IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Towards a complete software stack to integrate Quantum Key Distribution in a cloud environment

IGNAZIO PEDONE, ANDREA ATZENI, DANIELE CANAVESE, AND ANTONIO LIOY, *Member, IEEE*

Politecnico di Torino, Dip. Automatica e Informatica, Torino, Italy (e-mail: name.surname@polito.it)

Corresponding author: Ignazio Pedone (e-mail: ignazio.pedone@polito.it).

ABSTRACT The coming advent of Quantum Computing promises to jeopardize current communications security, undermining the effectiveness of traditional public-key based cryptography. Different strategies (Post-Quantum or Quantum Cryptography) have been proposed to address this problem. Many techniques and algorithms based on quantum phenomena have been presented in recent years; the most relevant example is the introduction of Quantum Key Distribution (QKD). This approach allows to exchange cryptographic keys among parties and does not suffer from the development of quantum computation. Problems arise when this technique has to be deployed and combined with modern distributed infrastructures that heavily depend on cloud and virtualisation paradigms. This paper addresses this issue by presenting a new software stack that effortlessly introduces QKD in such environments and involves a simulation tool for Quantum Key Distribution. This software stack allows for agnostic integration, monitoring, and management of QKD, independent from a specific vendor or technology. Furthermore, a QKD simulator is presented, designed, and tested. This latter contribution is suitable as a low-level testing device, as an independent software module to check QKD protocols, and as a testbed to identify future practical enhancements.

INDEX TERMS Quantum Cryptography, QKD, Quantum Communication, Softwarised Infrastructures

I. INTRODUCTION

Quantum Computing (QC) is a ground-breaking field that promises to solve problems otherwise impossible to tackle with classical computation. This holds for many scientific fields such as physics, chemistry, molecular biology, and computer science. However, some applications of QC introduce potential threats for IT systems. The primary concern is security, and in particular classical cryptographic algorithms. Some widely used public-key algorithms are endangered from QC advent, since this new paradigm can solve cryptographic problems in polynomial time.

Traditional public-key cryptographic techniques are used as a fundamental security pillar in modern IT infrastructures that are increasingly becoming "Softwarised infrastructures", an umbrella term used to indicate Cloud Computing, Network Functions Virtualisation (NFV) technologies, Edge or Fog Computing, and Internet of Things (IoT). Since the strength of these crypto-systems is at risk, the use of quantum-resistant cryptographic techniques in cloud environments is a desirable feature today, but it will become a vital requirement in the not-so-distant future. In fact, quantum threats appear consistent in the next 10-15 years[1].

Over the past decades, several approaches have been proposed to address the potential fall of the classical asymmetric algorithms. The more effective are *Post-quantum Cryptography* (PQC) and *Quantum Cryptography*. The first relies on new classical algorithms that shall be designed to be quantum-resistant and replace the current public-key ones. Some examples are SPHINCS^{+}[2] and Dilithium[3] for digital signatures, and NTRU[4] for key exchange. These algorithms have been submitted to the National Institute of Standards and Technology (NIST) PQC challenge[5] for replacing and standardising the new quantum-resistant public-

[1] https://globalriskinstitute.org/download/quantum-threat-timeline-report-2020/.
[2] https://sphincs.org.
[3] https://pq-crystals.org/dilithium/index.shtml.
[4] https://ntru.org.
[5] https://csrc.nist.gov/projects/post-quantum-cryptography.

key cryptographic algorithms. The second approach leverages Quantum Cryptography. In particular, Quantum Key Distribution (QKD). This technique is based on the no-cloning theorem and – for specific protocols – on entanglement [1]. The main idea is to detect an eavesdropper on the quantum channel when a key exchange takes place. In this paper, we mostly rely on this technique, even though we also use PQC to support specific tasks – mostly as a complement rather than an alternative.

The recent literature focuses on the simulation of Quantum Networks and quantum protocols. In addition, recent approaches underline the importance of integrating quantum-based technology within modern infrastructures. In particular, the European Telecommunications Standards Institute (ETSI) is working on defining clear specifications on how to build quantum devices, design QKD networks, and even implement standard APIs to facilitate the interconnection among devices coming from different vendors. However, the integration efforts in a softwarised infrastructure have been so far made mostly as high-level analyses, i.e. without considering real infrastructures or technologies that could fit the goal. Moreover, the standards are quite general, not envisioning consistent optimizations that could be adopted within the scope of modern infrastructures.

To fill such a gap, this paper discusses the integration of QKD in a softwarised infrastructures scenario, a challenging perspective considering the new constraints and requirements (i.e., a dedicated quantum channel, single-photon sources, public authenticated classical channel). We started from the ETSI GS QKD specifications [2] keeping compliance with the ETSI standard in terms of software interfaces. We devoted a significant effort to address the issues of designing and implementing a complete software stack for efficient QKD integration in modern infrastructure. In particular, we introduced the abstraction of the QKD devices up to a centralized Key Server to serve as an interface for security applications.

This paper also contributes to a framework to simulate QKD networks, towards a scalable and flexible simulation platform for QKD that enhances the testing of QKD protocols. We present a practical implementation suggesting technologies, such as Docker[6] containers, that could easily fit current cloud-native infrastructures facilitating the adoption of this kind of key distribution systems.

Adopting the aforementioned technologies as building blocks for a distributed simulation platform for QKD, we show that this can be integrated with the software stack and thus can be easily available. This allows experimenting on the different quantum simulation technologies to verify if they could fit different use cases.

Setting aside the design and implementation of the current version of both software stack and simulator, we aim at laying the basis for the evolution of these two parts separately. Therefore, we mainly propose a framework that we can evolve: for the software stack in extending coordination and routing functionalities among QKD network nodes; for the simulator to simulate complex Quantum Networks beyond QKD in an efficient and distributed approach.

To summarise, the main contribution of this work is the design and implementation of this software stack that could easily be integrated with modern infrastructures and leverages a cloud-native approach for development. This software stack allows the integration, monitoring, and management of many QKD systems within the infrastructure regardless of the specific vendor and technology. In addition, we designed and implemented a QKD simulator that could serve both as a low-level testing device and as an independent software module to test QKD protocols. In the end, we analysed a testing scenario in which we reproduce the exchange among two different nodes using the aforementioned QKD simulator. In this scenario, we also tested the performance of our simulation platform and proposed possible enhancement.

We report in the following the remaining structure of the paper. Section II provides a background on the technologies that we adopted and widely used in this paper. Section III discusses the threat model of a softwarised infrastructure mainly according to secure communications. Section IV describes the high-level architecture of our solution. Section V presents the implementation of the QKD module. Section VI describes the Quantum Key Server. Section VII provides further details about our QKD simulator. Section VIII explains the results of our testing and validation process. Section IX presents some work related to Quantum Networks, QKD, and quantum technologies in softwarised infrastructures. Section X discusses the achievements presented in this paper and possible future work. Finally, the appendices contain a description of the BB84 and E91 QKD protocols (Appendix A), our framework's APIs (Appendix B) and illustrates our software stack's complete workflow showing further details on the interaction among different components (Appendix C).

II. BACKGROUND

To completely understand this work's essence, we provide an overview of QKD, softwarised infrastructures and ETSI QKD GS specifications.

A. QUANTUM KEY DISTRIBUTION (QKD)

Quantum communication and quantum networks are topics of utmost importance for the future of the Internet and classical communication. Several papers [3], [4] show how a Quantum Internet will work alongside the classical one over the foreseeable future. The main idea is to leverage quantum phenomena such as entanglement to design and implement algorithms and protocols that do not have classical equivalents.

Quantum Cryptography is a Quantum Communication branch and involves algorithms and protocols that could be applied, depending on the specific case, to protect either

[6]https://www.docker.com

quantum or classical communication. The most suitable example is QKD.

This technique leverages quantum mechanical phenomena to exchange keys among different parties with Information-Theoretic Security [5]. QKD strongly relies on a quantum channel that allows many peers to exchange qubits encoded generally as photons. Beyond the quantum channel, QKD protocols rely on a public authenticated classical channel to exchange out-of-band information necessary to coordinate peers during the exchange. Roughly speaking, there are two main classes of QKD systems: discrete-variable QKD (DV-QKD) and continuous-variable QKD (CV-QKD). In the first type, encoding and decoding are performed leveraging qubits or other quantum systems with finite-dimensional Hilbert space. For this reason, it is also called qubit-based QKD. In CV-QKD, instead, keys are encoded in quadratures of the quantised electromagnetic field and decoded by coherent detections [6]. This kind of detection is beneficial in modern QKD implementations because it is compatible with existing telecom equipment and shows high detection efficiencies without the requirement to provide cooling [1]. This paper discusses only a subset of DV-QKD systems for the simulator part since their implementation is more straightforward with our current framework. In the future, we aim to extend this work also to CV-QKD. Regarding the software stack, that part is agnostic to the class of QKD adopted; moreover, we see in that context the QKD as a "black box".

So far, many DV-QKD protocols, such as BB84, E91, SARG04, BBM92, and COW [5], have been proposed. We could classify them into distinct categories depending on the adopted scheme: prepare-and-measure, entanglement-based, and Measurement-device-independent QKD (MDI-QKD). The first scheme involves only two actors (Alice and Bob), one preparing the qubits sent to the second, which performs the final measurements. The other two schemes involve three modules: Alice, Bob, and Eve. Eve is a third-party module used as an untrusted entangled-pairs sender in entanglement-based protocols and an untrusted receiver in MDI-QKD.

Appendix A presents two protocols falling in the two first categories: BB84 (prepare-and-measure) and E91 (entanglement-based). In section VII, we show how we implemented both protocols in our QKD simulator. We chose these two protocols as they are the most employed for basic experiments on QKD and serve as representatives for two relevant QKD approaches. It is also worth mentioning that companies such as ID Quantique[7] and MagicQ[8] already commercialise devices that allow performing point-to-point QKD (e.g., Clavis[3]).

The security definition for the QKD is available in [7], [8]. This is a composable security definition that is rigorously stated and widely accepted. Several composable security proofs are available ([1], [5], [9], [10]) for both DV-QKD and CV-QKD.

In order to work properly and to avoid Man-in-the-Middle attacks, a QKD system requires an authenticated classical channel to carry out the protocols. We can leverage Information-Theoretic Security (ITS) authentication for this channel with 2-universal hash functions [11]. Moreover, we could adopt an efficient authentication scheme using ϵ-almost strongly universal hash functions. We could retrieve the final security parameter ϵ by the composability statement of the respective security proofs of QKD and ITS authentication [12]. This parameter shows that the key is ϵ-close to the ideal one.

In the rest of the paper, we also introduce alternative authentication mechanisms (sections VII and VIII). From a practical perspective, these methods are more flexible in a cloud environment scenario. From a security standpoint, this comes at a price of a possible lower level of security. Thus, while it is effective for simulation environments, before using this protocol ensemble in a production environment, their composition requires further studies and formal security proofs to avoid the possible introduction of any security flaws.

Besides the point-to-point QKD, where only two peers share a quantum channel, some scenarios involve an entire network, namely a QKD network. This ensemble of nodes could exchange keys with each other, regardless of their position within the network. Two main strategies could be adopted to manage endpoints in a long-distance scenario: using trusted or quantum repeaters. In the first case, the assumption is to have intermediate trusted nodes that leveraging point-to-point communications build a chain that starts from the source and ends with the destination endpoint. The second strategy involves the usage of quantum repeaters based on entanglement swapping. This drops the hypothesis of using trusted repeaters and allows secure end-to-end exchanges. In this case, the endpoints verify the presence of an eavesdropper without assumptions on the repeaters.

B. QISKIT

Qiskit[9] is a Python framework that allows to work with quantum computers in terms of quantum circuit simulations, algorithms, and pulses. This framework allows to both simulate quantum circuits locally or submit jobs to IBM Q backends[10]. The main Qiskit elements are:

- *Terra*: this is the basis for all the other Qiskit components. It provides a layer for developing quantum programs by means of quantum circuits or pulses, a module for circuit optimization also according to specific device constrains, and interfaces to both facilitate the end-user experience and to access to diverse backends for simulation or execution on real devices.

[7]https://www.idquantique.com
[8]https://www.magiqtech.com
[9]https://qiskit.org.
[10]https://www.ibm.com/quantum-computing/.

- *Aer*: it provides an high-level performance quantum circuit simulator which could be used to simulate the circuits compiled using Terra. This is useful to quickly test and verify the functionalities of the designed quantum circuits. It also contains configurable noise models to perform realistic noisy simulations of the errors occurring on real devices.
- *Ignis*: it is a module which aims at providing tools for better characterize errors and run circuits in presence of noise. It has been designed for working on quantum error correction codes.
- *Aqua*: this is a module which contains algorithms that could be used in different domains (e.g., AI, Chemistry, finance). This module allows to analyze the benefits of using quantum computing in the respective domains and build on top the proposed algorithms customized solutions.

This framework is mainly adopted for simulating quantum algorithms and circuits. In our case we could leverage a quantum circuit representation of the QKD phases for simulation purposes.

C. ETSI QKD GS

The ETSI has a specific working group for the standardisation of QKD technologies: ETSI GS QKD[11]. The related standards collect many specifications regarding hardware devices and software interfaces needed to implement QKD systems and networks. Here a list of the specifications of interest for this paper:

- **ETSI GS QKD 004**: this specification on "Application Interface" [13] defines a standard API to interact with QKD devices. Moreover, a preliminary high-level architectural view of a QKD network site is provided.
- **ETSI GS QKD 014**: the "Protocol and data format of REST-based key delivery API" [2] specification defines a standard REST API that could be used by high-level security applications to request keys to a Key Management Entity (KME). The latter is a software layer between the security applications and the QKD devices. The KME within a specific QKD network site manages key exchanges and coordinates with other KMEs within the same network.

D. SOFTWARISED INFRASTRUCTURES

The expression "softwarised infrastructures" indicates all modern infrastructures that strongly rely on virtualisation technologies and allow the adoption of paradigms such as Cloud, Edge, and fog computing, NFV, and IoT. Nowadays, many virtualisation technologies are adopted depending on the specific use case and the service model that has to be offered.

A recent trend in application development proposes to divide them into smaller components, namely *microservices*. These microservices could be deployed using lightweight virtualisation technologies such as Docker, Podman[12], or cri-o[13] that run them in isolated environments called *containers*. Containers and the related application components can communicate with each other using a TCP/IP network. We refer to the applications running on such environments as *cloud-native*. To deploy and manage containers, an orchestration container platform (e.g., Kubernetes[14]) is usually adopted.

Regardless of the virtualisation technology, applications running on infrastructures may require exchanging cryptographic keys for different security protocols. Simple examples could be Transport Layer Security (TLS) for building a secure channel among two applications or Internet Key Exchange (IKE) in the scope of Virtual Private Networks (VPNs). As they may rely on RSA and Diffie-Hellman (DH) for the key exchange, TLS and IKE could be compromised by the advent of Quantum Computing and, in particular, Shor's algorithm.

III. THREAT MODEL

Most public-key cryptographic (PKC) schemes are based on the assumption that some mathematical problems are computationally intractable for present computers (e.g. prime number factorisation in RSA). Quantum computers (QCs) break this assumption, making such problems "easy", i.e. reduce the mathematical problems to polynomial complexity (for example, RSA can be attacked by exploiting Shor's algorithm). Less groundbreaking, but still a significant risk-incrementor for symmetric-key cryptography, is the QCs capability to investigate a 2^n large solution space in $2^{(n/2)}$ steps.

A softwarised infrastructure is composed of many virtualisation actors and agile deployment components (e.g. hypervisors, orchestrator, containers) that interact in a complex manner and can be the target of attacks (both in terms of single components and the communication channels among them). A large amount of mitigations to those attacks is based on authentication and confidentiality countermeasures, often based directly or indirectly on PKC and public-key infrastructure (PKI). Assuming that QC is available, no softwarised infrastructure can longer rely on traditional PKC since private keys based on standard algorithms would become vulnerable and insecure. The core point of the QKD is to provide Information-Theoretic Security to the process of key distribution, allowing robust security even after the advent of practical Quantum Computers.

For the inner nature of QC, a QKD based on quantum cryptography is particularly prone to Denial-of-Service (DoS) attacks. If an exchanged key exhibits an error rate above a specific threshold over an ideal channel, this shows an eavesdropping attempt. A robust security strategy is to discard the key and re-do the exchange. Thus, for an

[11] https://www.etsi.org/committee/qkd

[12] https://podman.io

[13] https://cri-o.io

[14] https://kubernetes.io

attacker would be sufficient to "read" the exchange to deny the service. While some amount of DoS possibility is accepted in almost any real case (e.g. any local wireless communication can be drastically reduced by pretty cheap and easy to buy WiFi jammers), a robust architecture must address this aspect by appropriate countermeasures, e.g. the presence of Intrusion Detection and Prevention systems to identify and isolate the attack source.

Even if the adoption of QKD provides in principle Information-Theoretic Security, implementations can violate theoretical assumptions; thus, many developed QKD systems suffer security flaws. The architecture we propose is agnostic to the practical realisation of the Quantum devices. So, while we do not discuss real device problems, we advise that theoretical key strength must be supported by robust security implementations. In particular, a large body of literature exists to analyse real device weaknesses, as well as how to manage practical imperfections to remove physical side channels [5].

Apart from the core threat to the involved cryptography, a softwarised infrastructure still suffers from usual key management and use issues. Typical key weaknesses involve scarce entropy (the key is not long enough and/or has been generated by a poor RNG), inappropriate re-use, both in terms of use in multiple scenarios and encrypting a large amount of data (since it jeopardises forward secrecy and increases brute-force attacks and key-relevant information leakage likelihood), insecure storage (obviously on Hard disk but even in device RAM, since in the absence of countermeasures even central memory could be accessed with well know vulnerabilities like, for example, Heartbleed and Spectre).

We assume in our solution the presence of either a True Random Number Generator (TRNG) or a Quantum Random Number Generator (QRNG) able to provide random numbers with a high source of entropy. As in the previous point, a practical implementation of these modules may suffer security flaws, but this is out of the scope of our research.

Key re-use in our architecture is treated flexibly. Since we propose different options, according to the scenarios, an appropriate security policy should be implied, guaranteeing an appropriate lifetime for the derived keys, as it happens in nowadays organisations. OSs and applications may suffer from implementation flaws, but again this is out of the scope of our investigation.

Even if robust QKD is involved, a key might still be compromised (e.g. due to an implementation defect), so a procedure to destroy it (i.e., securely delete the key and its traces beyond recoverability) should be available. This last aspect can be challenging since it conflicts with the need of logging the key lifecycle to identify possible breaches and apply revoking and destruction options, as well as forensics investigations. The adoption of QKD for generating the master secret to derive cryptographic keys does not change this aspect, which is a matter of the adopted organisational policies.

Finally, to counter insecure access to the key store, Key access must be limited on a need-to-use basis, complemented by separation-of-role based access control (e.g., an entity that uses a key should not be the entity that stores that key). Key recoverability after accidental loss must be assured through secure backup and recovery solutions in place. Best practice suggests internal custody of keys (or service at a provable similar security level), managed by central key storage technology. This key aspect in our architecture is managed by the Quantum Key Server, a flexible component that can adopt different configuration to adhere to organisational security policies, such as log level and adopted QKD protocols, and is flexible enough to exploit different existing secure secret manager (like Vault, in the concrete testbed described in this article).

IV. QKD SOFTWARE STACK HIGH-LEVEL ARCHITECTURE

The main goal of our architecture is to facilitate the adoption of QKD in software infrastructures (e.g., cloud environments). Developing the software stack depicted in Figure 1 brings this purpose closer. In particular, the problem of exchanging keys among two cloud instances or services is mapped in four logical layers and components:

- **QKD device:** this represents the QKD physical device capable of running QKD protocols on a real or simulated quantum channel. Since we designed and implemented a simulator for QKD that could be integrated into our software stack, this paper also refers to the previous level as "QKD simulator".
- **QKD Module (QKDM):** this level is an abstraction of the low-level QKD device and it provides a standard interface to interact with different apparatuses. Regardless of the technology used to implement the QKD, additional functionalities are provided to monitor, manage and use the underlying devices.
- **Quantum Key Server (QKS):** at this level, the QKD management across different nodes takes place. In particular, coordination is required among different QKSs to establish the key exchange process among distributed infrastructures. QKS provides an interface to high-level security applications that require cryptographic keys and selects the best path to follow in scenarios where there are several nodes between the two endpoints of the key exchange.
- **Security Application Entity (SAE):** this is the higher level and represents the security applications willing to use the QKD for specific purposes (i.e., to set up a VPN).

This architectural framework is compliant with the ETSI specifications [2].

Let us consider a point-to-point QKD connection; two infrastructures may be equipped, as in Figure 1, with QKD devices that are able to exchange keys at a specific rate. Those devices have to share a quantum and a classical

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

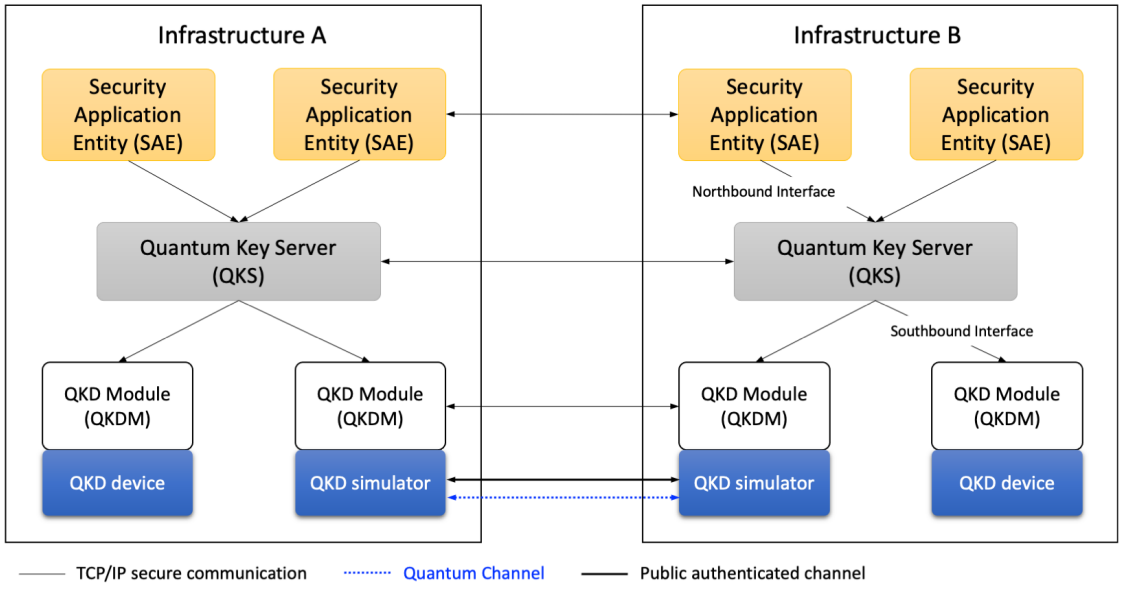

FIGURE 1. QKD software stack architecture.

authenticated channel to carry out a QKD protocol. In our architecture, the QKD device (or Simulator) identifies the physical system and the QKDM an abstraction for operating it.

If we switch to a more complex scenario, where a complete QKD network is in place, then we need to manage several aspects regarding the coordination of this network. Moreover, we must provide mechanisms to efficiently store the exchanged keys, routing functionalities to allow the exchange among each possible SAE combination, and monitoring mechanisms to log relevant events of the QKD-system life-cycle. Here comes into play the QKS. Each infrastructure shall expose at least one QKS, which represents the central management unit of the QKD systems.

SAEs running on top of the infrastructure can only see the QKS as a component, which from their perspective provides QKD-as-a-Service. They are able, in fact, to exchange quantum keys with all the other peers within the QKD network.

The underlying QKD network topology can vary based on low-level infrastructural choices. Two typical adopted schemes are switched QKD and trusted repeater networks [14]. In a switched QKD network, we typically find an SDN controller dynamically adjusting the paths among the endpoints (QKD devices). As an alternative, a network with trusted repeaters creates a chain of intermediate trusted nodes that must be traversed to perform the exchange. Regardless of the topology, the assumption is that every device leverages a quantum and a classical channel to carry out the QKD protocol.

If physical devices support more than one connection at a time (i.e., multiple quantum channels), then we could associate more than one QKDM to the same device. The basic idea is to have an abstraction where a QKDM pair identifies a single keystream over a quantum channel. Those keystreams and QKDMs can be efficiently operated by a QKS in charge of deciding when a key exchange has to start, stop, and keys must be retrieved to the higher-level applications.

We identified two primary interfaces in our architecture: the Northbound Interface and the Southbound Interface. These allow, in turn, SAEs-QKS and QKS-QKDMs communication.

In sections V, VI, and VII, we provide a more detailed description of the software stack's subcomponents. In Appendix C, we describe in details the interaction among them.

A. SCENARIOS

This section clarifies the potential applications of our software stack as well as the assumptions and limitations of the current solution. Our software stack is currently composed of a set of Docker containers deployable on a software infrastructure through Docker Compose. As a matter of fact, our software is a lightweight cloud-native application that could be deployed even on minimal infrastructures (i.e., edge-, fog- computing and IoT scenarios). This allows within that infrastructure to manage QKD as a cloud service, requiring keys on-demand (from a security application perspective).

Clearly, the end-user of this service is a SAE inside the infrastructure. Likely, a security application requires the keys derived from the QKD to set up a secure channel with another SAE in the same QKD network. This process may leverage protocols such as TLS and IKE [15]. These protocols must be adapted to be enhanced by QKD, and

our solution could simplify this process by providing a manageable way to access the keys. Even in this case, the arbitrary mixture of these protocols with QKD must be carefully handled from a security point of view (section II).

An even more suitable use case is integrating our software stack in Infrastructure-as-a-Service (IaaS) platforms such as OpenStack[15] or container orchestrator such as Kubernetes. As in the OpenStack case, this type of platform already manages secrets, keys, and certificates leveraging some of its specific services (e.g., Barbican[16]). In addition, utilities such as VPN-as-a-Service are available in the scope of OpenStack to create site-to-site VPN with other data centres. These utilities are integrated with Barbican to simplify the management of the cryptographic material required by the protocols (e.g., IKE).

A reasonable idea could be to integrate our software stack in the scope of these platforms and provide a way for those utilities to leverage QKD-exchanged keys directly. This kind of integration is feasible and straightforward due to the simplicity and modularity of our architecture.

The assumptions and the limitations of the current software, such as the available authentication mechanisms, are described in the following Sections V, VI, and VII.

In the end, our solution allows for various integrations and usages regardless of the size of the infrastructure and the specific context as long as we are in the scope of software infrastructures.

V. QKD MODULE

Along with this section we start describing the high-level software stack for the QKD integration in a softwarised infrastructure. The QKD module is an abstraction within an infrastructure aiming at providing a standard communication method with quantum devices. As we will see in section VII, the QKD node is the entity that maps within our solution the physical device (or simulator) that enables the QKD low-level exchange (e.g., ID Quantique Clavis[3]). In our paper, we use the QKD simulator to provide a complete solution without relying on physical devices. The role of the QKD Module is to act as a wrapper for the QKD node, exposing the standard API compliant to the ETSI QKD standard [13], and providing high-level functionalities to the upper layers such as retrieving a key exchanged with low-level devices. In our solution, the particular implementation and the vendor-specific devices adopted do not change the interaction between the QKS and the QKDM. In the following, we have a description of the APIs that shall be exposed, and the methods that have to be implemented/overridden to provides the functionalities expected from the quantum devices. We implemented a template of QKDM according to this strategy and provided a GitHub repository[17] that could be forked to implement a new QKDM for a specific device. The idea is that each device or simulator has its own QKDM, but the effort to implement it consists of overriding a few methods that we describe in the following. QKDM has been designed following modern cloud-native paradigms for developing applications and the Docker container technology. This simplifies the integration of QKDM component within the scope of cloud-native infrastructures (e.g., Kubernetes). A complete description of the QKDM APIs is available in appendix B-A.

A. ARCHITECTURE

As depicted in Figure 2, the QKDM architecture is composed of four different submodules: QKD node, *Key manager*, *Sync interface*, and *Southbound interface*. In this architecture, we also integrated the quantum device simulation part, which in Figure 1 is indicated as the QKD simulator. This provides a complete understanding of how the QKDM could interact with the underlying components.

The QKD node represents the quantum device. In this paper, this component is precisely the one described in section VII and includes a core component for QKD protocol simulation as well as a REST API interface to communicate with the other QKD nodes during the exchange (i.e., for the key sifting process). In a real use case scenario, the communication among devices and the implementation of the protocols strictly depend on the adopted technology. In that case, we could treat the QKD node as a black box with a provided interface.

The core component of the QKDM is the Key Manager, which is in charge of mapping the requests coming from the Southbound Interface to the instructions for the QKD node. It is also in charge of managing keys coming from the exchange process, typically relying on resources provided by the QKS (e.g., Vault in section VI).

The Southbound Interface is in charge of providing the communication among QKS and QKDM. This interface exposes the REST APIs proposed by the ETSI GS QKD 004 [13], especially the first three in table 3.

In the end, the Sync Interface is a set of REST APIs for the synchronization among different QKDM during the key exchange.

VI. QUANTUM KEY SERVER

As described in section V, QKDM is an independent component capable of managing the process of key exchange within a node. In principle, an infrastructural node only requires this module for integrating point-to-point QKD. Indeed, in our solution and besides the exchange process, the QKDM can store the collected keys in a specific secret engine provided by Hashicorp Vault. The only requirement is for a SAE to have access to this engine and get a new key.

Unfortunately, this might not work in complex infrastructures where different SAEs requires sharing QKD keys exchanged with one or more different destinations. Because of this, a software layer is required between SAEs and

[15]https://www.openstack.org
[16]https://wiki.openstack.org/wiki/Barbican
[17]https://github.com/ignaziopedone/qkd-module.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

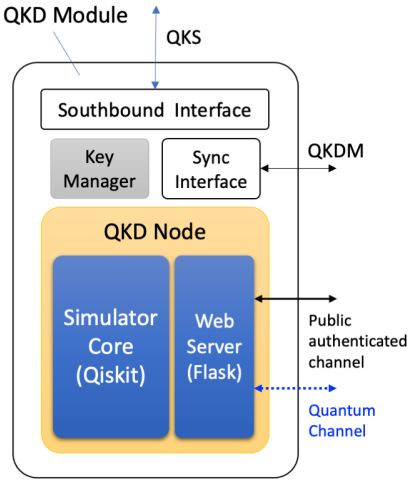

FIGURE 2. QKD module architecture.

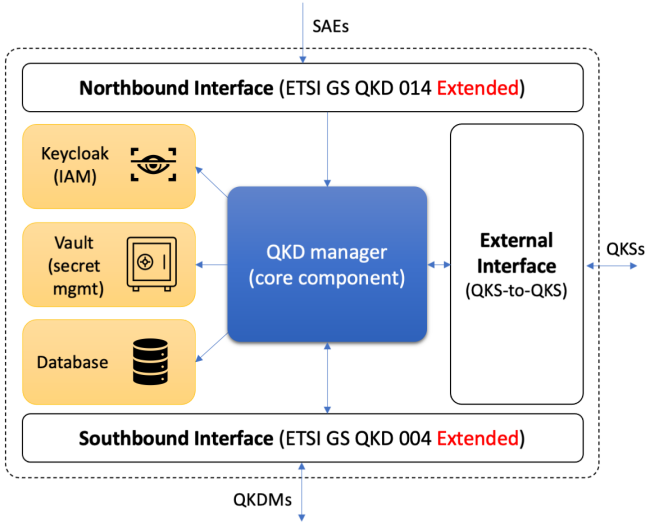

FIGURE 3. Quantum Key Server architecture.

QKDMs allowing to centralise the management of the exchange process, the key storage, the destination discovery, and the evaluation of the shortest path to reach them. This module is the QKS.

ETSI [2] focuses on the final security applications and foresees a strict binding among two SAEs belonging on two different nodes. This means that once a key exchange is required and communication among two QKDM takes place, a key stream is created and only the two involved SAEs could access it to get keys. The role of the Key Server is to facilitate this binding and managing the start and stop of the key exchange process.

Considering the complexity of softwarised infrastructures and their requirements in term of flexibility, our solution adopts a different approach. Even if we maintain most of the logic regarding the interactions among the entities, the interfaces and part of the data model, we change to some extent the role of the Key Server and design our QKS.

Our QKS acts similarly to a SAE in the sense that it is the one in charge of deciding whether or not a specific stream should be created and is the conceptual final "endpoint" of all key exchanges towards the node, albeit it is still in charge of the management of the exchange and the coordination with the other QKSs. In our solution, the QKS is a sort of middleware, which collects all the keys coming from all the destinations and provides them to all the registered SAEs, regardless of the binding that could happen between two specific SAEs. This allows to continuously exchange keys among QKDM point-to-point connections over the QKD networks and potentially use those keys for each SAE-to-SAE pairs over the same physical link. This is a relevant advantage for a virtualised scenario in which keys could be easily delivered among different virtual instances.

QKS has also been designed as a cloud-native application, which could be efficiently integrated into a modern infrastructure scenario. The interaction with the QKDM is pivotal. Indeed, QKS could support different QKD modules that shall register to it before starting any task. It is also the case of the SAEs that must be authenticated and authorised before accessing the key server and request any key. Further details on the interaction among all the components are in appendix C. In the following, we describe the architecture of the QKS.

A. ARCHITECTURE

As depicted in Figure 3, the architecture of QKS is composed of three interfaces and four main components as follows:

- **Northbound interface:** this is the interface towards the SAEs and provides these with an extended version of the ETSI API to query the QKS. This is mainly involved in the process of requesting keys from the SAEs.
- **Southbound interface:** this is the interface that allows the communication between the QKS and the QKDM. As reported in section V, this interface is mainly implemented on QKDM side. Nevertheless, this interface is bi-directional and also QKS exposes some calls to serve the QKDM.
- **External interface:** this interface serves as a synchronisation interface among QKSs. It is pivotal for sharing information such as the KSID and routing information.
- **QKD manager:** this is the core component in charge of providing all the core functionalities such as serving Northbound Interface requests or registering new QKDMs. This is also involved in the process of key aggregation to serve multiple keys of an arbitrary length to the upper layer.
- **Keycloak**[18] **(IAM):** this is an additional component that is useful in scenarios where the number of SAEs

[18]https://www.keycloak.org

and QKDM is large. Keycloak provides IAM functionalities that allow to authenticate and authorize SAEs and QKDMs. This is useful for both Northbound and Southbound Interfaces and could be easily extended for the External Interface as well.

- **Vault**[19] **(secret mgmt):** this is a secret management tool by Hashicorp which provides a flexible and scalable way for managing secrets. This tool allows the creation of separated secret engines (one for each QKDM), where the keys could be stored.
- **Database:** this component is used to store information regarding the whole QKD exchange process, the registered QKDMs, and the QKD protocols supported.

For the authentication and the authorisation of the Southbound and Northbound interfaces, we use Keycloak and *OpenID connect*. For the interaction among different QKSs we have not implemented those mechanisms yet. We could easily extend the Keycloak solution also in the case of the External Interface. The point is that, in a "quantum scenario", the communication over a secure channel within a QKD network should be protected with a quantum-resistant set of algorithms and protocols. Thus it is not just a matter of authentication and authorisation, but we aim at building quantum-resistant secure channels among them. We explored different possibilities that we would like to integrate with our solution. The first that we considered is to integrate post-quantum algorithms, such as SPHINCS$^+$ and NTRU, to build a secure TLS channel among QKSs. In addition, the authorisation part could still be performed using Keycloak. Another solution that we have lies in adopting TLS-PSK, which could be used in combination with the QKD keys already generated at the lower level. This envisions the QKS acting as a special SAE and able to require specific keys for the communication with other QKSs.

Even the communication within the infrastructure, among the different layers (i.e., QKS vs QKDM) has to be protected. According to ETSI GS QKD 014 [2] that communication could rely on the classical TLS v1.2 as a minimum requirement. We disagree on this, envisioning that in quantum scenario also the communication within the infrastructure should be protected with quantum-resistant approaches. In addition, it is not mandatory that the QKS has to be deployed on the same physical node as the quantum device, and this leaves room for other attack strategies. Our idea is to integrate within the infrastructure a secure communication mechanism based on post-quantum cryptography in order to completely overcome the aforementioned issue.

In Appendix C, we clarify other aspects regarding the interaction among all the components of our solution. The QKS's code is available on GitHub[20]. This implementation is completely based on a cloud-native approach and leverages Docker technologies. Each subcomponent has been modelled as a separate Docker container. To provide an effortless deployment, we additionally defined a Docker Compose[21] descriptor. In the end, we also describe the QKS APIs in appendix B-B.

VII. QKD SIMULATOR

To consistently test our solution and provide a different approach to the simulation of QKD protocols, we designed and implemented the QKD simulator described in the following. The current version of this software is at its early stages and provides the minimum functionalities for testing two QKD protocols: BB84 and E91. Even though the low-level simulation part has to be enhanced, our design principles allow to extend and scale our simulator providing all the required capabilities to become a full-fledged QKD simulator. In principle, it could also be extended to simulate protocols beyond QKD in the broader field of Quantum Networks.

The high-level QKD simulator architecture is depicted in Figure 4. All components have been designed as independent Docker containers. This allows them to run in lightweight virtual instances and communicate over classical TCP/IP networks with the others. Using Docker technology as *Container Runtime Interface (CRI)* provides an effortless way to scale the simulation across different infrastructural nodes. Thinking of this solution as being deployed on a Kubernetes cluster quickly arranges a way to test complex QKD network scenarios.

The main QKD simulator components are:

- **QKD node**: this is the component that models the QKD device. In the QKD scope, this is one of the parties involved in the key exchange (e.g., Alice, Bob). The QKD node is equipped with all the software capable of simulating the qubit encoding as well as the operation on the quantum states. It provides mechanisms to serialise qubit-related data and communicate with the other components over a TCP/IP network.
- **Quantum channel and Eve**: this entity is a particular QKD node, which includes the same underlying software for the quantum simulation, but it is designed to reproduce the specific effects of certain entities acting on a quantum channel (e.g., noise, eavesdroppers). It acts on the qubits encoded by QKD nodes leveraging the specifically chosen representation.
- **Entanglement pairs generator**: the latter provides pairs of qubits that are maximally entangled and still compliant to the chosen representation. This is useful within the scope of entanglement-based protocols.
- **QKD Simulator Manager**: this manager implements a central management unit that collects data about the simulation and gives a handy interface for triggering protocol simulations or configuring QKD nodes. Both command-line and graphic interfaces have been provided, even though they are at their early stages.

[19]https://www.vaultproject.io

[20]https://github.com/ignaziopedone/qkd-keyserver

[21]https://github.com/docker/compose

To replicate quantum phenomena and develop the underlying software to simulate qubit encoding, manipulation and measurement, we use Qiskit (section II). The basic idea is to leverage *State Vectors (SVs)* for the qubit encoding process and *Quantum Circuits (QCs)* to simulate the evolution of these SVs following the interaction among Alice, Bob, and Eve. In our scope, the first two act as parties involved in the key exchange and the last one acts as an eavesdropper on the quantum channel. Regardless of the specifically selected protocol, we discuss some aspects of our simulation platform. Afterwards, we present the implementation of both BB84 and E91 protocols.

The first interesting common aspect concerns the qubit encoding. In particular, we used the `Statevector` class to create an ensemble of qubits. This class allows to initialise the qubit vector with specific values (e.g., $|00000\rangle$), provides a method to evolve its state according to a supplied quantum circuit, and a method to measure the final state. These are three macro operations required to simulate all we need for QKD protocols.

Another crucial feature is each component's ability to communicate over a TCP/IP network for exchanging quantum and classical data. The idea is to serialise SVs objects and send them over a classical network, using HTTP at the application level. For this purpose, the `pickle` Python module for object serialisation has been used. Once an SV object has been serialised, we could embed it into the body of an HTTP request and send it to another entity within a QKD network. This process applies to all the communication that require quantum bits exchange within our simulated network. In our solution, we leverage Flask[22] web servers to expose REST-based API that could be used to communicate among peers. The use of HTTP and Flask web service is not mandatory and could be easily changed in the future, making room for a custom application-level protocol.

As depicted in Figure 4, all the communication among parties are mediated - see Alice and Bob - by an entity representing both the quantum channel and a possible attacker. This is because, directly on a TCP/IP network, we can neither simulate the quantum effects acting on transmission nor manipulate the qubits as an attacker over a quantum channel. Because of this, we proposed a new entity - Quantum Channel and Eve - that could apply those effects on quantum bits. Undoubtedly, this strategy could be extended by adding different intermediate nodes between Alice and Bob and make them affecting the qubits arbitrarily. The only capability that both QKD nodes and these "special nodes" have to share is a standard communication method leveraging an identical qubit representation.

In Section II, we claim that, regardless of the protocol that we adopt, we do need a public classical channel to exchange additional information during the QKD process. This channel has to be authenticated, and the technique that we use to perform this task should be flexible and scalable to fit real distributed use case scenarios. We proposed and implemented two different strategies: using SPHINCS$^+$, a post-quantum algorithm, or AES with Galois/Counter Mode (AES-GCM) authenticated encryption. The first solution requires signing the messages exchanged over the public channel. This is a highly flexible approach since it does not require pre-shared secrets among Alice and Bob, but only to know the public key of the counterpart. Some efforts are going towards a new "Post-Quantum PKI" which involves post-quantum X.509 certificates, giving a scalable solution to the authentication problem. The downsides are that it still relies on computational assumptions and post-quantum algorithms - including SPHINCS$^+$ - are not standardised yet. Fortunately, the authentication has to be granted only during the QKD exchange process, meaning that no sensitive information about the key could be extracted from data stolen from the public channel, and no attack could be performed using that information aftermath.

The second option involves the usage of an encrypted and authenticated channel through AES-GCM. This could be done using a pre-shared key between the parties that could be rotated using a portion of the key exchanged during the QKD process. Under the assumption of adopting a certain key length, this solution is also quantum-resistant, but it needs initial pre-shared secrets and a priori knowledge of those secrets among parties. In addition, this solution is vulnerable to *DoS attacks*. Indeed, if an attacker makes the exchange process continuously fail, the parties will consume all the pre-shared key material, leading to a denial of service. This could be mitigated by a fallback strategy in which, when that happens, instead of using a pre-shared key, a new secret could be obtained by using public-key cryptography.

The QKD simulator manager is in charge of monitoring and managing the key exchange process. Exposing both a CLI and a GUI, this module is able to trigger the exchange process between parties, set the parameters of this exchange (e.g., protocol, key length, eavesdropper presence) and show the simulation results. In principle, this component could be used as an orchestrator for the whole reproduced QKD network. This means it could potentially add new nodes and configure them. The current version is still at an early stage and only allows to manage Alice and Bob's exchange with the two supported protocol. Nevertheless, considering the adopted technologies, the simulator manager's improvement towards those features is not too ambitious.

A QRNG simulation module has been included within the QKD node. This allows reproducing the generation of random bits for the key to be exchanged. The module has a straightforward implementation consisting of a circuit acting on a quantum register that is initialised to $|0\rangle$. Applying a set of Hadamard gates to this register ($H^{\otimes n}$), where n is the number of qubits, the evolution of the circuit results in n qubits in the state $(|0\rangle + |1\rangle)/\sqrt{2} = |+\rangle$. If we measure all these qubits in the computational basis, we get either 0 or 1 with a 50% probability: we obtained the QRNG. Since

[22]https://flask.palletsprojects.com/en/2.0.x/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

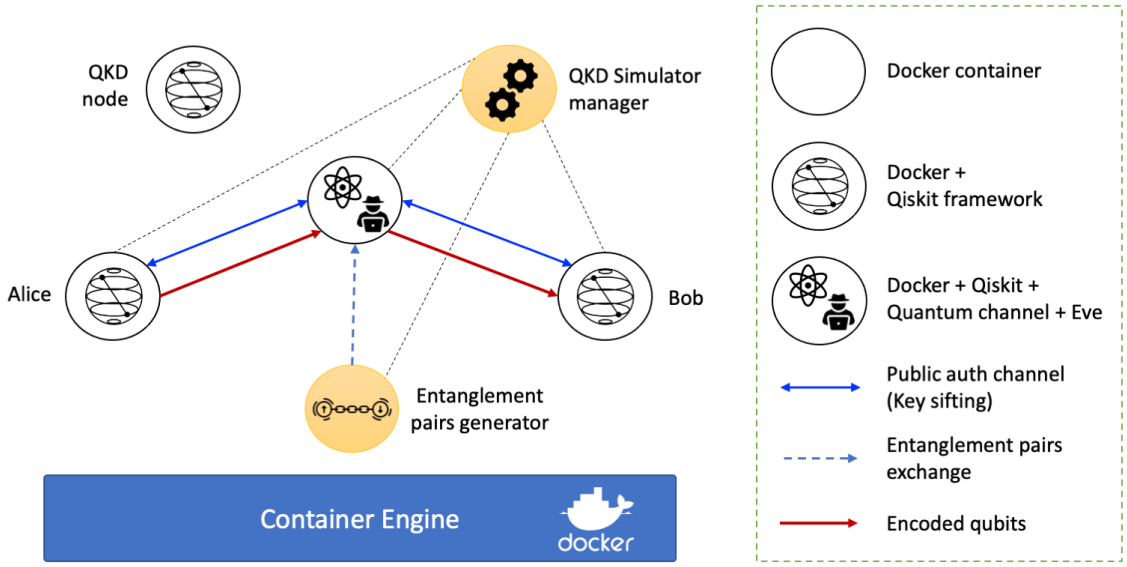

FIGURE 4. QKD simulator high-level architecture.

this module consistently affects the whole system in terms of performance, we also foresaw an option that allows using a classical pseudo-random number generator.

The last global consideration is according to the extension of our simulator to support other protocols. The current version of our software provides the abstract `QKD` class with the following methods that could be overridden:

- `begin`: this initialise the connection among two QKD nodes;
- `exchangeKey`: this starts the key exchange;
- `end`: this closes the connection among the nodes.

Maintaining all the features regarding the communication and the underlying Qiskit framework, it is possible to create a new version of our simulator that support other protocols simply overriding those methods.

Moreover, we are not even tied to the framework that we use for the simulation. Indeed, we could also extend the QKD node to support various frameworks, for instance, using the ones cited in section IX. Afterwards, depending on the protocol and the simulation requirements, it is possible to leverage the most suitable libraries for the specific implementation. For the sake of simplicity, we provided a Docker image on Docker Hub[23] with all the current underlying software required by the QKD node. This image could be used to deploy several QKD nodes and run the code related to the specific protocol simulation. Extending the simulator adopting other frameworks could be achieved by extending this Docker image. On a side note, it is reasonable to leverage diverse orchestration platforms for deploying that image and creating an arbitrary simulated QKD network. For this specific purpose, we automated our solution's deployment using Docker Compose. The QKD simulator's code and documentation are available on GitHub [24].

A. BB84 IMPLEMENTATION

In appendix A, we give a general description of the BB84 protocol. Now we discuss how it is possible to implement it with Qiskit. We assume that the architecture is the one depicted in Figure 4 and Alice and Bob are the parties wishing to perform the QKD. To simulate both quantum channel and Eve's attack, all traffic has to pass through the container that applies those effects. Each container has a single interface on the overlay network provided by the Container Runtime Interface. The two types of traffic - for both public and quantum channel - reach this interface when two nodes need to communicate.

Starting from these assumptions, a key exchange could start when one of the parties (assume Alice) requires it. The first phase requires that Alice generates two sequences of random numbers: one corresponding to the bits of the key to be exchanged, the other one to the sequence of bases in which we could measure those bits. In order to encode those bits in quantum bits, we need to prepare an array of Statevectors in the correct bases. A first consideration is that the number of bits required for a key (e.g., 256, 4096) is consistently larger than the number of qubits manageable in a Qiskit register. Because of this, we implemented a mechanism to manage an arbitrary number of bits iterating the quantum operations required in the process according to the size of the qubit register available in Qiskit. Moreover, we tested different register sizes (section VIII), and we chose the one that better fit according to the performance. Nevertheless, it is possible to change this dimension in the

[23] on hub.docker.com: ignaziopedone/qkd:simulator-1.2

[24] https://github.com/ignaziopedone/qkd-sim.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

simulator settings. Assuming that we are using a 5-qubit register and the key is of the same size, the BB84 scenario could be described by Figure 5.

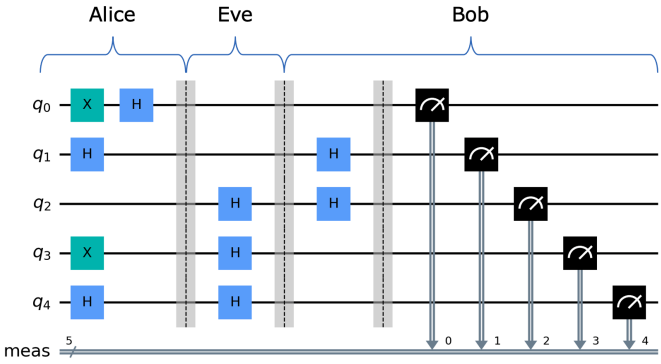

FIGURE 5. BB84 quantum circuit with 5-qubit register.

In practice, we start with a quantum register initialised to $|00000\rangle$ in the computational basis (z-basis). We could use as a basis either the computational basis or the Hadamard basis (x-basis). These bases are conjugate; namely, if we encode a qubit in z-basis and then perform a measurement in x-basis, we get a 50% chance to have either 0 or 1 and vice versa. According to BB84, we need four different states. Since all the qubits within a register are initialised to 0, if a 1 is required than the corresponding bit should be flipped. To perform this operation, we could use an X gate where we need a 1. Afterwards, we need to decide if we want to encode employing the computational or Hadamard basis. By default, we work with the computational basis, so to change this behaviour, we need to apply a Hadamard gate (H gate) where the change of basis has to occur. The result of this process (performed by Alice) is shown in Figure 5 according to the first four qubits:

- q_0 encodes a 1 in x-basis;
- q_1 encodes a 0 in x-basis;
- q_2 encodes a 0 in z-basis;
- q_3 encodes a 1 in z-basis.

Once the qubits have been correctly encoded in the chosen bases, Alice sends the corresponding SVs to Bob. This is possible by means of serialisation process. Afterwards, Bob could choose which bases he will use to measure the qubits and perform this action. As you may notice, the choice regarding the bases depends on whether we apply the Hadamard gate. So if both Alice and Bob make the same choice, they will have a consistent qubit, otherwise, there will be an error with a probability of $1/4$. This is exactly the principle that allows the detection of an eavesdropper (Eve). In our case, we introduce Eve, and the presence of a Hadamard gate indicates that she has chosen a wrong basis. There are different approaches to simulate BB84. One of them is to apply the basis as in the Alice and Bob case and measure the resulting state. Then it is possible to proceed with Bob's measurements. In our solution, we assume that when Eve chooses the same basis as Alice, we do not apply the H gate, otherwise, we do. This allows having the same result for BB84.

After the qubit exchange process, Alice and Bob start the Key Sifting process. They share their choices in terms of bases and exclude from the key the qubits where the basis does not match. In practice, according to Figure 5 only q_1 and q_3 are left. In the first case, Eve chooses the same basis, so no error is expected from Bob's side. This means that we can not detect Eve. In the second case, Eve chooses a wrong basis and then we have a 50% chance to detect her action.

The overall process could be repeated using the same register until we get all the qubits we need for the key. In section VIII, we discuss the intercept-resend attack and we show the results of its simulation.

The following steps involve the QBER estimation. This process could be done right after Alice and Bob publish a subset of bits within the key (in our case, half of them). QBER is the metric that allows us to establish Eve's presence. In literature [16], the classical threshold is the value of 11%. In practice, it depends on the key distillation process: for instance, adopting *Advantage Distillation* techniques enhances the performance up to 20%, as reported in [17]. In our case, we do not take into account non-idealities and quantum channel noise yet. Thus, the only error introduced is by Eve. If we assume that all the qubits of the key have been attacked independently of each other, then QBER should be around 25%. This threshold has to be calculated according to the PA technique adopted. In a real scenario, we have both quantum channel noise and Eve, so as we discuss in appendix A, we need error correction code to correct the bits flipped due to the noisy channel and also a technique to reduce the quantity of information gained by Eve. We have not implemented error correction and privacy amplification yet, but we are working on the integration in our prototype of the Cascade protocol [18] for error correction and PA techniques based on Toeplitz universal hash functions [19]. The general workflow already includes these two steps that have to be executed right after the QBER estimation.

B. E91 IMPLEMENTATION

The implementation of the E91 protocol is also at its early stage. We took inspiration from Qiskit community[25] to implement this one. According to the architecture, we provided another module to perform the entanglement pair generation. As depicted in Figure 6, we obtain a singlet state (maximally entangled) initialising q_0 and q_1 to $|1\rangle$, applying a Hadamard gate to the first qubit, and employing a *controlled NOT gate (C-NOT)* controlled by the first qubit and applied to the second one. This gives us the state described by the Equation 1. After this phase, the representation of the

[25]https://github.com/qiskit-community/qiskit-community-tutorials/tree/master/awards/teach_me_qiskit_2018/e91_qkd

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

entangled pair is sent to the quantum channel. Here we perform all the quantum operations without sending any qubits to Alice and Bob. At this stage, we used this strategy to simplify the entanglement simulation.

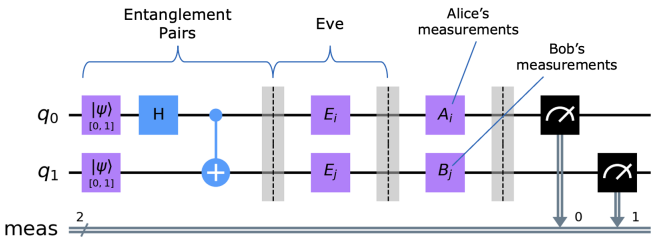

FIGURE 6. E91 quantum circuit.

For next releases we are working on a more effective solution. When the entangled pair representation reach the quantum channel, the latter performs the measurement according random choices provided by Alice and Bob nodes. The measurement choices for E91 are described in appendix A. Alice could choose among three different choices (A_j) and the same goes for Bob (B_j). In Figure 7 we have the implementation of these measurements as a circuit. In particular:

- Alice: q_0 ($\phi_3^a = 90°$), q_1 ($\phi_1^a = 0°$), q_2 ($\phi_2^a = 45°$);
- Bob: q_0 ($\phi_2^b = 90°$), q_2 ($\phi_1^b = 45°$), q_3 ($\phi_3^b = 135°$).

Eve could perform the exact measurements on both Alice and Bob's branches. In Figure 6 we did not add the measurement part regarding Eve, but it is implied.

After the measurement part, we divide the bits into two groups as described in appendix A: one for the key and one for the CHSH inequality verification. We calculate the value described in Equation 4, using also Equation 3 and 2. If the value is close to the correlation (anti-correlation) expected, then we keep the key, otherwise, we drop it.

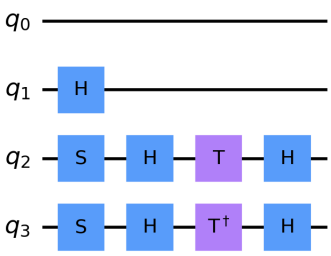

FIGURE 7. E91 Alice and Bob's possible measurements.

VIII. TESTING

For testing purposes, we developed a testbed following the schema depicted in Figure 1. Here are the details of the environment:

- two bare-metal nodes equipped with an Intel Core i5-5300U CPU @ 2.30 GHz, 16GB of RAM, and an Ubuntu 20.04 LTS server Linux distribution;
- Docker Engine CE v20.10.1 - API v1.41, Docker Compose v1.27.4, and our software stack integrated with the QKD node component of the simulator (all installed on both nodes).

For functional testing, we performed with success the following operations:

1) plug-and-play installation of the whole environment with Docker Compose;
2) registration of the QKD nodes to the respective QKSs;
3) start the key exchange process among the two nodes (through the QKSs);
4) installation of two SAEs - one on each node - and testing of `get_key` and `get_key_with_id` methods for key of different lengths (e.g., 256, 1024, 4096 bits).
5) repeat tests (2) and (3) with different protocols (e.g., BB84, E91) and public channel protections (e.g., SPHINCS$^+$, AES-GCM)

No major flaws have been identified during the functional testing.

As quantitative tests, we investigated the performance of the QKD simulator. We relied on two typical metrics for a QKD system: QBER and throughput (bit-rate) of the exchanged keys.

The first test in Figure 8 analyses the differences in using various quantum register lengths in Qiskit. In section VII, we show how quantum registers are used to support the simulation of the QKD and the possibility to set an arbitrary length in our solution. The test shows that, varying the number of qubits from 1 to 10, the performances are nearly the same with a pick when we choose a 5-qubit register. Because of this, we adopted this length for the other tests. Nevertheless, there is no significant enhancement in picking a specific length value. Quite the opposite the case of a length greater than 10. Indeed, in this case, the performances collapse due to the limit of qubits that we could simulate on our host. IBM Q[26] allows to simulate on their servers up to 32 qubits and also to execute the job on actual quantum devices up to 65 physical qubits and a quantum volume of 128. On our machine, as in the case of a regular laptop, we barely reach a 12 qubits simulation. In this case, the key exchanged was composed of 4096 bits.

Using the same number of qubits within a quantum register, we then tested, varying the length of the key, the time needed for a complete key exchange. We analysed the following four scenarios:

- using the QRNG and SPHINCS$^+$;
- using only SPHINCS$^+$;
- using only AES-GCM;
- using no protection on public channel and no QRNG (baseline).

[26]See https://www.ibm.com/quantum-computing/.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

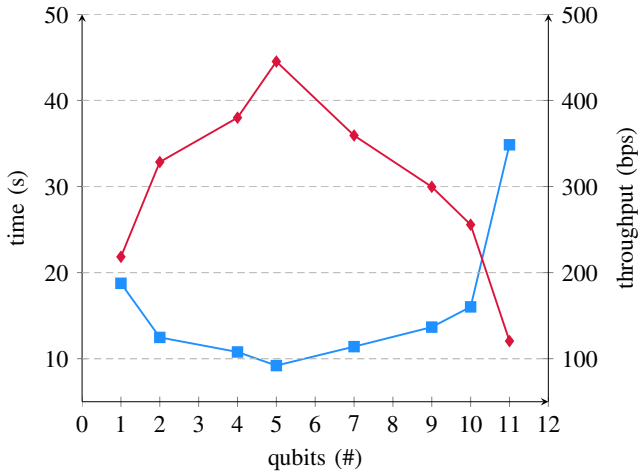

FIGURE 8. Time for a 4096 bits key exchange depending on the qubit register length.

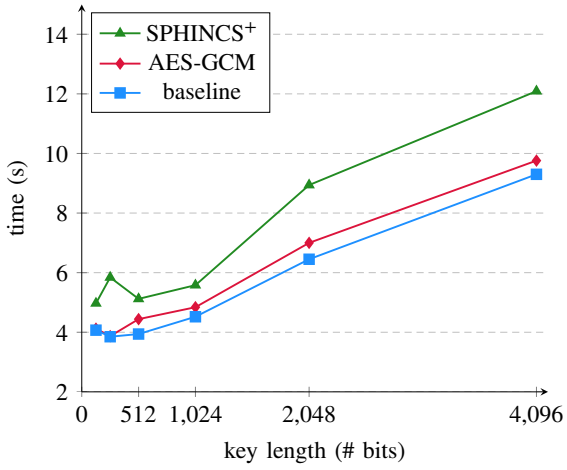

FIGURE 9. BB84 key exchange time depending on the key length.

TABLE 1. BB84 key exchange rate and time

SCENARIO	BIT-RATE (BPS)	EXCHANGE TIME (S)
Baseline	440.43	9.30
AES-GCM only	419.67	9.76
SPHINCS$^+$ only	338.79	12.09
QRNG + SPHINCS$^+$	24.75	82.78

As we depicted in Figure 10, the global trend is for the bit-rate to grow as the key length increases. This stops when we reach the maximum in terms of throughput, according to our system. In our case, the maxima with respect to the different scenarios are provided in Table 1.

The first three maxima were reached at the key length of 4096 bits. Going further and increasing the key length gave no more growth in terms of bit rate. The last maximum was reached at 2048 bits, even though the saturation started around 512 bits.

Further consideration on Figure 10 are related to the comparison among the different case scenario. Using a QRNG simulation is resource consuming, and the impact on the performance is clear looking at the blue line with the squared marker in Figure 10. For this reason, we do not proceed in showing further tests on this case scenario.

The other three lines are comparable. As we can see, the black line (with the circular marker) is the baseline. In that case, we got rid of all the overheads introduced by the authentication process and the QRNG. Adding to this baseline, the encryption and the authentication of the public channel with AES-GCM we have no consistent overhead. The red line with the diamond marker is really close to the black one. This is clear starting from a key length of 1024; indeed, until that threshold, the variance of the time needed for the quantum operations is mixed up with the other variables. In the case we only use SPHINCS$^+$ (green line with the triangular marker) instead, as we expected, it introduces a significant overhead compared to AES-GCM. Indeed, this post-quantum algorithm according to the variant we used (SHAKE256-128f) introduces an average overhead for signing and verifying in turn of 275 ms and 11 ms.

All the values have been evaluated as average over 100 iterations of a key exchange of a specific size. We also provided a vision of the same graph according to the time instead of the throughput in Figure 9. In this case, we left out the case of the QRNG, for the reasons we mentioned before. It is worth mentioning that most of the values depicted in Table 1 are comparable with real devices currently available on the market. In particular, if we refer to ID Qauntique Clavis[27] we could notice that the secret key rate is 1.4 kbps, which is close to the throughput of our simulator.

In the end, we also implemented an intercept-resend attack scenario on the BB84 protocol. According to Figure 4, all the traffic (related to Alice and Bob's communication) pass through Eve's node, which is capable of measuring an arbitrary quantity of qubits before they reach Bob's node. Once Eve measures a quantum bit, he could introduce (section VII) an error related to that bit with a probability of $1/4$. This happens because he has to choose random bases, as in the Alice and Bob's cases.

Moreover, this type of attacks could be more sophisticated than that in [5]. There are three types of attacks in theory: *individual*, *collective*, and *coherent*. The intercept-resend is an individual attack, which is the simplest type where each qubit is attacked independently. The other types are more sophisticated and involve the preparation of ancilla qubits for the interaction with the target qubits either to perform a collective measurement on them or to entangle the ancillas and then perform the measurements. These processes allow Eve to retrieve more information than the previous case. Today's technologies do not allow to perform collective and coherent attacks since they require quantum memories.

[27]https://www.idquantique.com/quantum-safe-security/products/clavis3-qkd-platform-rd/

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

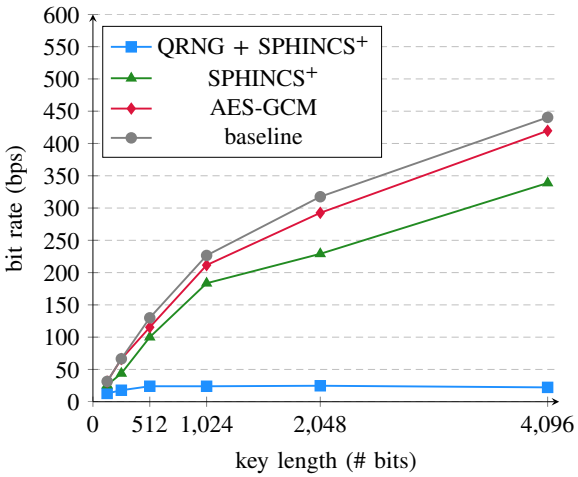

FIGURE 10. BB84 key exchange bit rate depending on the key length.

Returning to the intercept-resend attack, an attacker could decide to perform partial measurements as well as to measure all bits at once instead of doing that independently. We explored the primary case in which Eve either measure all qubits or a subset of them but always independently to each other.

In a real case scenario, the error introduced by the attacker could be tolerable, since with some PA techniques, we could reach the threshold of 11% for the QBER until we need to drop the key. This allows to reach better performances in term of bit-rate, and works well on very noisy quantum channels.

Setting aside the importance of the QBER, in Figure 11 we analyse the variation of the QBER as a function of the number of the attacked qubits. This operation has been performed for an exchange of an 8192-bit key. In particular, we consider our ideal scenario in which there is no noise over the quantum channel, and Eve attack an arbitrary number of qubits. As the number of qubits attacked (measured) by Eve increases, the QBER grows. We plotted all the data that we retrieved from our experiment and calculated the regression line, which shows accordingly to the theory that the QBER linearly grows. More specifically, if we zoom on the frame within the picture, we could see that the more we get closer to attack all the qubits, the closer we get to a 25% QBER value. This is the theoretical value for this type of attack in a situation where non-idealities are not in place. This shows how this first version of our simulator and the attack implemented work as expected.

IX. RELATED WORK

The literature on Quantum Computing and Quantum Communication has been particularly prolific in recent years. Indeed, a study published by Elsevier[28] highlights that publication in these fields have been steeply increasing since 2015. Among those publications, several works address this paper's issues; therefore, we report in the following some of the most captivating ones. We divide the papers into two groups: one addressing the integration of the QKD and the other the simulation.

The first group of papers discusses the integration of QKD in softwarised infrastructure scenarios. In [20], the authors present an interesting perspective from a telecommunication provider's point of view. They describe how, leveraging Software-defined networking (SDN), QKD can be deployed in modern infrastructures and provide insightful thoughts on real use case scenarios. The authors of [21] provide a solution for integrating QKD in an NFV scenario using SDN-controlled optical switches. Their work mainly focuses on the quantum channel's reconfiguration (using the programmable switches) rather than the key management within the infrastructure. Lopez et al. [22] demonstrate a practical QKD integration over a standard telecommunication network leveraging SDN. Finally, the authors of [23] present the *SDQaaS* framework. This framework endeavours to implement a QKD-as-a-Service (QaaS) approach, where the QaaS functions are developed within an SDN controller. Essentially, QaaS is a pattern that allows sharing of QKD services among different users. All these works leverage SDN for decoupling the control and management plane from the data plane (i.e., forwarding of the keys) within QKD networks. This is a valid strategy that allows dynamic changes in a QKD network, exploiting the available quantum channels effectively. Our solution is different because it focuses on the software stack that is required within an infrastructure to provide security applications with quantum keys. This is achieved by developing a cloud-native application that could run directly on the target infrastructure and do not require the adoption of the SDN paradigm. Nevertheless, according to the specific use case, SDN could still be adopted to optimize the usage of the quantum channel as discussed in [21] and to centralize the management of all QKD nodes. Moreover, it is possible to extend our solution by adopting a Software-defined QKD (SD-QKD) approach described in [24]. This implies that our QKS has to interoperate with an SDN controller (i.e., by means of an SDN agent within the infrastructure), which shall control the QKD modules. The latter adjusts the configuration of the modules according to the key exchange requests and retains information about the applications involved. So far, we believe that a more straightforward solution could be easily adopted in limited infrastructures and for diverse use cases. Nevertheless, our solution could be extended to support SD-QKD.

The second group of papers targets the topic of quantum simulation. Several works ([25], [26]) directly focus on simulating QKD protocols such as BB84 and B92. These works aim at capturing the peculiarities of the specific protocols, taking into account the non-idealities of the practical implementations. The aforementioned works also provide a framework to simulate a key exchange. Comparisons with real testbeds have been provided to show the consistency

[28]See https://www.elsevier.com/solutions/scopus/who-uses-scopus/research-and-development/quantum-computing-report.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

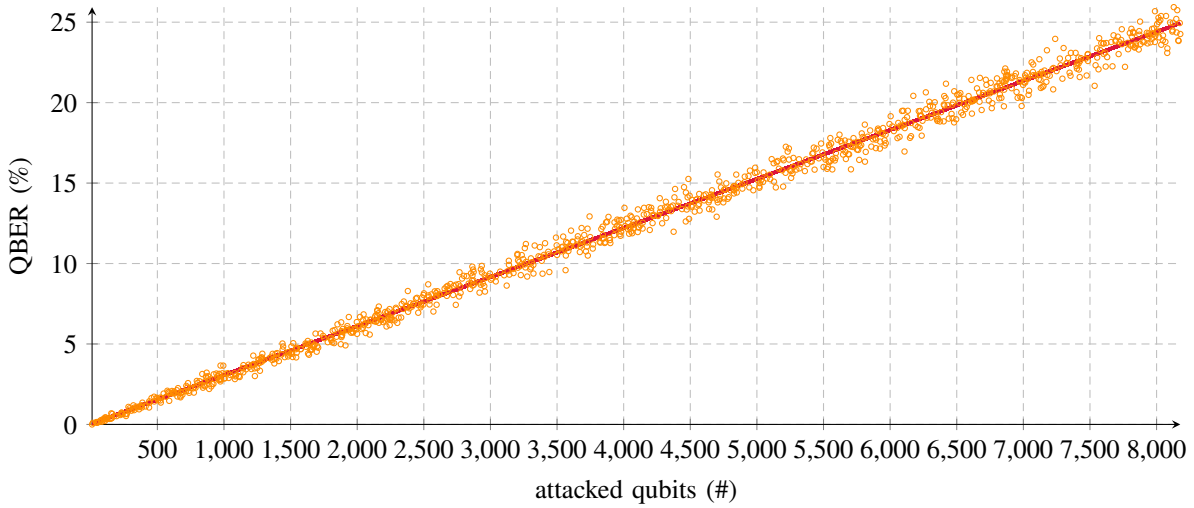

FIGURE 11. BB84 intercept-resend attack: QBER estimation.

of their solutions. Our approach, on the contrary, is more general and aims to reproduce quantum nodes that can communicate over a simulated QKD network, regardless of the type of protocols involved. In this sense, our approach is not tied to a specific technology or framework, but its goal is to provide a scalable framework that is adaptable to different use case scenarios.

Finally, several works address the problem of Quantum Network simulation ([27]–[30]). These works try to match some proposed abstracted quantum network architectures ([3], [31]) and simulate a Quantum Internet scenario with several quantum nodes. This is a difficult task since there is no standard or agreement on the evolution of these networks. One of the most recent and consistent approach is *NetSquid* [30]. This is a generic discrete-event based platform for simulating all the aspect of quantum networks and modular quantum computing systems: ranging from the physical to the application level. The authors of [30] also provide evidence of processing nodes based on NV centres in diamond and repeaters based on atomic ensembles. In this case, their solutions are more generic than the one proposed with the QKD simulators and focus on the quantum network protocols needed for building a Quantum Internet rather than Quantum Cryptography. In this sense, our solution is different because it aims at simulating QKD networks and providing a tool for testing QKD in real infrastructures. This does not exclude the possibility of evolving our solution in a more generic simulator which could also leverage some of those frameworks to simulate Quantum Networks using specific technologies (e.g., NV centres in diamond).

X. CONCLUSION

This paper describes a complete software stack for integrating QKD in softwarised infrastructures. The main contributions lie in a straightforward design and implementation of a QKS, which could be easily integrated into several kinds of infrastructure, and the development of a QKD simulator that could reproduce both BB84 and E91 protocols. These software components could efficiently be extended since they have been implemented using cloud-native technologies and according to the requirements of modern distributed infrastructures. Experimenting on both QKD and Quantum Computing frameworks and contributing to architectures and protocols related to Quantum Networks are of utmost importance for improving the state of the art of quantum technologies. This paper gives a different perspective on how to design and build software components that could enhance the adoption of quantum technologies.

As future work, it is possible to separately work on both the software stack for the QKD integration and the QKD simulation platform. This because the QKD simulator could evolve into something much more effective if extended to the case of the Quantum Networks and the Quantum Internet. We aim at building a scalable and flexible simulation platform where quantum nodes are independent entities that could be programmed to act as quantum devices. In this scenario, the communication over a TCP/IP network could serve as a mean of communication among them, and yet special nodes - as the case of the quantum channel - could reproduce non-idealities of the real case scenarios. Our next step will be to integrate within our framework the simulation of a noisy channel.

Regarding the work on the software stack, it could be improved by integrating a scenario in which the communication within the QKD network is extended beyond the point-to-point connections. Here trusted repeaters could be considered, and according to the ETSI standard, a real use case scenario could be implemented. In this case, routing mechanisms could be integrated at the QKS level, and tests on real infrastructures could be performed.

So far, two other research interests have arisen: the first regarding the error correction and privacy amplification

techniques and the second according to the authentication of the public channel. It would be interesting to integrate within the simulator protocols to implement the key distillation process. Moreover, this integration could provide a complete testing framework for QKD protocols. The authentication of the public channel could be achieved with several techniques. We show in section VII our approach. Nevertheless, the problem is far from being completely solved. In fact, the authentication mechanisms, especially when it comes to softwarised infrastructures, need to be flexible. Because of this, further studies are required to find an optimal solution to both preserve the security of the protocol and the flexibility of the QKD network. Also, for the secure communication with the QKS a standard approach has to be found, and it should be quantum-resistant.

.

APPENDIX A QKD PROTOCOLS

In this appendix we provide a description of two QKD protocols that we investigated in our work: BB84 and E91.

A. BB84

The BB84 protocol is a prepare and measure protocol which Bennett and Brassard describe in [32]. We provide a synthetic overview of the protocol starting from the Table 2.

BB84 deeply relies on the no-cloning theorem: an eavesdropper on the quantum channel could be detected when he measures one or many qubits since he introduces errors. BB84 could be divided into two principal phases: a quantum phase and a classical phase.

During the first phase, the idea is to encode a train of qubits starting from a random sequence of bits representing the classical key and a random sequence of bases that could be used to measure those qubits in a specific reference frame. This operation is performed by the sender (Alice), who chooses both bits and bases and encodes them in a sequence of qubits. Afterwards, she sends those qubits to Bob over the quantum channel. Once Bob receives the qubits, he chooses another random string of bases and measures Alice's qubits with these bases.

According to Table 2 (Quantum Channel), we see that the chosen bases are either "Rectilinear" (R) or "Diagonal" (D). This convention comes from a real use case related to the linear polarization of photons, which is traditionally used to implement QKD. Rectilinear means that we use the basis of vertical (↕) and horizontal (↔), corresponding in turn to the polarization angles of 0° and 90°. Diagonal means that we use the basis of (↙) and (↘), corresponding in turn to the polarization angles of 45° and 135°. The four possible encoded states in BB84 are: ↔ and ↙ for 0 in Rectilinear and Diagonal bases, and ↕ and ↘ for 1 in Rectilinear and Diagonal bases. Bob, measuring Alice's qubits with his bases, produces a classical key string that is a raw version of the final key.

After the quantum phase, there is a classical phase in which Alice and Bob have to share information regarding the chosen bases. This allows cancelling the bits corresponding to the bases that do not match. This process is called *Key Sifting*. Without considering non-idealities (e.g., quantum noisy channel), the key retrieved from this process holds if we demonstrate that no eavesdropper measured qubits on the channel. The basic idea is that, as in Bob's case, Eve has to choose random bases and measure all the qubits she wants during the exchange introducing an error on each qubit with probability 25% (see section VIII for further details). This refers to a simple intercept-resend attack; clearly, it is possible to perform more sophisticated attacks varying the error probability. In the simplest scenario without noise, we could drop every key that contains at least one wrong bit. In a real scenario with non-idealities, this does not hold. As in the Eve's case, a noisy channel introduces an error that could be estimated as *Quantum Bit Error rate (QBER)*. As a solution, it is possible to combine Error Correction (EC) and Privacy Amplification (PA) techniques to, on the one hand, correct potential error within the final key, and on the other hand, reduce the quantity of information that Eve could gain from the key at the expenses of several bits of the key. The metric that we use to establish whether or not a key is valid is the QBER itself. The state of the art techniques allows reaching 11% of QBER, even more with advantage distillation. This means that a key has to be dropped if it exceeds this error rate. Clearly, EC and PA have to be applied after the QBER estimation. This allows avoiding the correction of error introduced by Eve as they depended on the quantum channel. The role of EC is clear. PA, instead, serves the decisive purpose of increasing the number of valid keys during the exchange: even if a subset of qubits has been measured by Eve, we could keep the key as long as the quantity of information gained by her is limited.

The final step in which we apply both EC and PA is known as *Key Distillation*. After this classical phase, the exchange process is completed, and Alice and Bob could use the final key.

B. E91

E91 proposed by Artur Ekert [33] is an entanglement-based protocol. The idea is to leverage a source that emits pairs of spin-1/2 particles in a singlet state:

$$\phi = \frac{1}{\sqrt{2}}(|\uparrow_A\downarrow_B\rangle - |\downarrow_A\uparrow_B\rangle) \tag{1}$$

These particles are maximally entangled and anti-correlated. This means that if particle A is in the state $|0\rangle$ then particle B shall be in the state $|1\rangle$ and vice versa. This coordination is beyond any classical equivalent: when a measurement is applied to one of the particles, even if they are far away from each other, the other will assume the opposite state. After the generation phase, one of the entangled particles is sent to Alice and the other one to Bob over a quantum channel. Once qubits have been

TABLE 2. BB84 protocol steps. (source: [32])

QUANTUM CHANNEL															
Alice's random bits	0	1	1	0	1	1	0	0	1	0	1	1	0	0	1
random sending bases	D	R	D	R	R	R	R	R	D	D	R	D	D	D	R
photons Alice sends	↙	↕	↘	↔	↕	↕	↔	↔	↘	↙	↕	↘	↙	↙	↕
random receiving bases	R	D	D	R	R	D	D	R	D	R	D	D	D	D	R
bits as received by Bob	1		1		1	0	0	0		1	1	1		0	1
PUBLIC AUTH CLASSICAL CHANNEL															
Bob reports bases of received bits	R		D		R	D	D	R		R	D	D		D	R
Alice says which bases were correct			✓		✓			✓				✓		✓	✓
presumably shared information (if no eavesdrop)			1		1			0				1		0	1
Bob reveals some key bits at random					1									0	
Alice confirms them					✓									✓	
FINAL KEY															
remaining shared secret bits			1					0				1			1

received from the parties, they are measured according bases represented by unit vectors a_i and $b_j (i, j = 1, 2, 3)$. Given an x-y-z reference frame, suppose the particles travel according the z direction, than a_i and b_j lie on the x-y plane and could be described, starting from the x axis, by the following angles: on Alice's side $\{\phi_1^a = 0°, \phi_2^a = 45°, \phi_3^a = 90°\}$, on Bob's side $\{\phi_1^b = 45°, \phi_2^b = 90°, \phi_3^b = 135°\}$.

Superscripts a, b are related in turn to Alice and Bob's analysers. Alice and Bob shall randomly choose the orientation angle of the measure with a $1/3$ chance to pick up a specific value.

The quantity related to the Eq. 2 is the correlation of the measurements performed by Alice and Bob according to the bases a_i and b_j.

$$\begin{aligned} E(a_i, b_j) = & P_{++}(a_i, b_j) + P_{--}(a_i, b_j) \\ & - P_{+-}(a_i, b_j) - P_{-+}(a_i, b_j) \end{aligned} \tag{2}$$

The value $P_{\pm\pm}(a_i, b_j)$ is the joint probability of getting a ± 1 (+1 for $|0\rangle$, -1 for $|1\rangle$) along a_i and b_j.

When compatible bases are chosen (same orientation), as in the case of (a_2, b_1) and $(a_3.b_2)$, we have the total anticorrelation of the results: $E(a_2, a_1) = E(a_3, b_2) = -1$. In the other cases, we could define the quantity (3), which is the sum of all correlation coefficients when Alice and Bob used different orientations. This is the correlation value used in the CHSH inequality (one of Bell's inequalities).

$$S = E(a_1, b_1) - E(a_1, b_3) + E(a_3, b_1) - E(a_3, b_3) \tag{3}$$

In the specific case of maximally entangled particles, according to quantum mechanics, this correlation value has to be equal to (4), which is also known as Tsirelson's bound.

$$S = -2\sqrt{2} \tag{4}$$

This violates the CHSH inequality (5), which provides classical correlation boundaries. Violating this inequality demonstrates that the system exhibits a quantum correlation; in this case, the two particles are entangled.

$$|S| \leq 2 \tag{5}$$

Ekert also demonstrated that for every eavesdropping strategy and direction of Eve's measurements, the inequality (6) holds. This inequality violates (4), thus demonstrating that Bell's inequalities could be used as a practical mean to check the presence of an eavesdropper.

$$-\sqrt{2} \leq S \leq \sqrt{2} \tag{6}$$

Once measured and collected all the bit values, the next step of the algorithm is to exchange information regarding the measurement bases. That information could be used by Alice and Bob to divide the obtained bits into two groups: one with the results from compatible bases and another one with all the other measurements. The first group is used as the secret key, the second one for checking if the exchange has been intercepted. According to (3), Alice and Bob could calculate the correlation value and check if it is close to (4). If (5) has been violated, then the key has to be rejected; otherwise, we have a complete and successful key exchange.

APPENDIX B APIS

In the following paragraphs we present the web APIs of the QKDM and QKS components.

A. QKDM APIS

In the following, we detail the APIs exposed by a QKDM. We start from the Southbound Interface, which exposes the ETSI compliant APIs (the ones in bold) and some functionalities useful during the interaction with the QKS. The first group of calls gives access to functionalities related to the key exchange process, while the others serve as utilities:

- **`/api/v1/qkdm/actions/open_connect`**: this call reserves an association identified by a *Key Stream ID (KSID)* among two different QKDMs. This association represents a key stream between two QKD modules and, as a consequence, two Quantum Key Servers. Once this call is invoked, the key exchange

TABLE 3. Southbound interface API

API	ACCESS METHOD	ETSI'S METHOD NAME
/api/v1/qkdm/actions/open_connect	**POST**	**open_connect**
/api/v1/qkdm/actions/get_key	**POST**	**get_key**
/api/v1/qkdm/actions/close	**POST**	**close**
/api/v1/qkdm/actions/get_kids	GET	-
/api/v1/qkdm/available_keys	GET	-

process starts by means of the underlying QKD node component.

- **/api/v1/qkdm/actions/close**: this call stops the key exchange process, but the keys that have already been exchanged are still available.
- **/api/v1/qkdm/actions/get_key**: this call allows retrieving specific keys from a key stream using the `KSID` and `KIDs`. The latter contains a collection of identifiers associated with the keys (both `KSID` and `KIDs` are defined as UUID_v4 on 128 bit).
- `/api/v1/qkdm/actions/get_kids`: this call returns a set of `KIDs` associated with keys that could compose many aggregate keys of different length. The input parameters are `KSID` and `key_info`. The latter contains information such as the number and the length of the required keys. For the sake of simplicity, by default, this module stores keys of the same length (e.g., 128 bits). Because of this, we may want to aggregate more keys to create a larger one.
- `/api/v1/qkdm/available_keys`: this call retrieves information about the keys that are currently available at the QKDM level.
- `/api/v1/qkdm/actions/attach_to_server`: this call is available for an external management tool to trigger the process of registration of the QKDM to the QKS.

The Sync interface exposes instead the following API:

- `/api/v1/qkdm/actions/stream_create`: this call is used after the `open_connect` to notify the value of `KSID` to the peer QKDM.
- `/api/v1/qkdm/actions/sync_KID`: this method is used to notify the KID associated to a fresh generated key to the peer once the key exchange process has terminated.

B. QKS APIS

According to the ETSI standard QKD 014 [2], the *Northbound Interface* implements the communication between the SAE and the QKS. The following API calls (table 4) have been implemented; the bold ones are compliant with the standard while the other ones are an extension:

- **/api/v1/keys/{slave_SAE_ID}/status**: this call returns to the master SAE (the one which starts the key exchange) the status of a specific slave SAE (the target of a key exchange). In particular, it retrieves information regarding the available keys to be requested.
- **/api/v1/keys/{slave_SAE_ID}/enc_keys**: it returns the single or multiple keys requested by the master SAE for the communication to a specific slave SAE. In this case, the QKS in charge of managing the key exchange of the slave SAE shall be informed to reserve those keys to its own SAE.
- **/api/v1/keys/{master_SAE_ID}/dec_keys**: after receiving an *Aggregate Key ID (AKID)* from the master SAE, the slave SAE could call this method using that information to access the keys that have been reserved by its QKS.
- `/api/v1/preferences`: this call returns the current status of the preferences that have been set for the QKS. These preferences involves global settings regarding the Quantum Key Server, such as the log level, the preferred QKD protocols, and the timeout for the requests to other QKSs.
- `/api/v1/preferences/{preference_ID}`: this call allows to change a specific QKS setting.
- `/api/v1/information`: this call could be used by an administrator to retrieve specific info on the QKS (e.g., QKD devices, log).

At the moment, the QKS "side" of the *Southbound Interface* provides a single API call: `/api/v1/qks/actions/register`. This call allows registering a new QKDM to the QKS. The registration process involves the exchange of all the information required by the QKS to manage the specific QKDM. The registration also implies giving the QKDM limited access to the Database and the Vault resources.

In the end, we implemented the *External Interface API*. This allows the communication among different QKSs. In particular, we provide the following calls:

- `/api/v1/saes/{slave_SAE_ID}`: this call could be used to query a specific QKS to check if a SAE is reachable through it.
- `/api/v1/kids/actions/reserve_key`: this is used to reserve a specific key for the communication among two SAEs. Moreover, when a key has been chosen by the master QKS, then we have to reserve that key for the specific communication and make sure that the slave QKS has not used it yet.
- `/api/v1/keys/actions/send_KSID`: this is an

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

TABLE 4. Northbound interface API

API	ACCESS METHOD	ETSI'S METHOD NAME
/api/v1/keys/{slave_SAE_ID}/status	**GET**	**get_status**
/api/v1/keys/{slave_SAE_ID}/enc_keys	**POST**	**get_key**
/api/v1/keys/{master_SAE_ID}/dec_keys	**POST**	**get_key_with_ID**
/api/v1/preferences	GET	-
/api/v1/preferences/{preference_ID}	PUT	-
/api/v1/information	GET	-

utility call for supporting the key exchange among two QKDMs. In particular, this call is used to forward the *KSID* among two different QKSs.

APPENDIX C WORKFLOW

This appendix clarifies the interaction of all components within our solution. Three different phases could be distinguished within our software stack's operational workflow: the QKD module registration, the initialisation of the key exchange process, and the request of a single or multiple keys from a SAE. Before describing each one of these phases, we assume that a working system involves the correct deployment of at least a pair of SAEs and the related QKDMs and QKSs.

As depicted in Figure 12, we analyse these phases in a scenario in which Alice and Bob are aiming at starting a key exchange among their nodes, and Alice is the one opening the connection and requiring a key to communicate with Bob. The first phase (a) involves Alice and Bob's QKDMs registering themselves to the respective QKSs. In this phase, each module has to use the `register_module` call, providing information regarding the destinations that it could reach and the QKD protocols that it supports. Supplementary information could be added, such as the current size of the key buffer and the technology used for the key exchange. As a response, the QKS create a new isolated user on the DB and a new separated secret engine in Vault, returning the credentials to access both to the QKDM. To perform this request, the QKD module has to provide a token that has been generated using Keycloak. This means that only the QKDMs that possess a valid token could operate with the QKS. This process describes a new physical device that is installed on a specific node and wants to be integrated into the scope of the QKS.

The previous phase enables the QKD module to operate as an independent entity; that is, the key exchange process could be performed - once started - without the intervention of the Quantum Key Server, and the resulting keys could be stored directly with a specific `KID` within the secret engine. To optimise the usage of the dedicated channel, this process of exchange could be continuous until the key buffer is full. This should be unlikely, considering multiple SAEs continuously requiring keys for their security applications. To start the process, the QKS has to call - in phase (b) - the `open_connect` on the QKD module. This double-checks if the destination provided is reachable and send a `stream_create` to the peer module, sending information regarding the Quality-of-Service (refer to the standard [2]) and a new `KSID` which identifies the stream of key that will be exchanged among the QKDM pairs. Afterwards, the module retrieves the `KSID` information to the Quantum Key Server (Alice's QKS in Figure 12), which could propagate this information to the peer QKS on Bob's side. Once the latter receives this information, it could perform the same process starting the `open_connect`. After this, the QKD low-level devices (in our case, the QKD nodes) start exchanging their keys, and once an exchange is completed, the QKDMs store the fresh key in the secret engine. This process heavily depends on the implementation of the `open_connect` on both sides.

Moreover, other interactions (the calls presented in appendix B-A) are required to forward the information about the `KID` to assign to the key. In our solution, we define a parameter for the length of the individual keys at the QKDM level. This allows to combine them to form keys of arbitrary length. In a case with real devices, the length of the exchanged keys could vary, and it is also possible to work directly with longer streams. The QKDM abstraction could also serve as an interface to customise the length of resulting keys. We set this parameter to 128 bits for our experiments.

Perhaps the most interesting phase is the last one (c), in which we describe the SAE perspective and analyse Alice and Bob's request for a key. According to the ETSI standard [2], the Security Application Entity could use the `get_key` method to request a new key to the Quantum Key Server. The slave SAE and the characteristics of the key (`key_info`) have to be provided as parameters. In particular, information such as the key length and the number of keys have to be supplied. Once the request reaches the QKS, this latter asks for a collection of KIDs related to keys that could be used to serve the request. If granted, those KIDs are forwarded to the peer QKS with the `reseve_key`. The receiver checks if also on its side those KIDs are available and notify the result to the sender QKS. The idea is to reserve those keys for the communication among the pair of SAE. This mechanism replaces the strict binding of the original version of the standard, which imposes a one-to-one ratio among key streams and SAE pairs. Once we have reserved those keys, the next step involves the standard

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3102313, IEEE Access

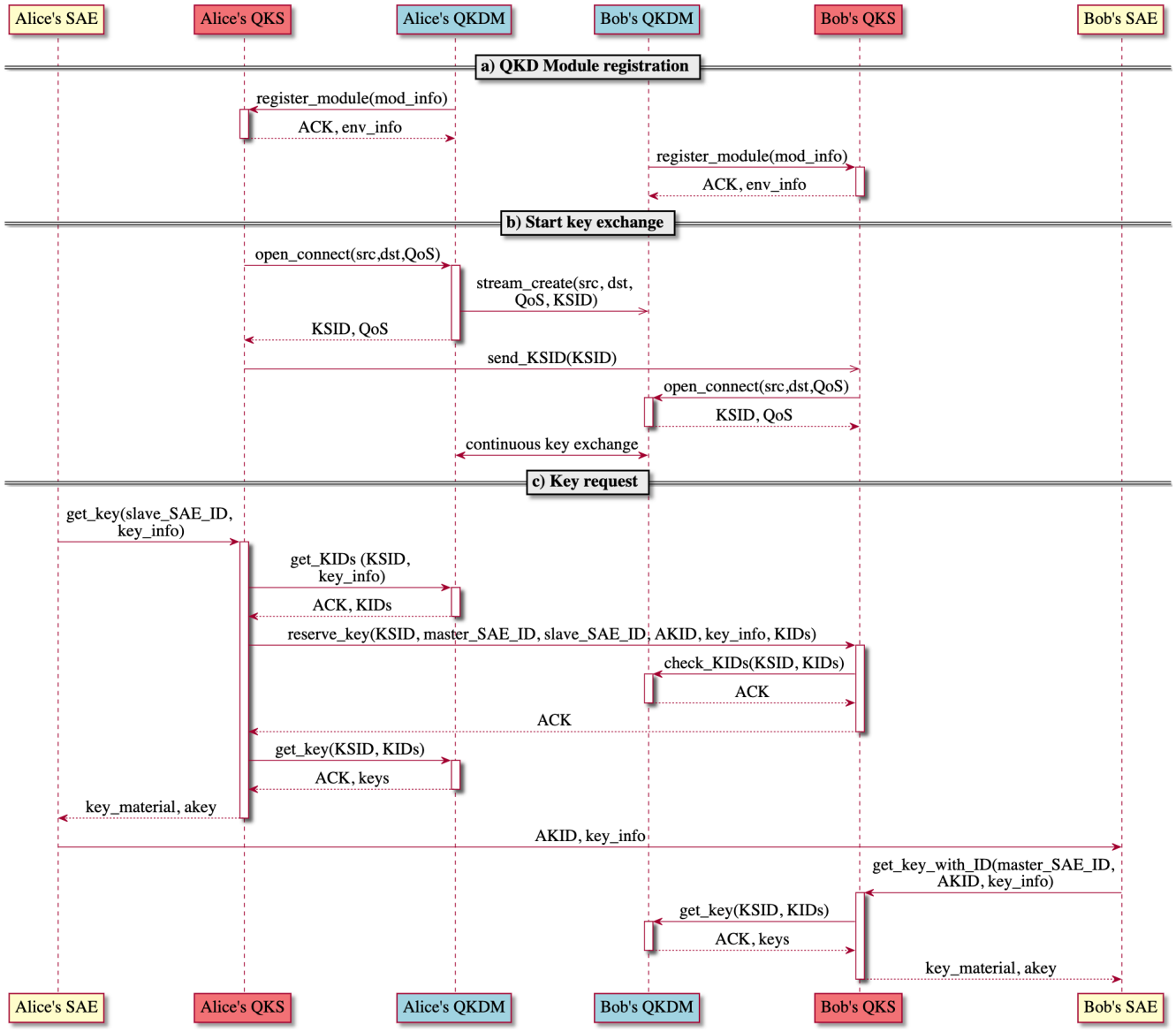

FIGURE 12. Workflow of the complete solution: a) registration phase; b) starting key exchange process; c) key request from a SAE and usage.

`get_key` to the QKDM to retrieve the actual keys and forward them to the SAE level.

At this point, we have on both sides the right keys reserved; we could proceed (as shown in [2]) to inform the peer SAE of the correct AKID to use. This Aggregate Key ID is pivotal because it allows retrieving the composed key all at once. After this, the receiver (Bob's SAE) shall perform the symmetric process and retrieve the required key with the method `get_key_with_ID` passing the right AKID. Clearly, the same mechanism could be easily extended working with different AKIDs. This means that a SAE could ask for multiple keys at a time to get back a map {AKID, key}.

ACKNOWLEDGMENT

We acknowledge Chiara Ruggeri, who received a M.Sc. degree in computer engineering from Politecnico di Torino, for her work.

REFERENCES

[1] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani *et al.*, "Advances in quantum cryptography," *Advances in Optics and Photonics*, vol. 12, no. 4, pp. 1012–1236, 2020.

[2] "Quantum key distribution (qkd); protocol and data format of rest-based key delivery api," European Telecommunications Standards Institute (ETSI), Tech. Rep., February 2019.

[3] S. Wehner, D. Elkouss, and R. Hanson, "Quantum internet: a vision for the road ahead," *Science*, vol. 362, pp. 1–9, October 2018.

[4] W. Kozlowski, A. Dahlberg, and S. Wehner, "Designing a quantum network protocol," in *16th International Conference on Emerging Networking Experiments and Technologies*, Barcelona (Spain), December 1–4 2020, pp. 1–16.

[5] F. Xu, X. Ma, Q. Zhang *et al.*, "Secure quantum key distribution with realistic devices," *Reviews of Modern Physics*, vol. 92, pp. 025 002–1–025 002–60, May 2020.

[6] C. Weedbrook, S. Pirandola, R. García-Patrón, N. J. Cerf, T. C. Ralph, J. H. Shapiro, and S. Lloyd, "Gaussian quantum information," *Reviews of Modern Physics*, vol. 84, no. 2, p. 621, 2012.

[7] R. Renner and R. König, "Universally composable privacy amplification against quantum adversaries," in *Theory of Cryptography Conference*. Springer, 2005, pp. 407–425.

[8] R. Renner, "Phd thesis," Ph.D. dissertation, Swiss Federal Inst. Technol. Zürich, 2005.

[9] C. Lupo, C. Ottaviani, P. Papanastasiou, and S. Pirandola, "Continuous-variable measurement-device-independent quantum key distribution: Composable security against coherent attacks," *Physical Review A*, vol. 97, no. 5, p. 052327, 2018.

[10] V. Scarani and R. Renner, "Quantum cryptography with finite resources: Unconditional security bound for discrete-variable protocols with one-way postprocessing," *Physical review letters*, vol. 100, no. 20, p. 200501, 2008.

[11] "Quantum key distribution (qkd); security proofs," European Telecommunications Standards Institute (ETSI), Tech. Rep., December 2010.

[12] B. Liu, B. Zhao, C. Wu, W. Yu, and I. You, "Efficient almost strongly universal hash function for quantum key distribution," in *Information and Communication Technology-EurAsia Conference*. Springer, 2015, pp. 282–285.

[13] "Quantum key distribution (qkd); application interface," European Telecommunications Standards Institute (ETSI), Tech. Rep., August 2020.

[14] M. Mehic, M. Niemiec, S. Rass *et al.*, "Quantum key distribution: a networking perspective," *ACM Computing Surveys*, vol. 53, pp. 1–41, September 2020.

[15] "Quantum key distribution (qkd); use cases," European Telecommunications Standards Institute (ETSI), Tech. Rep., June 2010.

[16] P. W. Shor and J. Preskill, "Simple proof of security of the bb84 quantum key distribution protocol," *Physical Review Letters*, vol. 85, pp. 441–444, July 2000.

[17] G. Murta, F. Rozpędek, J. Ribeiro *et al.*, "Key rates for quantum key distribution protocols with asymmetric noise," *Physical Review Letters*, vol. 101, pp. 062 321–1–062 321–10, June 2020.

[18] M. Toyran, M. Toyran, and S. Öztürk, "Optimized cascade protocol for efficient information reconciliation in quantum key distribution systems," *Quantum Info. Comput.*, vol. 18, no. 7–8, p. 553–578, Jun. 2018.

[19] B.-Y. Tang, B. Liu, Y.-P. Zhai, C.-Q. Wu, and W.-R. Yu, "High-speed and large-scale privacy amplification scheme for quantum key distribution," *Scientific Reports*, vol. 9, 2019.

[20] V. López, A. Pastor, D. López *et al.*, "Applying QKD to improve next-generation network infrastructures," in *European Conference on Networks and Communications*, Valencia (Spain), June 18–21 2019, pp. 283–288.

[21] A. Aguado, E. Hugues-Salas, P. A. Haigh *et al.*, "First experimental demonstration of secure nfv orchestration over an sdn-controlled optical network with time-shared quantum key distribution resources," in *42nd European Conference on Optical Communication*, Dusseldorf (Germany), September 18–22 2016, pp. 1–3.

[22] D. R. Lopez, V. Martin, V. Lopez, F. de la Iglesia, A. Pastor, H. Brunner, A. Aguado, S. Bettelli, F. Fung, D. Hillerkuss *et al.*, "Demonstration of software defined network services utilizing quantum key distribution fully integrated with standard telecommunication network," *Quantum Reports*, vol. 2, no. 3, pp. 453–458, 2020.

[23] Y. Cao, Y. Zhao, J. Wang, X. Yu, Z. Ma, and J. Zhang, "Sdqaas: software defined networking for quantum key distribution as a service," *Optics express*, vol. 27, no. 5, pp. 6892–6909, 2019.

[24] "Quantum key distribution (qkd); control interface for software defined networks," European Telecommunications Standards Institute (ETSI), Tech. Rep., March 2021.

[25] R. Chatterjee, K. Joarder, S. Chatterjee *et al.*, "qkdsim, a simulation toolkit for quantum key distribution including imperfections: Performance analysis and demonstration of the b92 protocol using heralded photons," *Physical Review Applied*, vol. 14, pp. 024 036–1–024 036–64, August 2020.

[26] L. O. Mailloux, J. D. Morris, M. R. Grimaila *et al.*, "A modeling framework for studying quantum key distribution system implementation nonidealities," *IEEE Access*, vol. 3, pp. 110–130, February 2015.

[27] X. Wu, A. Kolar, J. Chung *et al.*, "Sequence: A customizable discrete-event simulator of quantum networks," September 2020. [Online]. Available: http://arxiv.org/abs/2009.12000v1

[28] S. Diadamo, J. Nötzel, B. Zanger, and M. M. Bese, "Qunetsim: A software framework for quantum networks," April 2020. [Online]. Available: https://arxiv.org/abs/2003.06397

[29] A. Dahlberg and S. Wehner, "Simulaqron - a simulator for developing quantum internet software," *Quantum Science and Technology*, vol. 4, pp. 1–15, September 2018.

[30] T. Coopmans, R. Knegjens, A. Dahlberg *et al.*, "Netsquid, a discrete-event simulation platform for quantum networks," January 2021. [Online]. Available: https://arxiv.org/abs/2010.12535

[31] R. Van Meter, *Quantum networking*, I. S. Pub., Ed., 2014.

[32] C. H. Bennett and G. Brassard, "Quantum cryptography: public key distribution and coin tossing," *Theoretical Computer Science*, vol. 560, pp. 7–11, December 2014.

[33] A. K. Ekert, "Quantum cryptography based on bell's theorem," *Physical Review Letters*, vol. 67, pp. 661–663, August 1991.

IGNAZIO PEDONE received a M.Sc. degree in computer engineering from Politecnico di Torino. He is currently pursuing the Ph.D. degree in computer engineering and he is a member of TORSEC Security Group at Politecnico di Torino. His research interests include security of network infrastructures, quantum computing and cryptography, and trusted computing.

ANDREA ATZENI holds a MSc and a Ph.D. in Computer Engineering, both from Politecnico di Torino. He is currently Senior Research Fellow and Adjunct Professor in the TORSEC Security Group at the Politecnico di Torino. In last twenty years he contributed to a number of large scale European research projects under the FP5, FP6, FP7, CIP and Horizon 2020 programmes, addressing, among the others, the definition of security requirements in multi-platform systems, mobile security, modelisation of user expectation on security and privacy, security specification, risk analysis and threat modeling for complex cross-domain architectures, development of cross-domain usable security, digital and cloud forensics, development and integration of cross-border eIdentity, novel authentication mechanisms, malware analysis and modelling.

DANIELE CANAVESE received the M.Sc. and Ph.D. degrees in computer engineering from the Politecnico di Torino, in 2010 and 2016, respectively. He is currently a Research Assistant with the Politecnico di Torino. His research interests include security management via machine learning and inferential frameworks, software protection systems, public key cryptography, and models for network analysis.

ANTONIO LIOY , Full Professor of Cybersecurity, received the M.Sc. in Electronic Engineering and the Ph.D. in Computer Engineering from the Politecnico di Torino, where he currently leads the cybersecurity research group TORSEC. His research interests include electronic identity, PKI, trusted computing, and policy-based management of large IT systems.