Transport Layer Protocols for the Secoqc Quantum Key Distribution (QKD) Network

Mehrdad Dianati and Romain Alléaume
GET-ENST, Network and Computer Science Department, CNRS UMR 5141
46 rue Barrault F-75634 Paris Cedex 13 France

Abstract—Quantum Key Distribution (QKD) is an alternative key distribution technique that, unlike the classical approaches, can provide unconditionally secure keys for data communications over public communication networks. The European projet Secoqc (Secure Communication based on Quantum Cryptography) aims at developing a global network for unconditionally secure key distribution. This paper specifies the major elements of the transport layer protocols of the Secoqc QKD network.

I. INTRODUCTION

Implementation of many cryptographic schemes, such as encryption and authentication algorithms, rely on a proper key distribution scheme among the legitimate communicating parties. The strength of a cryptographic algorithm is directly linked to the difficulty of obtaining the secret keys by adversaries; key distribution schemes can be identified as one of the most sensitive parts of the security systems in communication networks. It is highly desirable to develop unconditionally secure key distribution systems, i.e., adversaries cannot obtain shared secrets regardless of their computing resources.

Quantum Key Distribution (QKD), developed in 1984 by Charles Bennett and Gilles Brassard [1], based on some earlier ideas of Stephen Wiesner [2], is an alternative solution for key distribution. Relying on the principles of quantum mechanics, a basic QKD system, as shown in Fig. 1, can establish a shared secret with unconditional security between two communicating parties, namely Alice and Bob [3][4]. Security of QKD relies on the fact that it is impossible to gain information about non-orthogonal quantum states without causing detectable perturbation [5]. This implies that Alice and Bob in Fig. 1 eliminate the compromised portion of the information that they exchange through quantum channel and establish unconditionally secure keys. Regardless of her computing resources, Eve cannot obtain useful information about the established shared secret between Alice and Bob.

Practical deployment of QKD systems is currently limited by few technical challenges. QKD links can only operate over point-to-point connections. Furthermore, the current QKD links have limited key generation rate and cannot be deployed over arbitrarily long distances. The development of QKD network architectures appears from this perspective as a necessary step towards effective deployment of QKD. A network of QKD devices can significantly address these problems by: 1) extending the range through chains of QKD devices; 2) improving utilization through resource sharing; and 3) providing point to multi-point communication structures.

The objective of the Secoqc project is to develop the first European QKD network for long-range high security communications. The Secoqc QKD network is built based on trusted classical relay nodes. The main originality of the Secoqc project, with respect to previous works, relies on the fact that we have opted for a dedicated key distribution network infrastructure that we call network of secrets. The functionality of the network of secrets is solely to forward session keys among communicating application pairs. The Secoqc QKD network is characterized by its dedicated link, network, and transport layers. The specific design concept of the Secoqc QKD network is the implementation of local key stores. QKD devices operate at their maximum key generation rate, and the surplus, when the traffic over a link is light, is stored in local key stores for future, when the traffic over the link is high. A pair of application programs access the network by connecting to a pair of network nodes. When one of the application programs requests a session key, the corresponding node generates a random number and forwards it through the QKD network to the other application program which is connected to another network node. The session keys are encrypted (one time padded) using the local keys over individual links in their way towards their destination nodes. To accomplish the task of key forwarding, similar to the conventional communication networks, a set of properly designed network architectures and protocols are required. The overall architecture and some components of the communication protocol stack of the Secoqc QKD network are specified in [6]-[8]. In this paper, we propose the general specification of the transport layer protocols for the Secoqc QKD network.

The rest of this paper is organized as follows. An overview...
of the architecture of the Secoqc QKD network is given in Section II. A logical view of the QKDTL is given in Section II. The concepts of QKDTL connection setup and termination are explained in Section III. In Section IV the concept of admission control is discussed. In Section V key synchronization and end-to-end reliability measures are specified. Flow and congestion control issues are addressed in Section VI.

A. Architecture of the Secoqc QKD Network

The architecture of the Secoqc QKD network is shown in Fig. 2. The network consists of Quantum BackBone (QBB) nodes that are connected by QBB links. QBB nodes are similar to routers in the conventional communication networks. A QBB node consists of multiple basic QKD devices. QBB links may be a single, as depicted in Fig. 1, or a combination of multiple parallel QKD links. QBB nodes are located inside secure and trusted sites. Users may access to the network through QBB or Quantum Access Node (QAN) nodes. A QAN node is a QBB node with limited routing functionalities, while it is more specialized in providing access point to many users. A QAN node is connected to a QBB node with a secure connection, e.g., a QKD link.

The application programs who intend to have secure communications over a public network, such as the Internet, use the QKD network for establishing unconditionally secure session keys which can be used for different cryptographic purposes. It has to be emphasized that, in a typical scenario, the QKD network does not carry real data traffic among application programs. The sole responsibility of the QKD network is to distribute secret keys with unconditional security. The main data communications are performed over the existing public networks such as the Internet. In a typical scenario, upon the request of an application program, namely the client, the ingress QBB or QAN node, which is the access point for the client application program, examines the possibility of providing a path to the requested destination. The destination application program, namely the server, obviously, must also be connected to another QBB or QAN node, which is referred to by the egress QBB or QAN node. If the QKD network has enough resources, the ingress node accepts the request; otherwise, the request is rejected. If the path is established successfully, the ingress QBB or QAN node generates random session keys and forwards them through the QKD network to the egress QBB/QAN node, which delivers the keys to the server application program. In its way through the QKD network the session keys are secured through hop-by-hop one-time-padding using the local keys. Local keys, that are theoretically secure, are established via QBB links between adjacent QBB nodes. QBB nodes are located inside private and secure sites. Regarding that the local keys are unconditionally secure and that the QBB nodes are inside secure locations, we can conclude that the QKD network can be used for distribution of unconditionally secure session keys. Note that the QBB links, which connect adjacent QBB nodes, are deployed over potentially unsecured locations; thus, they are vulnerable to different types of security threats. Furthermore, redundant paths over the QKD network can be utilized by the routing and forwarding protocols to combat Denial of Service (DoS) attacks.

II. OVERVIEW OF THE QKDTL

As specified in [8], the communication protocol stack of the Secoqc QKD network has a layered architecture, which is shown in Fig. 3. The protocol stack consists of four layers: 1) QKD Link Layer (QKDLL); 2) QKD Network Layer (QKDNL); 3) QKD Transport Layer (QKDTL); and 4) QKD Application Layer (QKDAL). The QKDTL complements the

---

**Fig. 2. The structure of the Secoqc QKD network**

**Fig. 3. The communications protocol stack of the QKD network**
network services by providing connection oriented reliable and synchronized key delivery for the client applications. This guarantees that the application pairs at the two ends of a QKDTL connection to receive keys with proper sequence and length information. Multiplexing and demultiplexing of the incoming/outgoing traffic at ingress and egress QBB nodes is also performed by the QKDTL. In addition, QKDTL maintains stability of the QKD network through flow and congestion control.

The logical architecture of the QKDTL, their internal, and external interactions are shown in Fig. 4. Admission control processes the connection request from the application programs and based on the status of network decides either to accept or reject the request. If the connection request is accepted, the connection management modules establish a QKDTL connection between the corresponding QBB nodes. A QKDTL connection controls transmission flow of session keys between ingress and egress QBB nodes.

All functionalities of the QKDTL are performed by exchanging QKDTL packets, Fig. 5, between a pair of end applications. A QKDTL packet contains the following fields: 1) Client and Server numbers uniquely identifies a QKDTL connection; 2) Sequence and ACK numbers identify a single data or acknowledgment during the life time of a connection; 3) Data Offset indicates the number of 32-bit words in the header; 4) Reserved bits are for future extensions; 5) flag bits (URG, ACK, and etc.) determines the type of a packet; 6) Window specifies the size of the sender’s and receiver’s window.

III. CONNECTION SETUP AND TERMINATION

As mentioned earlier the QKDTL provides the interface layer for the application programs who might use the QKD network. A typical scenario is illustrated in Fig. 6. The sequence diagram of the entire process and the QKDTL packet are shown in Fig. 7.
First, the ingress QBB node sends a QKDTL SYN message as shown in Fig. 8. The egress QBB node replies the SYN message by a SYN ACK message as shown in Fig. 9. A final ACK message from the client will complete establishment of the QKDTL connection. A connection is uniquely identified by the client and the server numbers. Note that if the same client opens another QKDTL connection with the same server, the initiating QBB node, ingress node, assigns a different client number to the new connection. The server number for each connection is also a unique number. Upon the successful establishment of the QKDTL connection, the client application establishes a conventional TCP connection with the server application over the public network.

In the secure communication phase, the client and the server
applications may request for session keys. To efficiently utilize the QKD network services, the connection manager, who receives a key request, generates a random number with the maximum QKDTL packet size and forwards it through the QKD network to the QBB node at the other end of a QKDTL connection. If the packet verification is passed successfully, the connection manager at the receiver side delivers the packet to the corresponding application and sends an ACK message back, indicating the correct reception of a certain packet which is identified by a unique sequence number. The sequence number is assigned by the connection manager at the initiator QBB node side. Note that at this phase, the initiator QBB node could be either the server side or the client side QBB node. When the QKDTL packet is acknowledged it can be delivered to the application program who has initiated the key request. Note that the QKDTL packet usually is a big chunk of key materials, around 64 kb, that may be broken into small pieces depending on the application. To improve the efficiency, each ACK packet may acknowledge multiple previously received packets. This will improve robustness of ACK messages. In the case that an ACK message is lost due to network congestion, the initiator connection manager does not drop the packet immediately, waiting for the upcoming ACK messages. After a reasonable timeout period, the connection manager at the initiator side assumes that the packet has been dropped due to network congestion and drops the packet. Note that the QKDTL connection is a full-duplex connection, i.e., both the client and the server may initiate a key request.

A secure communication must be properly terminated. Either client or server may initiate the termination process. For a successful termination: 1) the TCP connection between client and server must be terminated; 2) the QKDTL connection between the ingress and egress QBB nodes is closed; 3) the TCP connection between client and the corresponding QBB node must be terminated. The sequence diagram for connection termination are shown in Fig. 10 (initiated by the client) and Fig. 11 (initiated by server). If a server terminates a connection, the TCP connection between the server and its corresponding QBB node is not terminated. Note that, similar to the conventional TCP, each connection termination entails 4 QKDTL Messages.

Fig. 12 and Fig. 13 show the state diagrams of the client side and server side connections, respectively.

IV. ADMISSION CONTROL

The purpose of admission control is to prevent system overload that might cause significant performance degradation. A meticulous integration of admission control with access control and dynamic resource allocation policies is still a major challenge in implementing guaranteed QoS services in shared systems. The practical approach in public networks is to monitor network resources, spot bottlenecks, and increase the capacity of the overloaded devices. This might not be economically feasible for the QKD networks. Thus, a heuristic admission control might be a considerable option. Although such a scheme cannot be a robust base for guaranteeing QoS, it mitigates frequent congestions by limiting the number of concurrent connections through the QKD network.

Admission control of the QKD network can be performed during connection setup phase, when the QKDTL SYN and QKDTL SYN ACK messages are exchanged between ingress and egress QBB nodes. The payload of a SYN message, as shown in Fig. 8, contains the average required key rate requirement of the connection. Each QBB node in the forward path examines the SYN message; if the rate requirement cannot be supported the QBB node simply drops the SYN packet. Similarly, the payload of SYN ACK, as shown in Fig. 9, packet contains the required rate for ACK messages, which can be calculated by the egress QBB node. The QBB nodes in the reverse path examine the rate requirements in the reverse path, and drop the SYN ACK packet if the requested rate cannot be supported.
V. RELIABLE AND SYNCHRONIZED KEY DELIVERY

The QKD network protocols do not ensure reliable delivery of session keys. It is also important for an application pair to have a consistent numbering scheme for session keys in order to guarantee proper consumption of the session keys. To avoid the complexity of the network protocols of the QKD network, these important problems are addressed by the QKDTL protocol.

For key synchronization, each session key is assigned a unique sequence number. The initial value will be set during the connection establishment phase. Each session key is a big chunk of random bits, e.g., 64 KB, that may be consumed by the application programs in smaller blocks. For proper consumption of a session key, the application program may define a local pointer, as shown in Fig 14, to keep track of the consumed bits.

The proposed approach to enable reliable key deliver is based on feeding back the ACK packets by the receiver for successfully received packets. The transmitted session keys will be placed in transitional buffers. When the acknowledgment of a session key is received, it will be delivered to the application program and removed from the transition buffer. The acknowledgment scheme of the receiver is slightly different than that of the TCP. Upon successful reception of a session key, the receiver node will send an ACK packet to acknowledge the current and some arbitrary number of the previously received session keys. Acknowledging multiple packets can improve the impacts of lost ACK packets at the cost of larger ACK packets. However, since the ACK packets are not encrypted (but must be authenticated), their size does not increase key consumption. Note that, unlike TCP, there is no need for retransmission of the same packet.
after ACK timeout. In the case that a session key doesn’t get acknowledged for a relatively long period of time the sender node removes it from the transitional buffer.

VI. CONGESTION AND FLOW CONTROL

Congestion may occur communication networks when certain network resources are loaded beyond their capacity, causing packet dropping by the congested network devices. Congestion control mechanisms play important role in maintaining network stability and achieving a high utilization of the network resources. There are two principle approaches to reduce the probability of congestion or recover from a congested state: 1) resource reservation and traffic policing; 2) end-to-end congestion control. Although resource reservation looks a natural solution, its implementation incurs significant signaling overhead and complexity of the network protocols. In end-to-end approaches, such as the TCP congestion control, a sender node dynamically adjusts its transmission rate based on its observations of the network through the delayed or missing acknowledgement packets. End-to-end congestion control is simpler and more scalable than reservation based approaches. Furthermore, considering the variable nature of the network traffic, reservation may not be the best way of resource utilization. End-to-end congestion control is our choice as the current Seqoqc network protocols do not support reservation mechanisms.

A. Flow Control

For transport layer connections, a sender normally sends multiple packets before receiving an ACK to reduce the idle time of a connection as the round trip time is significantly larger than the transmission time of a single packet. However, this might lead to a situation that a fast sender overruns a slower receiver. Flow control is implemented in the transport layer of communication protocols to mitigate this problem. It is usually implemented through a sliding window mechanism which restricts the maximum number of packets that a sender can send before receiving an ACK. The limit is defined by a window size variable which is advertised by the receiver node during a connection setup. A sender must never increase its window size beyond the maximum advertised size by the receiver node. However, the transmission window size may be reduced by the sender due to network congestion (next subsection).

The sliding window mechanism of QKDTL is illustrated by an example in Fig. 15. In this example, the window size is 4; the sender sends 4 packets in row and waits for the first ACK. The arrival of an ACK triggers transmission of another packet. Note that there are always at most 4 unacknowledged packets in the sender side. Each ACK packet also contains acknowledgement for multiple packets.

B. TCP Congestion Control

Congestion occurs when some network links are loaded beyond their capacity. In such cases, network devices, i.e., routers, may drop packets as they run out of buffering memory. Thus, the transmission rate of the connections who share the congested link should be reduced to let the congested link recover back to a normal state. Congestion control is usually implemented in transport layer. A proper congestion control policy is the key factor of stable and efficient operation of packet switched networks. It is also an extremely complex issue as it entails dynamic collaboration of globally distributed and autonomous users. In this subsection we give an overview of the TCP congestion control mechanism; then, we specify that of the QKDTL.

The principle of the TCP congestion control is to allow a sender node to increase its transmission rate independently and react to observable congestion events which are learned through the pace of the ACK packets. The basic scheme, that has evolved by several later improvements, is called Additive Increase and Multiplicative Decrease (AIMD) [9] as shown in Fig. 16. In this scheme, a sender node controls its transmission rate using a variable, Congestion Window, (cwnd). Note that cwnd is a dynamic variable overriding the advertised window size that restricts the number of packets that can be sent prior to receiving an ACK. Starting from an initial window size, usually 1, the sender increases cwnd at a certain pace, e.g., 1 each Round Trip Time (RTT). The linear increase continues until the point that the receiver observes a missing ACK (timeout); then, assuming that the event indicates a congestion,
the cwnd is immediately halved to help the congested link recover back to a normal state. Note that the upper bound of cwnd is defined by the processing capability of the receiver node which is defined at the time of connection setup. AIMD is unnecessarily slow when the value of cwnd is small, e.g., in the beginning of a connection. This is know as the slow start problem. An improved version of TCP, known as Tahoe [10], proposes an exponential initial increase until a certain slow start threshold (ssthresh), i.e., cwnd is doubled each RTT. After ssthresh, cwnd is increased linearly. The initial value of ssthresh may be set equal to the advertised window size, the subsequent values are set to the half of cwnd when a missing ACK is observed. Further major improvements of TCP includes Fast Retransmit, Fast Recovery, Modified Slow Start (TCP Reno [10], TCP NewReno [11], SACK TCP [12], TCP Vegas [12], and TCP BIC [14].

For the QKD network, adopting a TCP-like congestion control might not be efficient as frequent packet dropping events, which is normal for TCP, can waste significant amount of local key materials. Therefore, we intend to take advantage of the specific characteristics of the QKD network to propose a congestion control policy that allows efficient utilization of network resources while minimizing packet dropping events. A proper estimation of RTT is essential for proper function of QKDTL. The both sides of a QKDTL connection implement a moving average filter to estimate RTT as follows.

\[
RTT(k) = (1 - \frac{1}{C})RTT(k - 1) + \frac{1}{C}rtt(k),
\]

where \(RTT(k)\) denotes the new estimate of RTT, \(rtt(k)\) is the new measurement of RTT, and \(C\) is the time constant of the filter.


ACK packets. The instantaneous RTT measured from postponed ACKs will not be used by the sender to update the average value of RTT.

\[
\text{Fig. 16. Additive Increase and Multiplicative Decrease (AIMD)}
\]

REFERENCES


