TSDSI TR 60XX V1.0.8

Interoperability of Multi-vendor QKD Hardware using SDN

09 2024

tsdsi
India's Telecom SDO

Telecommunications Standards Development Society, India
टेलिकम्यूनिकेशन्स स्टैण्डर्ड्स डेवलपमेंट सोसायटी, इण्डिया

TSDSI TR 60XX V1.0.8

Technical Report

Interoperability of Multi-vendor QKD Hardware using SDN

tsdsi

Telecommunications Standards Development Society, India

टेलीकम्यूनिकेशन्स् स्टैंडर्ड्स डेवलपमेंट सोसाइटी, इंडिया

(भारत का दूरसंचार मानक विकास संगठन)

Important Notice

Copyright Notification

Intellectual Property Rights

Telecommunications Standards Development Society, India

(TSDSI)

Registered Office Address
C-DOT Campus, Mandi Road, Mehrauli,
New Delhi, India – 110030
Tel: +91 124 4217 424
Fax: +91 124 4201 759

www.tsdsi.in

Table of Contents

List of Figures

Document Version & Release History

Version	Release Date	Change Description

Abstract

Quantum Key Distribution is emerging as a solution to overcome the challenges introduced by the Quantum computing on traditional asymmetric cryptography. The primary purpose of QKD is to share encryption keys between the QKD nodes which are directly connected through QKD links. The QKD works using the principles of quantum mechanics, and hence the QKD links are capable of detecting any unauthorized interception in the QKD transmission. QKD Networks are formed from multiple such QKD links connecting multiple QKD nodes. The current implementations of QKD protocols are proprietary in nature. This makes the interoperability of both QKD protocols and QKD Networks impossible. The problem of interoperability of QKD Network can be addressed at multiple levels of QKD network stack starting from the layer of quantum transmission. However, due to rapid and disruptive innovations are currently happening at the quantum level protocols, the standardization efforts are more focusing on higher layers including key management and key relay. The Software Defined Networking (SDN) concepts are also introduced for better manageability and controls of QKD networks and services. This study focusses on the problem of interoperability aspects of QKD Networks and the application of SDN for the same.

List of contributors & supporters

Name	Affiliation
Aneesh Kumar K B	CDAC, Thiruvananthapuram
Sajimon P. C.	CDAC, Thiruvananthapuram
Neena Suresh	ER&DCIIT, CDAC Thiruvananthapuram
Dr. A. Paventhan	ERNET
Ajay Pratap Shukla	CDOT
Yogesh Kumar	CDOT
Ashutosh Singh	IIT-M
Dr. Natarajan V	SETS

Abbreviations & Acronyms

QKD	Quantum Key Distribution
QKDN	Quantum Key Distribution Network
SDN	Software Defined Network
DV-QKD	Discrete-Variable QKD
CV-QKD	Continuous-Variable QKD
MDI-QKD	Measurement-Device-Independent QKD
LDPC	Low Density Parity Check
SD-QKD	Software-Defined Quantum Key Distribution
ETSI	European Telecommunications Standards Institute
ISO	International Organization for Standardization
ITU-T	International Telecommunication Union’s Telecommunication Standardization Sector
IEEE	Institute of Electrical and Electronics Engineers

1. Introduction

Quantum Key Distribution Networks (QKDNs) are emerging as a cornerstone of secure communication in the face of evolving cybersecurity threats, including those posed by quantum computing. By leveraging the principles of quantum mechanics, QKDNs provide unparalleled security, enabling the exchange of encryption keys that are resistant to both classical and quantum attacks. As these networks gain momentum, experimental deployments worldwide are showcasing their transformative potential.

However, the challenge of interoperability across diverse QKD systems remains significant. Variations in protocols, hardware designs, and vendor-specific implementations create barriers to seamless integration, limiting the scalability and practical deployment of QKDNs. Addressing this issue is essential for establishing a unified, standardized framework for global quantum communication.

Efforts to enhance interoperability are focusing on protocol compatibility, the development of standardization strategies, and the integration of Software-Defined Networking (SDN) technologies. These advancements are paving the way for the next generation of secure, scalable QKD networks. By overcoming these challenges, QKDNs are set to realize their full potential as a critical component of secure communication in the quantum era.

A typical context of QKD network interoperability is illustrated in the figure below. The diagram shows two QKD networks, with the nodes of each network color-coded differently. Nodes A and G, which belong to different QKD networks, are interconnected using both trusted and non-trusted classical links. In the case of a trusted classical link, both nodes are assumed to be within a single secure boundary. Conversely, in the case of a non-trusted classical link, the nodes are outside the secure boundary. This study focuses on the interoperability of these multi-vendor QKD nodes to ensure seamless key exchanges between any two nodes in the connected networks. For example, the key exchange between App A and App C in the diagram.

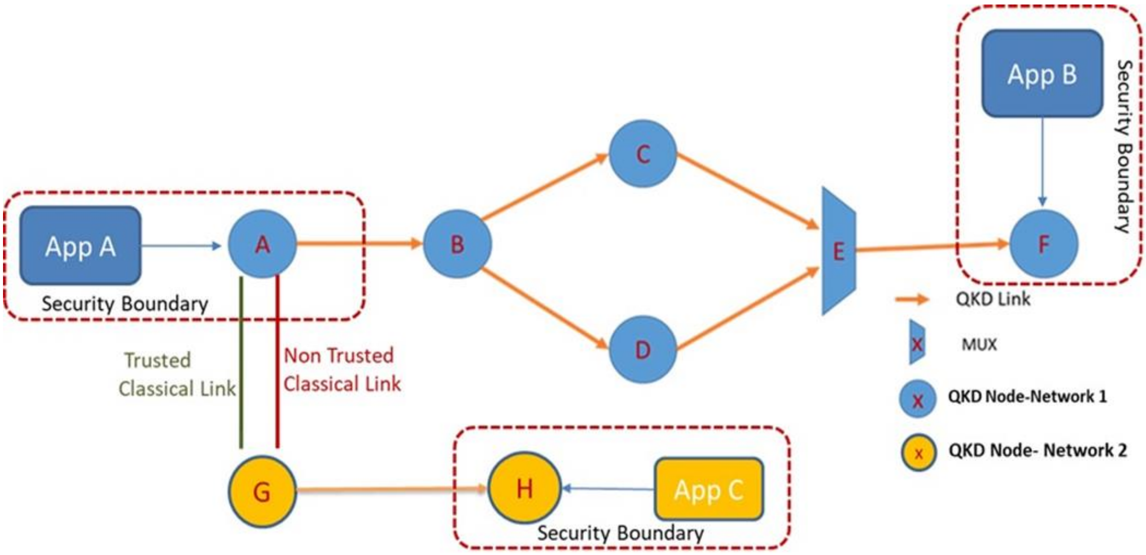


Figure 1: Context-QKD Network Interoperability

Software-Defined Networking (SDN) concepts are used for the control and management of QKD networks. Centralized control has been proven effective in QKD networks, where numerous control decisions must be made in consensus with multiple QKD nodes. The conventional decentralized approach adds significant overhead, as each QKD link comprises both quantum and classical channels, resulting in longer convergence times for QKD operations. This study also focuses on the application of SDN to achieve QKD network interoperability.

1.1 Purpose

The primary purpose of this project is to achieve the interoperability of QKD Networks through development of SDN based interface standards at the classical link to achieve interoperability of multi-vendor QKD Networks

1.2 Scope

The scope of this study is limited to interoperability aspects using the classical channel and SDN. Quantum channel interoperability is not considered, as the technology is still in its developmental stage, and it may be premature to standardize it

2. Overview of Quantum Key Distribution Network (QKD)

In 1995, Peter Shor proposed a quantum algorithm for finding the solution of the factoring problem in polynomial time. The classical counter part of the same has exponential running time which makes it practically impossible to factor an integer with thousand or more digits. The comparison of both algorithms is provided in the Figure 2. The growth and emergence in the field of quantum computing makes the Shores algorithm a near reality and it imposes a great threat to the public key cryptography which depends on the complexity of the prime factorization problem. The attack scenarios are depicted in the Figure 3, where the attacker Kevin is having a quantum computer and shall able to decrypt the encrypted communication between Alice and Bob while the classical attacker Eve still fails to find the key using a brute force attack.

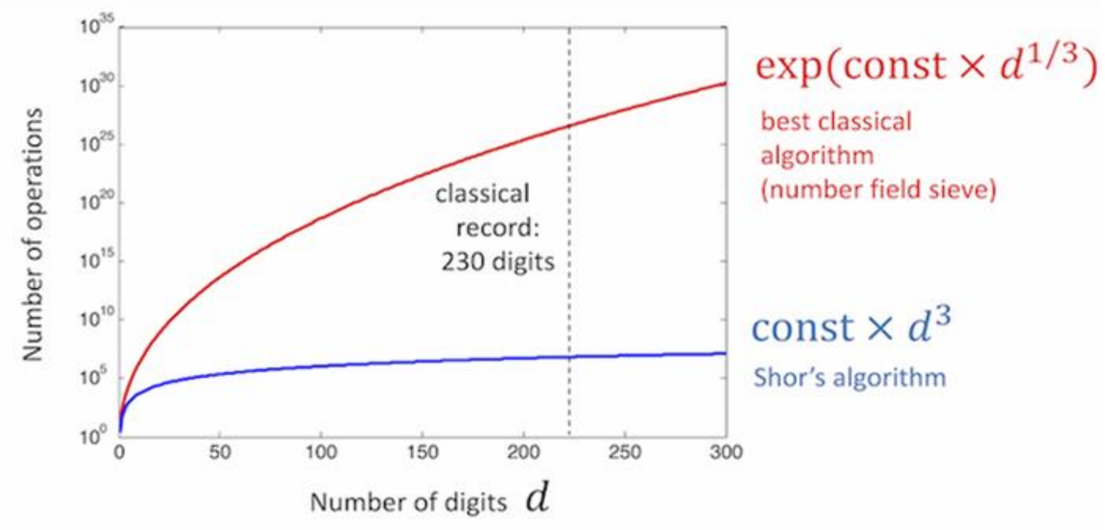


Figure 2:Comparison of factoring algorithms

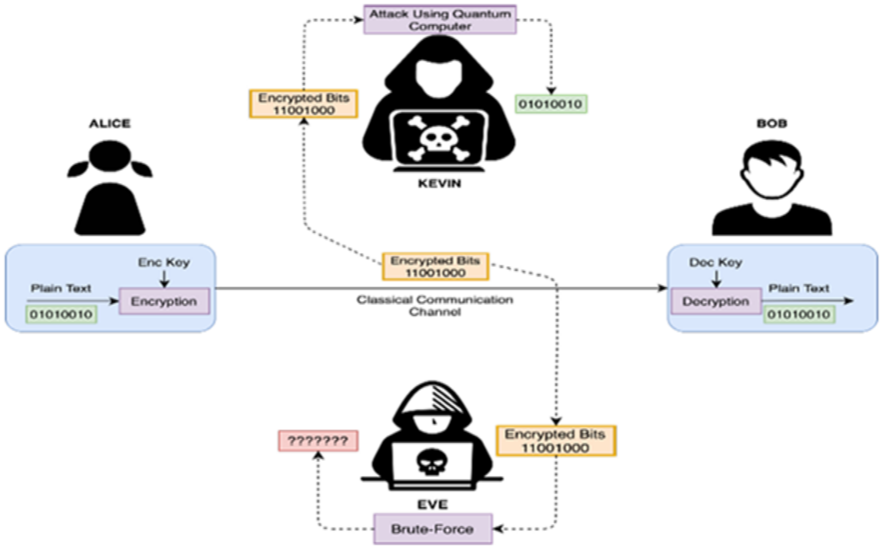


Figure 3: Attacking Scenario using Quantum Computer

The Quantum Key Distribution (QKD) is emerging as one of the cryptographic solutions to tackle the threat introduced by Quantum Computing to the public key cryptography. The main purpose of QKD is to generate and distribute symmetric cryptographic keys between geographically separated users in a quantum safe manner. QKD is a promising technology to achieve the information-theoretic secure (ITS) key exchange as it is based on the laws of physics rather than the computational complexity in determining the private keys. The principles of quantum mechanics on which the QKD is built include the following.

- **No Cloning Theorem :** The no-cloning theorem states that it is impossible to create an independent and identical copy of an arbitrary unknown quantum state.
- **Quantum Theory of Measurement and irreversibility**: Every measurement perturbs the system unless the measurement is compatible with the quantum state.
- **Non-Orthogonal Quantum States** : It is not possible to perfectly distinguish non-orthogonal quantum states.

CRYPTOSYSTEM	TYPE	IMPACT OF QUANTUM COMPUTER
RSA	Public Key	Insecure
Diffie-Hellman	Public Key	Insecure
ECC	Public Key	Insecure
AES	Symmetric Key	Require Large Key Size
One Time Pad (OTP)	Symmetric Key	Secure

Code Based	Post Quantum	Not Broken
Hash Based	Post Quantum	Not Broken
Lattice Based	Post Quantum	Not Broken
Multivariate	Post Quantum	Not Broken
QKD	Quantum	Secure

Table 1: Impact of Quantum Computer on Cryptographic Systems

2.1 Components of QKD Network

A Quantum Key Distribution (QKD) network consists of several key components that work together to securely generate and distribute encryption keys.

- **Quantum Transmitters and Receivers:** These are devices used to generate, send, and receive quantum bits (qubits), often using photons. The transmitter (usually referred to as Alice) encodes the key onto these qubits, and the receiver (usually called Bob) measures the qubits to extract the key.
- **Quantum Channels:** These are communication channels—typically fibre optic cables or free-space links—that carry the qubits from the transmitter to the receiver. Quantum channels are sensitive to interference and loss, so maintaining their integrity over long distances is a challenge.
- **Classical Communication Channel:** A conventional (non-quantum) communication link between the transmitter and receiver is used for exchanging information needed for error checking, key reconciliation, and verification. This channel is usually encrypted, but it does not carry the key directly, only auxiliary data.
- **Key Management and Delivery System**: This system handles the storage, management, and delivery of the generated keys once they are securely established between Alice and Bob. It interfaces with cryptographic applications that need secure keys, ensuring efficient key use and management.
- **Authentication Mechanism**: An authentication protocol is essential to verify the identities of the communicating parties and protect against man-in-the-middle attacks. This ensures the key is securely exchanged between trusted parties only.

2.2 Classification of QKD Network

Quantum Key Distribution (QKD) networks can be classified based on the network structure, the type of protocols used, and the implementation approach.

- **Based on Network Structure**
  - **Point-to-Point (Peer-to-Peer)**: Direct communication between two parties, often used for short distances. This is the simplest form of QKD and does not require any intermediary nodes.

  - **Star (Hub-and-Spoke)**: A central node connects multiple users, allowing each to communicate with the central hub, which relays keys as needed. This structure simplifies management but depends heavily on the hub.
  - **Mesh Network**: Multiple nodes are interconnected, allowing for multi-hop communication and redundancy. This type enhances flexibility and scalability, suitable for larger, interconnected networks.
  - **Ring Network**: Nodes are connected in a circular structure, with each node connected to two neighbouring nodes. This structure is reliable, as there is often an alternate path if one node fails, but it can be slower due to increased path length.

- **Based on QKD protocols**
  - **Discrete-Variable QKD (DV-QKD):** Information is encoded onto discrete quantum states, such as photon polarization. Protocols like BB84 and E91 use this approach, making it widely researched and suitable for high-security applications.
  - **Continuous-Variable QKD (CV-QKD):** Information is encoded onto continuous variables, such as the amplitude and phase of light waves. CV-QKD can be compatible with standard telecom equipment, though it is more susceptible to noise.
  - **Measurement-Device-Independent QKD (MDI-QKD):** In this approach, an untrusted third party performs the measurement of qubits, making the protocol immune to certain hacking attacks. MDI-QKD strengthens security by reducing the risk of device vulnerabilities.

- **Based on Implementation Approach**
  - **Fibre-Based QKD:** Uses fibre optic cables to transmit quantum information, ideal for urban or medium-range distances but limited in range due to signal loss.
  - **Free-Space QKD (Satellite-Based QKD):** Uses satellites or line-of-sight free-space links to transmit qubits over longer distances, such as between cities or continents, enabling global QKD networks.
  - **Hybrid QKD Networks:** Combines fibre-based and satellite-based QKD, often using satellites for long-distance transmission and fibre for shorter, ground-based connections.

2.3 Quantum Channel

The **quantum channel** is the medium through which quantum bits (qubits), typically photons, are transmitted from the sender (Alice) to the receiver (Bob). The working of QKD in the quantum channel relies on the principles of quantum mechanics, particularly the behaviour of quantum states and the disturbance caused by measurement. The process of sharing keys between a sender and a receiver can be explained step-by-step as follows:

1. Sender Prepares Qubits

- The **sender** creates **quantum bits (qubits)**. These qubits will later form the secret key shared with the receiver.
- Each qubit is **encoded in a quantum state**. This encoding could be based on a property like **polarization** of a photon, which can take different orientations, such as horizontal, vertical, or diagonal.
- The sender **randomly chooses a basis** for encoding each qubit. There are two common bases for encoding:

- **Rectilinear basis**: Horizontal or vertical polarization (representing bit values 0 or 1).
- **Diagonal basis**: 45° or 135° polarization (also representing bit values 0 or 1).

Table 1:Illustration of Quantum Encoding using basis

Basis	**Binary Bit**	**Encoding Polarization**
Rectilinear (+)	0	Horizontal (↔)
Rectilinear (+)	1	Vertical (↕)
Diagonal (x)	0	45-degree (⤡)
Diagonal (x)	1	135-degree (⤢)

2. Qubits are Transmitted through the Quantum Channel

- Once the qubits are prepared, the sender transmits them through the quantum channel (typically a fibre-optic cable or free space for photon transmission).
- **Quantum information** is carried by these qubits during transmission. This process is secure because **quantum states cannot be cloned or perfectly measured without disturbing them**, ensuring privacy.

3. Receiver Measures Qubits

- The receiver receives the qubits sent by the sender.
- The **receiver randomly chooses a basis** to measure each qubit. The basis selected could be the same as the sender's basis or different.
- The receiver **measures the qubit's state** based on their chosen basis:
  - If the receiver's basis matches the sender's encoding basis, the receiver will measure the correct bit value.
  - If the receiver's basis doesn't match the sender's, the measurement result will be random.

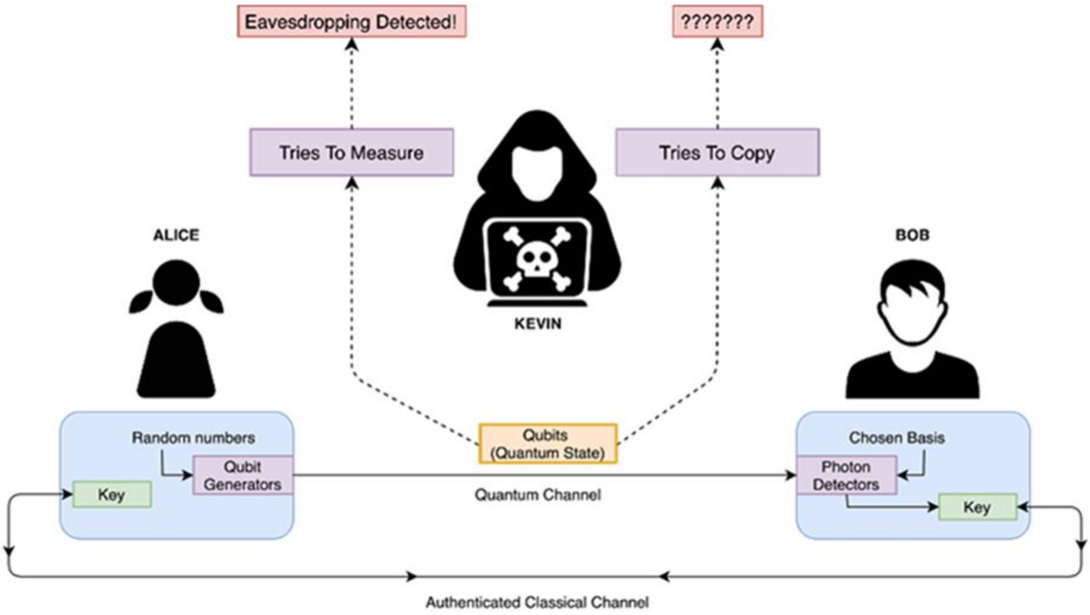


Figure 4:Quantum Key Distribution

2.4 Classical Channel

The **classical channel** enables the **sender** and **receiver** to coordinate and finalize a shared, secure key. This procedure is called post processing, where a classical post processing protocol is used to derive a symmetric key from the raw bits. The post processing message exchange happens through an authenticated classical channel. The post processing procedure in general include the following steps

2.4.1 Key Sifting

Key Sifting is the process through which the quantum transmitter (Alice) and receiver (Bob) eliminates all incompatible measurements by exchanging the information of basis used for measurement. Key sifting process is using classical channel and the output of the sifting process is the **Sifted data**. The sifted data is then passed on to other post processing steps like key reconciliation and privacy amplification to generate a symmetric shared key.

2.4.2 Key Reconciliation

In an ideal world with a perfect quantum channel and no eavesdropper, the sifted data which is the outcome of the key sifting process shall be identical at both sides. But in practical situations, errors are introduced due to various reasons like presence of eavesdropper, device imperfections or interference from environmental conditions. The purpose of reconciliation is to remove these errors and it mainly consists of the following steps.

2.4.2.1 Parameter Estimation

The purpose of Parameter estimation is to measure the error rate of the sifted data. In BB84, it is normally measured by picking a small random subset of bits from the sifted data and it is then publicly compared to detect the error count. The parameter estimation results in the probability of error in the sifted data called Quantum Bit Error Rate (QBER). The threshold of error rate, for which the estimated error rate should not exceed this threshold, is decided depending on the QKD protocols and possible error recovery codes in use. The error correction code can be chosen based on the maximum tolerable error rate of the QKD protocol. If the error rate is within the allowed limit, the parties can continue with the error correction or discard the entire process otherwise.

2.4.2.2 Error Correction

The error correction protocols are used to locate and correct the errors in the sifted data. The error correction in QKD has to be done without disclosing enough information that gives Eve chances to reconstruct the same data. There are multiple error correction protocols which are used in QKD and the examples include Cascade, Winnow, and Low Density Parity Check (LDPC). The output of the error correction step is a bit stream which is similar at both Bob and Alice ends.

2.4.3 Privacy Amplification

Privacy amplification is the last step of the classical post processing which results in a shared secret key between Alice and Bob. Privacy amplification is the art of distilling highly secret shared information, perhaps for use as a cryptographic key, from a larger body of shared information that is only partially secret. The purpose of privacy amplification is to remove any leftover information which might have exposed to the adversary during the post processing step. Generally, universal class of hash function is used for privacy amplification, which reconcile the key of length n to a finite key of length k and it is chosen randomly from a set of families of universal hash functions.

The final shared key which is generated from the classical post processing step is delivered to a Key Manager (KM) module. The Key Manager modules shall be implemented either in software or hardware which is responsible for the overall key management functions. The applications are interacting with the Key Manager for key requirements. The Key Manager is also responsible for synchronizing the keys across its peer KMs for its delivery to applications.

Table 2: Illustration of QKD using polarization

	Quantum Channel															
1	Alice Random Bits	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1
2	Random Sending Bases of Alice	x	+	x	+	+	+	+	+	x	x	+	x	x	x	+
3	Encoded Photons Alice Sends	⤡	↕	⤢	↔	↕	↕	↔	↔	⤢	⤡	↕	⤢	⤡	⤡	↕
4	Random Receiving Bases of Bob	+	x	x	+	+	x	x	+	x	+	x	x	x	x	+
5	Photons as received by Bob	↕		⤢		↕	⤡	⤢	↔		↕	⤢	⤢		⤡	↕
	Classical Channel															
6	Bob reports bases of received bits	+		x		+	x	x	+		+	x	x		x	+
7	Alice says which bases were correct			OK		OK			OK				OK		OK	OK
8	Presumably shared information			1		1			0				1		0	1
9	Bob reveals some key bits at random					1									0	
10	Alice conforms them					OK									OK	
	Outcome															
11	Remaining shared secret bits			1					0				1			1

2.5 Key Manager in QKD

In a Quantum Key Distribution (QKD) network, the Key Manager plays a pivotal role as it oversees the management, distribution, and security of keys generated by QKD devices like the sender and receiver. The Key Manager oversees several crucial tasks within the QKD system, including securely storing and tracking the keys generated through both the quantum and classical channels. By collaborating with various QKD devices, the Key Manager enables the QKD network to facilitate secure key exchange for encryption purposes between diverse nodes.

It guarantees the availability of keys for cryptographic operations, enforces access controls to prevent unauthorized key usage, and monitors the status of keys - whether they are active, expired, or revoked. Moreover, the Key Manager in QKD is responsible for overseeing multiple essential tasks to ensure the secure and effective distribution of keys.

Furthermore, the Key Manager collaborates with other network components to distribute keys among multiple users or devices within a QKD network. This feature allows seamless integration with traditional cryptographic systems. QKD-generated keys can be utilized for encryption and decryption in real-time communication. Overall, the Key Manager plays a vital role in converting the raw key material produced in QKD into a practical and secure key for cryptographic applications.

2.6 Types Of OKD Network

2.6.1 Trusted Relay QKD Network

A Trusted Relay in a Quantum Key Distribution (QKD) network serves as an intermediary node that facilitates the secure transmission of quantum keys over long distances. Since quantum communication is prone to losses and noise, especially over large distances, a direct quantum communication link between two distant parties can be inefficient or impossible. The Trusted Relay helps overcome these limitations by receiving quantum bits (qubits) from one node, forwarding them to another node, and ensuring the integrity of the transmitted information. It acts as a bridge in the QKD network, enabling the extension of the secure communication range without the need for direct long-distance quantum communication.

The working of a Trusted Relay in QKD involves several crucial steps. First, the relay receives qubits from the sender and stores them temporarily. It then forwards the qubits to the receiver after ensuring that no information is altered during transmission. The key point is that, while the relay does not measure or interfere with the qubits (to preserve quantum security), it must still be trusted to relay the information correctly. This trust is essential because if the relay were compromised, it could intercept and potentially alter the quantum information, jeopardizing the security of the key. Therefore, the relay’s role is limited to forwarding qubits without actively manipulating them, and its behaviour is often monitored to detect any unauthorized activities. The use of Trusted Relays is essential in expanding QKD networks to span greater distances and support secure communication between users located far apart.

2.6.2 Switched Path QKD Network

In a Quantum Key Distribution (QKD) network, a Switched Path is a crucial component that enhances the security and reliability of the key distribution process. A switched path refers to a dynamic routing system that allows the network to change physical optical path of the quantum based on current conditions. This flexibility is essential because it helps protect the system from external threats such as eavesdropping or disruptions caused by errors in the quantum channel. By dynamically adjusting the path, the QKD network ensures that qubits are transmitted securely and that the network remains robust even if part of the communication path is compromised or experiences high levels of noise.

The working of the switched path in a QKD network involves continuously monitoring the quantum and classical channels for any signs of interference or security breaches. If any disruption is detected in the quantum channel, such as an unexpectedly high error rate (which may indicate eavesdropping), the QKD network can quickly switch the path to a more secure route, ensuring that the quantum key exchange continues uninterrupted. This path-switching mechanism helps to maintain the security and reliability of the key distribution process, preventing potential attackers from gaining access to the shared key. Furthermore, it allows the QKD network to optimize performance by choosing the most stable and efficient communication paths based on real-time network conditions. Thus, the switched path mechanism in QKD plays a vital role in protecting the integrity of the key exchange process while ensuring the robustness of the overall network.

3. QKD Protocols

Quantum Key Distribution (QKD) protocols are the methods used to securely exchange cryptographic keys between two parties using the principles of quantum mechanics. These protocols rely on the properties of quantum states, such as superposition and entanglement, to ensure that any attempt at eavesdropping on the key exchange is detectable and are crucial for securing communications in the age of quantum computing, where traditional cryptographic methods may become vulnerable.

3.1 Classification of QKD Protocol

The classification of Quantum Key Distribution (QKD) protocols is based on several factors, such as the type of quantum states used, the underlying key exchange methods, the use of entanglement, and the security mechanisms incorporated

Category	Protocol Name	Description	Advantages	Limitations
Prepare-and-Measure Protocols	**BB84 Protocol**	Uses polarized photons in two bases. Alice prepares states; Bob measures randomly to establish a key.	High security, simplicity	Requires polarization control, synchronization
	B92 Protocol	Uses two non-orthogonal states. Bob detects presence/absence of photons to generate the key.	Simpler, fewer states	Potentially less secure, sensitive to noise
	Differential Phase Shift (DPS)	Encodes key bits in phase differences between successive pulses.	Resistant to photon-number-splitting attacks	Requires precise phase control, complex setup
Entanglement-Based Protocols	**E91 Protocol**	Uses entangled photons; correlations in Alice and Bob's measurements establish the key.	High security, entanglement detection of eavesdropping	Requires entangled photon sources
	BBM92 Protocol	Combines BB84 principles with entanglement for secure key exchange.	Robust security through entanglement correlations	Advanced entanglement setup required

Discrete-Variable (DV) QKD Protocols	**BB84 Protocol**	Same as above; utilizes discrete polarization states.	Foundational, secure	Synchronization needed
	B92 Protocol	Same as above; utilizes two-state non-orthogonal encoding.	Simplified setup	Higher sensitivity to noise
	Six-State Protocol	Extends BB84 with six polarization states, enhancing security.	Greater security than BB84	Lower efficiency
Continuous-Variable (CV) QKD Protocols	**Gaussian-Modulated Coherent States (GMCS)**	Encodes key bits in amplitude and phase quadrature of light waves.	Leverages telecom equipment	Sensitive to noise and losses
	Discrete-Modulated CV-QKD	Uses discrete modulation schemes within CV framework for robust error handling.	Improved error performance in noisy environments	Lower key generation rates
Device-Independent QKD (DI-QKD)	**DI-E91 Protocol**	Based on E91 but does not rely on trusted devices; security via Bell test correlations.	Secure even with untrusted devices	Technically complex
	DI-BBM92 Protocol	Device-independent version of BBM92; retains security with partially compromised devices.	High security with untrusted devices	Requires entangled sources and Bell tests
Measurement-Device-Independent (MDI) QKD	**MDI-BB84 Protocol**	Variant of BB84 where a third party performs measurements; sender and receiver don't need trusted detectors.	Secure against detector side-channel attacks	Additional complexity in setup
	Twin-Field QKD (TF-QKD)	Uses third-party measurements, extending range and key rate.	Enhanced distance and key rate	Advanced synchronization required

3.2 Architecture of QKD Protocol

The above discussed different architectures of QKD protocols are illustrated in Figure 7, Figure 8 and Figure 9 respectively. Generally, the scope of security evaluation is tightly related to the architecture of the implemented QKD protocol, which is clarified in the following.

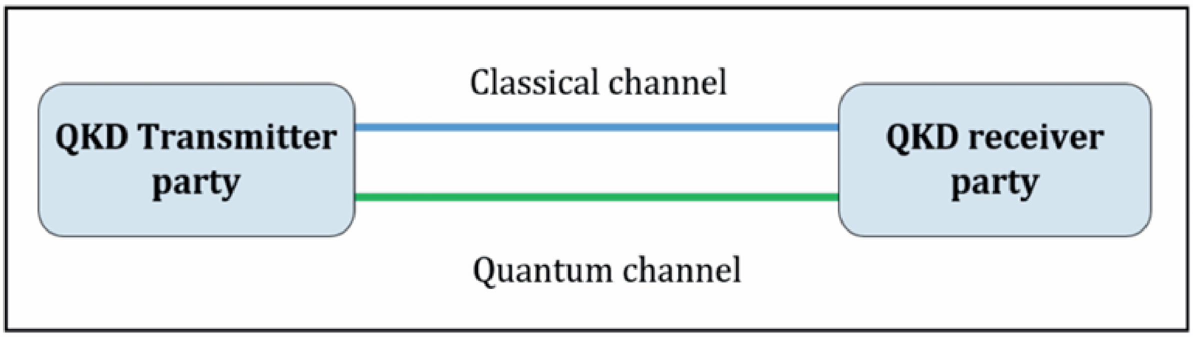


Figure 5:Prepare-and-measure QKD protocol

The architecture corresponding to the P&M-QKD protocol, illustrated in Figure 7, consists of a transmitter party and a receiver party connected by quantum and classical channels. The security of the practical QKD systems relies on the secure implementation of the functions of both parties, thus both of the implementation modules are required to be in the scope of security evaluation.

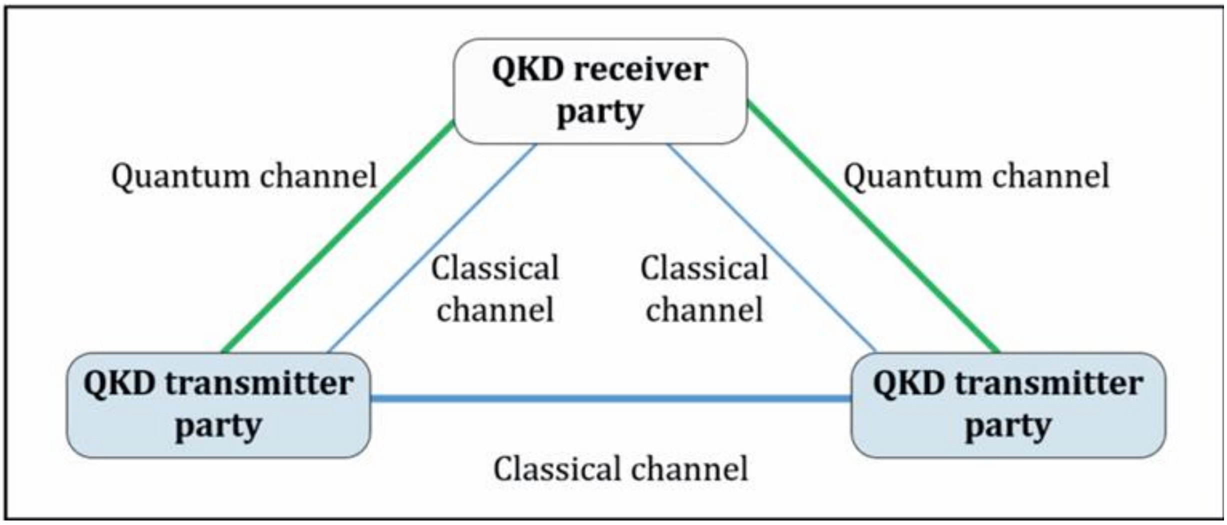


Figure 6:MDI-QKD protocol

In the MDI-QKD protocol, a middle party assumes the role of receiver party who connects with the two transmitter parties through a quantum channel and a classical channel, and the two transmitter parties are also connected via a classical channel, see Figure 8. After a successful QKD session, only the two transmitter parties know the final key.

NOTE : There are two types of classical channels in MDI-QKD. One is the classical channel connecting the two QKD transmitter parties, where data over the channel are required to be authenticated depending on the specific QKD protocols. The other type includes the two classical channels connecting each QKD transmitter party with the QKD receiver party. The latter type is used to transmit the measurement results of the QKD receiver party, but no message authentication is needed. In other words, it is assumed that the attacker can tamper with the measurement results sent from the QKD receiver party (via these classical channels) to the transmitter parties

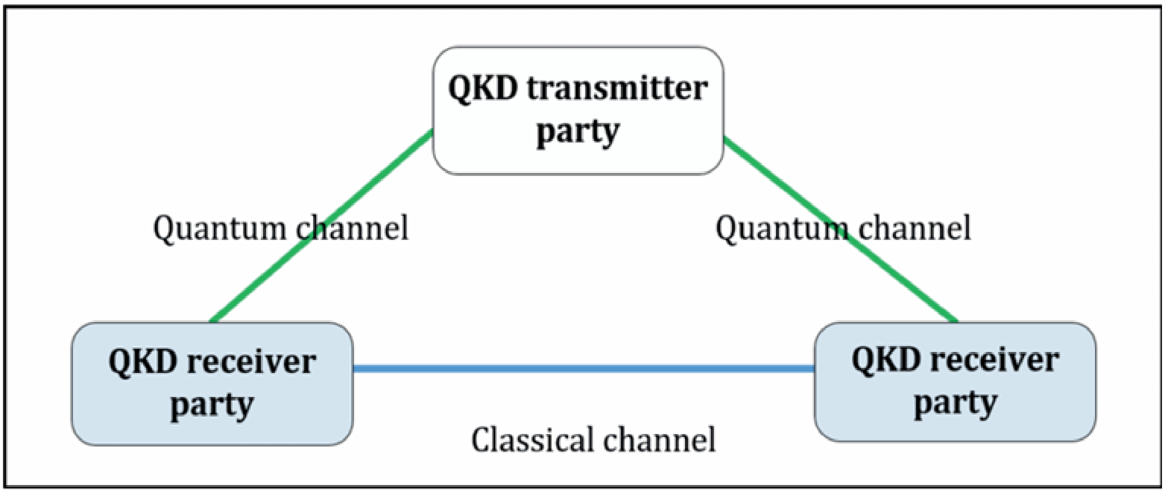


Figure 7: EB-QKD protocol

In the EB-QKD protocol, a middle party assumes the role of QKD transmitter party, who connects with each of the two receiver parties via a quantum channel, and the two receiver parties are again connected via a classical channel, see Figure 9. After a successful QKD session, only the two QKD receiver parties know the final key.

4. Software Defined QKD Network

A Software-Defined QKD Network (SD-QKD) integrates Quantum Key Distribution (QKD) with Software-Defined Networking (SDN) to provide secure communication. In this setup, SDN controllers manage and optimize the distribution of quantum keys across the network, using programmable software to dynamically adjust network paths and resources. QKD ensures secure key exchange through the principles of quantum mechanics, while SDN allows flexibility and efficiency in routing and resource management. This combination enhances both the security and adaptability of the network, ensuring reliable, scalable, and tamper-proof communication.

4.1 Software Defined Networking (SDN)

Software-Defined Networking (SDN) is a progressive network architecture that enables centralized control and management of network resources through software. By decoupling the **control plane**, which dictates how data should be processed, from the **data plane**, which routes the data, SDN offers increased flexibility, programmability, and scalability in network management.

Key Components:

- **SDN Controller**: The central component responsible for managing and controlling the network. It acts as the "brain" of the network, taking charge of decisions regarding data traffic routing, network policies, and optimization.
- **Data Plane:** Consisting of network devices such as switches and routers, it is responsible for the direct forwarding of data packets according to the commands issued by the SDN controller.
- **Control Plane:** The layer that handles the decision-making on data flow within the network, typically carried out through software implementation. In Software-Defined Networking, the control plane is distinct from the data plane, providing greater flexibility in network traffic management.
- **Southbound API (eg. OpenFlow)**: A communication protocol between the SDN controller and the network devices (data plane). OpenFlow is a widespread protocol that enables the controller to determine how traffic should be managed by the devices.
- **Northbound API**: An interface facilitating communication between the SDN controller and higher-level network applications, like network management, monitoring, or orchestration software.

Benefits:

- **Centralized Management**: Network operations can be controlled from a single point, facilitating easier configuration, management, and monitoring of the network.
- **Flexibility and Programmability**: SDN empowers dynamic network reconfiguration through software, providing the capability to automate network tasks and respond to real-time demands.
- **Cost-Efficiency**: Employing open standards and affordable hardware, SDN lessens reliance on proprietary solutions, leading to decreased operational expenses.

- **Enhanced Security**: Centralized control enables more efficient enforcement of security measures and faster identification of network irregularities or breaches. Scalability: SDN can effortlessly scale by incorporating new devices or adjusting configurations without requiring substantial hardware upgrades.

4.2 Architecture of SD-QKD Network

The architecture of a **Software-Defined Quantum Key Distribution (SD-QKD) Network** combines the principles of **Software-Defined Networking (SDN)** with **Quantum Key Distribution (QKD)** to create a flexible, secure, and programmable quantum communication infrastructure. The architecture integrates quantum communication elements (like quantum key exchange) with the network management capabilities of SDN.

An SDN-based QKD Network architecture is initially proposed by W. Yu et al.[12] in 2017. The architecture proposed by subsequent studies[10][11] on SDN-based QKD Networks consists of layers namely Application, Control, and Infrastructure. The infrastructure layer is further split into the QKD layer and the Data layer to segregate the QKD and physical optical functions. The control layer directly interacts with both the QKD layer and data layer in parallel in this architecture to fulfil the overall network functions. The QKD layer consists of QKD Nodes and the Data layer consists of optical network components. The QKD Nodes and optical network components are SDN aware and can interface with the SDN controller through the South Bound interfaces.

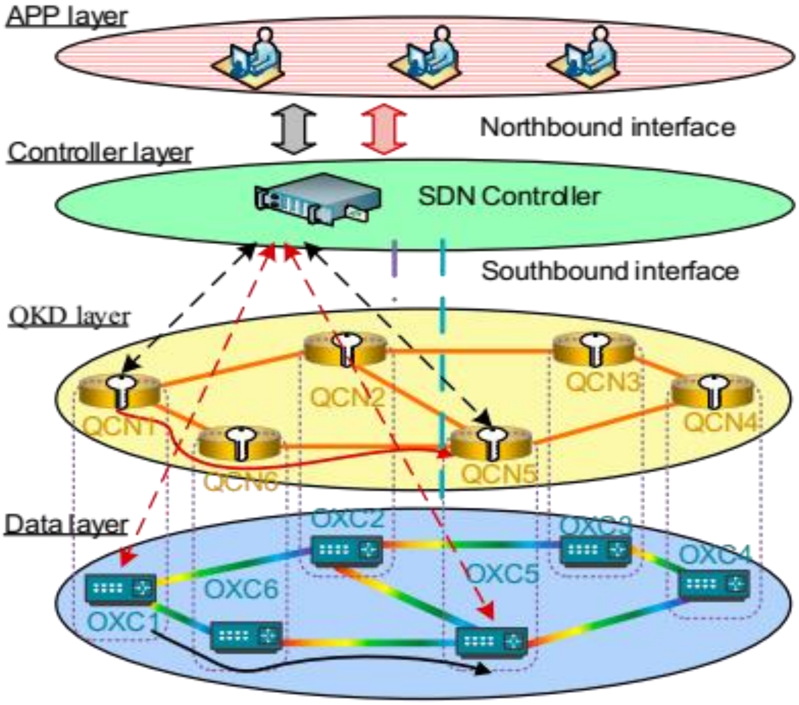


Figure 8: Four Layer Architecture of SD-QKD Network proposed by Y.Zhao et al.

The ITU-T has developed a functional architecture of the QKD Network[13]. This architecture followed a cross-layer design and consists of Quantum, QKD Network Management, Key Management, QKD Network Control, Service, and User Network Management layers. The important characteristics of this architecture are

- Separation of Network Control and Management functions

- Separation of Key Management into a separate layer and it is interacting directly with the service layer for providing keys and QKD module for key management. The service layer consumes the key pair provided by the Key Management Layer
- Introduction of a User Network Management Layer which interacts with both the management layer and service layer.

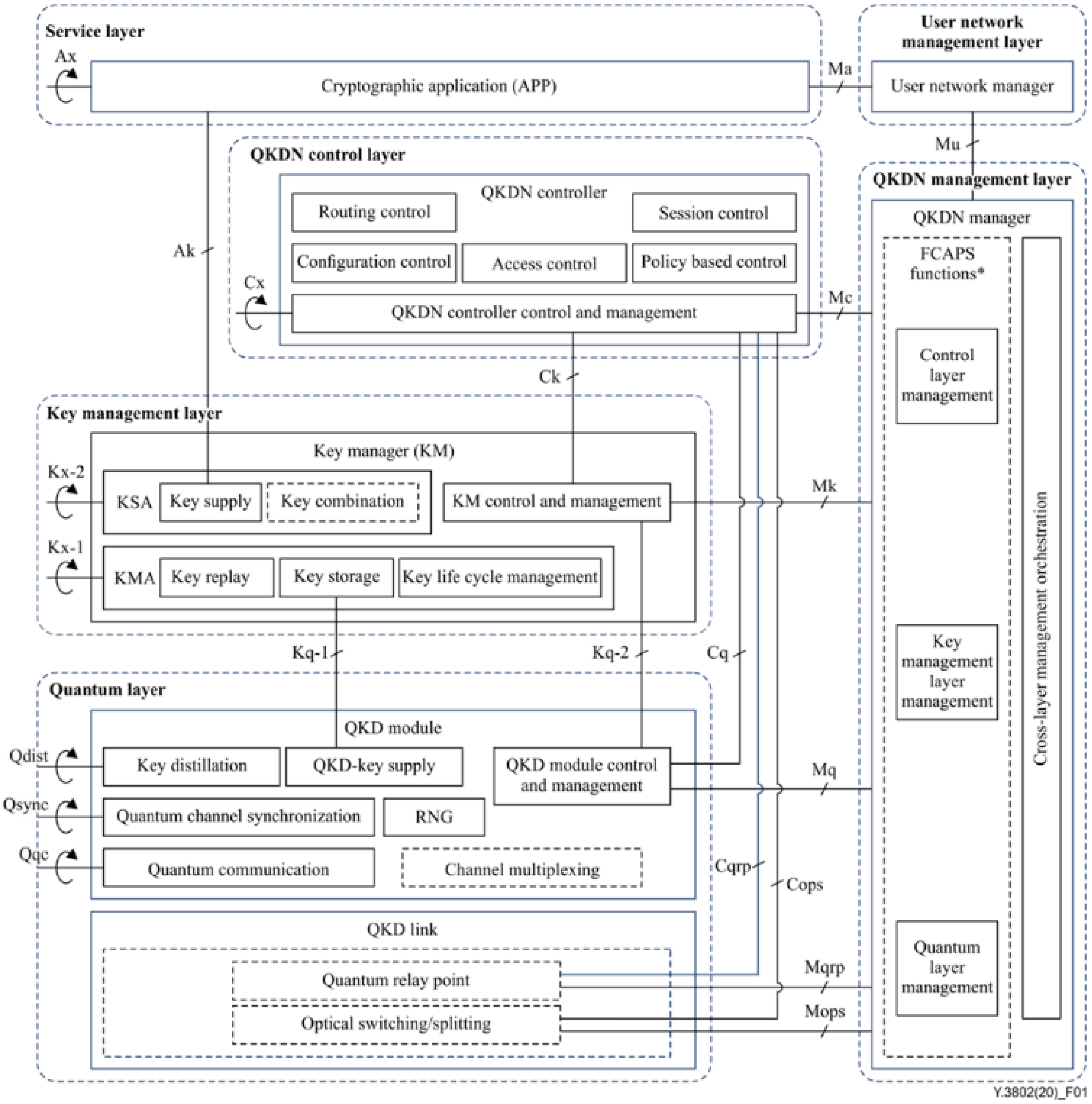


Figure 9: Functional Architecture of QKD Network by ITU-T

4.2.1 Architectural components of an SD-QKD network

- **SDN Controller**

  The **SDN controller** serves as the central management unit in the SD-QKD network, overseeing both the classical and quantum components. It is responsible for managing the distribution of classical data and quantum keys across the network, ensuring efficient and secure communication. The controller dynamically adjusts routing paths based on current network conditions and the performance of the quantum communication channels, optimizing the overall network efficiency. Additionally, the SDN controller continuously monitors the status of the quantum channels, ensuring their integrity by detecting any tampering or eavesdropping attempts, thereby maintaining the security of the network.

- **Quantum Devices** (QKD Devices)
  Quantum Devices are the physical components responsible for implementing Quantum Key Distribution (QKD) between different nodes in the SD-QKD network. These devices include **QKD transmitters**, which prepare and send quantum states (such as photons) to other nodes for key exchange. **QKD receivers** receive these quantum states, measure them, and perform error correction to generate the cryptographic key. In cases where the network spans large distances, **quantum repeaters** are used to extend the range of quantum communication by amplifying or relaying quantum signals, ensuring that the key exchange can occur over longer distances without loss of security or signal integrity.

- **Classical Network Infrastructure**
  The classical network in an SD-QKD setup is responsible for transmitting non-quantum data and integrating quantum key distribution with traditional network traffic. This network ensures seamless communication between quantum devices by handling the forwarding of classical data using classical routers and switches. These devices route non-quantum traffic between nodes while also managing the communication required for quantum key exchange. Once a quantum key is established, classical encryption algorithms like AES are used to secure the data. This encryption relies on the quantum-generated key, ensuring that the network traffic remains protected using the robust encryption derived from the quantum exchange.

- **Quantum Channel**
  The **quantum channel** plays a crucial role in transmitting quantum information, typically in the form of photons, between QKD devices. This channel can utilize different mediums depending on the distance and conditions of the network. **Optical fibres** are commonly used for quantum communication over short to medium distances, providing a stable and controlled environment for photon transmission. For longer distances or when optical fibres are not available, **free-space communication** is employed, though it is more challenging due to potential environmental interference, such as atmospheric conditions that can affect the quantum signals.

- **Key Management**
  The Key Management component in an SD-QKD network is responsible for managing the quantum keys generated by the QKD devices. It ensures that these secure keys are properly distributed across the network and are used for encrypting classical data within the network. The SDN controller oversees the key distribution, ensuring that quantum-generated keys are securely shared between the necessary nodes. Additionally, the system periodically performs key refreshment to maintain long-term security by updating keys at regular intervals. In some cases, post-quantum cryptography techniques are applied using the quantum keys, providing extra protection against potential future threats from quantum computers.

- **Security and Monitoring Layer**
  The **Security and Monitoring Layer** in an SD-QKD network plays a critical role in ensuring the integrity of the quantum key exchange and maintaining overall network security. It continuously monitors the quantum communication channels for any signs of interference or eavesdropping, with a built-in **eavesdropping detection** mechanism. If any tampering or unauthorized access is detected, the system is immediately alerted to take corrective action, preventing any potential

breach. Additionally, the **audit logs** component tracks and logs all key exchanges and activities within the network, providing a detailed record for auditing and enhancing security. This ensures transparency and allows for quick identification and resolution of any security incidents.

5. Service Perspective of SD-QKD Network

Software-Defined Quantum Key Distribution (SD-QKD) networks transform the delivery of quantum-secure communication solutions via their software-oriented framework. These networks facilitate adaptable, scalable, and compatible administration of quantum key distribution assets, guaranteeing effective key transmission and protected communication. The main service viewpoints of SD-QKD networks encompass Key Transport as a Service, QKD Network as a Service, and Entropy as a Service, with each focusing on distinct elements of secure quantum communication.

5.1 Key Transport as a Service

SD-QKD networks facilitate the efficient and reliable transportation of quantum keys across the network, ensuring secure communication for end-users. The key transport service can be envisioned as an overlay of the traditional key management process, enhanced with quantum-level security assurances.

- **Secure Distribution Mechanisms**: The SD-QKD network provides cryptographic keys using quantum protocols like BB84 or E91, ensuring they remain secure against eavesdropping attempts due to the principles of quantum mechanics.
- **Dynamic Path Selection**: With SDN control, quantum keys can be transported over dynamically selected paths that optimize resource usage and minimize latency or loss.
- **Multi-Tenant Support**: The service supports multiple users or organizations, allocating keys to each based on their specific requirements, ensuring isolation and security among tenants.
- **Integration with Classical Key Management**: The service integrates seamlessly with classical encryption systems, providing hybrid encryption solutions such as QKD-enhanced IPsec or TLS protocols.

5.2 Entropy as a Service

Entropy as a Service leverages the high-quality randomness generated by quantum processes to offer secure random number generation for cryptographic and other applications.

- **High-Quality Quantum Entropy Sources**: The service utilizes quantum processes to generate random numbers that provide a critical input for cryptographic systems needing robust entropy.
- **Scalable Distribution**: The Software-Defined Quantum Key Distribution architecture enables the scalable and secure delivery of entropy to multiple endpoints, ensuring consistent quality and availability across different systems.
- **Support for Diverse Use Cases**: Beyond cryptography, Entropy as a Service can be applied to simulations, secure authentication, and other applications that rely on high-quality randomness.
- **Subscription-Based Model**: The service can be offered on a pay-per-use or subscription basis, enabling organizations to access high-quality entropy without the need to maintain expensive quantum hardware.
- **Regulatory Compliance**: The service is designed to meet stringent security standards, ensuring that the generated randomness complies with cryptographic and regulatory requirements.

5.3 QKD Network as a Service

Quantum Key Distribution Network as a Service offers the capabilities of a quantum-secure network as an on-demand service, abstracting the complexity of its management. This approach enables users to benefit from quantum-secured communications without needing deep expertise or significant resources to operate the underlying infrastructure.

- **On-Demand QKD Resources**: Users can request QKD services when needed, scaling their usage according to their requirements without over-provisioning resources.
- **Centralized Control and Management**: Through the SDN controller, QKD nodes, channels, and devices can be monitored and managed centrally, ensuring network efficiency and reducing operational complexity.
- **Interoperability Across Multi-Vendor Environments**: By abstracting vendor-specific differences, It provides a unified interface for deploying and managing quantum networks, fostering collaboration among vendors.
- **Flexible Deployment Models**: It supports various deployment models, including private, public, or hybrid quantum networks, depending on the user's security and budget needs.
- **Fault Tolerance and Recovery**: SD-QKD networks ensure high availability by rerouting traffic or reconfiguring network paths in the event of failures, maintaining seamless service delivery.

The service perspectives of SD-QKD networks—Key Transport as a Service , QKD Network as a Service , and Entropy as a Service —represent significant advancements in how quantum technologies can be applied in real-world communication networks. By providing scalable, secure, and flexible services, SD-QKD networks make it possible for organizations to integrate quantum key distribution, entropy generation, and secure communication protocols without having to deploy complex and costly infrastructure. These services foster greater interoperability, cost efficiency, and security, enabling businesses to adopt quantum technologies as part of their cybersecurity strategies. As quantum technologies mature, the transition to SD-QKD-driven services will become increasingly vital, particularly as quantum-safe security becomes essential for future communication networks.

6. Standardisation in QKD Network

Standardisation is fundamental to achieving interoperability and ensuring the widespread adoption of Quantum Key Distribution (QKD) technology. Various global organisations are actively contributing to the development of protocols and frameworks to make QKD systems compatible across different vendors. ETSI plays a prominent role by defining standards for secure and efficient integration of QKD into existing telecommunications networks. ETSI's collaboration with ISO strengthens international efforts to harmonise these standards, while ITU-T focuses on creating global frameworks that address the technical and operational aspects of QKD implementation. IEEE contributes by defining essential protocols that support the scalability and management of QKD networks, particularly when combined with emerging technologies like Software-Defined Networking (SDN). Together, these bodies shape the landscape of standardisation, ensuring that QKD technology is secure, interoperable, and ready for deployment in diverse communication environments.

6.1 ETSI

The **European Telecommunications Standards Institute (ETSI)** [1] has been at the forefront of standardizing quantum technologies, including QKD. It has established the ETSI Industry Specification Group on QKD (ISG-QKD) to focus on defining architecture, security requirements, and implementation guidelines. By offering comprehensive technical standards, ETSI promotes compatibility between different vendors' hardware and software solutions.

In the context of multi-vendor QKD hardware using SDN, ETSI standards can streamline communication between QKD devices and the SDN controller. Adhering to these guidelines can ensure that hardware from multiple vendors can operate cohesively within an SDN framework, enabling effective key distribution and network management.

Notable ETSI Standards

Standard	Title	Content
ETSI GR QKD 003 V2.1.1	Quantum Key Distribution (QKD); Components and Internal Interfaces	Explores various use cases of QKD, including telecom networks, financial applications, and critical infrastructure.
ETSI GR QKD 007 V1.1.1	Quantum Key Distribution (QKD); Vocabulary	Provides methodologies for evaluating QKD system performance, including throughput, error rates, and security metrics.
ETSI GS QKD 004 V2.1.1	Quantum Key Distribution (QKD); Application Interface	Defines the security proofs for QKD protocols, focusing on robustness against attacks and quantum-safe cryptography.
ETSI GS QKD 008 V1.1.1	Quantum Key Distribution (QKD); QKD Module Security Specification	Analyses the applicability of QKD in different network scenarios and outlines integration strategies.
ETSI GS QKD 011 V1.1.1	Quantum Key Distribution (QKD);	Details testing and characterization methods for optical components like lasers, detectors, and fibers used in QKD.

	Component characterization: characterizing optical components for QKD systems	
ETSI GS QKD 012 V1.1.1	Quantum Key Distribution (QKD); Device and Communication Channel Parameters for QKD Deployment	Outlines procedures for verifying the performance and reliability of QKD system components.
ETSI GS QKD 014 V1.1.1	Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API	Defines key management architectures and protocols for QKD systems, ensuring secure storage and usage of quantum keys.
ETSI GS QKD 015 V2.1.1	Quantum Key Distribution (QKD); Control Interface for Software Defined Networks	Expands on the previous version with enhanced SDN features for multi-domain QKD networks and inter-networking.
ETSI GS QKD 018 V1.1.1	Quantum Key Distribution (QKD); Orchestration Interface for Software Defined Networks	Discusses methods for integrating QKD with secure communication platforms and existing cryptographic infrastructures.

6.2 ETSI/ISO

ETSI collaborates with the International Organization for Standardization (ISO) [1] [2] to enhance the global adoption of QKD standards. This partnership bridges regional standardization efforts with broader international frameworks, ensuring that QKD systems are universally applicable. By harmonizing standards, ETSI/ISO ensures that equipment and protocols are not restricted by geographical boundaries or vendor-specific constraints.

For a multi-vendor QKD network using SDN, ETSI/ISO standards help create a unified protocol stack, reducing the complexity of integration. This collaboration is particularly valuable for global deployments where diverse hardware and varied regulatory environments coexist. It facilitates a common language for SDN-enabled QKD networks, improving efficiency and reducing operational costs.

Notable ETSI/ISO Standards

Standard	Title	Content
ISO/IEC 23837-1:2023	Information technology security techniques — Security requirements, test and evaluation methods for quantum key distribution	Aims at securing systems using quantum cryptography, including QKD.

ISO/IEC 23264-1:2022	Quantum Key Distribution (QKD) Networks — Framework, Architecture, and Interfaces	Provides the architecture and interfaces for QKD networks.
ISO/IEC 27010:2020	Information Security Management for Inter-sector and Inter-organizational Communications	Focuses on secure communication between sectors, relevant for integrating QKD into cross-domain systems.

6.3 ITU-T

The **International Telecommunication Union's Telecommunication Standardization Sector (ITU-T)** [3] focuses on developing globally recognized standards for telecommunications, including QKD technologies. ITU-T recommendations address aspects like network architectures, protocol interoperability, and performance metrics, ensuring that QKD systems can seamlessly integrate into existing telecommunication infrastructures.

In SDN-driven multi-vendor QKD networks, ITU-T standards can guide the integration of QKD systems with legacy telecom networks, ensuring interoperability between classical and quantum components. These recommendations can help SDN controllers manage quantum and classical data flows, facilitating smooth key exchanges in hybrid network environments.

Notable ITU-T Standards

Standard	Title	Content
Y.3800	Overview on networks supporting quantum key distribution	Defines the fundamental principles, benefits, and challenges of QKD.
Y.3801	Functional requirements for quantum key distribution networks	Describes the architecture needed to enable operational QKD networks.
Y.3802	Quantum key distribution networks – Functional architecture	Focuses on control, monitoring, and management aspects for QKD networks.
Y.3803	Quantum key distribution networks – Key management	Discusses methods for key generation, distribution, storage, and lifecycle management in QKD systems.
Y.3804	Quantum key distribution networks – Control and management	Specifies performance indicators for evaluating QKD systems.
Y.3805	Software Defined Networking Control for Quantum Key Distribution Networks	Recommends the integration of SDN principles with QKD networks.

6.4 IEEE

The **Institute of Electrical and Electronics Engineers (IEEE)** [4] contributes to QKD standardization by focusing on the physical and networking layers of quantum communication systems. IEEE standards address critical aspects like quantum channel specifications, synchronization, and error correction, providing foundational guidelines for implementing QKD systems.

For multi-vendor QKD hardware integrated through SDN, IEEE standards ensure that the underlying quantum communication channels are robust and compatible. This harmonization allows the SDN controller to manage resources effectively, optimize network performance, and troubleshoot issues in a multi-vendor environment. IEEE's focus on innovation also fosters the development of advanced QKD techniques compatible with SDN frameworks.

Notable IEEE Standards

Standard	Title	Content
P1913	Software-Defined Quantum Communication	Defines architectures, protocols, and interfaces for quantum communication networks integrated with Software-Defined Networking (SDN).
P7130	Standard for Quantum Computing Definitions	Establishes a common vocabulary and definitions for quantum computing concepts, including hardware, algorithms, and systems.
P7131	Standard for Quantum Computing Performance Metrics & Performance Benchmarking	Defines performance benchmarks and metrics for quantum computing systems, including latency, qubit coherence, and error rates.

6.5 Key Differences Between ETSI, ETSI/ISO, ITU-T, and IEEE Standards

Aspect	ETSI	ETSI/ISO	ITU-T	IEEE
Scope and Focus	Focuses on QKD-specific standards for Europe and beyond.	Combines ETSI's technical expertise with ISO's global acceptance.	Concentrates on telecommunication-specific QKD integration.	Develops broader hardware and networking protocol standards.
Global Reach	Primarily regional (Europe) but with increasing global influence.	International collaboration for universal adoption.	Explicitly global, targeting telecom networks.	Widely recognized for global standards in networking and electronics.
Application Area	QKD systems, components, and practical deployment.	Harmonized standards for global applicability.	Integration of QKD within telecom infrastructures.	Networking protocols and cryptographic foundations that complement QKD

Each standardization body—ETSI, ISO, ITU-T, and IEEE—plays a crucial role in the development of QKD technologies, with different areas of focus ranging from regional to global standards, technical specifications, and integration with telecommunications infrastructure. Together, these standards contribute to the interoperability of multi-vendor QKD hardware in SDN networks, ensuring that secure communications can be achieved regardless of the underlying vendor equipment.

7. Interoperability of QKD Networks

Achieving interoperability in Quantum Key Distribution (QKD) networks is a critical step toward their widespread adoption and integration into existing communication infrastructures. QKD networks are inherently diverse, often composed of multi-vendor devices, each with unique protocols, architectures, and capabilities. This diversity, while fostering innovation, poses significant challenges for seamless communication and standardization.

By enabling devices and systems from different manufacturers to operate together, interoperability enhances the scalability, reliability, and efficiency of QKD networks. This effort not only ensures the secure exchange of quantum keys across heterogeneous platforms but also aligns with the broader goals of global quantum communication networks. Fostering such compatibility requires innovative solutions, including standardized protocols, software-defined networking (SDN), and advanced relay technologies that bridge the gap between varying systems while maintaining the fundamental principles of quantum security.

7.1 Network Architecture for Interoperability

Two network architecture scenarios are considered for the study of interoperability. The first approach uses a centralized controller where all QKD nodes belonging to different networks are logically attached to a single controller. A second approach in which each independent QKD network is controlled by a QKD Network controller and an orchestrator solution acts a controller of controllers. Each scenarios is defined below. In both scenarios, the classical connectivity for interoperability include both trusted and non-trusted link.

7.1.1 Interoperability of QKD Networks Using Centralised SDN Controller

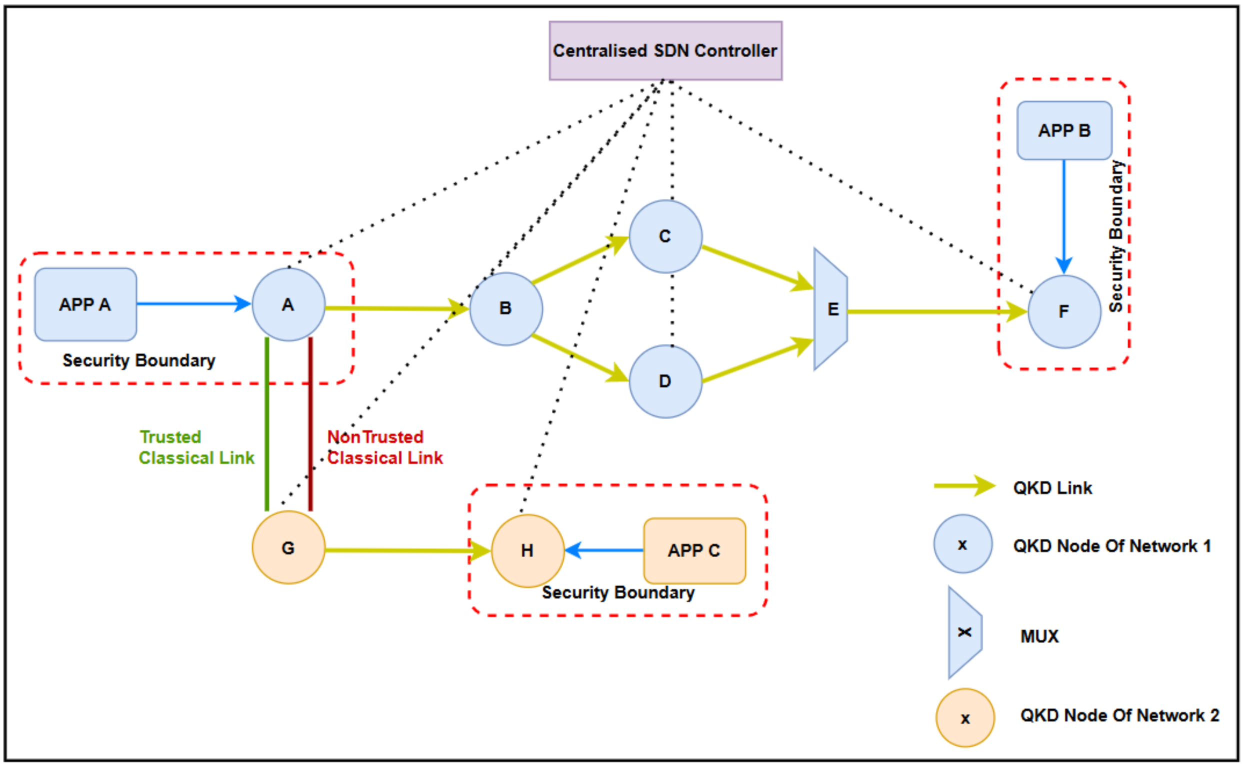


Figure 10 : Architecture of Interoperability of Two QKD Networks with Centralised SDN Controller

The scenario of centralized SDN-Controller where Orchestrator and

- **Network 1** is a Software Defined QKD Network connecting four locations. It allows key exchange between any nodes connecting these locations. Another QKD **Network 2** which need to interoperate with Network 1 for the Key Management and Key Delivery.
- The objective of this integration is to achieve the key exchange between any two nodes of Network 1 & Network 2.
- The interconnection of Network 1 and Network 2 can be established either through a **trusted link** or **non-trusted link**.
- The **Centralised SDN Controller** would Control both Network 1 and Network 2 nodes for Key exchange.

7.1.2 Interoperability of Two QKD Networks Using QSDN Controller and Orchestrator

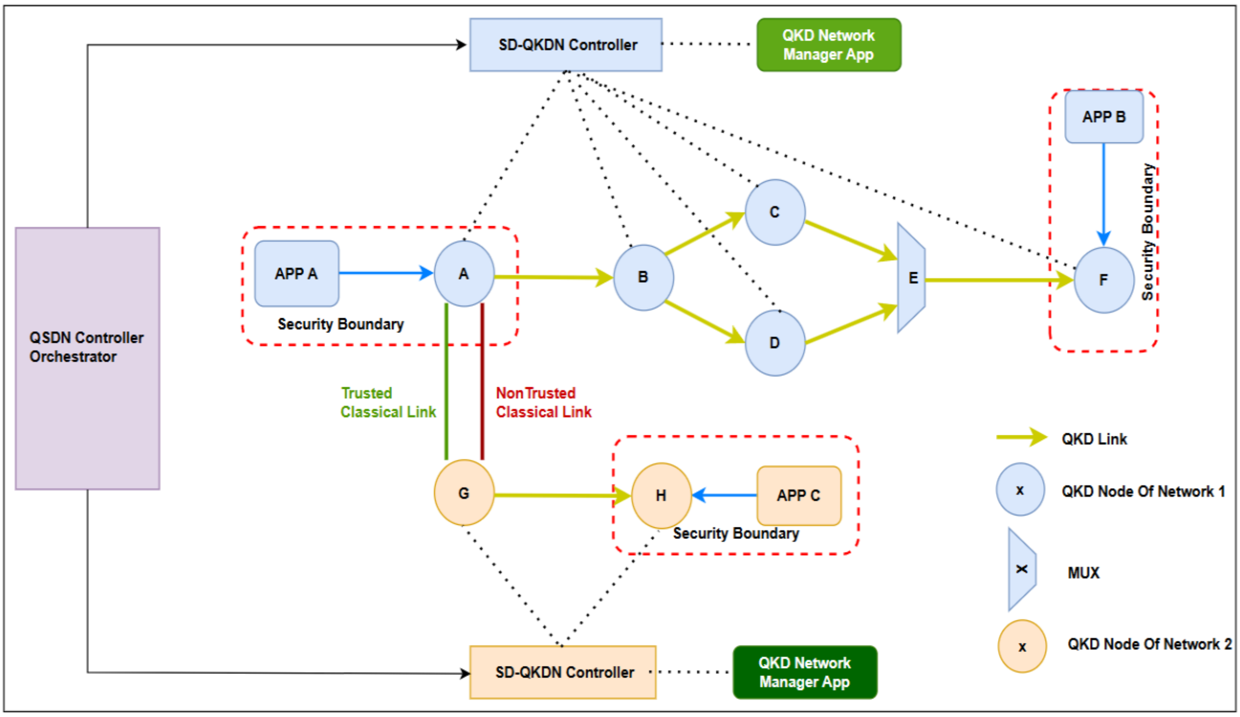


Figure 11 : Architecture of Interoperability of Two QKD Networks with QSDN Controller Orchestrator

- **Network 1** is a Software Defined QKD Network connecting four locations. It allows key exchange between any nodes connecting these locations. Another QKD **Network 2** which need to interoperate with Network 1 for the Key Management and Key Delivery.
- The objective of this integration is to achieve the key exchange between any two nodes of Network 1 & Network 2.
- The interconnection of Network 1 and Network 2 can be established either through a **trusted link or non-trusted link**.
- Each Network Posses Sperate **SD-QKDN Controller** and each Controller would be connected to the **QSDN Controller Orchestrator**.

- This QSDN Controller Orchestrator and SD-QKDN of each Network would communicate with each other for Key exchange between both Networks.

7.1.3 Pros and Cons of Centralised SDN Controller and QSDN Controller Orchestrator

In multi-vendor Quantum Key Distribution (QKD) networks, effective coordination and management of diverse technologies are essential for ensuring seamless operation and interoperability. Software-Defined Networking (SDN) plays a crucial role in addressing these challenges, with two primary approaches: deploying a centralized SDN controller to manage the entire network or using an SDN controller orchestrator to coordinate independently operated local controllers. Each approach offers unique advantages and trade-offs, influencing scalability, fault tolerance, and operational complexity.

Centralized SDN Controller

Advantages	Disadvantages
▪ Unified control and management for the entire network. ▪ Global network visibility for efficient resource allocation. ▪ Simplifies interoperability through enforced standards. ▪ Ensures consistent policy implementation across the network. ▪ Centralizes troubleshooting and fault management.	▪ Scalability issues as the network size increases. ▪ Single point of failure risks disrupting the entire network. ▪ High complexity in managing diverse vendor protocols. ▪ Communication delays impacting real-time QKD operations. ▪ Potential resistance from vendors due to reduced autonomy.

SDN Controller-Orchestrator Combination

Advantages	Disadvantages
▪ Scalable by distributing control across local controllers. ▪ Localized failures have minimal impact on the overall network. ▪ Retains vendor autonomy, encouraging collaboration. ▪ Modular addition of new networks and vendors. ▪ Simplifies distributed management across multiple domains.	▪ High complexity in coordinating diverse controllers. ▪ Challenges in ensuring policy consistency across domains. ▪ Increased latency due to inter-controller communication. ▪ Limited visibility into individual network domains. ▪ Significant effort required for standardizing and integrating interfaces.

7.2 Workflow of Interoperability

7.2.1 Centralised SDN Controller

Network 1 and **Network 2** are two Software Defined QKD Network , which need to establish a connection to enable secure key exchange between any two nodes across the networks. Here the Key exchange is monitored by a **Centralised SDN Controller** , which will help for Link establishment between nodes of two different network. In this scenario the Key Exchange is between **QKD NODE A** of Network 1 to **QKD NODE H** of Network 2 which in monitored and regulated by the **Centralised SDN Controller.**

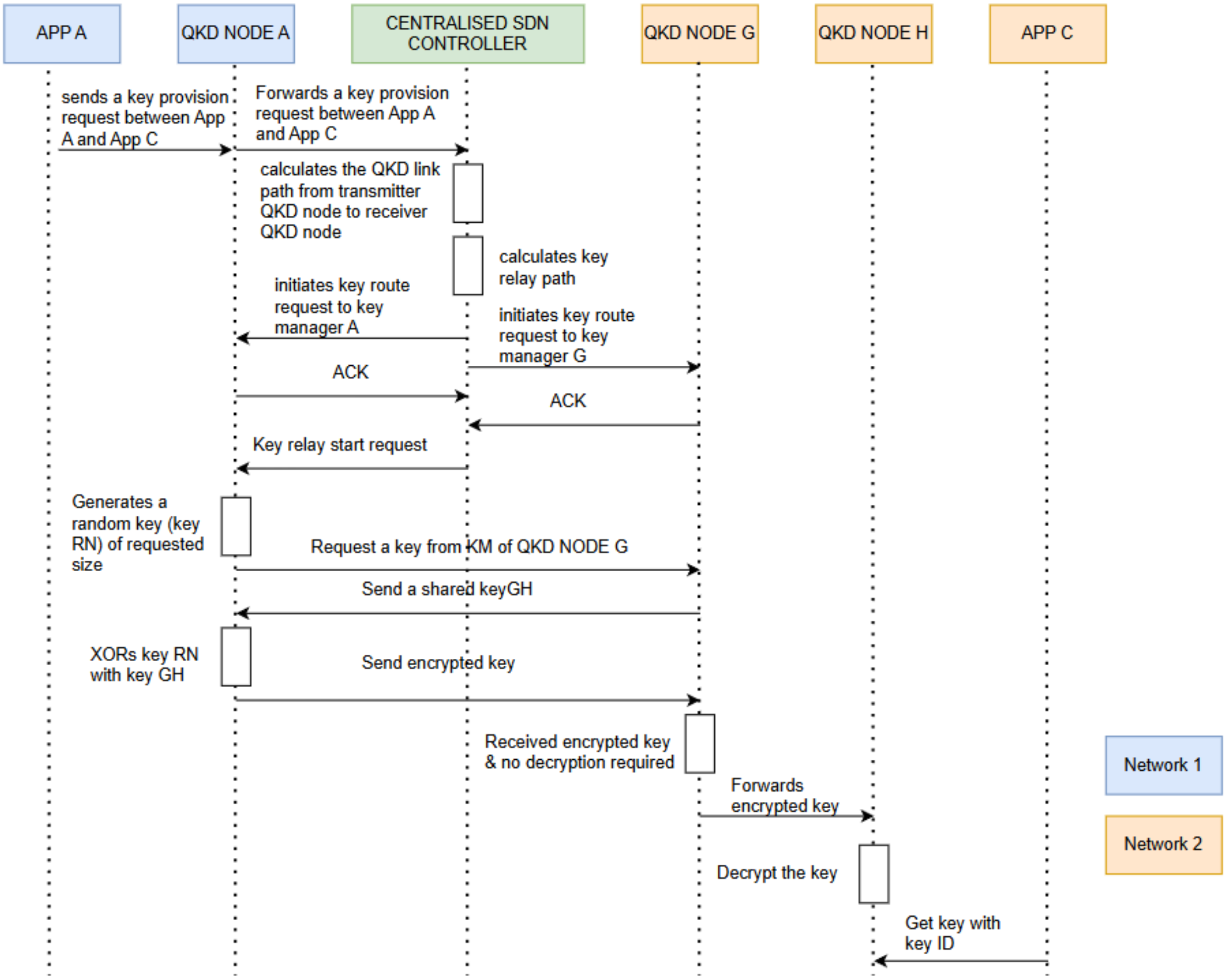

APP A
QKD NODE A
CENTRALISED SDN CONTROLLER
QKD NODE G
QKD NODE H
APP C
sends a key provision request between App A and App C
Forwards a key provision request between App A and App C
calculates the QKD link path from transmitter QKD node to receiver QKD node
calculates key relay path
initiates key route request to key manager A
initiates key route request to key manager G
ACK
ACK
Key relay start request
Generates a random key (key RN) of requested size
Request a key from KM of QKD NODE G
Send a shared keyGH
XORs key RN with key GH
Send encrypted key
Received encrypted key & no decryption required
Forwards encrypted key
Decrypt the key
Get key with key ID
Network 1
Network 2

Figure 12 : KEY RELAY USING CENTRALISED SDN CONTROLLER

PROCEDURE

- **Application A** of Network 1 sends a key provision request to **QKD NODE A** using the KeyProvision API.
- **QKD NODE A** forwards a key provision request to the centralized SDN controller.
- Controller calculates the QKD link path from transmitter QKD node to receiver QKD node. If there is no direct QKD link path, controller calculates key relay path between two Networks using the Keyrelayroute API**.**

- Controller initiates key route request to key managers of **QKD NODE A** and to **QKD NODE G** using the Keyrelaystart API.
- The key route request will be having the information like path to which key managers have to transfer the relayed keys.
- When the Centralised Controller receive the response from key managers for key route request, controller initiates a key relay start request to transmitter key manager using the setKeyRelayStatus API**.**
- The transmitter key manager KM-A upon receiving key relay start request creates a random key **(KeyRN**) of the requested size.
- Then the **QKD NODE A** sends a request to KM of **QKD NODE G** to provide a Key for Encryption, Since both these nodes are in different networks using the keyprovision API **.**
- **QKD NODE G** will share the **Shared KeyGH** (**Shared key between QKD NODE G and QKD NODE H**) to **QKD NODE A.**
- As Transmitter receives the Key for Encryption , **KeyRN** XORed with **KeyGH** ,then the XORed key is transmitted to **QKD NODE G** using the keyManager API**.**
- As the encrypted key is reached **QKD NODE G** , here the key is directly forward to the **QKD NODE H** and no decryption is done because its Encrypted by **Shared KeyGH** using the km_destination API.
- The **KeyRN** is generated at **QKD NODE H** by Decrypting the received key using **Shared KeyGH**.
- Finally the **Application C** of Network 2 sends a request to get key with key with key ID to **QKD NODE H.**

7.2.2 QSDN Controller Orchestrator

Network 1 and **Network 2** are two Software Defined QKD Network, which need to establish a connection to enable secure key exchange between any two nodes across the networks. Here the Key exchange is monitored by the **SDN Controller** of each Network and **QSDN Controller Orchestrator**, which will help for Link establishment between two nodes of two different network. In this scenario the Key Exchange is between **QKD NODE A** of Network 1 to **QKD NODE H** of Network 2 which in monitored and regulated by the **SDN Controller** of each Network and **QSDN Controller Orchestrator**.

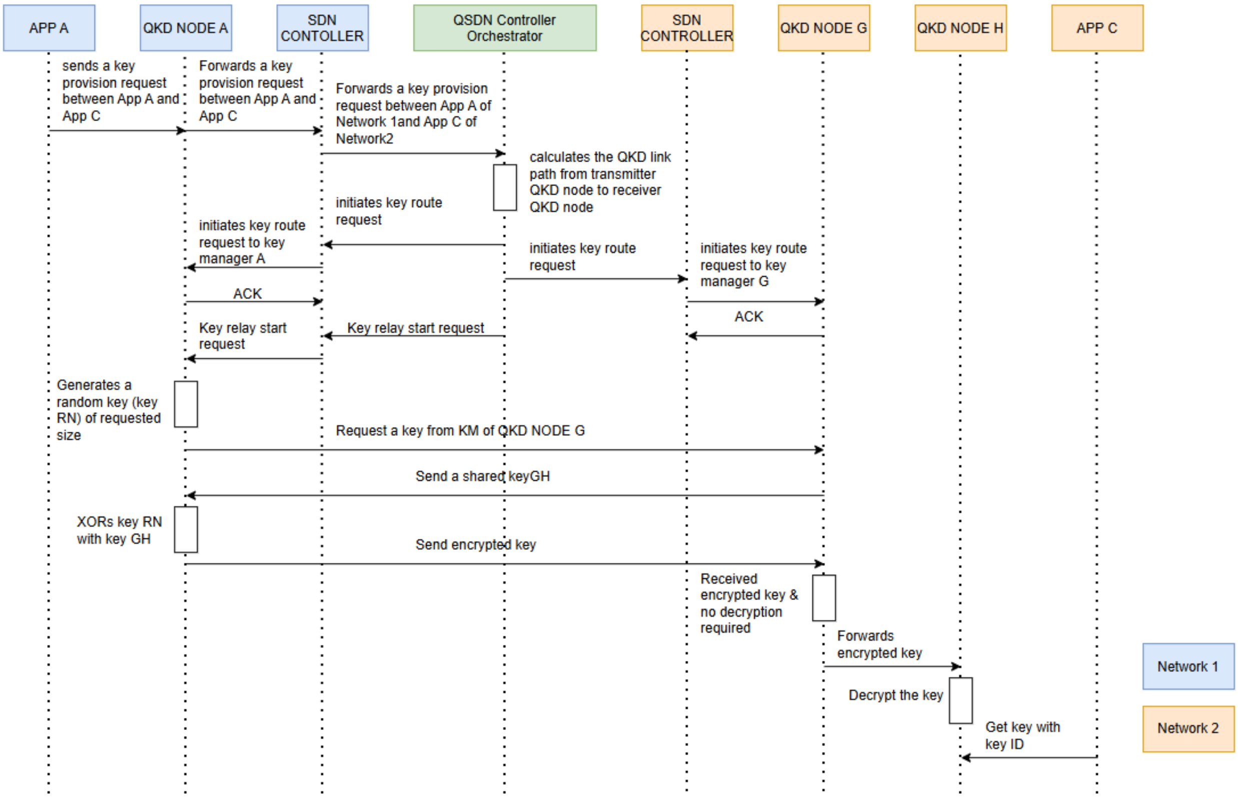

APP A
QKD NODE A
SDN CONTOLLER
QSDN Controller Orchestrator
SDN CONTROLLER
QKD NODE G
QKD NODE H
APP C
sends a key provision request between App A and App C
Forwards a key provision request between App A and App C
Forwards a key provision request between App A of Network 1and App C of Network2
calculates the QKD link path from transmitter QKD node to receiver QKD node
initiates key route request
initiates key route request to key manager A
initiates key route request
initiates key route request to key manager G
ACK
ACK
Key relay start request
Key relay start request
Generates a random key (key RN) of requested size
Request a key from KM of QKD NODE G
Send a shared keyGH
XORs key RN with key GH
Send encrypted key
Received encrypted key & no decryption required
Forwards encrypted key
Decrypt the key
Get key with key ID
Network 1
Network 2

Figure 13 :KEY RELAY USING QSDN CONTROLLER ORCHESTRATOR

PROCEDURE

- **Application A** sends a key provision request to **QKD NODE A** using the KeyProvision API.
- **QKD NODE A** forwards a key provision request to the SDN controller of Network 1.
- The SDN controller forwards the key provision request to **QSDN Controller Orchestrator**.
- **QSDN Controller Orchestrator** calculates the QKD link path from transmitter QKD node to receiver QKD node. If there is no direct QKD link path, controller calculates key relay path between two networks using the Keyrelayroute API.
- SDN Controller of Network 1 initiates key route request to key managers of **QKD NODE A** using the Keyrelaystart API**.**
- SDN Controller of Network 2 initiates key route request to key managers of **QKD NODE G** using the Keyrelaystart API.
- The key route request will be having the information like path to which key managers have to transfer the relayed keys.
- When the each Controller receive the response from key managers for key route request, controller initiates a key relay start request to transmitter key manager setKeyRelayStatus API**.**
- The transmitter key manager KM-A upon receiving key relay start request creates a random key **(KeyRN**) of the requested size.
- Then the **QKD NODE A** sends a request to KM of **QKD NODE G** to provide a Key for Encryption, Since both these nodes are in different networks using the keyprovision API.
- **QKD NODE G** will share the **Shared KeyGH** (**Shared key between QKD NODE G and QKD NODE H**) to **QKD NODE A.**

- As Transmitter receives the Key for Encryption, **KeyRN** XORed with **KeyGH** ,then the XORed key is transmitted to **QKD NODE G** using the keyManager API.
- As the encrypted key is reached **QKD NODE G** , here the key is directly forward to the **QKD NODE H** and no decryption is done because its Encrypted by **Shared KeyGH** using the km_destination API.
- The **KeyRN** is generated at **QKD NODE H** by Decrypting the received key using **Shared KeyGH**.
- Finally the **Application C** of Network 2 sends a request to get key with key with key ID to **QKD NODE H.**

7.3 Interfaces for Interoperability

7.3.1 Centralised SDN Controller Interface

Four types of interfaces are used in Network 1 and Network 2 which are depicted in the figure below.

- The **interface 1** is a REST based API which is used by the QKD Node to update its inventory details with the Centralised SDN controller.
- The **interface 2** is a REST based API used by Key Manager inside each node for supporting the registration of Encryption apps and to provision end-to-end keys with Centralised SDN Controller. The **interface 3** is ETSI Key Delivery API which is used by the Encryption App to register the application and retrieve the end-to-end key from the QKD Node.
- The **interface 4** is used for the key relay between the QKD Nodes which is required only if a shared key is provisioned between nodes of Network 1 and Network 2.

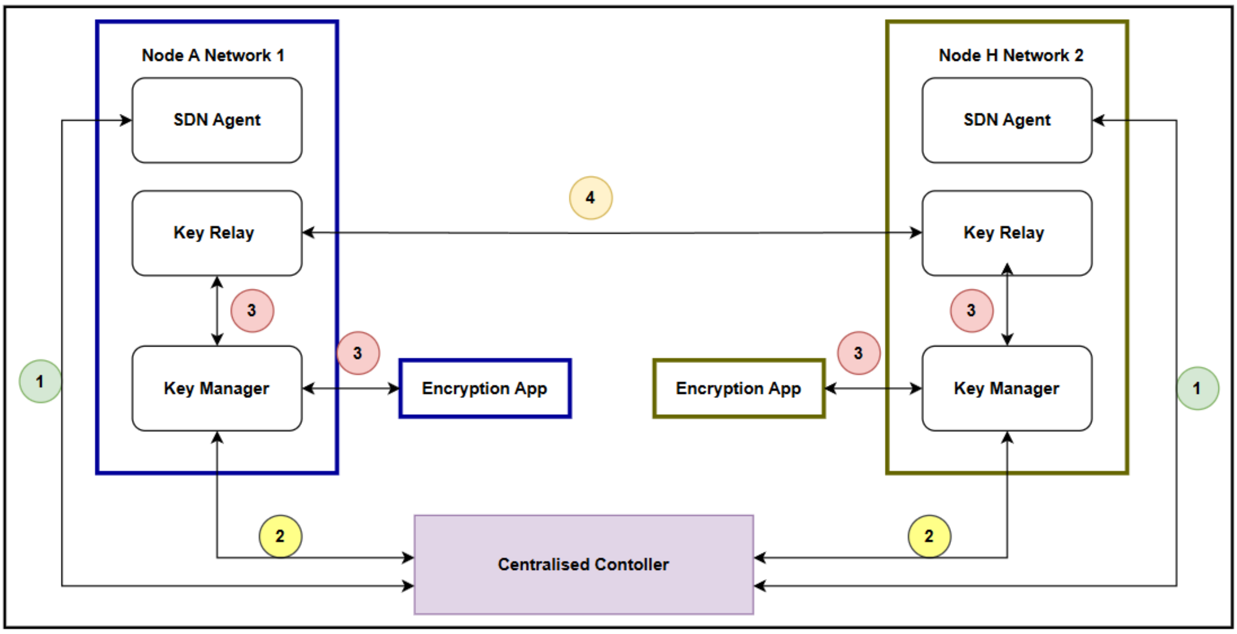


Figure 14: Centralised SDN Controller Interface

7.3.1.1 API Specifications of Interface 1 : SDN-Agent to SDN-Controller

The **interface 1** is a REST based API which is used by the QKD Node to update its inventory details with the Centralised SDN controller.

7.3.1.1.1 Device Init

The 'DEVICE_INIT' message is sent to the controller when the system is powered on.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/status
HEADER	NA
BODY	

```
{
  "MESSAGE_ID":"1",
  "NODE_ID":"node_x1",
  "NODE_NAME":"1@CDOT",
  "DOMAIN": "cdot.qkd.net",
  "NODE_IP":"172.18.1.1",
  "NODE_PORT":"5507",
  "STATUS_TYPE":"DEVICE_INIT",
  "TIMESTAMP":"18-02-2025 05:00:38",
  "KM_IP":"172.18.1.1",
  "KM_PORT":"5508",
  "QUANTUM_CHANNEL":[
    {
      "QC_ID":"X1X2",
      "QC_NAME":"CDOT1-CDOT2",
      "QC_STATUS":"UP",
      "QC_TYPE":"TX",
      "NEIGHBOUR_NODE_NAME":"2@CDOT",
      "NEIGHBOUR_NODE_ID":"node_x2"
    },
    {
      "QC_ID":"BX1",
      "QC_NAME":"CDOT-CDOT",
      "QC_STATUS":"UP",
      "QC_TYPE":"TX",
      "NEIGHBOUR_NODE_NAME":"Bob@ERNET",
      "NEIGHBOUR_NODE_ID":"node_b"
    }
  ],
  "CLASSICAL_CHANNEL": [
    {
      "CC_ID":"eth1",
      "CC_STATUS":"UP",
      "CC_NAME":"1@CLASSICAL-CDOT",
      "XD_LINK": false,
      "XD_NEIGHBOUR_NODE_ID":"NA"
    },
    {
      "CC_ID":"eth2",
      "CC_STATUS":"UP",
      "CC_NAME":"2@CLASSICAL-XL-CDOT",
      "XD_LINK": true,
      "XD_NEIGHBOUR_NODE_ID":"node_b"
    }
  ]
}
```

RESPONSE

```
{
  "response": "success",
  "type": "status"
}
```

CURL

7.3.1.1.2 Status Update

The 'STATUS_UPDATE' message is sent to the controller every five minutes.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/status
HEADER	NA

BODY

```
 {
  "MESSAGE_ID": "2",
  "NODE_ID": " node_x1",
  "NODE_NAME": "1@CDOT ",
  "DOMAIN": "cdot.qkd.net"
  "NODE_IP": "172.18.1.1",
  "NODE_PORT": "5507",
  "STATUS_TYPE": "STATUS_UPDATE",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "KM_IP": "172.18.1.1",
  "KM_PORT": "5508",
  "QUANTUM_CHANNEL": [
    {
      "QC_ID": "X1X2",
      "LINK_STATUS_QUANTUM": "UP",
      "LINK_ID_QUANTUM": "IITM-ERNET",
      "QBER": 0.0,
      "KEY_RATE": 0.0,
      "KEY_GEN_STATUS": ""
    }
  ],
  "CLASSICAL_CHANNEL": [
    {
      "CC_ID": "eth1",
 "CC_STATUS": "UP",
"CC_NAME": "1@CLASSICAL-CDOT"
        }
    ]
}
```

RESPONSE

```
{
  "response": "success",
   "type": "status"
}
```

CURL	

7.3.1.1.3 Agent Heart beat

The ‘NO_UPDATE’ message is sent to the controller every few seconds (configured in agent) to let controller know that the node is up.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/status
HEADER	NA

BODY

```
{
   "MESSAGE_ID": "2",
   "NODE_ID": "node_x1 ",
   "NODE_NAME": "1@CDOT ",
    "DOMAIN”: "cdot.qkd.net"
   "NODE_IP": "172.18.1.1",
   "NODE_PORT": "5507",
   "STATUS_TYPE": "NO_UPDATE”, // Heartbeat
   "TIMESTAMP": "18-02-2025 05:00:38",
   "KM_IP": "172.18.1.1",
   "KM_PORT": "5508",
   "QUANTUM_CHANNEL": [
      {
         "QC_ID": "X1X2",
         "LINK_STATUS_QUANTUM": "UP",
         "LINK_ID_QUANTUM": "IITM-ERNET",
         "QBER": 0.0,
         "KEY_RATE": 0.0,
         "KEY_GEN_STATUS": ""
      }
   ],
   "CLASSICAL_CHANNEL": [
      {
         "CC_ID": "eth1",
         "CC_STATUS": "UP",
         "CC_NAME": "2@CLASSICAL-XL-CDOT"
      }
   ]
}
```

RESPONSE

```
{
   "response": "success",
   "type": "status"
}
```

CURL	

7.3.1.1.4 Command Request (Link Provision)

The commands are issued from controller to the SDN agent.

Method	POST
URL	http://{{sdn_agent_ip}}:{{sdn_agent_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "1",
  "TIMESTAMP": "04-08-2023 15:11:41",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A1F1",
  "COMMAND": "QUANTUM_CHANNEL_PROVISION",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "OUTGOING_LINK": {
        "DEST_NODE": "node_x2",   //Node ID
        "SRC_NODE": "node_x1",
        "LINK_ID": "X1X2"
      },
      "INCOMING_LINK": {
        "DEST_NODE": "NA",
        "SRC_NODE": "NA",
        "LINK_ID": "NA"
      },
      "NODE_ID": "node_x1",
      "NO_OF_NODES_IN_LINK": 4,
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1"
}
```

RESPONSE

```
{
"response": "success",
"type": "command"
}
```

CURL

7.3.1.1.5 Command Response (Link Provision)

The command response is sending from SDN Agent to Controller as response to the requests issues by controller.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/command
HEADER	NA

BODY

```
           {

  "MESSAGE_ID": "3",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_PROVISION",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X1F1",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
  ]
}
```

RESPONSE

```
{
"response": "success",
"type": "command"
}
```

CURL	

7.3.1.1.6 Command Request (Link Provision Start)

Method	POST
URL	http://{{sdn_agent_ip}}:{{sdn_agent_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "3",
  "TIMESTAMP": "04-08-2023 15:12:02",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A2F2",
  "COMMAND": "QUANTUM_CHANNEL_START",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1"
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL	

7.3.1.1.7 Command Response (Link Provision Start)

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/command
HEADER	NA

BODY

```
          {

  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_START",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X2B2",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
  ]
}

     {

  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_START",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X2B2",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "FAILED"
    }
  ]
}
```

- {

```
  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "<KEY GENERATION STEPS>",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X2B2",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "FAILED"
    }
  ]
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL

7.3.1.1.8 Command Request (Link Re-Provision)

Method	POST
URL	http://{{sdn_agent_ip}}:{{sdn_agent_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "7",
  "TIMESTAMP": "04-08-2023 15:12:54",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A3F2",
  "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "OUTGOING_LINK": {
        "DEST_NODE": "node_x2",
        "SRC_NODE": "node_x1",
        "LINK_ID": "X1X2"
      },
      "INCOMING_LINK": {
        "DEST_NODE": "NA",
        "SRC_NODE": "NA",
        "LINK_ID": "NA"
      },
      "NODE_ID": "node_x1",
      "NO_OF_NODES_IN_LINK": 4,
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1"
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL	

7.3.1.1.9 Command Response (Link Re-Provision)

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/command
HEADER	NA

BODY

```
     •   {

  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X3B3",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
  ]
}
     {

  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X3B3",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "FAILED"
    }
  ]
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL	

7.3.1.1.10 Command Request (Link Provision Stop)

Method	POST
URL	http://{{sdn_agent_ip}}:{{sdn_agent_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "5",
  "TIMESTAMP": "04-08-2023 15:12:02",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A2F2",
  "COMMAND": "QUANTUM_CHANNEL_STOP",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1"
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL	

7.3.1.1.11 Command Response (Link Provision Stop)

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/AgentPush/command
HEADER	NA

BODY

```
    •   {

  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_STOP",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X4B4",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
  ]
}
```

```
•   {
  "MESSAGE_ID": "5",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_STOP",
  "NODE_ID": "node_x1",
  "COMMAND_ID": "X4B4",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "FAILED"
    }
  ]
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL

7.3.1.2 API Specifications of Interface 2 : Key Manger to SDN Controller

The **interface 2** is a REST based API used by Key Manager inside each node for supporting the registration of Encryption apps and to provision end-to-end keys with Centralised SDN Controller.

7.3.1.2.1 Application Registration

To register a cryptographic application and get SAE_ID.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/application/registerApp
HEADER	NA

BODY

```
{
  "source_KME_ID": "node_x1"
  “Application_Type”:”keymanager/xd_keymanager” // optional
  “SAE_D”: “10000001” //Optional if provided by xd_km or application
}
```

RESPONSE

```
{
“SAE_ID” : “10000001” //Same as supplied if it is accepted otherwise a fresh ID
}
```

CURL

7.3.1.2.2 Key reserve

To reserve certain no. of keys in the key store

Method	POST
URL	http://{{km_ip}}:{{km_port}}/keyManager/reserveKeys
HEADER	NA

BODY

```
{
  "key_ID": "8424fa33-b21f-4cdf-9ff1-2c240c17c1a5",
  "source_SAE_ID": "10000001",
  "target_SAE_ID": "10000002",
  "Number": 1,
  "Size": 128,
  "type": "Quantum"
}
```

RESPONSE

```
{
"key_ID" : "8424fa33-b21f-4cdf-9ff1-2c240c17c1a5",
"provision_status"  : "Complete",
"provision_message"  :  "Complete",
"type":"Quantum" / "Relay" / "PQC"
}
```

CURL	

7.3.1.2.3 Clear Key store

To clear keys from the key store.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/keyManager/clearKeys
HEADER	NA
BODY	NA

RESPONSE

```
{

  "status": "Complete"
}
```

CURL	

7.3.1.2.4 Key Provision

To reserve a certain no. of key in key store as requested by cryptographic application

Method	POST

URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/keyProvision/provision
HEADER	NA

BODY	{
	"source_SAE_ID":"10000001",
	"target_SAE_ID":"10000002",
	"Number":1,
	"Size":128,
	"Application_Type":"video" / "audio" / "keymanager"
	}

RESPONSE	{
	"key_ID" : "8424fa33-b21f-4cdf-9ff1-2c240c17c1a5",
	"provision_status" : "Complete",
	"provision_message" : "Complete",
	"type":"Quantum" / "Relay" / "PQC"
	}

CURL	

7.3.1.2.5 Key Provision Status

To reserve a certain no. of key in key store as requested by cryptographic application

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/keyProvision/status/
HEADER	NA

BODY	{
	"key_ID" : "8424fa33-b21f-4cdf-9ff1-2c240c17c1a5",
	}

RESPONSE	{
	"key_ID" : "8424fa33-b21f-4cdf-9ff1-2c240c17c1a5",
	"provision_status" : "Complete",
	"provision_message" : "Complete",
	"type":"Quantum" / "Relay" / "PQC"
	}

CURL	

7.3.1.3 API Specifications of Interface 3 : Application to Key Manager

The **interface 3** is ETSI Key Delivery API which is used by the Encryption App to register the application and retrieve the end-to-end key from the QKD Node.

7.3.1.3.1 Application Registration

To register a cryptographic application for both transmitter and receiver and get SAE_ID.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/register
HEADER	NA

BODY	{
	"source_KME_ID":"node_a",
	"target_KME_ID":"node_b",
	"Application_Type":"video" / "audio" / "keymanager"
	“SAE_ID”:”10000001” //Existing SAE_ID if, optional
	}

RESPONSE	{
	“SAE_ID” : “10000001” //Existing SAE_ID if requested and accepted otherwise a fresh ID
	}

CURL	

7.3.1.3.2 Key Provision

To reserve a certain no. of key in key store as requested by cryptographic application.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/keyprovision
HEADER	NA

BODY	{
	"source_SAE_ID":"10000001",
	"target_SAE_ID":"10000002",
	"Number":1,
	“Size”: 128
	}

RESPONSE	{
	“key_ID” : “a3f2c1d4-89b7-4e32-b0df-9c4a57e8f912”,
	“provision_status” : “Success”,
	“provision_message” : “Success” ,
	"type":"Quantum" / “Relay” / “PQC”
	}

CURL	

7.3.1.3.3 Key Provision Status

To reserve a certain no. of key in key store as requested by cryptographic application

Method	POST

URL	http://{{km_ip}}:{{km_port}}/keyprovisionStatus
HEADER	NA

BODY	{
	“key_ID” : “a3f2c1d4-89b7-4e32-b0df-9c4a57e8f912”,
	}

RESPONSE	• {
	“key_ID” : “a3f2c1d4-89b7-4e32-b0df-9c4a57e8f912”,
	“provision_status” : “Success”,
	“provision_message” : “Success” ,
	"type":"Quantum" / “Relay” / “PQC”
	}
	• {
	“key_ID” : “a3f2c1d4-89b7-4e32-b0df-9c4a57e8f912”,
	“provision_status” : “Failed”,
	“provision_message” : “<Failure message>”
	"type":"Quantum" / “Relay” / “PQC”
	}

CURL	

7.3.1.3.4 Get Key with Id

To retrieve key from key manager using key_ID

Method	POST
URL	http://{{km_ip}}:{{km_port}}/api/v1/keys/SAE_ID/dec_keys
HEADER	NA

BODY	{
	"source_KME_ID": "node_x1",
	"target_KME_ID": "node_x2",
	"master_SAE_ID": " 1010",
	"key_ID": " a3f2c1d4-89b7-4e32-b0df-9c4a57e8f912",
	"type" : “Quantum" / “Relay” / “PQC”
	}

RESPONSE	{"keys": [{"key": "MTAwMDAwMDEwMTExMDExMA=="}]} // Base64 (UTF-8(key))
CURL	

7.3.1.3.5 Get Status

To get the key store status from key manager

Method	POST

URL	http://{{km_ip}}:{{km_port}}/api/v1/keys/SAE_ID/status
HEADER	NA

BODY

```
{
  "source_KME_ID": "node_x1",
  "target_KME_ID": "node_x2",
  "master_SAE_ID": "1010",
  "type": "All" / "Quantum" / "Relay" / "PQC"
}
```

RESPONSE

- If type is All:

```
{
  "Quantum": {
    "source_KME_ID": "node_x1",
    "target_KME_ID": "node_x2",
    "master_SAE_ID": "NA",
    "slave_SAE_ID": "NA",
    "key_size": 544,
    "stored_key_count": 2,
    "max_key_count": 1024,
    "max_key_per_request": 1,
    "max_key_size": 1024,
    "min_key_size": 64,
    "max_SAE_ID_count": 0,
    "max_key_store_size": 20480
  },
  "Relay": {
    "status": "Key Store - Not Available "
  },
  "PQC": {
    "status": "Key Store - Not Available "
  }
}
```

- If type is Quantum, Relay or PQC

```
{
  "source_KME_ID": "node_x1",
  "target_KME_ID": "node_x2",
  "master_SAE_ID": "NA",
  "slave_SAE_ID": "NA",
  "key_size": 576,
  "stored_key_count": 2,
  "max_key_count": 1024,
  "max_key_per_request": 1,
  "max_key_size": 1024,
  "min_key_size": 64,
  "max_SAE_ID_count": 0,
  "max_key_store_size": 20480,
  "status": "Key Store Available"
}
```

CURL

7.3.1.4 API Specifications of Interface 4 : Key Relay

The **interface 4** is used for the key relay between the QKD Nodes which is required only if a shared key is provisioned between nodes of Network 1 and Network 2.

7.3.1.4.1 SDN Controller to Key Manager : Key Relay Route

Controller raises key route to each of the key managers in the key relay path.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/keyRelay/route
HEADER	NA

BODY

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "node_id":"node_b",
  "type":"intermediate",
  "source_id":"node_a",
  "destination_id":"node_x2",
  "relay_type":"normal" / "inter_domain",
  "relay_path":[
    {
      "relay_source":"node_b",
      "relay_destination":"node_x1",
      "relay_action": "encrypt_forward" / "decrypt_encrypt_forward" / "decrypt"/ 
"xd_encrypt_forward"/  "xd_decrypt_encrypt_forward"/ "xd_decrypt" / forward
    }
  ],
"relay_provision":[
    {
      "provision_source":"node_x1",
      "provision_destination":"node_x2",
      "provision_type": "qkd_provision" / "xd_key_provision"
     “source_KM_SAE_ID”:”1000002”
     “target_KM_SAE_ID”:”1000003”
    }
  ],
  "live_key_required":true,
  "Number":1,
  "Size":128,
  "neighbournode_ip":"10.176.27.85",
  "neighbournode_port":"7071",
  "source_SAE_ID":"1000001",
  "target_SAE_ID":"1000002",
  "source_ip":"10.176.27.85",
  "source_port":"7060"
}
```

RESPONSE

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "keyRoute_Status":"COMPLETE",
  "keyRoute_Message":"Key relay routing success!!!"
}
```

CURL

7.3.1.4.2 Key Manager to SDN Controller : Key Relay Status

Key managers informs controller about the status of the key relay during each step. The destination key manager will set the status as COMPLETE if the hash match is success.

Method	POST
URL	http://{{sdn_controller_ip}}:8080/MAQAN_CONTROLLER/keyRelay/setKeyRelayStatus
HEADER	NA

BODY

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "keyRelay_Status":"COMPLETE",
  "keyRelay_Message":"Key Relay Completed",
  "current_node_id":"node_a",
  "neighbour_node_id":"node_c"
}
```

RESPONSE

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "keyRoute_Status":" COMPLETE ",
  "keyRoute_Message":" Key Relay Completed”
}
```

CURL	

7.3.1.4.3 SDN Controller to Key Manager : Key Relay Start

Controller raises key relay start to the source key manager informing it to start key relay. This will be initiated by the controller only if the key route request in success.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/keyRelay/relaystart
HEADER	NA

BODY

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "node_id":"node_a",
  "type":"source",
  "source_id":"node_a",
  "destination_id":"node_c",
  "Number":1,
  "Size":128,
  "destination_ip":"10.176.27.85",
  "destination_port":"7072",
  "source_SAE_ID":"",
  "target_SAE_ID":""
}
```

RESPONSE

```
{
  "relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "keyRoute_Status":"INPROGRESS",
  "keyRoute_Message":"Key relay Started!!!"
}
```

CURL

7.3.1.4.4 Key Manager to Key Manager : Transfer Key to neighbour KM

Request to send XORed relay key to neighbour key manager.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/Key_Relay/keyManager
HEADER	Key-id: a5a04e21-a61c-4096-be4c-7f000b549a3d Target-node-id: node_x2

BODY

```
{
  "relayed_key":
"MEEwMDcxMDg3OTcwMDg3NTA1NzYwQzA2MDE3MzcwMDQwNjc1MDcwMjdFMDQw
NjAwMDIwNDc3\nMDEwMDBDN0M3MA==\n",
  "source_id": "node_x1",
  "destination_id": "node_x2",
  "relayId": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "appid": "1010",
  "size": 128,
  "Number": 1
}
```

RESPONSE

```
{
  "keyRoute_Status":"INPROGRESS"
}
```

CURL

7.3.1.4.5 Key Manager to Key Manager: Transfer Key to destination KM

Request to send XORed relay key to destination key manager.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/Key_Relay/km_destination
HEADER	Key-id: a5a04e21-a61c-4096-be4c-7f000b549a3d Target-node-id: node_x2

BODY

```
{
  "relayed_key": "MEEwMDcxMDg3OTcwMDg3NTA1NzYwQzA2MDE3MzcwMDQwNjc1MDcwMjdFMDQwNjAwMDIwNDc3\nMDEwMDBDN0M3MA==\n", // Base64 Encoded
  "source_id": "node_x1",
  "destination_id": "node_x2",
  "relayId": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "appid": "1010",
  "size": 128,
  "Number": 1
}
```

RESPONSE

```
{
  "keyRoute_Status":"INPROGRESS"
}
```

CURL	

7.3.1.4.6 Key Manager to Key Manager: Inform hash to Source KM

This request will be raised by destination key manager to source key manager. Used to compare hash values of relayed key at source and destination key managers. Key relay is success if hash values are same.

Method	POST
URL	http://{{km_ip}}:{{km_port}}/Key_Relay/inform_source
HEADER	NA

BODY

```
{
  "relayed_key": "038DDEC1251D9BA407583A38C489BA3843AB5CC79783FF6D187A97F89AB93267", //Hash – SHA-256
  "keyRoute_Status": "INPROGRESS",
  "keyRoute_Message": "Successfull received relay key at destination",
  "relayId": "a5a04e21-a61c-4096-be4c-7f000b549a3d"
}
```

RESPONSE

```
{
  "keyRoute_Status":"COMPLETE"
}
```

CURL	

7.3.1.5 API Mapping of Centralised SDN Controller Interface

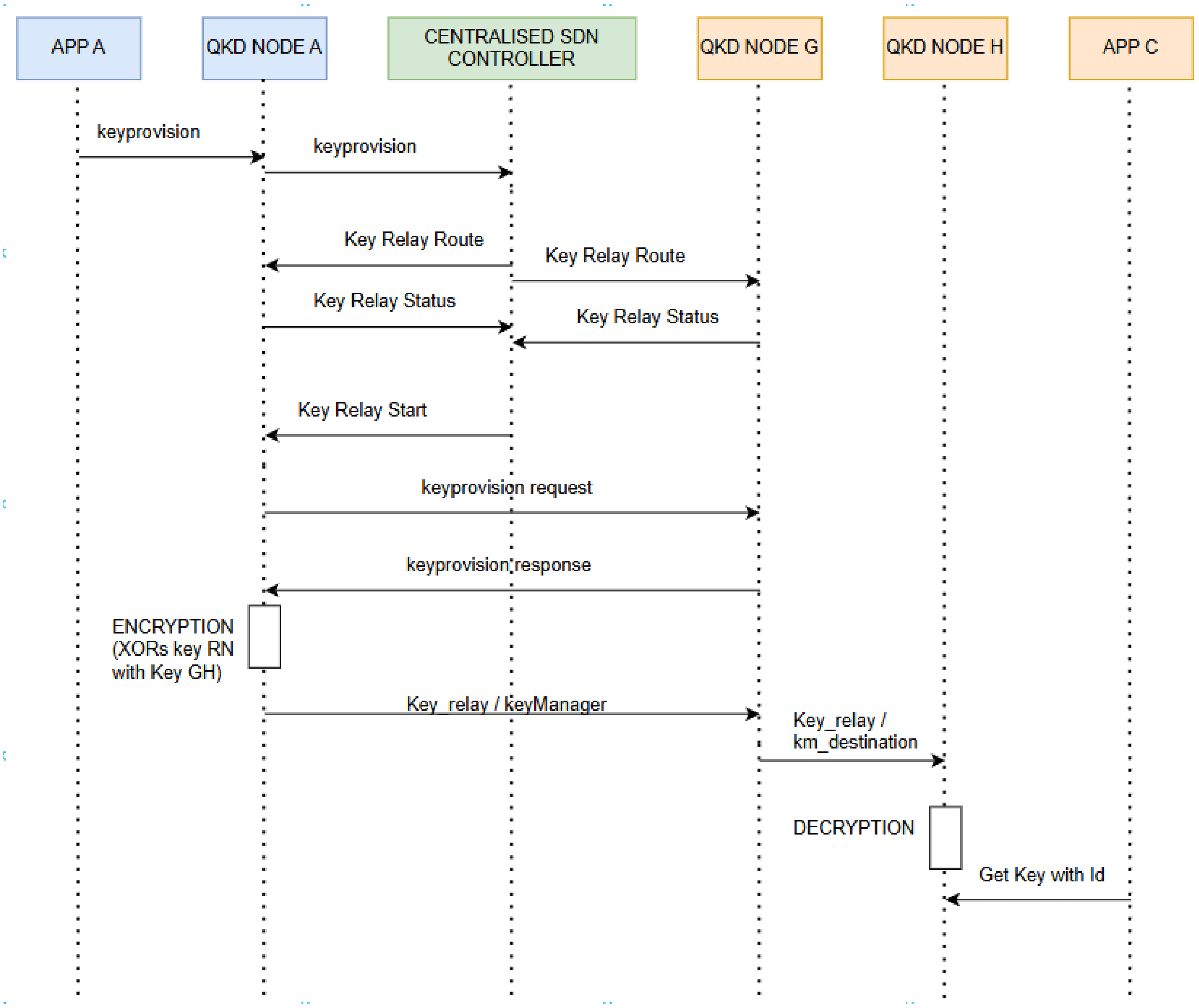


Figure 15 : API Mapping of Centralised SDN Controller

7.3.2 Controller Orchestrator Interface

Five types of interfaces are used in Network 1 and Network 2 which are depicted in the figure below.

- The **interface 1** is a REST based API which is used by the QKD Node to update its inventory details with the Centralised SDN controller.

- The **interface 2** is a REST based API used by Key Manager inside each node for supporting the registration of Encryption apps and to provision end-to-end keys with Centralised SDN Controller. The **interface 3** is ETSI Key Delivery API which is used by the Encryption App to register the application and retrieve the end-to-end key from the QKD Node.
- The **interface 4** is used for the key relay between the QKD Nodes which is required only if a shared key is provisioned between nodes of Network 1 and Network 2.
- The **interface 5** enables communication between the SDN Controller of two Networks and the QSDN Controller Orchestrator. This interface supports the integration and orchestration of classical and quantum resources, enabling resource allocation & link provisioning across both networks.

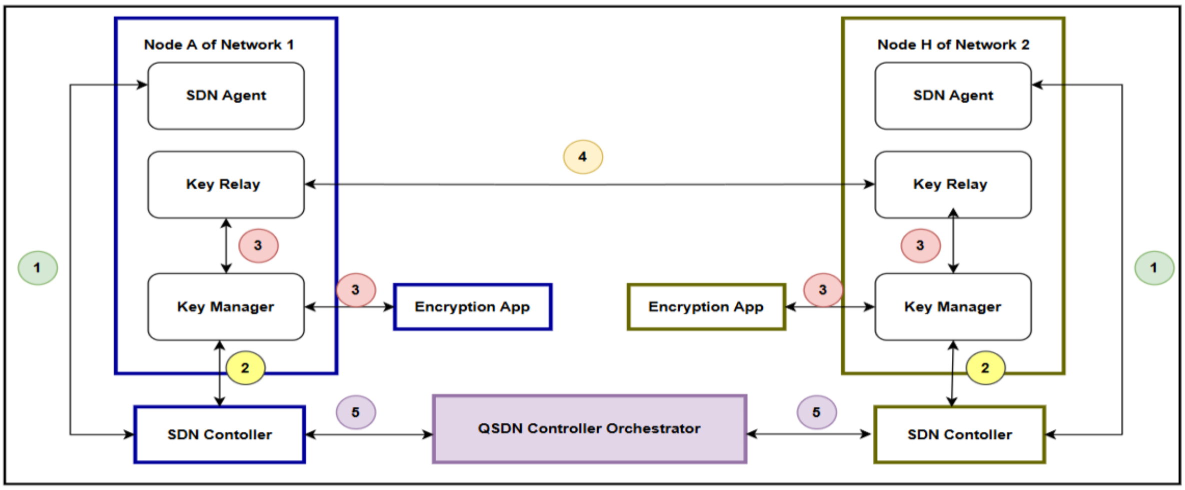


Figure 16: QSDN Controller Orchestrator Interface

The API Specification for **Interface 1, Interface 2, Interface 3, and Interface 4** have already been detailed in **Sections 7.3.1.1, 7.3.1.2, 7.3.1.3, and 7.3.1.4**, respectively. While these interfaces API specification are similar for Centralised SDN Controller Interface and Controller Orchestrator Interface. Please refer to the respective sections for details on these interfaces.

7.3.2.1 API Specifications of Interface 5: SDN Controller to QSDN Controller Orchestrator

The **interface 5** enables communication between the SDN Controller of two Networks and the QSDN Controller Orchestrator. This interface supports the integration and orchestration of classical and quantum resources, enabling resource allocation & link provisioning across both networks.

7.3.2.1.1 Controller Registration

To register SDN Controller of each network with the QSDN Controller Orchestrator.

Method	POST
URL	http://{{orchestrator_ip}}:8080/controller/register

HEADER	NA

BODY

```
{
  "CONTROLLER_IP": "192.168.20.75",
  "CONTROLLER_PORT": 8080,
  "NETWORK_ID": "network1",
  "LOCATION": "NA",
  "TIMESTAMP": "2025-02-18T10:17:00Z"
}
```

RESPONSE

```
{

  "STATUS": "SUCCESS",
  "MESSAGE": "Controller registered successfully",
  "CONTROLLER_ID": "controller 1", // Orchestrator-assigned ID
  "TIMESTAMP": "2025-02-18T10:17:10Z"
}
```

CURL	

7.3.2.1.2 Inventory/deviceinit

This API is sent to the QSDN Controller Orchestrator upon a QKD device's power-on, to register the device and its initial quantum and classical channel inventory.

Method	POST
URL	http://{{orchestrator_ip}}:8080/inventory/deviceinit
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "1",
  "NODE_ID": "node_x1",
  "NODE_NAME": "1@CDOT",
  "DOMAIN": "cdot.qkd.net",
  "NODE_IP": "172.18.1.1",
  "NODE_PORT": "5507",
  "STATUS_TYPE": "DEVICE_INIT",
  "TIMESTAMP": "18-02-2025 05:00:38",
  "KM_IP": "172.18.1.1",
  "KM_PORT": "5508",
  "CONTROLLER_ID": "controller1",
  "QUANTUM_CHANNEL": [
    {
      "QC_ID": "X1X2",
      "QC_NAME": "CDOT1-CDOT2",
      "QC_STATUS": "UP",
      "QC_TYPE": "TX",
      "NEIGHBOUR_NODE_NAME": "2@CDOT",
      "NEIGHBOUR_NODE_ID": "node_x2",
      "NEIGHBOUR_NETWORK_ID": "NA"
    },
    {
      "QC_ID": "BX1",
      "QC_NAME": "CDOT-ERNET",
      "QC_STATUS": "UP",
      "QC_TYPE": "TX",
      "NEIGHBOUR_NODE_NAME": "Bob@ERNET",
      "NEIGHBOUR_NODE_ID": "node_b",
      "NEIGHBOUR_NETWORK_ID": "NA"
    }
  ],
  "CLASSICAL_CHANNEL": [
    {
      "CC_ID": "eth1",
      "CC_STATUS": "UP",
      "CC_NAME": "1@CLASSICAL-CDOT",
      "XD_LINK": false,
      "XD_NEIGHBOUR_NODE_ID": "NA",
      "XD_NEIGHBOUR_NETWORK_ID": "NA"
    },
    {
      "CC_ID": "eth2",
      "CC_STATUS": "UP",
      "CC_NAME": "2@CLASSICAL-XL-CDOT",
      "XD_LINK": true,
      "XD_NEIGHBOUR_NODE_ID": "node_b",
      "XD_NEIGHBOUR_NETWORK_ID": "NA"
    }
  ]
}
```

RESPONSE

```
{
  "status": "success",
  "message": "Inventory updated"
}
```

CURL	

7.3.2.1.3 Application registration

To register the cryptographic application by SDN controller (after receiving the application registration request from KM) and get SAE_ID from Orchestrator.

Method	POST
URL	http://{{orchestrator_ip}}:8080/QSDN_ORCHESTRATOR/application/registerApp
HEADER	NA

BODY

```
{
  "source_KME_ID": "node_a",
  "controller_id": "controller1"
}
```

RESPONSE

```
{
“SAE_ID” : “10000001” ,
 "CONTROLLER_ID": "controller 1"
```

	}
CURL	

7.3.2.1.4 Link Provision (Request)

The commands are issued from Orchestrator to Controller of each network.

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/SDN/command
HEADER	NA

BODY

```
{
   "MESSAGE_ID": "1",
   "TIMESTAMP": "2025-02-18 12:00:00",
   "MESSAGE_TYPE": "REQUEST",
   "COMMAND_ID": "LP001",
   "COMMAND": "QUANTUM_CHANNEL_PROVISION",
   "COMMAND_PARAMS": [
      {
         "TRANSMITTER": "node_a",
         "MODE": "TX",
         "OUTGOING_LINK": {
            "DEST_NODE": "node_h",
            "SRC_NODE": "node_a",
            "LINK_ID": "AH_LINK"
         },
         "INCOMING_LINK": {
            "DEST_NODE": "NA",
            "SRC_NODE": "NA",
            "LINK_ID": "NA"
         },
         "NODE_ID": "node_a",
         "NO_OF_NODES_IN_LINK": 2,
         "RECEIVER": "node_h"
      }
   ],
   "NODE_ID": "node_a",
   "CONTROLLER_ID": "controller1"
}
```

RESPONSE	{
	"response": "Success",
	"Type": "command"
	}

CURL	

7.3.2.1.5 Link Provision (Response)

To report the status of the link provisioning command back to the Orchestrator from particular SDN Controller.

Method	POST
URL	http://{{orchestrator_ip}}:8080/Controller1/AgentPush/command
HEADER	NA

BODY

-
  ```
  {

   "MESSAGE_ID": "3",
   "MESSAGE_TYPE": "RESPONSE",
   "COMMAND": "QUANTUM_CHANNEL_PROVISION",
   "NODE_ID": "node_a",
   "COMMAND_ID": "A1F1",
   "TIMESTAMP": "2025-02-18 12:55:35",
   "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
   ],
   "CONTROLLER_ID": "controller1"
  }
  ```

-
  ```
  {
   "MESSAGE_ID": "4",
   "MESSAGE_TYPE": "RESPONSE",
   "COMMAND": "QUANTUM_CHANNEL_PROVISION",
   "NODE_ID": "node_a",
   "COMMAND_ID": "A1F1",
   "TIMESTAMP": "2025-02-18 12:56:35",
   "RESPONSE_PARAMS": [
    {
      "STATUS": "FAILED",
      "ERROR_MESSAGE": "Unable to establish quantum channel: Resource
  unavailable"
    }
   ],
   "CONTROLLER_ID": "controller1"
  }
  ```

RESPONSE	{
	"response": "Success",
	"Type": "command"
	}

CURL	

7.3.2.1.6 Link Provision Start(Request)

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "3",
  "TIMESTAMP": "2025-02-18 13:10:02",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A2F2",
  "COMMAND": "QUANTUM_CHANNEL_START",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1",
  "CONTROLLER_ID": "controller1"
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL

7.3.2.1.7 Link Provision Start (Response)

Method	POST
URL	http://{{orchestrator_ip}}:8080/ Controller1/AgentPush/command
HEADER	NA

BODY

```
    •  {
        "MESSAGE_ID": "5",
        "MESSAGE_TYPE": "RESPONSE",
        "COMMAND": "QUANTUM_CHANNEL_START",
        "NODE_ID": "node_x1",
        "CONTROLLER_ID": "controller1",
        "COMMAND_ID": "A2F2",
        "TIMESTAMP": "2025-02-18 12:55:35",
        "RESPONSE_PARAMS": [
         {
           "STATUS": "SUCCESS"
         }
    ]
  }
```

-
  ```
  {
    "MESSAGE_ID": "6",
    "MESSAGE_TYPE": "RESPONSE",
    "COMMAND": "QUANTUM_CHANNEL_START",
    "NODE_ID": "node_x1",
    "CONTROLLER_ID": "controller1",
    "COMMAND_ID": "A2F2",
    "TIMESTAMP": "2025-02-18 12:56:35",
    "RESPONSE_PARAMS": [
      {
        "STATUS": "FAILED",
        "ERROR_MESSAGE": "Failed to start quantum channel: Device offline"
      }
    ]
  }
  ```

-
  ```
  {
    "MESSAGE_ID": "7",
    "MESSAGE_TYPE": "RESPONSE",
    "COMMAND": "",
    "NODE_ID": "node_x1",
    "CONTROLLER_ID": "controller1",
    "COMMAND_ID": "A2F2",
    "TIMESTAMP": "2025-02-18 12:57:35",
    "RESPONSE_PARAMS": [
      {
        "STATUS": "FAILED",
        "ERROR_MESSAGE": "Failed to start quantum channel: Configuration error"
      }
    ]
  }
  ```

RESPONSE

```
{
   "response": "success",
   "type": "command"
}
```

CURL

7.3.2.1.8 Link Re-Provision(Request)

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/SDN/command

HEADER	NA

BODY

```
{
 "MESSAGE_ID": "7",
 "TIMESTAMP": "2025-02-18 15:12:54",
 "MESSAGE_TYPE": "REQUEST",
 "COMMAND_ID": "A3F2",
 "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
 "COMMAND_PARAMS": [
  {
   "TRANSMITTER": "node_x1",
   "MODE": "TX",
   "OUTGOING_LINK": {
    "DEST_NODE": "node_x2",
    "SRC_NODE": "node_x1",
    "LINK_ID": "X1X2"
   },
   "INCOMING_LINK": {
    "DEST_NODE": "NA",
    "SRC_NODE": "NA",
    "LINK_ID": "NA"
   },
   "NODE_ID": "node_x1",
   "NO_OF_NODES_IN_LINK": 4,
   "RECEIVER": "node_x2"
  }
 ],
 "NODE_ID": "node_x1",
 "CONTROLLER_ID": "controller1"
}
```

RESPONSE

```
{
  "response": "success",
  "type": "command"
}
```

CURL	

7.3.2.1.9 Link Re-Provision(Response)

Method	POST
URL	http://{{orchestrator_ip}}:8080/ Controller1/AgentPush/command
HEADER	NA

BODY

```
• {

  "MESSAGE_ID": "8",
  "MESSAGE_TYPE": "RESPONSE",
  "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
  "NODE_ID": "node_x1",
  "CONTROLLER_ID": "controller1",
  "COMMAND_ID": "A3F2",
  "TIMESTAMP": "2025-02-18 05:00:38",
  "RESPONSE_PARAMS": [
    {
      "STATUS": "SUCCESS"
    }
  ]
}
```

- {

```
    "MESSAGE_ID": "9",
    "MESSAGE_TYPE": "RESPONSE",
    "COMMAND": "QUANTUM_CHANNEL_REPROVISION",
    "NODE_ID": "node_x1",
    "CONTROLLER_ID": "controller1",
    "COMMAND_ID": "A3F2",
    "TIMESTAMP": "2025-02-18 05:00:38",
    "RESPONSE_PARAMS": [
      {
        "STATUS": "FAILED"
      }
    ]
  }
```

RESPONSE

```
{
   "response": "success",
   "type": "command"
}
```

CURL

7.3.2.1.10 Link Provision Stop(Request)

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/SDN/command
HEADER	NA

BODY

```
{
  "MESSAGE_ID": "5",
  "TIMESTAMP": "2025-02-18 15:12:02",
  "MESSAGE_TYPE": "REQUEST",
  "COMMAND_ID": "A2F2",
  "COMMAND": "QUANTUM_CHANNEL_STOP",
  "COMMAND_PARAMS": [
    {
      "TRANSMITTER": "node_x1",
      "MODE": "TX",
      "RECEIVER": "node_x2"
    }
  ],
  "NODE_ID": "node_x1",
  "CONTROLLER_ID": "controller1"
}
```

RESPONSE

```
{
   "response": "success",
   "type": "command"
}
```

CURL	

7.3.2.1.11 Link Provision Stop (Response)

Method	POST
URL	http://{{orchestrator_ip}}:8080/ Controller1/AgentPush/command
HEADER	NA

BODY

-
  ```
  {

   "MESSAGE_ID": "10",
   "MESSAGE_TYPE": "RESPONSE",
   "COMMAND": "QUANTUM_CHANNEL_STOP",
   "NODE_ID": "node_x1",
   "CONTROLLER_ID": "controller1",
   "COMMAND_ID": "A2F2",
   "TIMESTAMP": "2025-02-18 05:00:38",
   "RESPONSE_PARAMS": [
    {
     "STATUS": "SUCCESS"
    }
   ]
  }
  ```

-
  ```
  {

   "MESSAGE_ID": "11",
   "MESSAGE_TYPE": "RESPONSE",
   "COMMAND": "QUANTUM_CHANNEL_STOP",
   "NODE_ID": "node_x1",
   "CONTROLLER_ID": "controller1",
   "COMMAND_ID": "A2F2",
   "TIMESTAMP": "2025-02-18 05:00:38",
   "RESPONSE_PARAMS": [
    {
     "STATUS": "FAILED"
    }
   ]
  }
  ```

RESPONSE

```
{
   "response": "success",
   "type": "command"
}
```

CURL	

7.3.2.1.12 Key Relay Route

The Orchestrator initiates the Key Relay Route by sending a request to each SDN Controller involved in the key relay path.

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/keyRelay/route
HEADER	NA
BODY	

```
{
  "relay_id": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "source_id": "node_1",
  "destination_id": "node_6",
  "relay_type": "normal / inter_domain",
  "network_controllers": [
    {
      "network": "Network_1",
      "controller_ip": "192.168.1.1",
      "controller_port": "8080"
    },
    {
      "network": "Network_2",
      "controller_ip": "192.168.2.1",
      "controller_port": "8081"
    }
  ],
  "relay_path": [
    {
      "relay_source": "node_1",
      "relay_destination": "node_4",
      "relay_action": "encrypt_forward / decrypt_encrypt_forward / decrypt /
xd_encrypt_forward / xd_decrypt_encrypt_forward / xd_decrypt"
    },
    {
      "relay_source": "node_4",
      "relay_destination": "node_5",
      "relay_action": "encrypt_forward"
    },
    {
      "relay_source": "node_5",
      "relay_destination": "node_6",
      "relay_action": "decrypt_encrypt_forward"
    }
  ],
  "relay_provision": [
    {
      "provision_source": "node_4",
      "provision_destination": "node_5",
      "provision_type": "qkd_provision / xd_key_provision"
    },
    {
      "provision_source": "node_5",
      "provision_destination": "node_6",
      "provision_type": "qkd_provision / xd_key_provision"
    }
  ],
  "live_key_required": true,
  "Number": 1,
  "Size": 128,
  "neighbournode_ip": "10.176.27.85",
  "neighbournode_port": "7071",
  "source_SAE_ID": "NA",
  "target_SAE_ID": "NA",
  "source_ip": "10.176.27.85",
  "source_port": "7060"
}
```

RESPONSE

```
{
  "relay_id": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "keyRoute_Status": "COMPLETE",
  "keyRoute_Message": "Key relay routing success!!!",
  "keyRelay_Path": ["node_1", "node_4", "node_5", "node_6"]
}
```

CURL	

7.3.2.1.13 Key Relay Status

Each SDN Controller receives updates from its respective Key Managers about the key relay status at every step. The controller then forwards this status update to the Orchestrator to ensure synchronization across networks.

Method	POST
URL	http://{{orchestrator_ip}}:8080/Controller1/keyRelay/setKeyRelayStatus
HEADER	NA

BODY

```
{
  "relay_id": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "controller_id": "controller_1",
  "keyRelay_Status": "IN_PROGRESS / COMPLETE / FAILED",
  "keyRelay_Message": "Key relay is in progress / Key relay completed / Key relay failed",
  "current_node_id": "node_1",
  "neighbour_node_id": "node_4"
}
```

RESPONSE

```
{
"relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
"keyRoute_Status":" COMPLETE ",
"keyRoute_Message":" Key Relay Completed”
}
```

CURL	

7.3.2.1.14 Key Relay Start

The Orchestrator sends a key relay start request to the SDN Controller of the source network, instructing it to initiate the key relay process.

Method	POST
URL	http://{{sdn_controller_ip}}:{{sdn_controller_port}}/keyRelay/relaystart
HEADER	NA

BODY

```
{
  "relay_id": "a5a04e21-a61c-4096-be4c-7f000b549a3d",
  "controller_id": "controller_1",
  "action": "key_relay_start",
  "node_id": "node_a",
  "type": "source",
  "source_id": "node_a",
  "destination_id": "node_c",
  "Number": 1,
  "Size": 128,
  "destination_ip": "10.176.27.85",
  "destination_port": "7072",
  "source_SAE_ID": "",
  "target_SAE_ID": "",
  "status": "INITIATED"
}
```

RESPONSE

```
{
"relay_id":"a5a04e21-a61c-4096-be4c-7f000b549a3d",
"keyRoute_Status":"INPROGRESS",
"keyRoute_Message":"Key relay Started!!!"
}
```

CURL

7.4 Field Tests and Evaluation Result

The evaluation of the proposed APIs for the interoperability of multi-vendor QKD networks, is conducted in the interdomain QKD testbed at Chennai connecting Metro Area Quantum Access Network(MAQAN) and the CDOT QKD Network(CQN). The main goal of these tests is to assess the secure key exchange process between application endpoints connected to two different QKD Network (MAQAN and CQN). The topology of the integrated testbed connecting MAQAN and CDOT is provided in Figure 17. The MAQN Node B and the CQN Node G are collocated at ERNET and are connected through a secure trusted classical link. All the QKD Nodes of both MAQAN And CQN are implemented the proposed interoperability APIs and are connected to a Software Defined QKD Network (SD-QKDN) Controller running at ERNET. The SD-QKDN controller is also implemented the proposed APIs.

The filed test is conducted to relay secure keys between MAQAN Node A and CQN Node H and subsequent shared key provision between 'App A' and 'App C' in MAQAN and CQN respectively. 'DARPAN-Q,' the SDN based QKD Network Management application of MAQAN is used for the same.

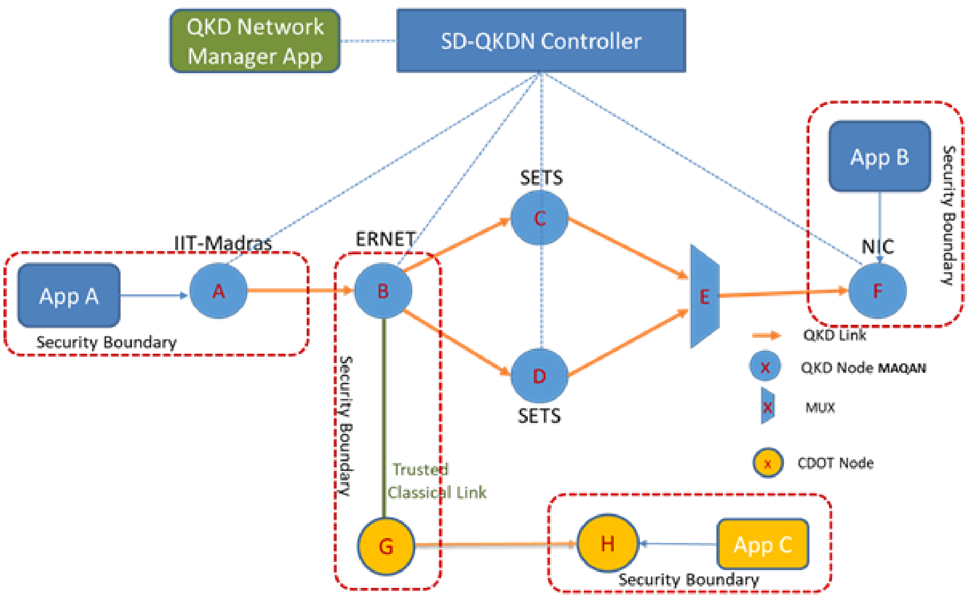


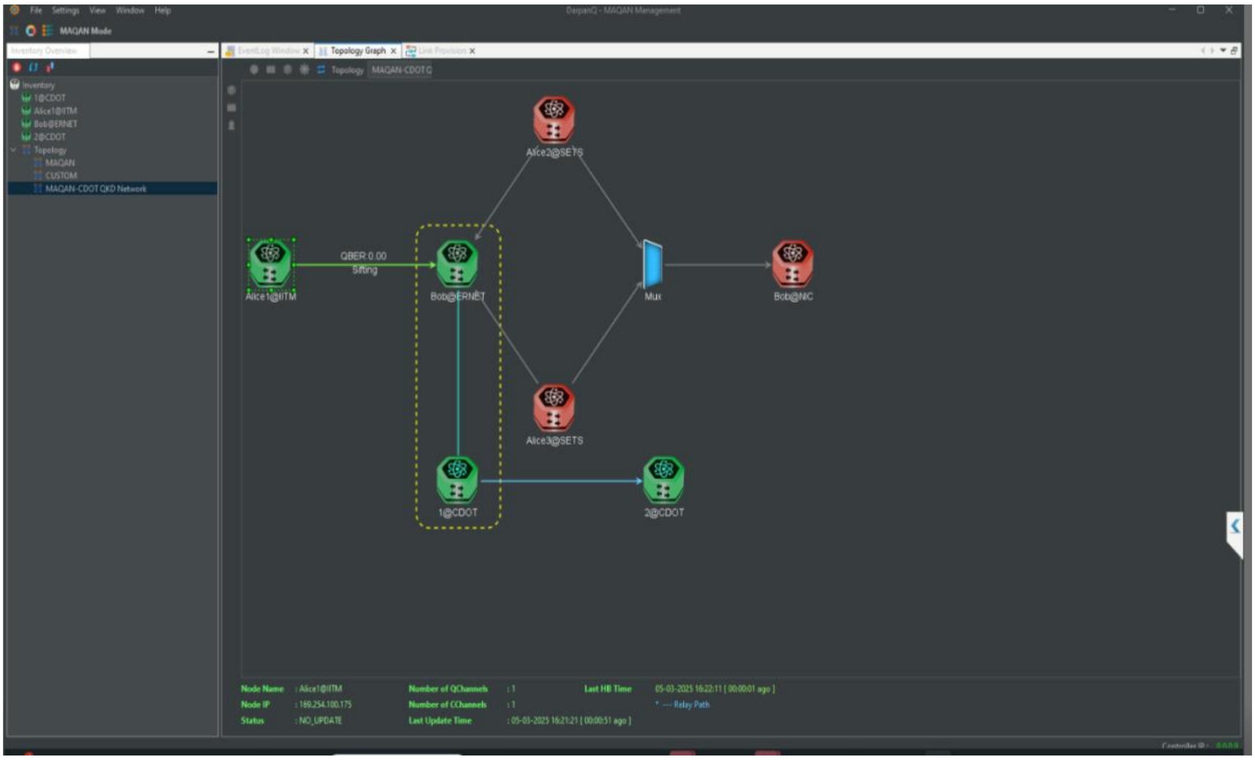
Figure 17 : Demonstration of Interoperability of Multi-Vendor QKD Network using SDN

The primary objective was to assess the secure transmission of keys between Node A of the MAQAN network and Node H of the CDOT network. Here Node B of the MAQAN network and Node G of the CDOT network were within the same security boundary, establishing a classical communication channel between them.

The key transmission process followed these steps:

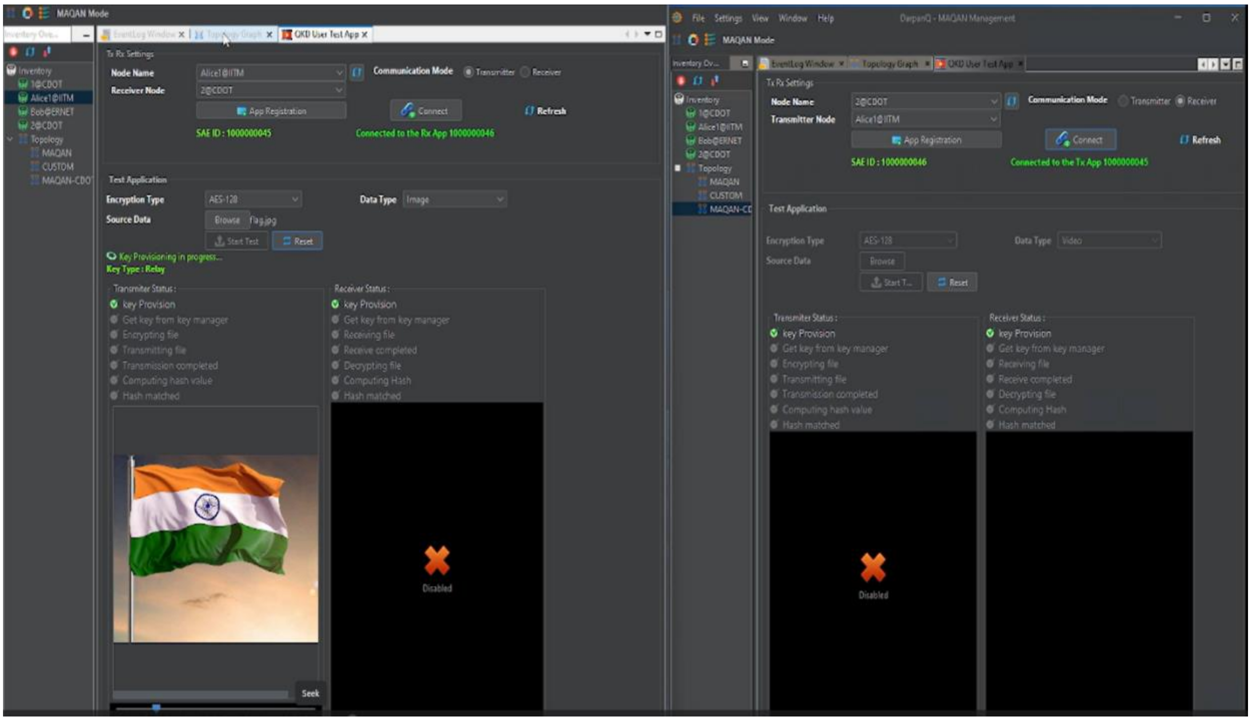
- **QKD NODE A** forwards a key provision request to the centralized SDN controller.
- Controller calculates the QKD link path from transmitter QKD node to receiver QKD node. If there is no direct QKD link path, controller calculates key relay path between two Networks **.**
- Controller initiates key route request to all key managers between **QKD NODE A** and **QKD NODE G.**
- When the Centralised Controller receive the response from key managers for key route request, the controller initiates a key relay start request to transmitter key manager in **Node A.**
- The transmitter key manager KM-A upon receiving key relay start request, encrypt the random key **(KeyRN**) and send to **Node B** using a shared Quantum key as these nodes are belong to the same QKD network and connected through a quantum channel**.**
- Then the **QKD NODE B** sends a request to KM of **QKD NODE G** to provide a shared Key for Encryption between KM-B and KM-G, since both these nodes are in different networks.
- **QKD NODE G** will share the **Shared KeyGH** (**Shared key between QKD NODE G and QKD NODE H**) to **QKD NODE B.**
- As Transmitter receives the Key for Encryption , **KeyRN** XORed with **KeyGH** ,then the XORed key is transmitted to **QKD NODE G**
- As the encrypted key is reached **QKD NODE G** , here the key is directly forward to the **QKD NODE H** and no decryption is done because its Encrypted by **Shared KeyGH**
- The **KeyRN** is generated at **QKD NODE H** by Decrypting the received key using **Shared KeyGH**.
- Then a shared key is provisioned between 'App A' and 'App C' and the same is used to securely transmit an image across the applications.

1.4.1 Screenshots



Node Name : Alice1@IITM
Node IP : 169.254.100.175
Status : NO_UPDATE
Number of QChannels : 1
Number of CChannels : 1
Last Update Time : 05-03-2025 16:21:21 [00:00:51 ago]
Last HB Time 05-03-2025 16:22:11 [00:00:01 ago]

Figure 18 : MAQAN-CDOT Network



Tx Rx Settings
Node Name Alice1@IITM
Receiver Node 2@CDOT
Communication Mode ● Transmitter ○ Receiver
SAE ID : 1000000045
Connected to the Rx App 1000000046

Test Application
Encryption Type AES-128
Data Type Image
Source Data flag.jpg
Key Provisioning in progress...
Key Type : Relay

Transmiter Status :
- ✓ key Provision
- ○ Get key from key manager
- ○ Encrypting file
- ○ Transmitting file
- ○ Transmission completed
- ○ Computing hash value
- ○ Hash matched

Receiver Status :
- ✓ key Provision
- ○ Get key from key manager
- ○ Receiving file
- ○ Receive completed
- ○ Decrypting file
- ○ Computing Hash
- ○ Hash matched

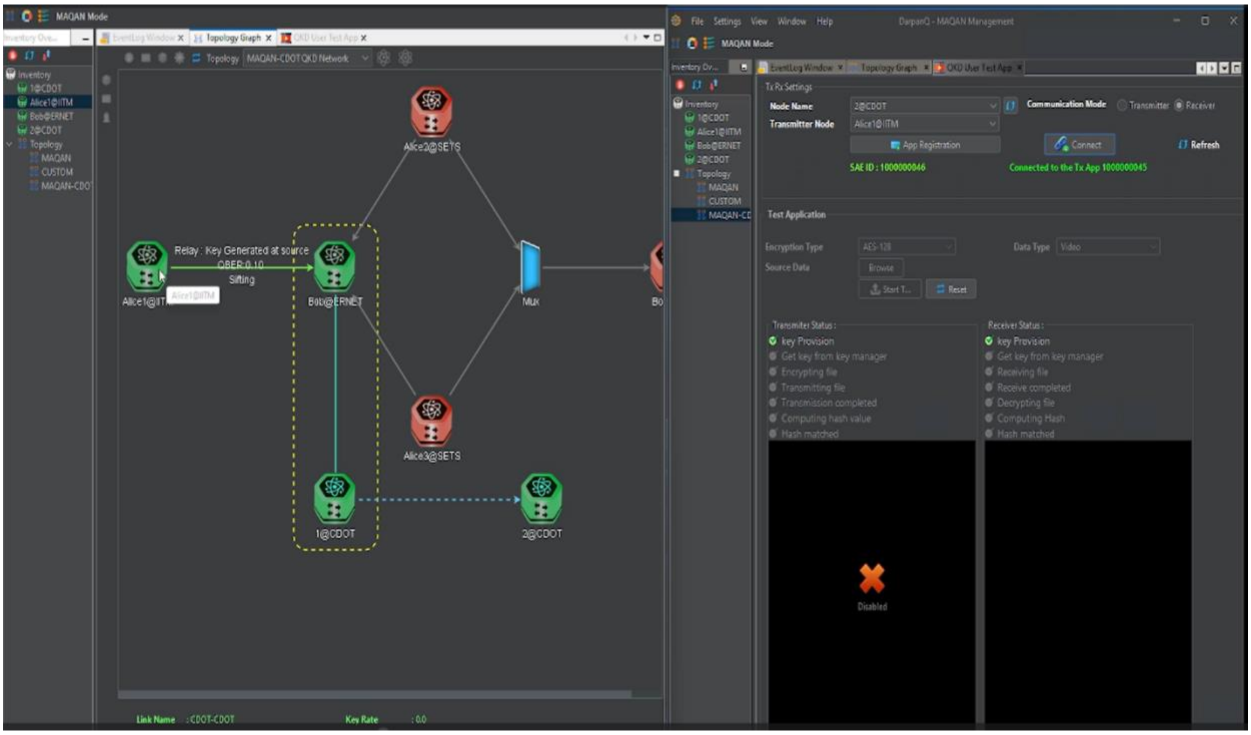
Tx Rx Settings
Node Name 2@CDOT
Transmitter Node Alice1@IITM
Communication Mode ○ Transmitter ● Receiver
SAE ID : 1000000046
Connected to the Tx App 1000000045

Test Application
Encryption Type AES-128
Data Type Video

Transmiter Status :
- ✓ key Provision
- ○ Get key from key manager
- ○ Encrypting file
- ○ Transmitting file
- ○ Transmission completed
- ○ Computing hash value
- ○ Hash matched

Receiver Status :
- ✓ key Provision
- ○ Get key from key manager
- ○ Receiving file
- ○ Receive completed
- ○ Decrypting file
- ○ Computing Hash
- ○ Hash matched

Figure 19 : Key Provisioning between Alice@IITM to 2@CDOT



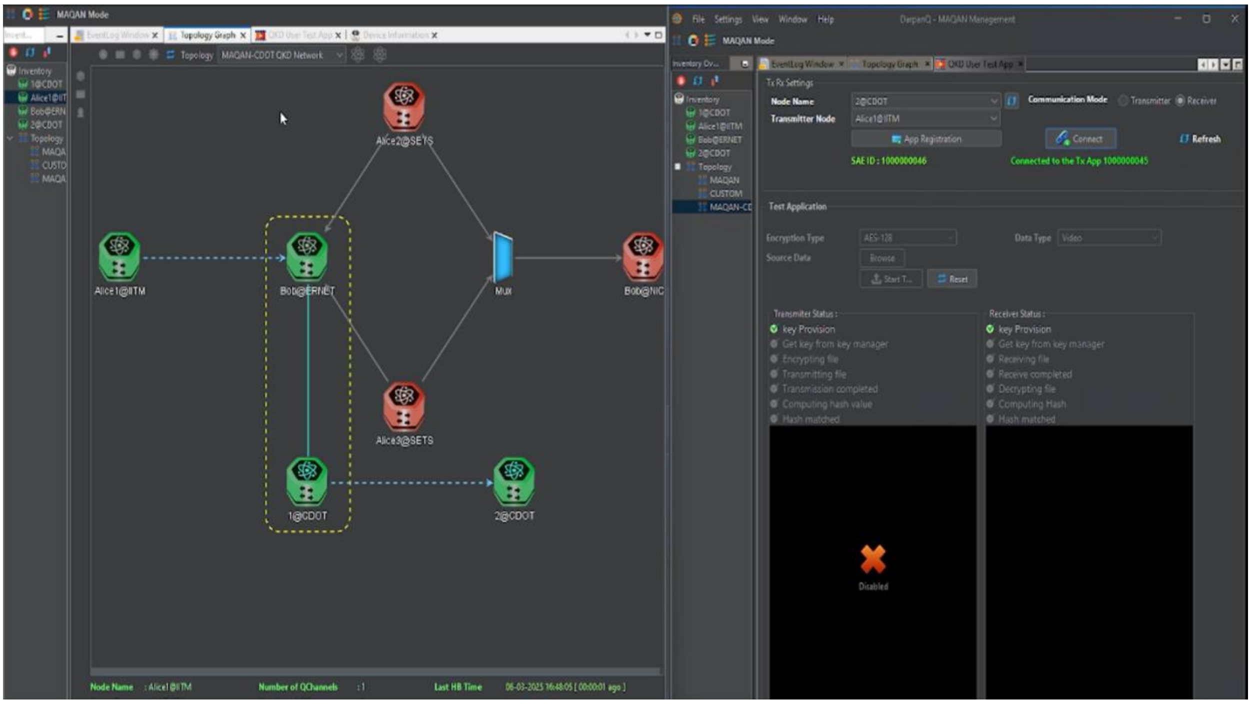
Link Name : CDOT-CDOT Key Rate : 0.0

Node Name: 2@CDOT
Transmitter Node: Alice1@IITM
Communication Mode ○ Transmitter ● Receiver
SAE ID : 1000000046
Connected to the Tx App 1000000045
Encryption Type: AES-128
Data Type: Video

Transmitter Status :
- key Provision
- Get key from key manager
- Encrypting file
- Transmitting file
- Transmission completed
- Computing hash value
- Hash matched

Receiver Status :
- key Provision
- Get key from key manager
- Receiving file
- Receive completed
- Decrypting file
- Computing Hash
- Hash matched

Figure 20 : Relay :Key Generates at Source Node



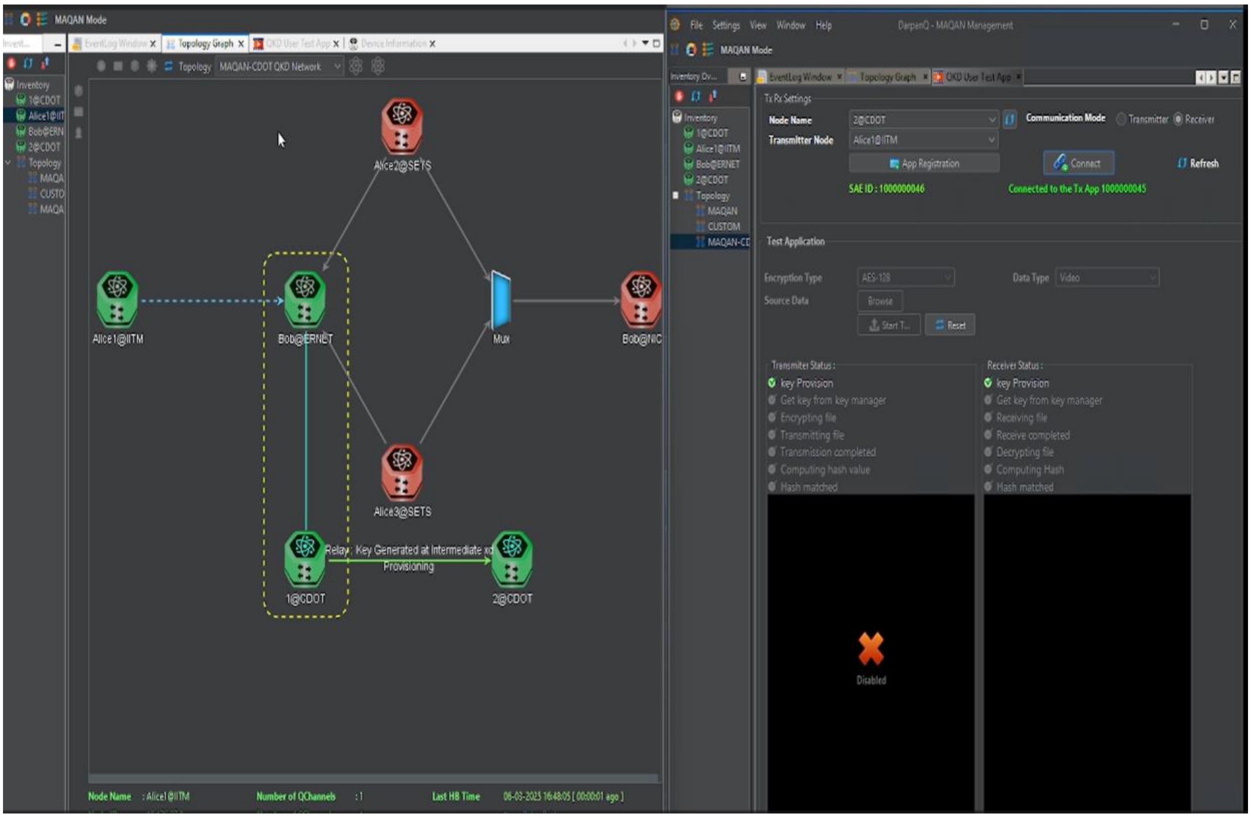
Node Name : Alice1@IITM Number of QChannels : 1 Last HB Time 06-03-2025 16:48:05 [00:00:01 ago]

Node Name: 2@CDOT
Transmitter Node: Alice1@IITM
Communication Mode ○ Transmitter ● Receiver
SAE ID : 1000000046
Connected to the Tx App 1000000045
Encryption Type: AES-128
Data Type: Video

Transmitter Status :
- key Provision
- Get key from key manager
- Encrypting file
- Transmitting file
- Transmission completed
- Computing hash value
- Hash matched

Receiver Status :
- key Provision
- Get key from key manager
- Receiving file
- Receive completed
- Decrypting file
- Computing Hash
- Hash matched

Figure 21 : Key Provision Ongoing



Relay : Key Generated at Intermediate xd Provisioning

Node Name : Alice1@IITM Number of QChannels : 1 Last HB Time 06-03-2023 16:48:05 [00:00:01 ago]

Node Name 2@CDOT

Transmitter Node Alice1@IITM

Communication Mode ○ Transmitter ● Receiver

SAE ID : 1000000046

Connected to the Tx App 1000000045

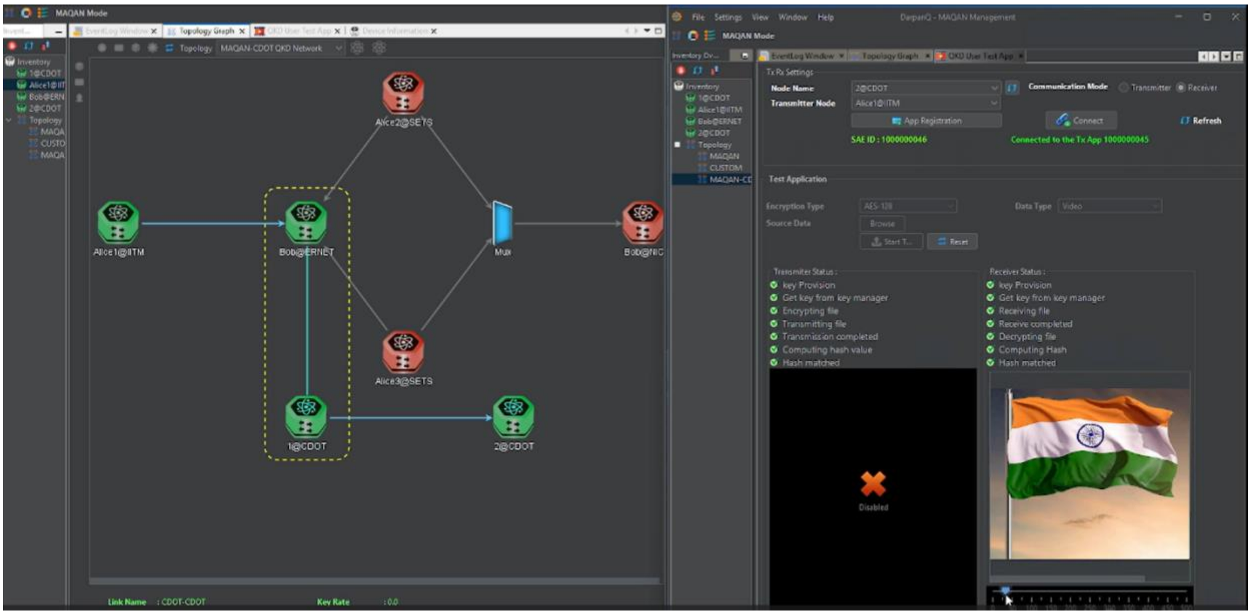
Encryption Type AES-128

Data Type Video

Transmiter Status :
- ✓ key Provision
- Get key from key manager
- Encrypting file
- Transmitting file
- Transmission completed
- Computing hash value
- Hash matched

Receiver Status :
- ✓ key Provision
- Get key from key manager
- Receiving file
- Receive completed
- Decrypting file
- Computing Hash
- Hash matched

Figure 22 : Relay : Key Generated at Intermediate Node



Link Name : CDOT-CDOT Key Rate : 0.0

Node Name 2@CDOT

Transmitter Node Alice1@ITM

Communication Mode ○ Transmitter ● Receiver

SAE ID : 1000000046

Connected to the Tx App 1000000045

Encryption Type AES-128

Data Type Video

Transmiter Status :
- ✓ key Provision
- ✓ Get key from key manager
- ✓ Encrypting file
- ✓ Transmitting file
- ✓ Transmission completed
- ✓ Computing hash value
- ✓ Hash matched

Receiver Status :
- ✓ key Provision
- ✓ Get key from key manager
- ✓ Receiving file
- ✓ Receive completed
- ✓ Decrypting file
- ✓ Computing Hash
- ✓ Hash matched

Figure 23 : Transmission Completed

2. Conclusion and Recommendations

Interoperability of Quantum Key Distribution (QKD) networks is essential in the age of quantum computing, which is having a significant impact on traditional cryptographic systems. As quantum computing technology progresses, encryption techniques like RSA and ECC will become inadequate, rendering quantum-safe communication necessary. Nevertheless, the absence of interoperability between multi-vendor QKD hardware has stood as a significant obstacle to the establishment of extensive quantum communication networks.

This research study, **'Interoperability of Multi-vendor QKD Hardware using SDN'**, successfully created and validated a framework for interoperability based on SDN for QKD networks. The research aimed to achieve interoperability through the use of both a centralized SDN controller and a controller-orchestrator model, ensuring secure key exchanges between nodes across various QKD networks. The application of APIs permitted the smooth integration of multi-vendor QKD systems, enabling key provisioning, key relay, and key route management.

Two main network architectures were examined. The centralized SDN controller model utilized a single controller to oversee key exchanges among multiple QKD networks, guaranteeing uniform control and policy application. The controller-orchestrator model distributed the control among independent network controllers, coordinated by a supreme orchestrator, improving scalability and fault tolerance. A comparative evaluation of both architectures underscored their respective strengths and weaknesses in achieving seamless interoperability of QKD networks.

The research successfully met its defined goals by addressing interoperability issues through the creation of SDN interfaces, protocols, and APIs. Numerous elements of QKD interoperability, management based on SDN, and existing QKD standards were thoroughly examined and incorporated into the suggested framework. The successful implementation of the approach based on the centralized SDN controller validated the practicality of interoperability in multi-vendor QKD scenarios. However, the controller-orchestrator model remains in the study phase, needing further implementation and validation using real-world applications to assess the impact of distributed control. The successful evaluation of the API in the field, make it eligible for further standardisation.

3. Reference

[1] https://www.etsi.org/technologies/quantum-key-distribution
[2] https://www.iso.org/obp/ui/en/#iso:std:iso-iec:23837:-2:ed-1:v1:en
[3] https://www.itu.int/rec/T-REC-Y.3804-202009-I/en
[4] https://ieeexplore.ieee.org/document/7063507
[5] https://www.iso.org/obp/ui/en/#iso:std:77309:en
[6] *Information technology security techniques-Security requirements, test and evaluation methods for quantum key distribution-Part 1: DRAFT INTERNATIONAL STANDARD*. (2022).
[7] https://www.itu.int/en/ITU-T/focusgroups/qit4n/Documents/D2.5.pdf
[8] Lopez, B., Vidal, I., Valera, F., Lopez, D. R., & Pastor, A. (2024). Unleashing Flexibility and Interoperability in QKD Networks: The Power of Softwarized Architectures. Proceedings - 2024 International Conference on Quantum Communications, Networking, and Computing, QCNC 2024, 216–220. https://doi.org/10.1109/QCNC62729.2024.00041
[9] Sim, D. H., Shin, J., & Kim, M. H. (2023). Software-Defined Networking Orchestration for Interoperable Key Management of Quantum Key Distribution Networks. Entropy, 25(6). https://doi.org/10.3390/e25060943
[10] Y. Zhao et al., “Resource Allocation in Optical Networks Secured by Quantum Key Distribution,” IEEE Commun. Mag., vol. 56, no. 8, pp. 130–137, 2018, doi: 10.1109/MCOM.2018.1700656.
[11] W. Yu, B. Zhao, and Z. Yan, “Software defined quantum key distribution network,” in IEEE International Conference on Computer and Communications Software, 2017, vol. 3, pp. 1293–1297.
[12] Zhao, Yongli, et al. "Software Defined Quantum Key Distribution Networks." Asia Communications and Photonics Conference. Optica Publishing Group, 2019.
[13] Draft Recommendation Software Defined Networking Control for Quantum Key Distribution Networks, ITU-T Y.3805
[14]