the alphabet in table 1.

Implementations shall ensure that padding used in the base64 data encoding scheme is never used as key material. This includes the zero, two, or one "=" padding characters at the end of the final encoded unit of output where the final quantum of encoding input is exactly 24 bits, 8 bits, or 16 bits, respectively.

When non-integer-byte-size keys are used, it is essential to strip any padding bits with value zero that were added (on the right) when decoding. It is not safe to strip all bits with value zero from the end of the decoded key since this can bias keys. Decoding needs to make use of independent knowledge of the requested key size to correctly strip such padding to recover a valid key. (The base64 data encoding scheme and the "=" padding character rules it includes can only indicate the size of the encoding input in integer byte sizes. The final character of the encoded output or the final character before the first "=" padding character can include information from padding bits with value zero that were added during encoding in the case of non-integer-byte-size keys.)

Support for non-integer-byte-size keys is optional and vendors may choose to support only integer byte sizes.

The data format shall be a string.

EXAMPLE: An example of "value" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"wHHVxRwDJs3/bXd38GHP3oe4svTuRpZS0yCC7x4Ly\u002Bs="
```

7.23 version

Version "version" shall be a string indicating a version of the API supported by the host.

The data format shall be a string.

Maximum length: 16

EXAMPLE: An example of "version" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
"v1"
```

7.24 version_container

Version container "version_container" shall be an object to indicate supported "versions" and "capabilities" of a host along with optional "extension" information.

The data format shall be as specified in Table 7.24-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.24-1: Data type of "version_container"

Field Name	Type	Description
versions	array of string schema: versions	***REQUIRED*** Versions "versions" shall be an array of supported API versions "version". *Maximum items:* 1 000
capabilities	array of string schema: capabilities	Capabilities "capabilities" shall be an array of strings each indicating a "capability" supported by the host. Where a host supports the optional synchronous mode it should report this capability of "synchronous_mode". Clients should confirm this capability before sending requests for synchronous mode. *Maximum items:* 1 000
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024

EXAMPLE: An example of "version_container" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "versions": [
    "v1"
  ],
  "capabilities": [
    "synchronous_mode"
  ]
}
```

7.25 versions

Versions "versions" shall be an array of supported API versions "version".

The data format shall be an array.

Maximum items: 1 000

EXAMPLE: An example of "versions" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
[
  "v1"
]
```

7.26 void_container

Void request container "void_container" shall be an object used in the "ext_keys/void" endpoint to request keys to be voided.

The data format shall be as specified in Table 7.26-1 encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2].

Table 7.26-1: Data type of "void_container"

Field Name	Type	Description
key_ids	array of string schema: key_ids	***REQUIRED*** Key IDs "key_ids" shall be an array of "key_id".
initiator_sae_id	string schema: initiator_sae_id	***REQUIRED*** Initiator SAE ID "initiator_sae_id" shall be the ID of the SAE that originally requested to share key(s) with one or more target SAE(s). It can be used to refer to key(s) in "ext_keys". It can also be used in "ext_keys/void" or "ext_keys/ack" to refer to key(s) from a previous call (or calls) to "ext_keys". A KME can act as an (internal) SAE and request the sharing of key(s) with other KME(s) that are also acting as SAE(s). If no distinction between the initiator SAE and target SAE(s) applies to the origin of a request, an implementation may arbitrarily assign one as "initiator_sae_id" and other(s) "target_sae_id". SAE IDs shall follow the requirements under clause 4.6 in the main description. *Maximum length:* 64
target_sae_ids	array of string schema: target_sae_ids	***REQUIRED*** "target_sae_ids" shall be an array of "target_sae_id" that are the IDs of target SAEs relevant to a request. A single target or multiple targets may be specified. Each target SAE in addition to the initiator SAE are the SAEs eligible to receive an identical copy of a key. KMEs may support only one target SAE (a single SAE ID in the array) or limit the maximum number of target SAEs supported. Requests exceeding the capabilities of available KMEs may be rejected with a "503" ('Error of server side') response code.
ack_callback_url	string schema: ack_callback_url	Acknowledgements callback URL "ack_callback_url" to which acknowledgement(s) should be sent after all or part of the request completes or fails. For the required asynchronous mode, this parameter is required and shall be specified. If this parameter is omitted the request shall be interpreted as a request to invoke the endpoint in the optional synchronous mode. Otherwise, it shall be interpreted as a request to invoke the endpoint using the required asynchronous mode. Refer to clause 4.2 for the required asynchronous and optional synchronous modes of operation. Before submitting synchronous mode requests without an "ack_callback_url" to a new KME, the "/kmapi/versions" endpoint should be called to confirm that the new KME supports the optional synchronous mode "capability". Where provided, "ack_callback_url" shall be a complete URL starting with the protocol, i.e. starting "https://". The authority may optionally include a port number, in addition to a domain or IP address. *Format:* uri *Maximum length:* 1 024
extension	object schema: extension	Extension "extension" shall be an object containing any number of properties that each represents an extension specified by ETSI or another organization (such as a vendor). KMEs may choose to implement a set of extensions. Specifications of extensions can include requirements and KMEs that implement an extension shall comply with such requirements. Further requirements are given in clause 4.5. *Minimum properties:* 1 *Maximum properties:* 1 024
all_confirmation	boolean schema: all_confirmation	"all_confirmation" acts as a confirmation flag that shall be included with a value of "true" to confirm that voiding of all keys between the specified SAEs is intended when no key IDs are specified in a call to "ext_keys/void". If "all_confirmation" is omitted or its value is not "true" when no key IDs are specified in a call to "ext_keys/void" the response shall be "400" ('Bad request format') with a string named "no_all_confirmation" within "details" in the response.

EXAMPLE: An example of "void_container" encoded as media type "application/json" in JSON format as specified in IETF RFC 8259 [2]:

```
{
  "key_ids": [
    "550e8400-e29b-41d4-a716-446655440000",
    "373b0b2c-d841-4765-af6f-c6232cda6531",
    "65380d2d-7de4-4445-9a1e-23f77218e61d"
  ],
  "initiator_sae_id": "encryptor1",
  "target_sae_ids": [
    "encryptor2"
  ],
  "ack_callback_url": "https://kme1/kmapi/v1/ext_keys/ack",
  "extension": {
    "E32473_extension1": "Some string",
    "E32473_extension2": {
      "property1": 10111,
      "property2": "Some text",
      "property3": {
        "subprop1": 21,
        "subprop2": true
      }
    }
  }
}
```

Annex A (normative): OpenAPI representation

The latest version of an OpenAPI representation of the API in the present document is available at the following URL:

- https://forge.etsi.org/rep/qkd/gs020-interop-kms

And the version associated with this V1.1.1 publication will remain available at:

- https://forge.etsi.org/rep/qkd/gs020-interop-kms/-/tree/v1.1.1

This OpenAPI representation may be considered equivalent to the specification in the present document when interpreted according to the "OpenAPI Specification v3.1.2" specified in [10]. The present document was largely generated from the OpenAPI representation.

History

Version	Date	Status
V1.1.1	June 2026	Publication