Chapter 8
QKD Networks Based on Q3P

O. Maurhart

This chapter explains how single QKD links can be connected together via Q3P to form a QKD network. The term "network" emphasizes the need to interconnect more than two nodes. In this network arbitrary user data is transmitted from and to non-adjacent parties.

Here Q3P – the Quantum Point-to-Point Protocol – is described. This protocol expands the concepts of the well-known Point-to-Point Protocol (described in [12]) enabling and extending its prime ideas to adopt them in a quantum cryptography environment. Thus Q3P can guarantee information theoretical security between nodes of a QKD network with respect to encryption and authentication.

Based on Q3P, additional protocols for routing and message transmission are discussed which respond to QKD network differences in comparison to classical ones. Availability of key material, considerations of resend mechanisms, and trusted paths are among the properties which clearly separate QKD networks from other network types.

The protocols presented have been designed and developed during the EU founded SECOQC project and were successfully presented in Vienna, 2008 (see [9]).

8.1 QKD Networks

By means of QKD a shared secret key can be established on two sides, likely labeled Alice and Bob. Furthermore, quantum cryptography provides mechanisms and techniques to allow an eavesdropper (Eve) to be detected on the shared communication media.

The various techniques involved in QKD always use a quantum channel to transmit quantum information as well as a classical channel for further actions. In order to successfully establish a key between Alice and Bob to be used for one-time-pad

O. Maurhart (✉)
Safety & Security Department, Quantum Technologies, AIT Austrian Institute of Technology GmbH, Lakeside B01A, 9020 Klagenfurt, Austria, oliver.maurhart@ait.ac.at; http://www.ait.ac.at

Maurhart, O.: *QKD Networks Based on Q3P*. Lect. Notes Phys. **797**, 151–171 (2010)
DOI 10.1007/978-3-642-04831-9_8

(OTP) and other cryptographic techniques of information theoretical security processing a number of steps have to be undertaken. Recalling previous chapters of this book to a bare minimum these are as follows:

- Sifting: This allows Alice and Bob to remove bits from their raw key due to wrong basis choice. After this step Alice and Bob share a *sifted* key. There have been various protocols developed in the recent years, starting with [1] and are still emerging since new quantum technologies are still discovered requiring new sifting protocols.
- Error Detection and Correction: Still there can be errors present in both keys. This is due to some quantum bits that may have been damaged or flipped during transmission or quantum detection failures. Note that errors are also probably introduced by an eavesdropper. Therefore the system has to decide whether the detected errors can reasonably be found by technological and/or physical shortcomings *or* are introduced due to the presence of an attacker and more importantly whether relevant security limits are exceeded. If the error rate can still be tolerated an error correction scheme is to be applied, otherwise the current key distillation round has to be terminated and the results skipped. The most popular error correction technique is CASCADE, but other more refined versions are researched and uprising. The result of this step is a *reconciled* key.
- Privacy Amplification: An eavesdropper – Eve – is always expected. As Eve is conservatively assumed to be the source of all errors occurring in the raw key acquisition phase and is supposed to listen on the insecure public channel on which the error correction steps above are performed, she may gather enough knowledge to deduce part of the key. Hence Alice and Bob now try to minimize Eve's knowledge by appropriately reducing the length of the reconciled key. After this step Alice and Bob have a *shared secret* key.

However, the benefit of QKD is restricted to a common shared quantum media. Therefore the key distillation steps described above are only applicable on a peer-to-peer basis, where Alice and Bob have access to the same quantum media. Also with respect to different quantum techniques key establishment may be expensive both in time and computational resources.

QKD networks now try to exceed point-to-point limitations, enabling Alice to send information to Bob without sharing a quantum channel. They are rather connected along a path of intermediate nodes. Therefore a QKD network is described by

- A quantum network consists of nodes connected by quantum channels. The quality and design of the quantum channels are limited by the certain technology used at and the physical attributes they impose. Therefore, quantum networks may be composed of quantum channels of different kind regarding distance and bandwidth compared to cost to establish and maintain.
- All nodes are connected to a classical communication media, which ought to be public and insecure. The nodes need this media for their specific QKD-protocols.

The issue, whether this media is the same or not as used for the quantum channel (i.e., fiber optic), is irrelevant from a conceptional point of view.

- A quantum channel *itself* is always a point-to-point link between exactly two nodes. One-to-many links are seen as multiple logical point-to-point links in spatial or time division manner. Routing protocols on top bypass the peer-to-peer limitations and enforce multi- and broadcast capabilities on the system.
- A quantum network node may participate in more than one, single quantum channel, thus making it a relay station.
- Messages between two adjacent nodes can be transferred with information theoretical security using at least one-time-pad cipher for encryption on the classical channel.
- Messages between two adjacent nodes can be authenticated with information theoretical security.

This results in delivering a message on a hop-by-hop basis as illustrated in Fig. 8.1. On receiving, each node decrypts the message using a local key shared with its neighbor and then retransmits it with a key it has for its other neighbor. The keys are discarded thereafter. Exhibiting such behavior qualifies QKD nodes to be viewed as "Store & Forward" network devices.

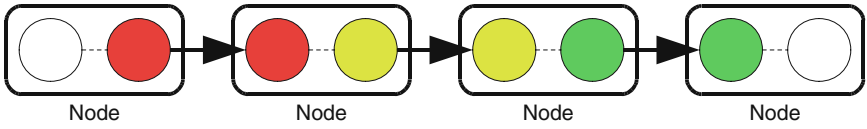


Fig. 8.1 Hop-by-hop transmission of a message between nodes

Having such a set of interconnected QKD nodes which transmit messages on a hop-by-hop basis constitutes to a *trusted repeater QKD network*. Topologies of such networks include mesh- and starlike networks. Within a trusted repeater QKD network, keys are shared between two parties along a path of intermediate and trustworthy nodes. This is opposed to quantum switching network in which the quantum channel is switched. The later model does connect more than two peers but is bound in distance. As QKD switched networks create selective point-to-point connections on demand by short-circuiting QKD channels, attenuation is seen as a major concern and problem. With the help of trusted repeaters any number of participating users connected across arbitrary distances can share secrets on a QKD network.

Obviously a trusted repeater QKD network can be built using more than one single QKD technology. The requirement to integrate different QKD techniques and methods into one single network interconnecting more than two users is indispensable to realize trusted repeater QKD networks. This reasons the need for a layer to be introduced which separates key usage from the technical details of key generation. This is much like today's Internet in which the client browser does not care about the concrete transport technique (Ethernet, WLAN, ...). This layering is solved by Q3P.

The prime building block of a quantum network is Q3P. As Q3P itself is deeply inspired by PPP, the classical, traditional protocol for interconnecting two machines, a brief summary of PPP has to be stated.

8.2 PPP

The Point-to-Point Protocol [12] standardized 1994 is used to establish a connection between two nodes in a network. It is primarily used as dial-up connection mechanism and superseded the former SLIP [11]. SLIP lacked error detection and was bound to serial lines as well to IP messaging solely.

PPP holds

- Encapsulation of various datagram protocols like IP, IPX, and AppleTalk
- Error Detection
- Link Control Mechanism
- Authentication

PPP does not rely on a certain connection technology as SLIP does (serial line). Instead PPP imposes a few requirements on the lower layer:

- The link provides full-duplex simultaneous bi-directional operation.
- All packets arrive in order at the responder side (in-order delivery).

PPP enforces no sequencing or fragmentation on packets which requires a collection and reordering facility at the responder side. PPP does not expect this options of the lower layer requiring to send information packed in one single PPP frame. However, information datagrams of higher layers can be distributed across several packets but it is the responsibility of the higher level protocol to reassemble the fragments orderly.

The feature list of PPP is as follows:

- Encapsulation: PPP is capable of encapsulating various protocols situated at the next higher layer when speaking in terms of the ISO/OSI model [13]. The list of encapsulated protocols includes IP, IPX, AppleTalk, and others. The mechanism provided in PPP is of a much general term, allowing any arbitrary protocol to be used within PPP.
- Link Control Protocol (LCP): a major part of PPP is dedicated to establish and configure the link.
- Network Control Protocol (NCP): to be independent of any succeeding network protocol running on top of PPP, a framework for negotiating several aspects and options of the higher protocol is presented. For each network protocol collaborating with PPP a specific NCP is written, i.e., IPCP – the Internet Protocol Control Protocol for IP [7].

PPP is used as a layer 2 protocol for the well-known ISO/OSI reference model but is also implemented on top of already existing layer 2 protocols like PPPoE [6]

– "Point-to-Point Protocol over Ethernet" –, PPtP [4] – "Point-to-Point Tunneling Protocol," – and PPPoA [3] – "Point-to-Point Protocol over ATM."

The intention of PPP is to be self-configuring and functionable right out-of-the box. Simplicity in maintenance and configuration and flexibility and expandability are key terms of PPP.

8.3 Q3P

Q3P is the *Quantum Point-to-Point Protocol*. Q3P uses the ideas in the well-known PPP to form a PPP-*like* protocol between two adjacent nodes in a quantum network.

The whole Q3P design serves different aspects:

- Provide a communication primitive between exactly two adjacent nodes in a quantum network to enable sending and receiving of messages enhanced by information theoretical security.
- Enforce overall security of a QKD node by being the very single but neat interface to the outside world on the classical channel.
- Utilize different and yet even unknown QKD protocols and devices for key generation.
- Utilize different and yet even unknown cryptographic algorithms for information theoretical security for encryption and authentication.
- Separate key production from key usage. Even more: leverage Quality of Service methods on key usage regardless of any concrete QKD machine producing them.

With respect to some ISO/OSI-like hierarchical structure, viewed "from top" Q3P offers the very same interface of level 2 regardless of any concrete quantum technique used underneath. This is required to build stable network protocols on top of it without the need to tackle with QKD implementation specifics and quirks.

8.3.1 Q3P Building Blocks

Figure 8.2 shows the very basic setup for a Q3P implementation. The setup shown consists of two QKD devices, a Q3P engine, a classical channel, and an application using Q3P facilities (information theoretical secure encryption and authentication to another direct connected peer). The central point is that both – the QKD devices and the application – use Q3P to communicate with their peers on the classical, public channel. The Q3P engine acts as a singleton, that is: there must be only one single instance in a system running.

The sketch shows two instances of QKD devices, each of them having their own dedicated quantum channel. It is possible to connect more than one QKD device to a single Q3P engine. Having more than one QKD device connected to a Q3P engine opens up multiple connections to different peers, the basic building block for a network.

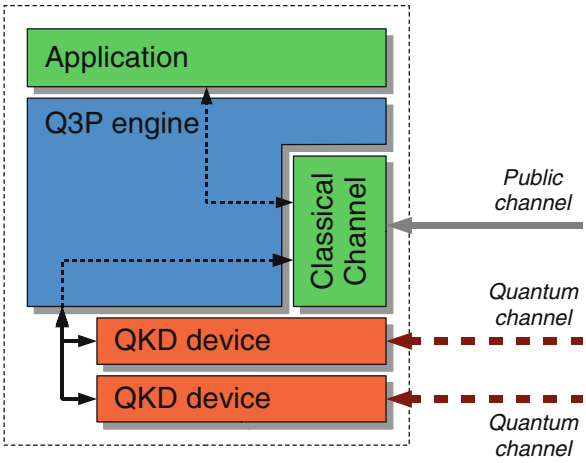


Fig. 8.2 The "Q3P Engine" where all Q3P processing takes place

Examining the Q3P engine further we see:

1. The *Q3P node*. This entity represents administration and coordination unit. It creates Q3P link entities on demand. Whenever a connection from the outside – be it from a QKD device, a peer Q3P node or any application utilizing Q3P – is made the Q3P node is the very first to be contacted. It dispatches requests and commands to the proper Q3P link processes. As the node is responsible for creating the Q3P link instances it also observes, terminates, and relaunches them as appropriate.
2. The *Q3P link*. A Q3P link is a concrete instance of an established connection between two peers. The Q3P link entity is responsible for encryption, authentication, and authentication check. A single Q3P link instance serves all QKD to QKD connections between two nodes. That is, on Q3P node *A* and node *B*, we attach an arbitrary number of QKD devices and interconnect them each: still there is only one single Q3P link instance serving all QKD to QKD connections. Any messages which are sent and received by Q3P pass a Q3P link to get encrypted and authenticated or authentication checked and decrypted, respectively. As all messages are passed through this instance a Q3P link maintains *Channels* for each application to application (or QKD device to QKD device) communication line (Fig. 8.3).
3. The *Key Store*. Each Q3P link has an associated Key Store which buffers available key material for future use. Key Stores receive their key material from the QKD devices underneath. However, once a key has been delivered to such Key Stores any tags of information, stating the origin of the key material or the concrete date and time when it has been created, are removed. As such, a quantum key is nothing but a bitstream with quantum cryptography-based randomness.
4. Crypto Engine. The Crypto Engine acts as a container for any security – or more precisely cryptographic – relevant algorithms. Whereas OTP is rather

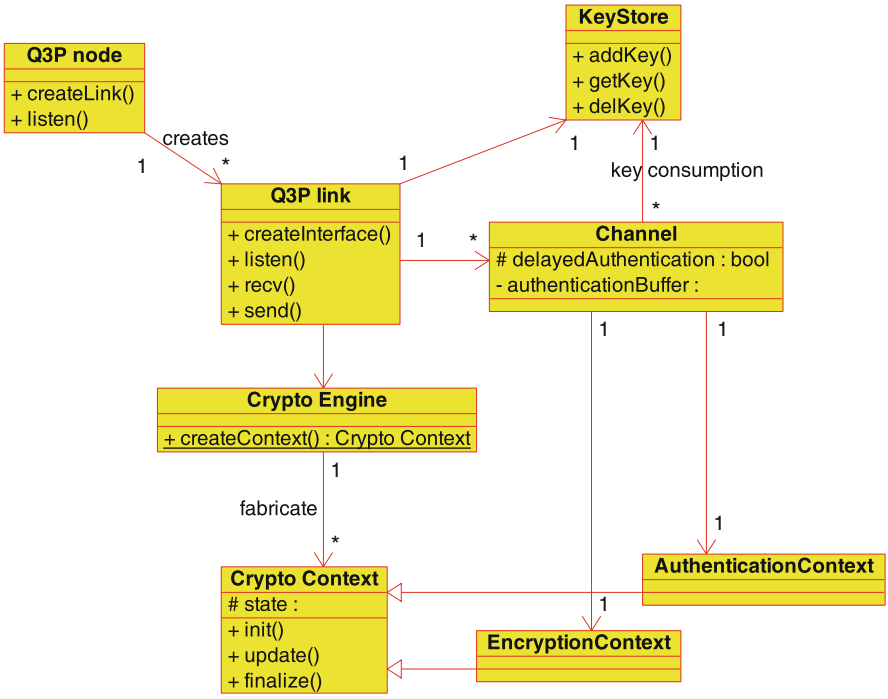


Fig. 8.3 Q3P class diagram in UML

trivial, information theoretical authentication is not. The Crypto Engine is capable of running different authentications with different tag sizes in parallel. This is achieved by creating *Crypto Contexts* dedicated either to the realm of encryption or authentication. Each channel has access to its unique encryption and authentication context.

8.3.2 Message-Based Streaming

Q3P connects two adjacent nodes. Such communication line which is hold up by the Q3P link entities on both sides is called a *Q3P Association*. A single Q3P Association serves more than one single application or QKD device. Each pair of QKD devices or applications utilizing a Q3P Association maintain a set of *Q3P Connections*. That is to say: a single Q3P Connection interconnects two QKD devices (or two applications, respectively). As QKD devices and applications may wish to send and receive in parallel each Q3P Connection can be further divided into *Q3P Channels*. It is these Q3P Channels to which cryptographic routines are attached and operated on.

Q3P is a message-stream based protocol. A single unit of communication for Q3P is a message. This is different for well-established protocols like TCP (see

[10]) for they are byte-streamed. Byte-streamed based protocols work on streams of bytes without interpretation of these streams in any way. On the other side, applications do have special semantics and meanings introduced into such byte-streams. Hence, even a video-stream encoded as MPEG or a VoIP-stream is split into several "messages." In order to preserve the message semantics each application utilizing a byte-streamed based protocol has to undertake additional steps to signal message starting and ending to its peer. For message-based protocols this is not needed. And for Q3P this comes natural since for a security centric protocol the key element for encryption and authentication schemes widely used is a message, not a byte-stream.

A message is defined as a sequence of bytes having a well-defined starting and ending position within the transport mechanism delivering the information. Further on – for Q3P – a message is not been interpreted; the message's content is of no concern to Q3P. However, messages are uniquely identifiable over a reasonable period of time. Ideally, there is no number clash of a message's ID during the lifetime of an established Q3P Connection. The size of a message is not limited and may vary, but performance considerations and hardware do impose restrictions.

The disadvantage of using a message-based view resides in the "store and forward" principle of nearly all networking facilities. When sending a message the receiver waits till the whole message arrived *and then* dispatches it for next computation (or routing). When messages are rather large then the transmission delay, the timespan encountered between receiving the very first byte of the message to the very last one, is as large as well. Thus enlarging the overall delay in communication.

However, knowing this, applications have the capability to increase (and reduce) performance on their own, by splitting their information into smaller messages and even drop some if their content does not provide essential meaning but simply increases quality of the data perceived. This is definitively true for media streams like MPEG which may drop B- and P-Frames off the whole message stream.

As Q3P has no knowledge of the concrete content of a message, it cannot decide which message can be dropped or not. Leaving this capability to entities having the knowledge of the concrete content of a message liberates Q3P to marginally rebuild parts of higher level protocols (Fig. 8.4).

8.3.3 Security Modes

The main goal of Q3P is to enforce a transmission of any messages by means of information theoretical security between two nodes. Q3P offers to send messages with these schemes:

- encrypted and authenticated
- authenticated
- plain

This is very contrary to most established security-focused network protocols like IPSec (IPSec starting with [5] and ongoing). Common security architectures within

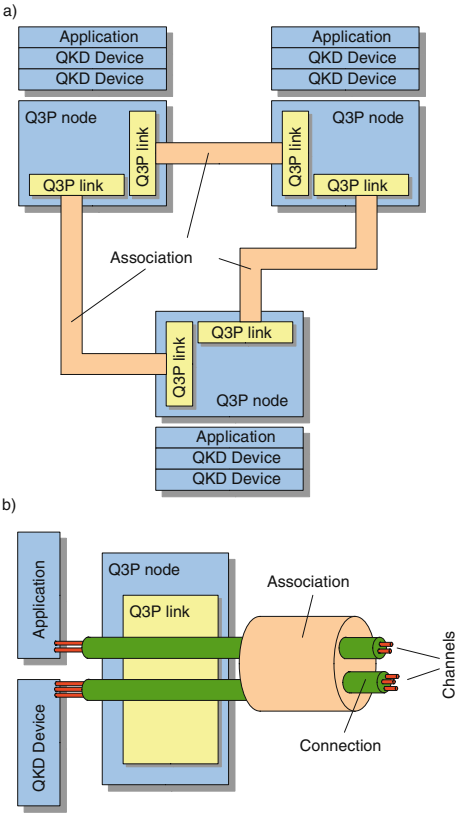


Fig. 8.4 Relationship between Q3P Associations, Q3P Connections, and Q3P Channels. (**a**) shows three Q3P nodes interconnected by their Q3P link instances via Q3P Associations. On each Q3P node an application and two QKD devices are attached communicating with their peers. (**b**) shows the situation on one single Q3P node: an application has a connection established. Within this Q3P Connection it utilizes two Channels. On the very same Association a QKD device has another Connection but with three Channels up and running

the networking community usually introduce an *all-or-nothing* principle whereas either the whole communication is encrypted and authenticated or no encryption and authentication is applied at all.

Secure transmission within Q3P is done by information theoretical security. As for the time of writing no *non*-key consuming algorithm with information theoretical secure quality is known, be it for encryption or be it for authentication. Therefore, any key used for encryption and/or authentication when sending messages is wasted if the message content is not of sensible nature for secrecy or integrity.

This is not a problem if QKD devices do produce much more keys than needed by applications. But today quantum key material is a very scarce resource and has to be used very sensitive. As bandwidth throughput is limited by the available key material it is counterproductive to waste such keys on messages not sensitive enough

to be encrypted. This is the rationale in offering this variety of communication schemes.

Another reason is this: let M be the message. The keys used if the message is to be transmitted encrypted and authenticated are donated as k_E for the encryption and k_A for the authentication. As for OTP $|k_E| = |M|$ and for any arbitrary information theoretical secure authentication $|k_A| > 0$ holds, clearly stating $|M| < |k_E| + |k_A|$; with full encryption and authentication turned on, slightly more key bits are consumed than transmitted.

As mentioned earlier Q3P acts also as the only interface for the QKD devices on the classical channel and the latter use the classical channel to do their sifting, error correction, and privacy amplification to produce a new key. Therefore, the sum of the key material used for all messages exchanged to create a new key may now exceed the length of the new key produced if encryption and authentication is activated for each and every message. These results in having more key material consumed than created by the generation process.

To avoid this effect Q3P offers the encryption and authentication combined modes, authentication without encryption mode, and the plain mode without any security attached. The latter two are reasonable whenever information has to be exchanged of non-confidential nature. One even may omit authentication at all, if the integrity of the information sent and received can be determined in future steps, e.g., the application's communication can be mapped on a transaction processing concept.[1]

Another concern is authentication for many small messages. A huge load of small messages to be authenticated is a characteristic of common QKD error correction protocols like CASCADE. As QKD devices produce such messages and due to the nature of some error correction algorithms lacking possibilities to accumulate such messages and send them in once, another authentication scheme is introduced: *Delayed Authentication*. This authentication technique is offered along the *Instant Authentication* in which each and every message is authenticated at once.

Within Delayed Authentication (see Fig. 8.5), messages to be sent are locally copied to an internal Q3P buffer and then transmitted across the network to the peer. This particular transmission is *not* authenticated but sent in plain instead, consuming no key material at all. Gradually, the Q3P authentication buffer is filled with a series of small messages as the QKD device communicates with its peer.

Finally, the device requests an authentication-buffer check from the Q3P implementation at a suitable position within its processing QKD protocol. Q3P now computes the authentication tag and verifies this tag with the one received by the peer's Q3P. The result of this verification is responded back to the invoking QKD device. If authentication succeeded the device can be sure that all buffered messages so far were not altered by an adversary.

---

[1] A transaction concept having the ACID – Atomicity, Consistency, Isolation, and Durability – properties. A transaction has a clearly defined starting point and a clearly defined endpoint. Any action undertaken between these two points does not have any consequence to the surrounding environment until the full transaction is committed at the endpoint.

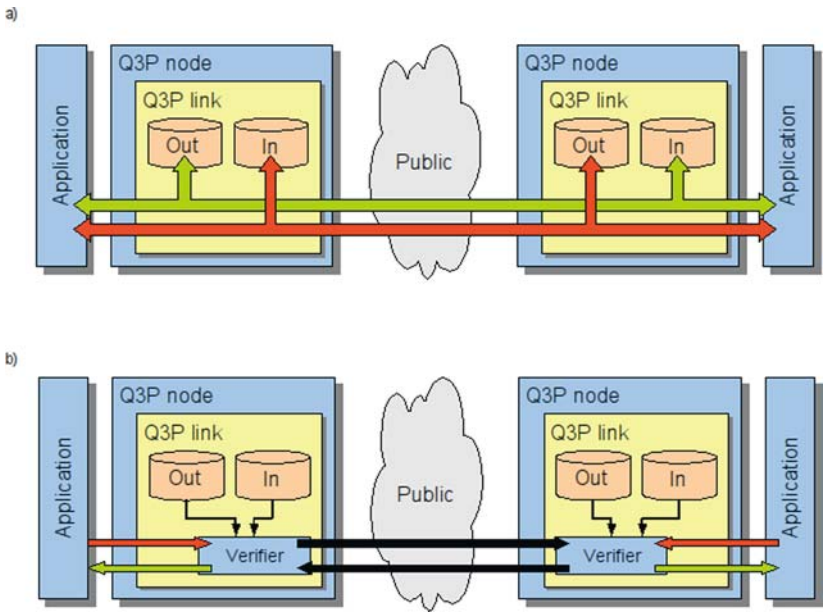


Fig. 8.5 Delayed Authentication. (**a**) shows the situation during normal message processing: all messages are copied to the outbound buffer before sending. On receiving the message is copied to the inbound buffer before handing out to the application (or QKD device). (**b**) illustrates the authentication check: both applications *have to* issue an authentication check. This stops ordinary message processing until the peer also ordered an authentication check as well. The authentication check is then performed by the Q3P Link instance (Verifier) and the Q3P Channel is unblocked on success again

Note that Q3P maintains a send and receive buffer for the delayed authentication separately. When performing delayed authentication now the authentication check is done for both directions, ingoing and outgoing at once. On receiving the inbound message buffer tag from the peer in this process the local party can also decide if information sent by it has been received untainted. This effect is quite uncommon in today's authentication protocols: giving the *sender* the ability to prove the authentication of the messages sent as well as the receiver. Though it seems that this design is memory consuming, authentication algorithms usually act on message blocks modifying some intermediate state S. When finalizing the authentication process some additional computation is done on the intermediate state S. This hints the way for memory optimization since for authentication algorithms which can be laid out this way only S is of concern and the message blocks once computed can be discarded for the authentication afterward.

Clearly, Q3P cannot state when such authentication checks have to be done. This is up to the application for only it knows the concrete context and find a proper position along its own, possibly proprietary application protocol. When doing OLTP (Online Transaction Processing) such steps ought to be found right before or along the COMMIT instruction.

Having the freedom of choice for a variety of several different security modes to choose for a single Q3P Channel introduces responsibility as well. If Q3P receives an authenticated message then it hands out the message if the authentication check succeeds. If the latter fails the message is silently discarded. But Q3P cannot decide whether a received message has to be encrypted, authenticated, or none of both in advance without knowing the message context, e.g., a message has been received in plain without any authentication tag attached. This *must be* checked by the application by inspecting the mode flags of a received message and comparing this to the internal application protocol: Is an unauthenticated message acceptable at the current stage of processing? Q3P has no means to tell if an unauthenticated message should be received authenticated – Q3P lacks the domain knowledge of the concrete application protocol to decide. This is a drawback since now applications are burdened with additional security checks introducing more responsibility on the application designers when it comes to security.

8.3.4 The Key Store

At the very heart of Q3P is the Q3P Key Store. A Q3P Key Store differs from any classical key storing and management infrastructure: once a key has been used, it is ought to be discarded. This is certainly true for OTP encryption and for any information theoretical authentication scheme currently used in Q3P.

These arguments request a total different approach than common public key systems or common symmetrical key buffers do imply – because they usually do not trash the key at first usage.

Four primitive functions have to be fulfilled:

1. Store a key.
2. Shred a key. The key store removes a key qualified by a given key identifier. This is natural if the key has been used. The notion “to shred” indicates that the key really has to be destroyed instead of simply “deleting” or “dropping” a key. Shredding a key may find its limitations by operating system memory management and by physical attributes of the memory media.
3. Retrieving a key. The key store offers a method to retrieve the *next* key with a given length l. A second technique is to retrieve a key in the database based on a key identifier.
4. Query the state of the key store. The key store has to give knowledge about various internal states to enforce QoS. That is, the amount of key material within the key store, statistics about recent key usage, and reasonable assumptions about key material growth.

Keys are delivered by QKD devices. However, they may do that in any variation of order. Next, the key material is used in symmetric key cryptographic algorithms and thus is not prequalified for any particular direction. When a party decides to send something to its peer it takes *any* key. There is no direction of data flow attached to a key nor any specific task label, like mutual dedication for encryption or

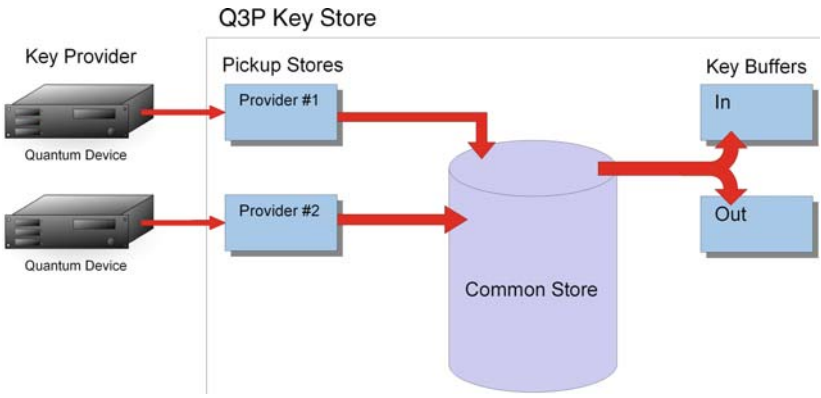


Fig. 8.6 Q3P Key store buffers: Pickup-stores, common store and in-out-buffers

authentication. To send and receive messages *some* keys are used. As the peer proceeds likewise clashes in key access are obvious leading to collisions when occasionally both parties decide to send data utilizing the very same key.

To solve this problem, three strategies are subject of concern: (a) try and detect collisions and, in case, recover, (b) use dedicated send and receive key material buffers on key delivery of a QKD device, or (c) authorization of key handout. Following the last two strategies the Q3P Key Store addresses these issues by introducing a *Master–Slave* relationship on multi-level key buffer.

Figure 8.6 depicts the several key buffers used in Q3P:

1. The key material pushed by the QKD devices is collected in the *Pickup Stores*. Here the keys may appear in any order. Q3P periodically picks up key material and verifies its existence with the peer.
2. Once it is assured that key material is present in both stores the keys are transferred to the *Common Store*. The Common Store represents a big mass of homogeneous key bits organized in blocks. Whereas keys found in the Pickup Stores can be associated with a certain QKD device and thus may even have some certain properties attached, like the error rate when producing this key, time stamp and others, keys in the Common Store don't have this additional information: key identifiers, time stamp, and any other attributes are detached. New keys transferred to this store are merged with already existing key bits there. Though key material is accessible by its BlockID at chunks of varying sizes.
3. The *In-Out-Buffers* hold key material dedicated for receiving and sending. Both buffers are symmetrically interlocked with the peer's buffers. That is, the In Buffer is synchronized with the peer's Out-Buffer and vice versa (see Fig. 8.7).

As opposed to the Pickup Store and the In-Out-Buffers the Common Store is the only key buffer which is persistent. Both other key storage devices are transient, meaning they are emptied at the start and at the end of a Q3P instance lifetime. Any key material placed in the Pickup Stores or the In-/Out-Buffers is lost if Q3P terminates. The Common Store places its key material on some media (e.g., Secured

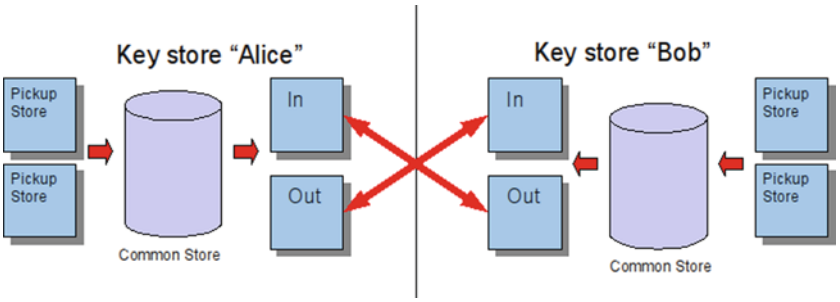
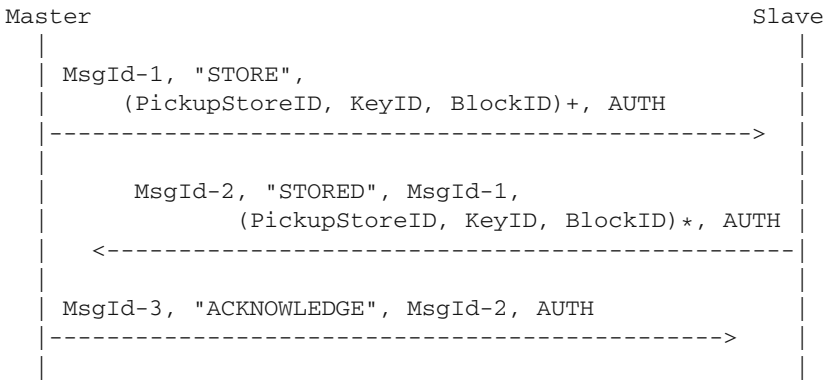


Fig. 8.7 Interleaved Q3P In-Out-buffers

Hard Disk, Smart Card, ...). On starting the Common Store is reincarnated with already collected and synchronized key material. This behavior is to some extent crucial since authentication protocols depend on preshared secrets, which can be now served from the Common Store.

In order to keep these buffers in synchronization, Q3P introduces a Master–Slave relationship. One of the Key Stores in a Q3P Association is the Master, the second the Slave. The central point thereby is that only the Master authorize key material to be transferred from a key buffer to another one.

Among the protocols which are needed for the Key Store to function properly the two most important subprotocols are described as follow. In order to transfer Key Material from the Pickup Store to the Common Store the Master Key Store ignites a STORE subprotocol.



Q3P key Store 3-phase STORE subprotocol

The data sent from the Master to the Slave is a MsgId-1, holding a unique number for this message, a STORE command identifier, followed by a minimum of 1 (PickupStoreID, KeyID, BlockID) tuple addressing the Pickup Store, the key within that Pickup Store and the BlockID to be assigned within the Common Store. The message itself is authenticated with an AUTH tag.

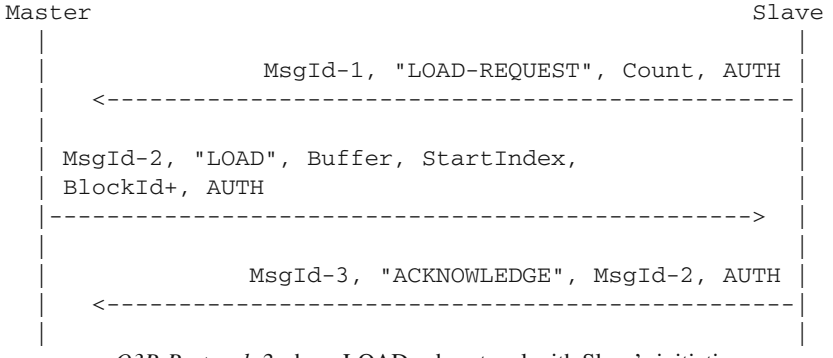
On response the Slave issues another MsgId-2, the number for the Slave’s response message, a STORED keyword, MsgId-1 pointing to the former message of the Master for which this message is a response to, a series of (PickupStoreID, KeyID, BlockID) tuples – which may be empty – and finalized with another AUTH tag.

Finally, the Master responses in the same manner with an ACKNOWLEDGE keyword adding the message id of the Slave to which is an acknowledgment.

As the communication may break at any time the keys are not directly pushed to the Common Store at once. At step 1 – the Master sending the STORE command – the Master Key Store holds the series of keys taken from the Pickup Stores in suspense and does not place them in the Common Store now. On reception of the STORE command, the Slave selects the keys he has and creates the answer STORED command. However, the Slave pushes the keys to the Common Store but labels them as *potentially in sync* with the Master's Common Store. If the Master receives now the STORED answer from the Slave, the Master knows that the Slave has withdrawn these keys from the Pickup Store and therefore are present in the Slave's Common Store. The Master now writes the keys to the Common Store and labels them as *real in sync*. As the Slave gets the ACKNOWLEDGE message from the Master, he now relabels these keys in the Common Store as *real in sync*.

After the STORE subprotocol Master and Slave do have several keys which have been pushed by the QKD devices in sync. But in order to utilize these keys they have to be dedicated for a particular direction: inbound or outbound. Keys found in the Out-Buffer are dedicated to be used for sending messages. Therefore, on sending a key of suitable length is withdrawn from the Out-Buffer for encryption and/or authentication. Contrary, the In-Buffer holds keys which are used for decryption and authentication checks of received messages.

In order to fairly fill these two buffers, Master and Slave observe their In-Buffer. If the Master detects a shortage of key material in its In-Buffer it selects key material from the Common Store which is labeled as *real in sync* and issues the Slave to load this key material. This is done by the LOAD subprotocol. If the Slave encounters shortage of its In-Buffer it sends the Master a request to initiate a LOAD subprotocol but with different target Buffers. With a single LOAD command as many key material as possible is to be loaded but not more than $\lfloor |CommonStore|/2 \rfloor$. By having $d = |InBuffer| - |OutBuffer|$ "shortage" is present if $d \leqq 0$ on Master side and $d < 0$ on Slave side. The In-/Out-Buffers do have a maximum capacity, which may not be exceeded, since every key material present in these buffers is lost if the system goes down. As the encryption key is taken from the Out-Buffer the maximum length of a single message is defined by this limit.



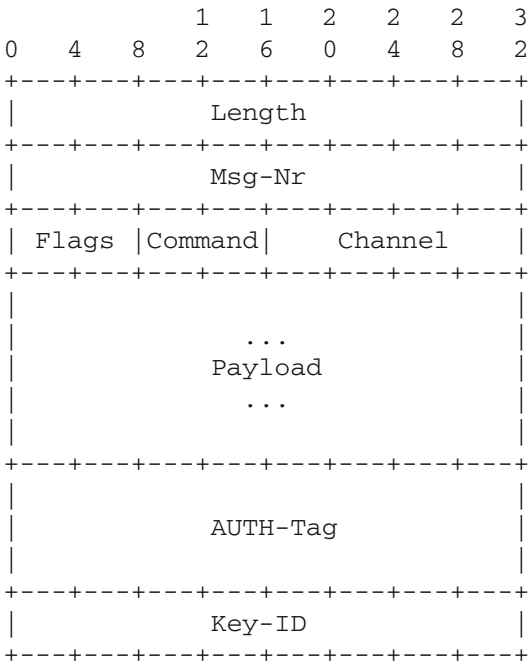
Q3P-Protocol: 3-phase LOAD subprotocol with Slave's initiative

It is important to note that only the Master decides which key material is to be distributed to which Buffer. Since the Master chooses only *real in sync* key material he knows that this material must be present at the Slave's side; the same is not true for *potentially in sync* keys on the Master side. Also by observing the In-Buffer the communication paradigm is receiver-driven: the sender may not send more information than the receiver can compute; the receiver dictates the speed of transmission.

On startup the operator of a Q3P link decides which side becomes a Master and which one the Slave. As preshared secrets are needed for the first authentication by convention the Master picks up the first block of the Common Store and loads it into its Out-Buffer and the second into its In-Buffer. The Slave does likewise vice versa.

8.3.5 Q3P Packet Layout

A Q3P packet is defined as:



Q3P packet layout

At first the length of the packet is given, including the total header and trailer, followed by the message number. Next a flag indicates if the message has been encrypted and/or authenticated. The channel field holds the channel identifier. Then we find the payload. If the encryption flag is set the payload is the cipher else it is the plaintext. Presence of the authentication flag in the flags field requires the authentication tag to follow the payload part. Finally, the key identifier is added. This key identifier points into the In- or corresponding Out-Buffer as an offset. As

encryption and authentication are always performed sequentially the address of one continuous stream of keys in the buffer is sufficient.

Hence that the authentication tag may have varying sizes. The concrete authentication algorithm to use and therefore the generated tag size must be set already by the operator out-of-band on a per Association basis. Further, the packet does not indicate the nature of the enclosed payload like a protocol ID or whatsoever. Q3P requires the parties listening on the channels to understand and interpret the given data and hands it out as-is.

8.4 Routing

With the help of Q3P, point-to-point connections are established which are capable of sending and receiving messages in an information theoretical secure manner. Q3P is now capable to encapsulate protocols; one can do this with the well-known IP to achieve routing. But for dynamic routing each hop or node has to maintain its routing tables and in order to establish some sort of QoS we have to examine known interior gateway protocols like OSPF and adopt these to QKD needs.

When using OSPF (see [8]) messages about the current state of the network are "announced" by so called LSA[2] packets. These announcements are flood periodically in the network to inform neighbor nodes of the current "best" routes. Each node states its current costs to other nodes/networks it knows. Based on this information the participating nodes can recalculate their routing tables to estimate the shortest paths to known nodes/networks. "Shortest" actually means "cheapest" in terms of cost, whereas the cost function should be flexible, taking key store capacities and key generation rate into account. If no further action is undertaken each and every LSA is authenticated and encrypted, which seems too much:

1. It is unclear to which extend LSA packets have to be encrypted. What can an attacker do with the information about the link states of each QKD node?
2. As key consumption regarding encryption equals message size a constant key consumption rate is ongoing without the network processing user data.

These and other issues are addressed by the QKD-NL – Quantum Key Distribution - Network Layer – protocol (see [2]).

As the messages are passed via Q3P we can choose to only authenticate the messages and not to encrypt them. This already leverages (2) although the background key consumption of the network is not zero, since every LSA packet receives at least an authentication tag. This tag costs for key consumption equals tag size and occurs on every message sent.

We can further tweak background key consumption by shorten or lengthen the announcement timespan. By enlarging the timespan we can spare key material and lower network background key burning. However, the bigger the timespan the slower network changes are propagated through the network and this may result

[2] *L*ink *S*tate *A*nnouncement

in inefficient network usage, since things may change quicker than communicated or lots of failed connection attempt since nodes may not be present any longer as suggested on the ingress side.

As opposed to standard OSPF the new QKD-NL protocol considers more than one path to a destination node. This is done in order to achieve some kind of load balancing. In OSPF the forwarding decision is based solely on the shortest path and thus every packet addressing a certain network will pass through one single port. In QKD-NL, however, a local load balancing is introduced. Along with the standard routing table an *Extended Routing Table* is present, which lists all costs from one node to each target nodes in increasing magnitude of cost. Next a *Load State Database* keeps track of the approximated load of each outgoing link. Whenever a packet is to be transmitted the Load State Database is looked up and if the local load of the outgoing link is below a threshold the packet is transmitted on exactly this link. If the current load on the link exceeds the limit the next best link is looked up and so forth.

8.5 Transport

A QKD network enables two parties, Alice and Bob, to share the same key on an end-to-end connection across several intermediate hosts. After operating jointly on a common view of the QKD devices and the key sources via Q3P and having an optimized routing scheme in the intermediate host given by QKD-NL such end-to-end key distribution systems can be designed.

Based on the well-known TCP the QKD-TL – *Quantum Key Distribution Transport - Transport Layer* – protocol addresses

- Reliability
- Congestion control
- Connection management
- Multiplexing network services

QKD-TL packets are enclosed in QKD-NL packets, which in turn are enclosed in Q3P packets. As QKD-TL orients itself tightly to TCP (as QKD-NL does with IP) the main differences to TCP are described here. The principle of QKD-TL is to pick a random number, preferentially from a true random number source, and send it hop-by-hop across the network. On the receiver side this random number is treated as key. Superfacial this can be achieved already with TCP, but differences do exist.

First, the messages which are transmitted over this network do not have arbitrary content. These are quantum generated keys as depicted in Sect. 8.1 and – as such – random numbers. For every hop an additional key plus the key material for an authentication tag is consumed. Now, whenever some node along the transmission path has to drop such message due to congestion pure TCP results in a timeout on the sender side, halving the windows size and resending the packets starting from the last good acknowledgment.

In QKD-TL things are different. Since we do not know where the packet has been dropped we might sacrifice a lot of key material for classical TCP resend. Even worse: the key might have been delivered but the corresponding ACK packet got lost. Now resending the whole key again for no additional information benefit besides the sender knowing the key has already reached the receiver is not optimal. Based on the fact, that within QKD-TL a message content is actual a series of pure random bits without any further informational impact or application intention besides using this data as a cryptographic key, there is no resend mechanism. This has a deep impact in subprotocols to follow.

Congestion deals with the problem on how to lower network traffic in such a way that every participating party has a fair amount of bandwidth. The resources in question are the QKD keys needed for transport in each intermediate host. The technique to calculate the reserves is to watch the outgoing key stores of Q3P plus the approximated key generation rate on a single Q3P link. If these values drop below a certain threshold a key shortage is likely to happen. Now QKD-TL introduces a single new Bit in the flags section of the TCP header field: *Congested*. This Bit is set, whenever a node sees its key material on an outgoing line below its label. The destination node suppresses the sending of an ACK packet back whenever this Congested Bit is set in packet it reads. This results in a timeout on the sender side and thus halving the window size which in turn lowers network bandwidth and key consumption.

Missing a resend facility within QKD-TL raises new questions on how to hand out key material. For this, QKD-TL cannot be seen as a stream of key material but has been rather designed utilizing a client–server approach. Figure 8.8 pictures the basic setup. It shows a typical setup: two areas are drawn which share access to a QKD network on the bottom and access to a classical public network on top. On the left side resides a client which wants to share a key with a server on the right side. This scenario is true for companies having several branches connecting to some central unit running various application servers.

Itemizing the steps in a simplified manner:

1. The application server registers itself within the QKD network and states it readiness to accept key requests by clients through the QKD network. It does this registration on its next QKD network node running a QKD-TL connection manager.
2. A client contacts its nearest QKD network node and requests the residing QKD TL connection manager there to establish a connection with the server.
3. Throughout the QKD network a connection is established, session IDs exchanged. Along these sessions, the client side QKD-TL manager starts generating keys along with a key ID, sends them across the QKD network to the server's QKD-TL manager. The latter hands the keys out to the server and acknowledges the key reception. On receiving the acknowledgment the client's QKD-TL manager hands out the key to the client.
4. Based on the session ID and on the key ID the client and the server negotiate keys for concrete application usage.

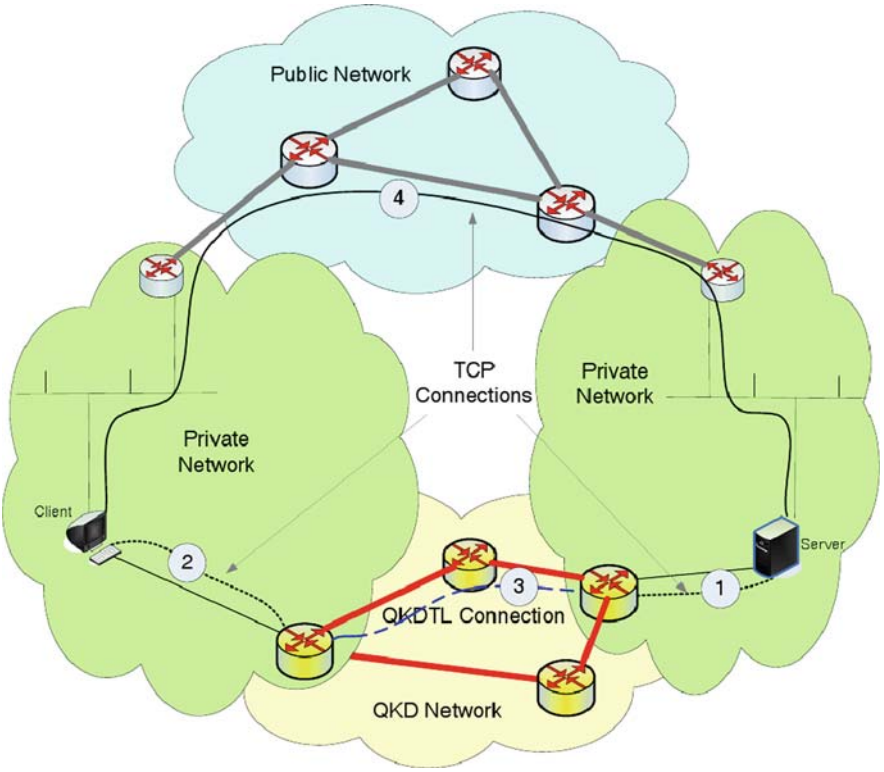


Fig. 8.8 QKD-TL and client-server (source: [2])

As QKD-TL does not have a resend mechanism any key transmitted from the client's side which lacks a corresponding acknowledgment may be lost. QKD-TL itself does not provide a key handshaking protocol yet but whenever one is going to be designed this idiosyncrasy prevails: due to the fact that not the key packet but an ACK packet may have been lost in the QKD network, the server may have keys the client lacks at his disposal. Therefore, any key negotiation must ground on client key material for the some of the server's keys have to be dismissed.

References

1. Bennett, C.H., Brassard, G.: Quantum cryptography: Public key distribution and coin tossing. In: Proceedings of the IEEE International Conference on Computers, Systems, and Signal Processing, pp. 175–179 (1984) 152
2. Dianati, M., Allaume, R., Gagnaire, M., Shen, X.S.: Architecture and protocols of the future European quantum key distribution network. In: Security and Communication Networks. John Wiley & Sons, New York (2009) 167, 170
3. Gross, G., Kaycee, M., Li, A., Malis, A., Stephens, J.: RFC 2364: PPP over AAL5 (1998). URL ftp://ftp.internic.net/rfc/rfc2364.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2364.txt 155

4. Hamzeh, K., et al.: RFC 2637: Point-to-point tunneling protocol (pptp) (1999). URL http://www.ietf.org/rfc/rfc2637.txt 155
5. Kent, S., Atkinson, R.: RFC 2401: Security architecture for the Internet Protocol (1998). URL ftp://ftp.internic.net/rfc/rfc1825.txt, ftp://ftp.internic.net/rfc/rfc2401.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1825.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2401.txt 158
6. Mamakos, L., et al.: RFC 2516: A method for transmitting ppp over ethernet (pppoe) (1999). URL http://www.ietf.org/rfc/rfc2516.txt 154
7. McGregor, G.: RFC 1332: The PPP Internet Protocol Control Protocol (IPCP) (1992). URL ftp://ftp.internic.net/rfc/rfc1332.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1332.txt 154
8. Moy, J.: RFC 2328: Ospf version 2 (1998). URL ftp://ftp.internic.net/rfc/rfc2364.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2364.txt 167
9. Poppe, A., Peev, M., Maurhart, O.: Outline of the secoqc quantum-key-distribution network in vienna. In: International Journal of Quantum Information IJQI, Vol. 6. World Scientific Publishing (2008). URL http://arxiv.org/abs/0804.0122 151
10. Postel, J.: RFC 793: Transmission control protocol (1981). URL ftp://ftp.internic.net/rfc/rfc793.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc793.txt 158
11. Romkey, J.L.: RFC 1055: Nonstandard for transmission of IP datagrams over serial lines: SLIP (1988). URL ftp://ftp.internic.net/rfc/rfc1055.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1055.txt 154
12. Simpson, W.: RFC 1661: The point-to-point protocol (PPP) (1994). URL ftp://ftp.internic.net/rfc/rfc1548.txt, ftp://ftp.internic.net/rfc/rfc1661.txt, ftp://ftp.internic.net/rfc/rfc2153.txt, ftp://ftp.internic.net/rfc/std51.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1548.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc1661.txt, ftp://ftp.math.utah.edu/pub/rfc/rfc2153.txt, ftp://ftp.math.utah.edu/pub/rfc/std51.txt 151, 154
13. Zimmermann, H.: Osi reference model–the iso model of architecture for open system interconnection. IEEE Transactions Communications Vol. COM-28 (1980). URL http://www.comsoc.org/livepubs/50_journals/pdf/RightsManageme% nt_eid=136833.pdf 154