Design and Implementation of the QKD Control and Management Layers for Access Network Deployments

E. Kosmatos[(1)], A. Stavdas[(1)], A. Lord[(2)]

[(1)] OpenLightComm Ltd., The Ross Building, Adastral Park, Ipswich, IP5 3RE, UK
[(2)] Applied Research, BT, Polaris House, Adastral Park, Ipswich, IP5 3RE, United Kingdom

Abstract *The integration of QKD technology for 6G deployments facilitates advanced protection against attacks on critical infrastructures. We detail, implement and demonstrate the Key Management and Control Layer operations of a Free Space Optics-based QKD link for network access including key delivery with QoS guarantees.*

Introduction

Quantum technologies have the potential to instigate advanced security in 6G network infrastructures. Quantum Key Distribution (QKD) provides this security inherently as the keys cannot be accessed without tampering [1]. Among the candidate QKD technologies, Free-Space Optics (FSO) [2] may deliver high-grade security in a cost-effective way due to the high throughput and high-beam directivity.

Amongst the cryptographic services based on FSO-QKD, the UK-funded AirQKD project [3] is considering the secure handover between 6G Base Stations (BSs) for critical services like Vehicle-to-Infrastructure (V2I). For the V2I use-case, 6G X-Haul data is carried securely over FSO-QKD transceivers with an optical reach of 150-200m, placed on building rooftops to exchange secret keys within Line-of-Sight (see Fig.1).

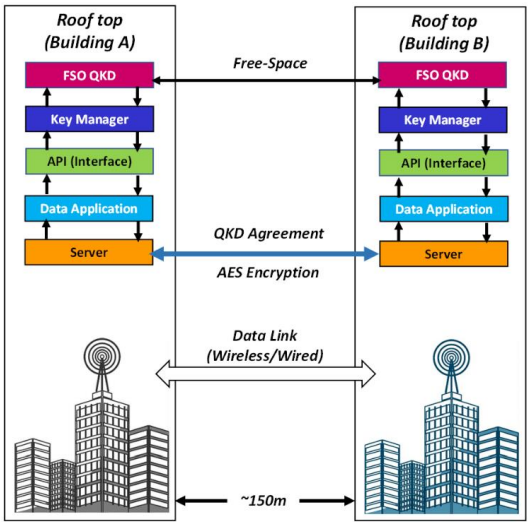

Fig. 1: Processes for secure message exchanges between rooftops based on FSO-QKD technology.

A 6G V2I application that is hosted in 'servers' requires a cryptographic service to encrypt a "message" via a classic protocol, e.g., Advanced Encryption Standard (AES) and thus it requests a quantum key to implement this encryption. A quantum key is generated via the FSO link by means of the BB84 QKD protocol. These keys are used to secure the classic encryption and, hence, the messages are exchanged between the two terminal points securely.

Overview of the FSO-QKD Architecture

Following various standards recommendations [4][8], the AirQKD end-to-end cryptographic service is made possible by means of coordinated actions between four layers; namely, i) the Quantum Layer; ii) the Key Management Layer (KML); iii) the QKD Control Layer; and iv) the Service Layer as shown in Fig.2.

In particular, the KML stores and manages the QKD keys as well as the synchronisation and reformatting of the bit strings. The Key Management Module (KMM) together with the QKD module and QKD driver reside in the same Trusted Node (TN). The KMM exposes interfaces to various cryptographic applications that aim to create a secure data link by means of QKD symmetric keys. The KMM receives key requests from Secure Application Entities (SAEs), and it allocates the appropriate number of keys to these entities. The KMM synchronizes with other KMMs, authenticates the keys, and supplies them under the appropriate format and the agreed Quality of Service (QoS) to both SAE ends.

The KMM supports the following functions: i) Key authentication; ii) Key storage; iii) Key protection; iv) Key identification; v) Key provision to applications on request; vi) Key replacement on request vii) Key destruction (based on decided key lifetime; viii) Management of the key pool; ix) Allocation of keys to applications based on the agreed QoS performance; x) Synchronisation between KMM entities for key exchange; xi) Key relay. The (x) and (xi) functions are essential to support a hop-by-hop key delivery between successive 6G BSs as in Fig.1. Finally, it is pointed out that since the KMM receives digital

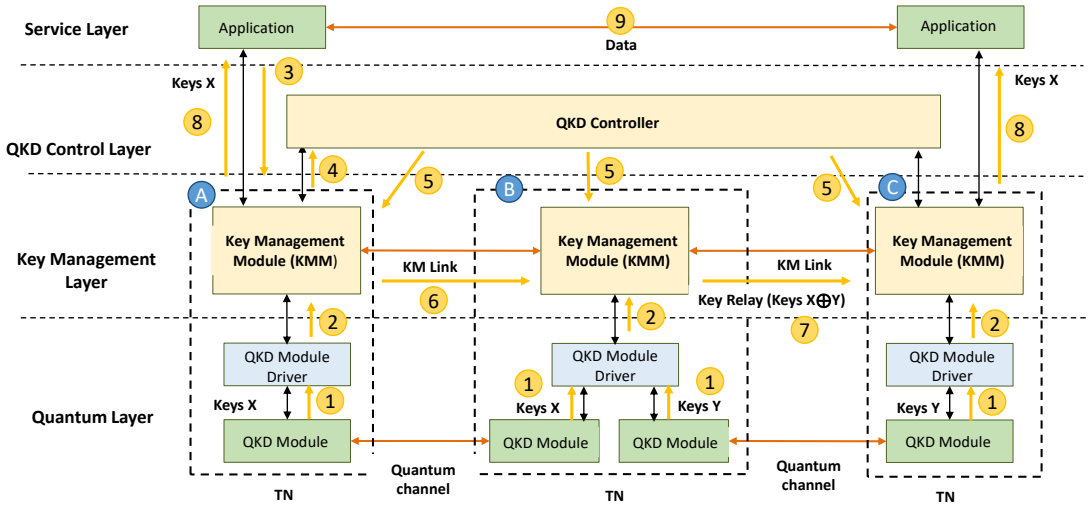

Fig. 2: A schematic illustration of the QKD Layer Architecture and QKD logical sequence

bits, it is agnostic to the specific QKD technology used, while the KML/TN node presented architecture is rather generic.

Overall, the KMM manages the pool of stored keys and keep them in a stable, managed state (e.g. by writing off any expired keys etc). A source SAE may request a secure data connection to a destination SAE by means of the corresponding interface (API) exposed by its local KMM module. The KMM is responsible for the management of the stored keys including the functions of key protection, provisioning, and destruction.

Each KMM module consists of several s/w sub-modules as follows: the *Event Manager* that is tasked with registering the events originating from the NBI and SBI interfaces while it includes an *App Request Handler* to process new requests from the SAEs. The *Key Allocation Element* (KAE) allocates the generated keys to the applications in an orderly fashion.

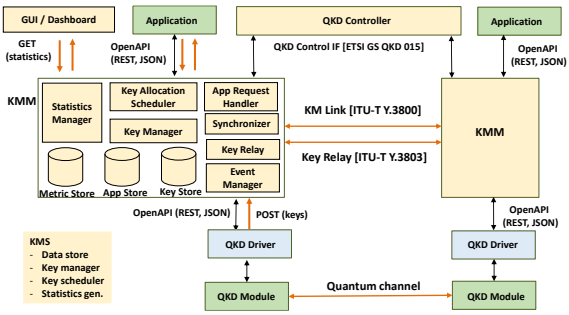

Fig. 3: The KMM architecture.

The KAE ensures that key allocation complies with a) the agreed QoS parameters set during the request phase; b) the instructions from the QKD Controller. The *Synchronizer* implements the synchronisation between KMM modules within the KM Link as of [4]. Similarly, the *Key Relay Element* (KRE) serves to deliver keys on a hop-by-hop basis between roof-tops following [8]. The *Statistics Manager Element* (SME) collects and analyses various statistics (e.g. unallocated keys, allocated keys, etc). Finally, *Key Store* caches the symmetric keys, the *App Store* registers application related information (including app characteristics and requested QoS) while the *Metric Store* stores the raw metrics collected and the corresponding analyzed statistics.

The Sequence of QKDN Operations

In the proposed framework, the end-to-end delivery of quantum keys is made possible following the steps as in Fig.2. QKD modules create symmetric keys in the form of quantum bits (1). The corresponding digital byte strings are delivered to the KMM modules using the appropriate interfaces (2). Each key is represented by its identification number (key_id) and the actual key data (key_raw). After *Application* requests a secure connection (3), the source SAE requests a new Key Distribution Service (KDS) from the local KMM (4) which provides the connection details (e.g. the SAE addresses) and the desired QoS performance. The 'source' KMM forwards the request to the QKD Controller which identifies to both SAE-ends the physical path and it forwards the necessary notifications to the relevant KMMs (5). These KMMs create a KDS and turn to 'standby' to allocate keys to the specific application.

With an appropriate message the Controller notifies the KMMs for each KMM role (i.e. Master or Slave), the QoS parameters and whether key relay is necessary. In Fig.2, the KMM(A) is Master, and the KMM(B) is Slave for the connection A->B while for the connection B -> C, KMM(B) is Master and the KMM(C) is Slave.

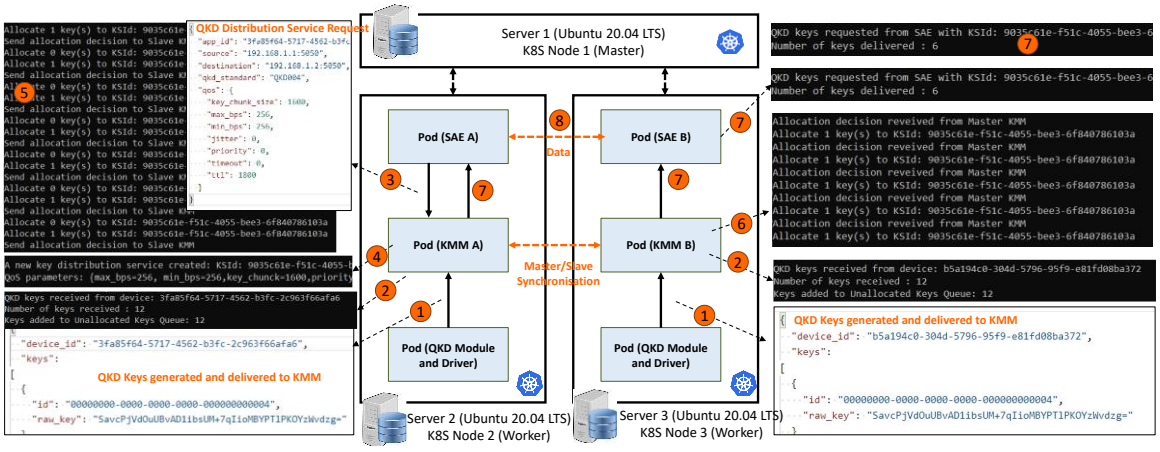

Fig. 4: Testbed and reference implementation deployment

A KMM Master allocates keys to the specific KDS and informs the KMM Slave of this decision. The KMM(A) allocates {n keys, X} (where X is the first QKD Module pair) to the specific application and informs KMM(B) (6). The same process is repeated for B->C (7) and KMM(C) allocates {n keys, Y} to the application (where Y is the second QKD Module pair). In this way, the two SAEs receive the same keys since, based on the key relay functionality realised between KMM(B) and KMM(C), the KMM(B) sends the XOR (X⊕Y) of the allocated keys, so the KMM(C) retrieve the keys X, by combining the keys X⊕Y and keys Y. Finally, SAEs use the keys received to encrypt/decrypt the data exchanged in the classical network link between the SAEs (9).

Reference Implementation and Deployment

The KMM of Fig. 3 is implemented, deployed, and evaluated. In detail, the KMM entity was developed as a RESTful Web Service using Java JAX-RS and was deployed on an Apache Application Server. NBI/SBI and inter-KMM interfaces were realized in REST/JSON following [5],[6] while the interface between KMM and the QKD Controller is as in [7].

To validate the implementation, two QKD Drivers are developed generating QKD 64-based encoded bit array symmetric keys of 32 bytes in size. In addition, a couple of SAEs are implemented with conventional encryption to emulate the encrypted data communication of a classical channel. Also, a QKD Controller is developed, capable of controlling two KMMs.

As the components are running as services, the architecture of Fig.3 is deployed in a Kubernetes cluster as shown in Fig.4. All components are deployed as Pods (building blocks of Kubernetes), while the communication between the pods is realized as a set of Kubernetes services. The Kubernetes cluster includes three nodes: one is the ‘Master’ (top Fig.4), while others are ‘Workers’ (bottom of Fig.4) emulating the two TNs of a QKD link between two rooftops.

After Fig.4, the emulated digital QKD keys are generated and pushed to the corresponding KMMs (1). The keys are stored in the unallocated keys queue of KMMs (2). After a SAE launches a KDS request to the source KMM (3), the KMM analyses the QoS parameters of the request and creates a new QKD service (4). Here, the SAEs requests 1 key (of 32 bytes length) per sec. The key allocation service runs periodically (every 0.5s in our case) and allocates any spare keys to the appropriate service queues based on their QoS parameters (5). During a particular allocation cycle, the master KMM (A here) sends a key allocation message to slave KMM (B here) and the slave KMM reciprocates (6) creating a symmetric service. Then both source-destination SAEs receive the keys from the respective KMMs (7). During the final step, the two SAEs are synchronised, so they use the digital quantum keys for data encryption or decryption, respectively.

Conclusions

We have designed, implemented, deployed, and evaluated the basic functionality of QKD Key Management and Control Layers to support the secure handover of messages between 6G BSs deployed at building rooftops. Open-source s/w was used to implement the KMM and the interfaces to other QKDN layers following ETSI and ITU-T recommendations. Although the AirQKD consortium considers FSO-QKD technology, the proposed KML architecture is technology-agnostic and, therefore, of wider interest.

Acknowledgements

This work was supported by the Innovate UK project AIRQKD.

References

[1] Tariq, M. R. Khandaker, K.-K. Wong, M. A. Imran, M. Bennis, and M. Debbah, "A speculative study on 6g," IEEE Wireless Communica- tions, vol. 27, no. 4, pp. 118-125, 2020.

[2] K. Matsuda et al., "Field Demonstration of Real-time 14 Tb/s 220 m FSO Transmission with Class 1 Eye-safe 9-aperture Transmitter," Optical Fiber Communications Conference and Exhibition (OFC), San Francisco, CA, USA, 2021, pp. 1-3

[3] https://gtr.ukri.org/projects?ref=45364

[4] ITU-T Y.3800, "Overview on networks supporting quantum key distribution", October 2019

[5] ETSI GS QKD 004, "Quantum Key Distribution (QKD); Application Interface", December 2020

[6] ETSI GS QKD 014, "Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API", February 2019

[7] ETSI GS QKD 015, "Quantum Key Distribution (QKD); Control Interface for Software Defined Networks", March 2021

[8] ITU-T Y.3803, "Quantum key distribution networks - Key management", December 2020