ETSI GS QKD 004 V1.1.1 (2010-12)

Group Specification

Quantum Key Distribution (QKD); Application Interface

Disclaimer

This 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-0004_APPLINTF

Keywords
quantum cryptography, Quantum Key Distribution, use case

ETSI

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

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

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

Contents

Intellectual Property Rights

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

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

Foreword

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

1 Scope

The present document is intended to describe the interface between security applications and a QKD key management layer, which is an additional layer that sits between the QKD systems and various applications.

Key Management in general, covers the exchange, storage, protection, use, identification, installation, replacement and destruction of cryptographic keys. A QKD system may provide keys for a Key Management System.

QKD, like most key distribution protocols, requires a distributed key management process that operates in a symmetric (vs. server/client) mode. So both key management peers shall negotiate and verify all reservations and allocations.

The QKD protocol generates a pool of ordered secure bits. The function of this key management layer is to demultiplex these bits into separate, ordered groups, where each group is used independently by applications and thus shall be synchronized between the two communication end points. By synchronized we mean that a group of secure bits reserved at one communication end point are identical to the associated group at the other communication end point. It is also required that these same secure bits are then discarded by this layer once they are used and never revealed to anyone else.

2 References

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

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

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

2.1 Normative references

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

Not applicable.

2.2 Informative references

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

[i.1] OMG IDL Syntax and Semantics, Object Management Group Inc., formal/02-06-39 (CORBA 3.0 - OMG IDL Syntax and Semantics chapter).

NOTE: Available at http://www.omg.org/cgi-bin/doc?formal/02-06-39.

3 Definitions and Abbreviations

3.1 Definitions

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

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

Key Management Interface: interface between an application and a Key Management Layer

Key Management Interoperability Protocol (KMIP): protocol for the communication between enterprise key management systems and encryption systems

NOTE: The KMIP is directed by the OASIS initiative.

Key Management Layer: abstraction in a layered model including physically distributed key management systems, e.g. on two network nodes connected with a QKD Link

NOTE: The Key Management Layer sits between the QKD Link and various applications.

Key Management System: part of a cryptography system managing the exchange, storage, protection, use, identification, installation, replacement and destruction of cryptographic keys

Link Encryptor: device performing link encryption, i.e. the communication security process of encrypting information between two peers on the data link level

Organization for the Advancement of Structured Information Standards (OASIS): global consortium that drives the development, convergence and adoption of e-business and web service standards, incluing KMIP

QKD Link, QKD System: pair of two QKD Modules, interconnected by a quantum channel and a classical channel

QKD Module: set of hardware and software components that implements cryptographic functions and quantum optical processes, including cryptographic algorithms and protocols and key generation, and is contained within a defined cryptographic boundary

QKD protocol: list of steps that have to be performed in a QKD module to distill a secure key out of the measurement data of the underlying quantum optical subsystem

Quality of Service (QoS): ability to provide different priority to different applications or users of a QKD system.

Transport Layer Security (TLS): cryptographic protocols used to encrypt the segments of network connections above the Transport Layer, using symmetric cryptography for privacy and a keyed message authentication code for message reliability

3.2 Abbreviations

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

API	Application Program Interface
APPA	Application A
APPB	Application B
KMIP	Key Management Interoperability Protocol
OASIS	Organization for the Advancement of Structured Information Standards
QKD	Quantum Key Distribution
QoS	Quality of Service
TLS	Transport Layer Security

4 Introduction to the QKD Key Management Layer

The QKD key management layer is an additional layer that sits between the QKD protocol and various security applications. This is a distributed process and shall operate in a symmetric (vs. server/client) mode. So all key management peers shall negotiate and verify all reservations and allocations that are associated with them. The QKD protocol generates a pool of ordered secure bits. The function of this key management layer is to demultiplex these bits into separate, ordered groups, where each group is used independently by applications and thus shall be synchronized between the two ends of the communication end points. By synchronized we mean that a group of secure bits reserved at one end point are identical to the associated group at the other end point. It is also required that these same secure bits are then discarded by this layer once they are used (delivered to the application) and never revealed to anyone else.

The Key Management Interface describes the interface of the applications with this layer.

It is assumed that applications that call upon the services of this layer do so via the application programming interfaces (APIs) specified in the following clauses and that this interchange is accomplished within the security perimeter of the QKD System. Manufacturers shall supply and implement these API function calls when a key management layer is provided. Manufacturers may provide additional APIs and expanded functionality as they deem fit. Furthermore this layer is not required if the manufacturer is using the QKD system in a dedicated single application, such as a link encryptor product where they are the only user.

5 QKD Application Interface Specification Description

The QKD key management layer shall demultiplex the ordered QKD pool of secure bits into separate independent groups that are synchronized at both ends of the QKD link and pass those groups to their associated applications. This requires that each local key manager shall communicate with its peer at the other end of the QKD link to perform this service. In addition, some communication between the peer applications may also be necessary to establish a common key association. Communication between the local applications and the local QKD key manager is assumed to occur within the security perimeter of the local system, as shown in the Figure 1.

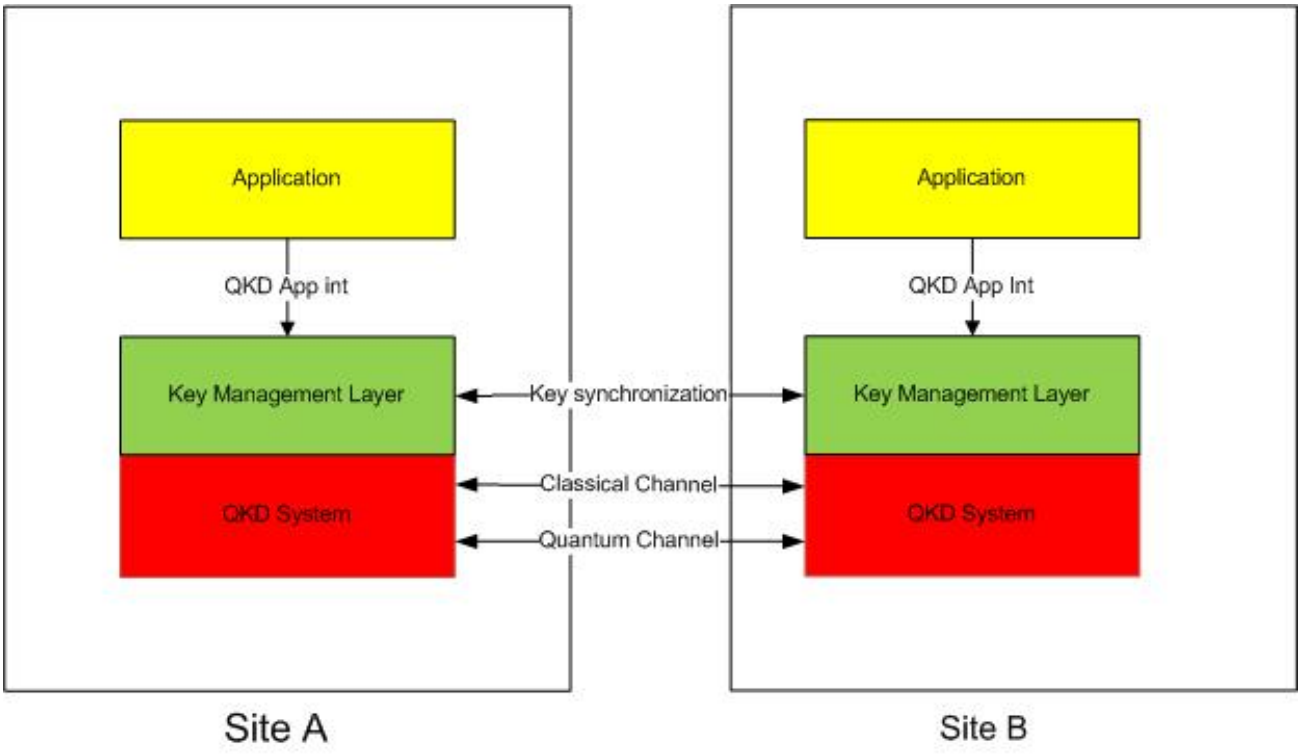

Figure 1: QKD Application Interface and peer relationship

Scenario for a QKD network with multiple QKD links per node is described in Annex C.

6 QKD Application Interface API Specification

The QKD key provider mode must provide the following API functions.

Name	Description
QKD_OPEN	Reserve an association (key_handle) to a set of future keys at both ends of the QKD link through this distributed Key Management Layer and establish a set of parameters that define the expected levels of key service. This function shall return immediately and not block.
QKD_CONNECT_NONBLOCK	Verifies that the QKD link is available and the key_handle association is synchronized at both ends of the link. This function shall not block and returns immediately indicating that both sides of the link have rendezvoused or an error has occurred.
QKD_CONNECT_BLOCKING	Verifies that the QKD link is available and the key_handle association is synchronized at both ends of the link. This function shall block until both sides of the link have rendezvoused, an error is detected, or the specified TIMEOUT delay has been exceeded.
QKD_CLOSE	This terminates the association established for this key_handle and no further keys will be allocated for this key_handle. Due to timing differences at the other end of the link, the peer operation will happen at some other time and any unused keys shall be held until that occurs and then be discarded.
QKD_GET_KEY	Obtain the required amount of key material requested for this key_handle. Each call shall return the fixed amount of requested key or an error message indicating why it failed. This function may be called as often as desired, but the key manager only needs to respond at the bit rate requested through the QOS parameters, or at the best rate the system can manage. The key manager is responsible for reserving and synchronizing the keys at the two ends of the QKD link through communication with its peer. This function may be blocking (wait for the key or an error) or non-blocking and always return with the status parameter indicating success or failure, depending on the request made via the QKD_OPEN function. The TIMEOUT value for this function is specified in the QKD_OPEN() function.

The syntax of these functions is as follows (according to OMG IDL syntax defined in [i.1]).

```
Interface QKD_AppInt{

   QKD_OPEN (in destination, in QoS, inout key_handle , out status);


   QKD_CONNECT_NONBLOCK (in key_handle, out status);


   QKD_CONNECT_BLOCKING (in key_handle, in timeout, out status);


   QKD_GET_KEY (in key_handle, out key_buffer, out status);


   QKD_CLOSE (in key_handle, out status);

   }
```

NOTE: The parameter "key_handle" is an output parameter when the caller is initially opening the connection and the value passed in is "NULL". If the value is not "NULL" the value is assumed to be either a "key_handle" recently established by the peer end or a specifically desired/predefined "key_handle". For use in all other functions "key_handle" is an input. If the "key_handle" is already in use by some other application, the QKD_OPEN function will return with an error.

The parameters are described in Table 1.

Table 1

Name	Description	Type	Comments
key_handle	A unique handle to identify the group of synchronized bits provided by the QKD Key Manager to the application	64 octet string (512 bits)	It contains a reference to the necessary information to locate a key, but does not contain any key material. Key material cannot be derived from the key_handle. It shall be represented as a character array (char *), with the high order octet being the most significant octet (i.e., octet[63] is the most significant, of 0 to 63) Maybe passed and stored in other systems It shall be unique and both peers will use the same value to reference an application key stream. The "key_handle" may be previously agreed upon between the peer applications or sent between them by a public channel. If local applications and the QKD system are within the same security perimeter, the "key_handle" may be sent over a public channel to its peer.
key_buffer	Buffer containing the current stream of keys.	Array of bytes (octets)	Key buffer is an array of bits packed into octets (char *) ordered such that bit[0] of octet[0] is the 1st bit and bit[7] of octet[n] is the 8*n+8th bit.
destination	Address of Peer Application	IP address structure	Initially we are assuming the IP address will be sufficient to specify the peer node, but future consideration may use the port number for reserved applications, such as IPSec, TLS, or other protocols.

QOS	A Structure describing the characteristics of the requested key source			
	Requested_length	Length of the key buffer requested by the application	32 bit unsigned integer	If the requested key amount cannot be provided, an error shall be returned in the status parameter.
	Max_bps	Maximum key rate requested in bits per second	32 bit unsigned integer	This is intended to provide guidance to the Key Manager on expected demand. The Key Manager may choose to reject the request if it cannot be met or do its best which may result in a lower rate than requested.
	Priority	Priority of the request	32 bit unsigned integer	This is intended to provide guidance to the Key Manager about the priority level of the request. The handling of this information is left to the specific implementation.
	Timeout	Time in ms, after which the call will be aborted, returning an error.	32 bit unsigned integer	If this much time has passed and the function has not completed, the function shall return a TIMEOUT_ERROR in the status parameter This value shall be expressed in milliseconds.
status		Success/Failure of the request	32 bit unsigned integer	The values will be the following 0 -> Successful 1 -> QKD_GET_KEY failed because insufficient key available 2 -> No QKD connection available. 3 -> QKD_OPEN failed because the "key_handle" is already in use. 4 -> TIMEOUT_ERROR The call failed because the specified TIMEOUT has occurred Values 5-1 023 are reserved for future use of the present document. Values above 1 023 are left for vendor specific implementations.

Important considerations:

- An application may manage several independent key streams through different "key handles" (e.g. one set of keys for output messages and another for input messages, where each set is identified by a separate "key_handle"). There is no limit on the number of "key_handles" that one application can request.
- The key_handle shall be generated in such a way that uniquely identifies the key material and that key material cannot be derived from it.
- It has not been considered how the internal operation of this QKD service is to be accomplished, except for error messages back to the application through the status parameter. This will be left to the implementer.
- It has not been considered how the QOS priority parameter shall be handled by the internal operation of this QKD service. This shall be left to the implementer.

- The lifetime of the keys in the QKD bit pool that is managed by this layer is not specified nor is any action specified at that time. This shall be left to the implementer.

6.1 Sequence diagrams for QKD Application Interface

In all sequences we have four actors:

Application at host A, APPA:

1) Application at host B, APPB
2) QKD key manager at host A, QKDA
3) QKD key manager at host B, QKDB

We show four different sequences:

APPA and APPB do not have a predefined key handle and execute successfully:

1) APPA and APPB do not have a predefined key handle and timeout on the QKD_CONNECT_BLOCKING() call.
2) APPA and APPB both know the predefined key handle that they will use for the key association and execute successfully.
3) APPA and APPB both know the predefined key handle that they will use for the key association and timeout on the QKD_CONNECT_BLOCKING() call.

For the scenarios covered in this clause, one may assume that the communication between local applications and the key manager layer is accomplished within the security perimeter of the QKD System.

6.1.1 Case 1: Undefined key handle

In this case, APPA starts the sequence with a call to QKD_OPEN() indicating the level of service desired and a NULL value in the key_handle parameter. QKD_OPEN() returns a key_handle value and status information indicating success. APPA sends this key_handle value to APPB on the public channel and then calls QKD_CONNECT_BLOCKING() to wait for APPB to also connect. APPB is waiting on the public channel from APPA for a message with the key handle that they will share. Once APPB gets the key handle it shall call QKD_OPEN() to register its key_handle but it does not need to repeat the level of service desired as long as APPA has already done so. APPB shall check the status parameter returned by QKD_OPEN for success and then call QKD_CONNECT_BLOCKING() to rendezvous with APPA. At this point the key association has been verified and established and if the QKD system is operational the status shall be returned as successful. After that, both APPA and APPB start making calls to QKD_GET_KEY() until they have no more need for secure keys and call the QKD_CLOSE() function in order to terminate the key_handle association in a timely manner.

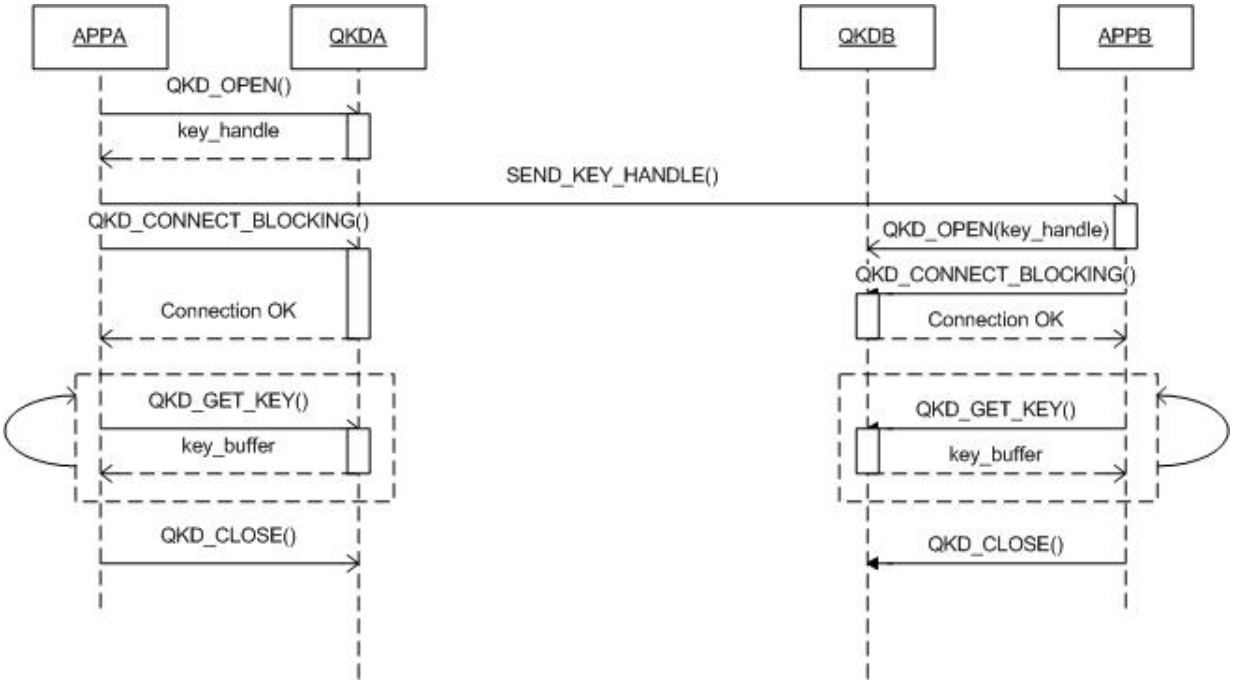

Figure 2: Case 1 - Undefined key handle

NOTE: SEND_KEY_HANDLE() is not in the scope of the QKD Application Interface. It is included to illustrate the functionality of the case.

6.1.2 Case 2: Undefined key handle and failed blocking call

This is the same as in case 1, except for some reason QKDA cannot exchange the necessary information with QKDB in the time specified by the TIMEOUT parameter of QKD_CONNECT_ BLOCKING(). Both APPA and APPB receive a TIMEOUT ERROR in the status parameter from their calls to QKD_CONNECT_ BLOCKING().

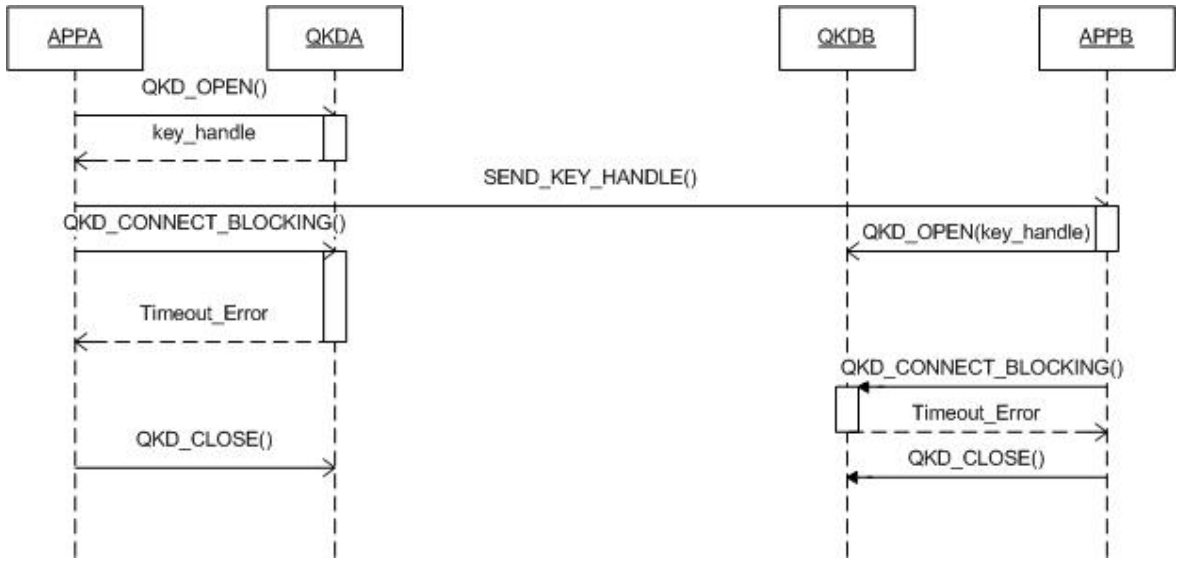

Figure 3: Case 2 - Undefined key handle and failed blocking call

NOTE: SEND_KEY_HANDLE() is not in the scope of the QKD Application Interface. It is included to illustrate the functionality of the case.

6.1.3 Case 3: Predefined key handle

In this case, it is irrelevant who starts the sequence. Both APPA and APPB call QKD_OPEN() indicating the level of service desired and a predefined value in the key_handle parameter. QKD_OPEN() verifies that the specified key_handle is not already in use and returns status information indicating success. No key_handle information needs to be sent between APPA and APPB. Both APPA and APPB then call QKD_CONNECT_BLOCKING() to rendezvous. At this point the key association has been verified and established between QKDA and QKDB, and if the QKD system is operational the status will be returned as successful. After that, both APPA and APPB start making calls to QKD_GET_KEY() until they have no more need for secure key and call the QKD_CLOSE() function that terminates the key_handle association in a timely manner.

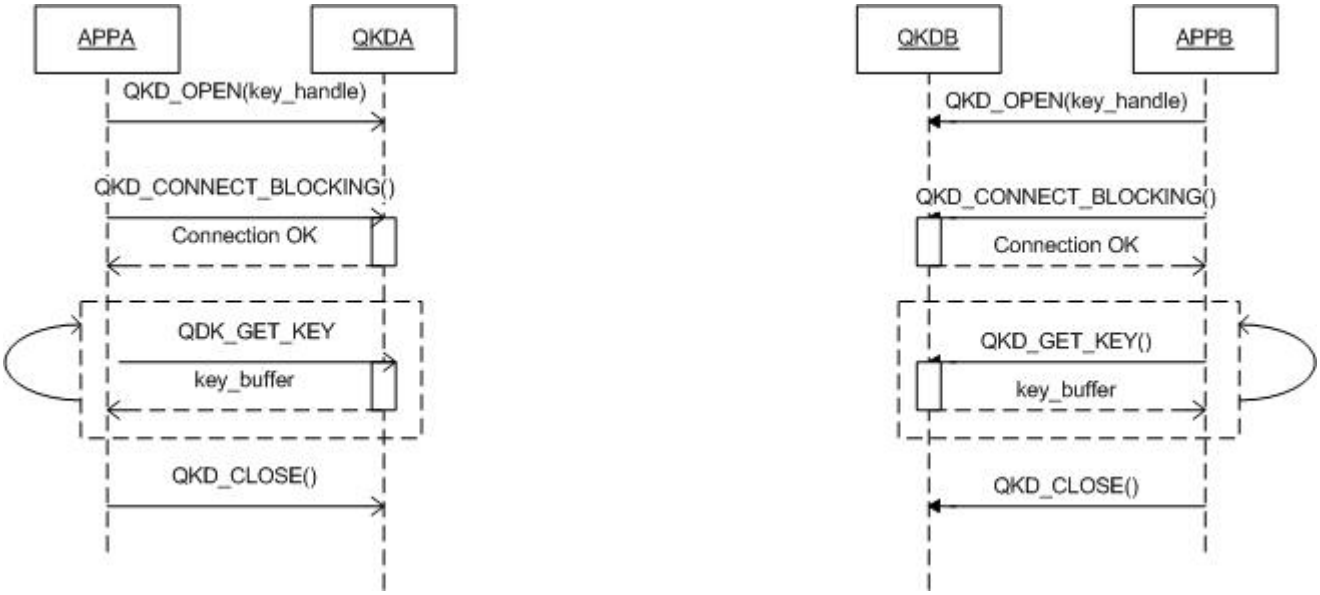

Figure 4: Case 3 - Predefined key handle

6.1.4 Case 4: Predefined key handle and failed blocking call

This is the same as case 3, except for some reason QKDA cannot exchange the needed information with QKDB in the time specified by the TIMEOUT parameter of QKD_CONNECT_BLOCKING(). Both APPA and APPB receive a TIMEOUT ERROR in the status parameter from their calls to QKD_CONNECT_BLOCKING().

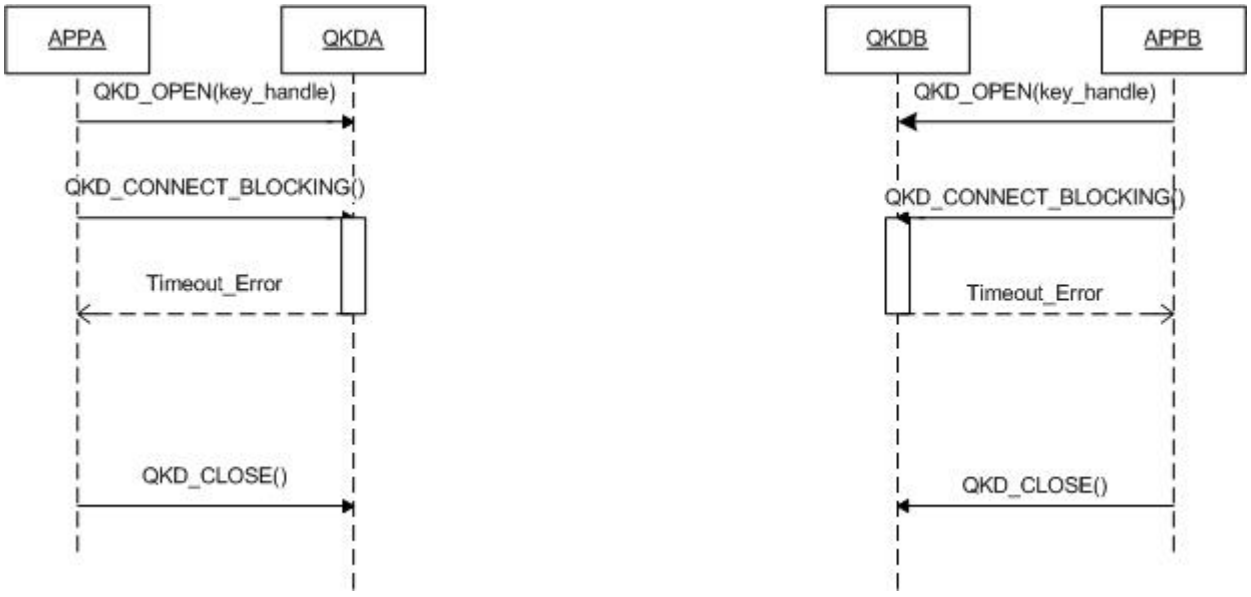

Figure 5: Case 4 - Predefined key handle and failed blocking call

Annex A (informative): Authors and Contributors

The following people have contributed to the present document (by alphabetical order):

- Jorge Dávila, Universidad Politécnica de Madrid (www.upm.es)
- Daniel Lancho, Universidad Politecnica de Madrid (www.upm.es)
- Vicente Martín, Universidad Politécnica de Madrid (www.upm.es)
- Alan Mink, National Institute of Standards and Technology (www.nist.gov)
- Mercedes Soto Rodriguez, Telefónica Investigación y Desarrollo (www.tid.es)

Annex B (informative): Conventional Key Management Systems

Conventional key management systems were investigated for use within a QKD system. One such protocol, the KMIP initiative within the OASIS open standards consortium <http://www.oasis-open.org/home/index.php> was considered. It specifies a protocol and interface for communication between clients and a key server. QKD instead operates under a distributed paradigm, where the keys are already distributed to the QKD end points but demultiplexing them into independent synchronized streams and distributing them to various applications, at those communication end points, is needed.

Some thought was given to adopting the KMIP framework for QKD System as well as using QKD to strengthen the KMIP as follows:

1) To enhance security of KMIP authentication protocols. KMIP does not specify, as part of the protocol, the mechanisms by which key management systems and cryptographic clients identify themselves to each other. Rather, KMIP relies on existing standards for mutual authentication that specify how this identification is to be established. KMIP currently defines two authentication profiles, the first based on TLS, the second on HTTPS. In both profiles, digital certificates are used by the client and the server to identify themselves as participants in KMIP requests and responses. Registration mechanisms by which the enterprise key manager learns the identity of cryptographic clients are not defined in KMIP. The credentials used by the cryptographic client to identify itself can be included in the protocol as part of a request message, to simplify processing of the request by the key management system. However, the credential element is not guaranteed to be authenticated and is therefore not intended for use in authentication. Message integrity for KMIP exchanges, as well as entity authentication, is provided by TLS. Other mechanisms (for example with the inclusion of QKD inside TLS or HTTPS key exchange process) that could also be used for enhanced security of KMIP messages are not currently defined for KMIP.

2) To enhance security of a Key Management system using QKD one-time-pad tunnels for the interchange of KIMP messages.

3) To include QKD keys as a third party source of keys for KMIP servers.

B.1 KMIP Draft Documents Version 0.98

- "KMIP: Key Management Interoperability Protocol" (online: http://xml.coverpages.org/KMIP/KMIP-v0.98-final.pdf).
- "Key Management Interoperability Protocol Usage Guide" (online: http://xml.coverpages.org/KMIP/KMIP-UsageGuide-v0.98-final.pdf).
- "Key Management Interoperability Protocol Use Cases" (online: http://xml.coverpages.org/KMIP/KMIP-UseCases-v0.98-final.pdf).
- Key Management Interoperability Protocol (KMIP) FAQ Document (online: http://xml.coverpages.org/KMIP/KMIP-FAQ.pdf).

B.2 Other Material

- "Key Management Interoperability Protocol (KMIP): Addressing the Need for Standardization in Enterprise Key Management". (online: http://xml.coverpages.org/KMIP/KMIP-WhitePaper.pdf) KMIP White Paper. Version 1.0. May 20, 2009. 22 pages. Previous Version, Draft 0.9, April 05, 2009.(online: http://xml.coverpages.org/KMIP/KMIP-v0.98-final.pdf).
- "Key Management Interoperability Protocol (KMIP)" (online: http://xml.coverpages.org/KMIP/KMIP-Webinar20090402.pdf). Prepared for a Webinar Presentation, April 02, 2009.

Annex C (informative): Scenario for a QKD network

In this clause we outline guidance for a more complex network scenario instead of a single point-to-point QKD system. A new architectural element, the Key Server is introduced to manage several QKD links at any given node, while individual Local Key Managers handle keys from only one QKD link, as shown in Figure C.1.

There is no change to the API presented in the present document. The difference is purely internal to the Key Management Layer that now needs an additional management structure, the Key Server, to coordinate the multiple QKD links within a node. The Key Server can be responsible for communicating with multiple peers for the synchronization, exchange, storage, and destruction of the keys generated by the QKD Systems.

Figure C.1: Multilink QKD Network scenario

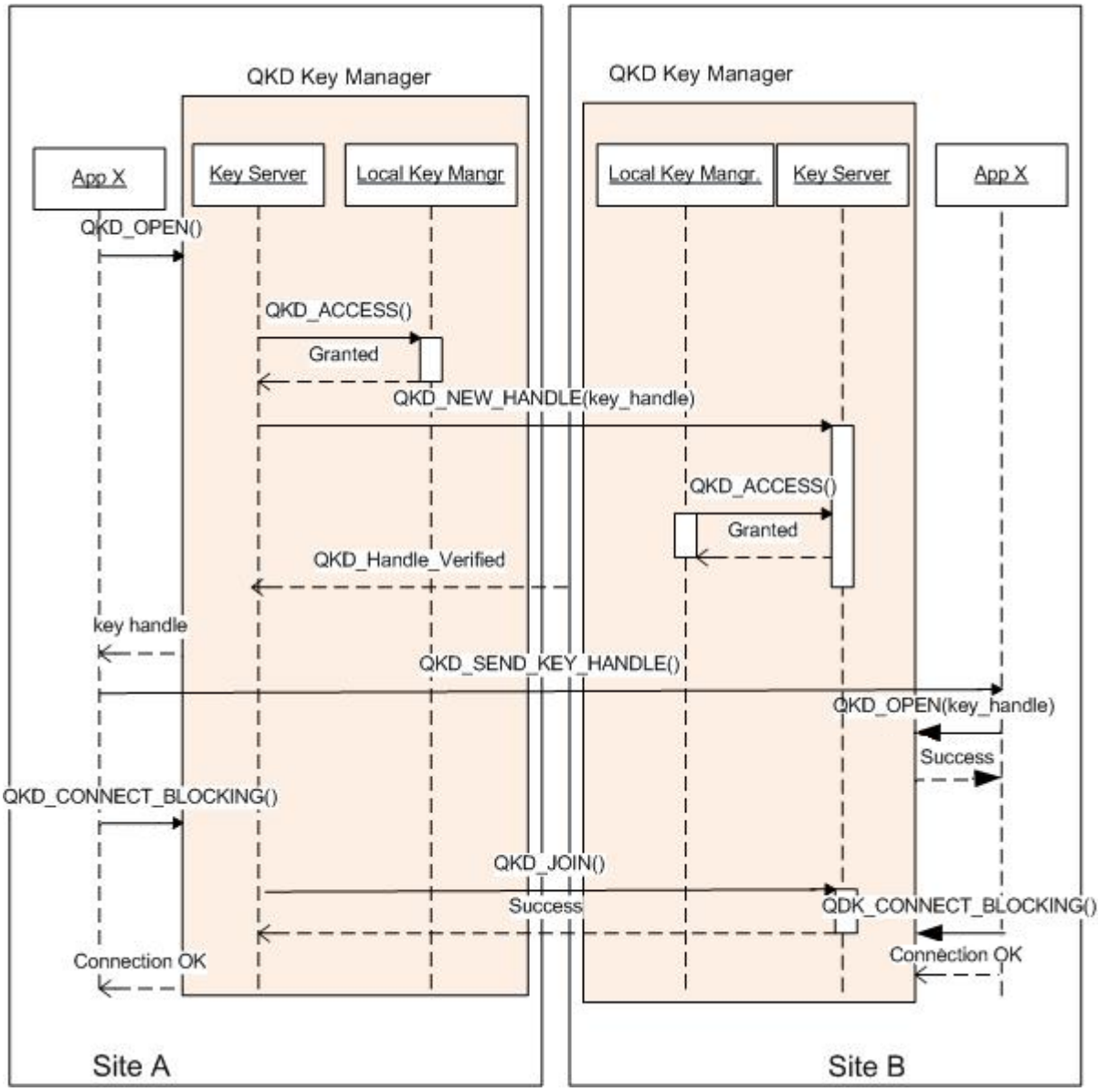

Figure C.2: Network Message Exchange scenario

The scenario, depicted in Figure C.2, is intended to show one possible internal communication sequence between the Key Server and a Local Key Manager as it relates to an application open request.

In this case, an application at site A initiates a QKD session with its Key Management layer. The Key Server then proceeds to initiate communication with the proper Local Key Manager and its remote peer. The remote peer Key Server establishes communication with its proper Local Key Manager and together they all validate the proposed key-handle, verifying that it is unique and determining if the services requested by the application can be successfully supplied. Once determined, a success indicator is return to the application along with the key_handle. The peer applications at the communication end points can use the key_handle to recover synchronized key material.

Annex D (informative): Bibliography

C.H. Bennett and G. Brassard: "Quantum Cryptography: Public Key Distribution and Coin Tossing", Proceedings of IEEE International Conference on Computers Systems and Signal Processing, Bangalore India, December 1984, pp 175-179.

NOTE: Online at http://www.research.ibm.com/people/b/bennetc/bennettc198469790513.pdf.

Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel and Hugo Zbinden: "Quantum cryptography, Reviews of Modern Physics", Vol 74, 145-195 (2002).

NOTE: Online at http://www.gap-optique.unige.ch/Publications/PDF/QC.pdf.

Valerio Scarani, Helle Bechmann-Pasquinucci, Nicolas J. Cerf, Miloslav Dušek, Norbert Lütkenhaus, and Momtchil Peev: "The security of practical quantum key distribution", Vol. 81, 1301-1351 (2009).

NOTE: Online at http://arxiv.org/abs/0802.4155.

D. Gottesman, H.-K. Lo, N. Lütkenhaus, and J. Preskill: "Security of quantum key distribution with imperfect devices", Vol. 5, 325-360) (2004).

NOTE: Available at http://arxiv.org/abs/quant-ph/0212066.

"White Paper on Quantum Key Distribution and Cryptography", Preprint arXiv:quant-ph/0701168, Alléaume R, Bouda J, Branciard C, Debuisschert T, Dianati M, Gisin N, Godfrey M, Grangier Ph, Länger T, Leverrier A, Lütkenhaus N, Painchault P, Peev M, Poppe A, Pornin Th, Rarity J, Renner R, Ribordy G, Riguidel M, Salvail L, Shields A, Weinfurter H, Zeilinge, A, 2006 SECOQC.

UQC Report: "Updating Quantum Cryptography", Quantum Physics (quant-ph); Cyptography and Security. Donna Dodson, Mikio Fujiwara, Philippe Grangier, Masahito Hayashi, Kentaro Imafuku, Ken-ichi Kitayama, Prem Kumar, Christian Kurtsiefer, Gaby Lenhart, Norbert Luetkenhaus, Tsutomu Matsumoto, William J. Munro, Tsuyoshi Nishioka, Momtchil Peev, Masahide Sasaki, Yutaka Sata, Atsushi Takada, Masahiro Takeoka, Kiyoshi Tamaki, Hidema Tanaka, Yasuhiro Tokura, Akihisa Tomita, Morio Toyoshima, Rodney van Meter, Atsuhiro Yamagishi, Yoshihisa Yamamoto, and Akihiro Yamamura, 2009.

NOTE: Available at http://arxiv.org/abs/0905.4325.

Menezes A. J., van Oorschot P. C. and Vanstone S. A.: "Handbook of Applied Cryptography", (Boca Raton: CRC Press) 1997.

Schneier B.: "Applied Cryptography", 1996, (New York: John Wiley).

History

Document history		
V1.1.1	December 2010	Publication