Politecnico di Torino

ScuDo
Scuola di Dottorato ~ Doctoral School
WHAT YOU ARE, TAKES YOU FAR

Doctoral Dissertation
Doctoral Program in Computer and Control Engineering (35^{th}cycle)

Cybersecurity and Quantum Computing: friends or foes?

By

Ignazio Pedone

******

Supervisor(s):
Prof. Antonio Lioy, Supervisor

Doctoral Examination Committee:
Prof. Sokratis Katsikas, Referee, Norwegian University of Science and Technology
David Arroyo, Ph.D., Referee, Consejo Superior de Investigaciones Científicas
Prof. Stefano Pirandola, University of York
Marco Gramegna, Ph.D., Istituto Nazionale di Ricerca Metrologica
Prof. Bartolomeo Montrucchio, Politecnico di Torino

Politecnico di Torino
2023

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Ignazio Pedone
2023

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Abstract

The introduction of Quantum Technologies (QTs) has a dual impact on cybersecurity: on the one hand, Quantum Computing jeopardises current classical public-key cryptosystems, such as RSA and ECDH, and on the other hand, Quantum Cryptography can be a powerful countermeasure against quantum attacks and also a building block for next-generation cryptography. The objective of this thesis work is to explore the current relationship between QTs and cybersecurity in three central aspects: the use of Quantum Cryptography and Quantum Key Distribution (QKD) to protect modern Software-Defined infrastructures (SDIs) against quantum attacks, the adoption of Quantum Annealing techniques to optimise SDIs that can be used as a mean to deploy and manage responsive on-demand network security services, and the analysis of the impact of Quantum Computing on current cryptosystems. As the first result of this work, the reader can appreciate the design and development of a software stack to integrate QKD in SDIs, the Quantum Software Stack (QSS). The QSS is a cloud-native application that can be easily integrated into SDIs and includes a QKD simulator based on a quantum circuit model simulation. A second result is the introduction of a generic Quadratic Unconstrained Binary Optimisation (QUBO) formulation to describe Virtual Network Functions Embedding Problems (VNFEPs). These optimisation problems are part of SDIs, and solving them helps telecommunication providers optimise the management of their infrastructures. The presented QUBO formulation has been validated using a quantum annealer, compatible with this type of formulation, and classical solvers. The final result, even if it can be considered as a preliminary study, regards the analysis of Shor's algorithm for solving Discrete Logarithm Problem (DLP) and Elliptic Curve DLP (ECDLP). The main objective is to analyse the available optimisation for Shor's algorithm and estimate the required resources to run it on real hardware. The implementation of the related quantum algorithms uses Qiskit as a toolkit for describing and simulating quantum circuits.

Contents

Chapter 1

Introduction

Over the last few years, Quantum Computing (QC) has attracted more and more attention due to the increasing hype for the development of a quantum computer powerful enough to run valuable applications for several fields, among others: physics, finance, chemistry, computer science, and molecular biology. The advent of this new computational paradigm is undoubtedly promising, even though it still requires significant efforts to become a reality. For instance, it is not yet clear which the best type of technology for manufacturing quantum processors is (e.g., superconducting qubits, ion trap, photons), how to deal with quantum noise and decoherence to create physical qubits stable enough for running complex quantum algorithms, and the progress of Quantum Error Correction (QEC), which is an extremely active research field, pivotal to achieve the ultimate goal of the fault-tolerant quantum computation. Quantum computing is not the only cutting-edge quantum technology (QT) that gained hype and became of interest to multiple research fields. Indeed, one can count quantum communication, cryptography, and sensing, among others, as equally relevant fields. For some of them, even practical applications are available on the market or close to becoming commercially accessible.

One of the central questions in this thesis work is how the rapid evolution of these technologies is related to cybersecurity. Moreover, what impact in terms of benefits and disadvantages the introduction of this revolutionary advancement may cause. Cybersecurity is a vibrant field, especially for research and development, and, at least for the classical part, it is presently relevant from a technological standpoint. From a general perspective, in recent years, security awareness has increased in different aspects, including software protection, secure communication in computer networks, privacy, protocols and algorithm standardisation, and the introduction of the security-by-design approach where systems

are oriented at security starting from their very first design. As an attempt to answer the previous question, the first impact of QC on cybersecurity regards Cryptography, one of its building blocks. In particular, Shor's algorithm jeopardises the current classical Public-Key Cryptography (PKC) algorithms, i.e., with a quantum computer powerful enough, one can break these cryptosystems with a catastrophic impact on a massive number of operating IT systems. As a reaction, currently, efforts exist to replace those algorithms, e.g., the introduction of Post-Quantum Cryptography (PQC, subsection 2.2.1). Beyond this harmful effect, QT's impact on security has many bright sides. Quantum Cryptography, a branch of Quantum Communication (section 2.4), proposes techniques that allow going beyond the limit of classical cryptography. One example is Quantum Key Distribution (QKD), which enables information-theoretic security for cryptography key exchange and counteracts Quantum Computing's advent as an alternative to PQC.

Another question central to this thesis is how simple integrating Quantum Cryptography into current IT systems is. As an example, more and more distributed systems are based on Software-Defined Infrastructure (SDI) which represents the future of managing and scaling distributed infrastructures and applications. Notably, SDIs are, by definition, flexible and not tied to special-purpose hardware. Current cybersecurity solutions for SDIs, in terms of software protection, monitoring, and secure communication, try to preserve this flexibility. Quantum Cryptography and available QKD solutions, instead, require a significant commitment to special-purpose hardware. This makes the effort of integrating QKD and similar technologies in SDIs a non-trivial task.

The third question indirectly involves how QTs can support cybersecurity by improving SDIs. Moreover, Quantum Computing is particularly valuable in solving complex optimisation problems beyond classical capabilities, e.g., NP-Hard problems with Quantum Annealing strategies (section 2.7). Cybersecurity primarily relies on SDIs to offer on-demand security services, unlocking a relevant opportunity to improve the infrastructures that manage these services.

This thesis work provides a view on all these aspects. First, it discusses the design and implementation of a framework to integrate QKD in SDI. Second, it provides an example of a QKD simulator based on quantum circuit model simulation. This simulator can be integrated into SDI. Third, it presents a formulation for a class of problems related to SDI optimisation and uses it with a quantum annealer as a solver. Fourth and final, it provides an in-depth analysis of the quantum threat posed by Shor's algorithm and an attempt to estimate the resources needed to break current cryptosystems.

The core results of this work show how it is possible to integrate QKD in SDI with a cloud-native software framework designed and implemented by the author. They also provide evidence of how this is appropriate in modern infrastructures, a validation for the QKD simulator, and a full integration into the current de facto standard for managing distributed infrastructures. Beyond the thesis's core results, other achievements come from the optimisation of SDI through Quantum Annealing. The author demonstrates the feasibility of solving SDI-related optimisation problems with a specific formulation and the QPU solver by D-Wave. He also provides a quantitative comparison with alternative classical solvers. Finally, the ultimate results show the analysis of Shor's algorithm for solving Discrete Logarithm Problem (DLP) and Elliptic Curve DLP (ECDLP). This activity also delivered as outcomes the implementation of different variants of Shor's algorithm with the Qiskit toolkit from IBM and a performance comparison among them.

In more detail, the rest of the work is organised as in the following:

- **Chapter 2:** contains a brief background review on QTs and SDIs. This helps the reader find the necessary references to revisit the base knowledge to appreciate this work better.
- **Chapter 3:** is one of the core chapters and introduces the first design and implementation of the QKD integration in SDI. It also presents the first version of the QKD simulator.
- **Chapter 4:** describes the first major improvement of the software stack to integrate QKD in SDI. It introduces the first integration with the current de facto standard container orchestration platform and several new features.
- **Chapter 5:** discusses a new version of the QKD simulator, enhanced for what concerns the infrastructural aspects. This version allows for distributed simulations.
- **Chapter 6:** introduces a class of optimisation problems relevant to SDI and cybersecurity. The author proposes a formulation for these problems and discusses the results of tests with both quantum and classical solvers.
- **Chapter 7:** introduces Shor's algorithm, its impact on current cryptography, available and proposed optimisation, a Qiskit implementation, and an attempt to resource estimation for its execution on real hardware. The reader may wonder why the chapters on the impact of QTs on modern cryptography are at the end of the work. The first reason is that the three main subtopics of the thesis are decoupled and can be treated

independently. The second reason is the choice first to treat the core argument of the thesis.

- **Chapter 8:** extends the work of the previous chapter to ECDLP by presenting the state of the art on the topic and potential implementations.
- **Chapter 9:** this chapter concludes the thesis discussing outcome and future work.

Chapter 2

Background

This chapter provides a brief and general background for the concepts, topics, and technologies discussed in this thesis work. Beyond the introduction to concepts, the references provided in this chapter are helpful for revisiting some details that allow appreciating this work better. As a remark for the reader, since this thesis investigates many subtopics, in the following chapters, it is also possible to find additional preliminary details on each examined topic.

2.1 Quantum Computing

Quantum computing (QC) is a vibrant field for both academia and businesses. QC is widely considered a ground-breaking area with the potential to address issues previously intractable using traditional computing. One may find examples confirming that from several scientific disciplines, including physics, chemistry, molecular biology, computer science, and cryptography. Unfortunately, QC impact on these fields may not always be an improvement, but sometimes can also be a risk for classical systems. As an example, it jeopardises existing cryptosystems with its capability of solving, in polynomial time, problems that were believed to be untractable in terms of computational complexity. This clearly has a significant short-to-medium-term impact on current technologies and also society. In order to stress more the increasing relevance of this emerging paradigm, in 2021, the government of the United States funded QC research and development with $1.2 billion over five years via the National Quantum Initiative Act [1].

From a technical standpoint, the authors of [1–3] presented a technical and detailed exposition of the basics of QC, such as the definition of a qubit as a unit of quantum information,

quantum mechanical principles such as superposition, decoherence, and entanglement as well as application and quantum algorithms. In this work, since we discuss several aspects of quantum information, computing and communication, we assume that the reader is familiar with these fundamental concepts as well as with the definition of state vector, density matrix and quantum measurements. However, it is possible to follow the references provided at the beginning and in the following for a complete and consistent introduction to the various QTs.

A qubit is a basic unit for quantum information corresponding to the classical analogue bit and can be represented using a two-dimensional Hilbert space. A pure state of a qubit, in general, can be defined as $|\Psi\rangle = a|0\rangle + b|1\rangle$ in the computational basis, where $a, b \in \mathbb{C}$ and $|a|^2 + |b|^2 = 1$. One may also adopt different bases. As an example replacing $a = 1/\sqrt{2}$ and $b = \pm\sqrt{2}$, one obtains the so-called conjugate basis. In practice, a qubit can be realized by associating a two-dimensional space or a subspace of a physical system with it, e.g., a photonic system. Certain conditions, such as the interaction with the environment, can decrease a qubit's purity, leading to a different qubit representation, a statistical mixture of pure quantum states (mixed state). A general description of a state can then be obtained using a positive operator ρ, called the density operator (density matrix). Unitary transformations can be applied to a single qubit and describe reversible, probability-preserving operations on qubits (quantum gates). Examples of these gates are the Pauli gates (X, Z, Y) that have the effect of a phase flip, a bit flip, and a combination of the two, respectively. Other single-qubit gates have been defined, as well as multi-qubit gates [2] to form complex quantum circuits for describing quantum algorithms. The measurement process on a quantum system in a state ρ can be generally described by a set of linear (Kraus) operators [4]. The outcome of such a measurement is a value with a certain probability of occurrence.

From a more general point of view, there are different quantum computation models. The most adopted is the quantum circuit model, which we briefly introduced and is discussed in [1] and that we adopted in section 3.4, for the QKD simulator, and in chapter 7, for the analysis of Shor's algorithm. In this model, one can start from a quantum register, initialised to a specific value, apply a series of quantum gates [2] that represent the quantum state evolution and measure the final result. Quantum algorithms can be expressed using this model, e.g., Grover, Deutsch–Jozsa, Shor, Simon, and Quantum Phase Estimation [2]. Other possible quantum computation models exist, such as Quantum Turing Machines, Adiabatic Quantum Computing (from which the quantum annealing discussed in section 2.7 derives), quantum walks model and dissipative quantum computing. Several technologies are available to implement qubits in practical architectures: superconducting qubits, in which many companies such as D-Wave, Rigetti Computing, Google and IBM are investing; trapped ions, where we have Quantinuum,

IonQ; photonics with Xanadu and PsiQuantum; neutral atoms; Nuclear Magnetic Resonance (NMR); Nitrogen vacancy defects in diamond [1].

The toolkit generally adopted for our works, Qiskit, is based on the quantum circuit model and has a reference hardware architecture from IBM. Qiskit textbook [2] is a valuable resource for getting details about quantum gates, algorithms, noise models, and practical programming examples generally tied to the IBM architecture.

2.1.1 Qiskit toolkit

In the following chapters, the reader will find several implementations based on the quantum circuit model and, in particular, the Qiskit toolkit[1]. This section briefly presents Qiskit and gives useful references to collect more in-depth details about this technology.

Qiskit is a software toolkit that works with quantum computers in terms of quantum circuit simulations, algorithms, and pulses. This toolkit allows simulating quantum circuits locally and submitting jobs to IBM Q backends[2]. The main components of Qiskit are:

- *Terra*: all other Qiskit components are built on top of this. Terra offers a layer for creating quantum programmes using quantum circuits or pulses, a module for optimising circuits according to particular device constraints, and interfaces to improve end-user usability and enable access to various backends for simulation or actual device execution.

- *Aer*: it offers a high-performance quantum circuit simulator that may be used to model circuits created by Terra. This is helpful for rapidly evaluating and confirming the functionality of the created quantum circuits. Additionally, it has programmable noise models that may be used to create accurate noisy simulations of errors on actual devices.

- *Ignis (Qiskit Experiments)*: it is a module that offers resources for better characterising errors and operating circuits when noise is present. Recently it has been replaced by Qiskit Experiments. It was created to be used with quantum error correction codes.

- *Aqua*: quantum algorithms might be applied to a variety of fields, including finance, chemistry, and artificial intelligence. Aqua was a software library that made possible to evaluate the advantages of using quantum computing in the relevant fields and create unique solutions on top of the suggested methods. Recently Aqua has been

[1] https://qiskit.org
[2] https://www.ibm.com/quantum-computing/

deprecated and divided into several more specific software components: Finance, Nature, Optimisation, and Machine Learning.

Even though Qiskit and IBM Q are leading technologies for this sector, it is worth mentioning that several other toolkit for Quantum Computing are available. Examples are Cirq[3], PennyLane[4], and Rigetti forest[5].

2.2 Quantum Computing vs Cybersecurity

In this section, the reader can find a high-level description of the threats and benefits of QTs.

2.2.1 The Quantum Computing threat

Shor's algorithm [5], presented by Peter Shor in 1994, showed for the first time how it would be possible to solve integer factorisation and Discrete Logarithm Problem (DLP) by using a quantum computer in polynomial time. This event unlocked a plethora of scenarios where classical cryptosystems, particularly those based on specific problems (e.g., RSA, Diffie-Hellman), became potentially insecure against a future full-fledged quantum computer. At that time, the idea of building such a powerful quantum computer was far away in time; in contrast, today, we are closer and closer to that achievement. This is clear looking at the roadmap to implement quantum hardware by manufacturers such as IBM [6] and the concern about the matter shown by several government agencies [7]. In addition to this concern, many optimisations and possible implementations of Shor's algorithm have been presented so far, as well as extensions [8] to make it work on other cryptosystems based on Elliptic Curve DLP (ECDLP), such as ECDH.

It is currently impossible to run Shor's algorithm to break the state-of-the-art classical cryptosystems due to a lack in terms of quantum hardware maturity. Nevertheless, to prepare fsdeuture cryptosystems for the so-called "Q-day" (when a quantum computer powerful enough will be available), two main approaches have been proposed: one based on quantum mechanics and related to the field of Quantum Cryptography, the other based on classical algorithms and called Post-Quantum Cryptography (PQC).

[3] https://quantumai.google/cirq
[4] https://pennylane.ai
[5] https://pyquil-docs.rigetti.com/en/v2.7.2/

Quantum Cryptography is a branch of Quantum Communication (section 2.4) which explores secure communication. Quantum Key Distribution (QKD) (section 2.5) is one of the few approaches of Quantum Cryptography available today which can be implemented in practical systems. QKD is generally based on the no-cloning theorem and - for specific protocols - on entanglement [9].

PQC is an alternative to Quantum Cryptography against the quantum threat, which relies on developing classical quantum-resistant algorithms to substitute the current Key Encapsulation Mechanisms (KEM) and algorithms for digital signature.

In 2016, the National Institute of Standards and Technologies (NIST) started the standardisation process of these algorithms, announcing a call for proposals[6]. Several algorithms have been proposed during the first three rounds, which are based on the specific mathematical problem considered: lattice-based, code-based, multivariate, hash-based and isogeny-based. Recently NIST has announced the first group of winners: CRYSTALS-Kyber as KEM (lattice-based), CRYSTALS-Dilithium and Falcon for digital signatures (lattice-based), and SPHINCS$^+$ (hash-based) also for digital signatures. NIST also announced a fourth round of selection for the following KEM algorithms: BIKE, Classic McEliece, HQC (code-based), and SIKE (isogeny-based). As a remark for the reader, after the NIST announcement, a vulnerability of the SIKE scheme has been found [10]. This vulnerability compromises the current version of the algorithm.

Beyond PKC schemes and Shor's algorithm, there is another quantum algorithm that, in principle, may represent a threat to cybersecurity, particularly to symmetric algorithms and hash functions: Grover's algorithm. The latter is an algorithm for unstructured search (e.g., unsorted database) that can succeed with high probability performing $\mathcal{O}(\sqrt{N})$ queries instead of the $\mathcal{O}(N)$ of the classical case. As a remark for the reader, since Grover's algorithm can only provide at most quadratic speedup over the classical solution for unstructured search and classical NP-complete problems require an exponential-time algorithm to be solved, Grover can only offer for this problem quadratic speedup; thus it cannot solve them in polynomial time.

For symmetric cryptography, as shown in [11, 12], this result means a reduction of the security of a key to half: AES-256 (key of 256 bits) will be secure as the current AES-128. AES-256 so far is considered to be quantum-safe[7] especially looking at the resource estimations provided by [13, 14]. These estimations highlight how to break the security of

[6] https://csrc.nist.gov/Projects/post-quantum-cryptography/post-quantum-cryptography-standardization
[7] https://media.defense.gov/2022/Sep/07/2003071836/1/1/0/CSI_CNSA_2.0_FAQ_.PDF

AES-256, a quantum circuit with a depth of roughly 2^{145} is required, which translates into a reasonable security level according to NIST.

Grover's algorithm can also threaten hash functions [15, 16]: quadratic improvement of the classical pre-image and birthday paradox attacks. Despite these improvements and considering, as in the case of symmetric cryptography, the resource estimation of the required quantum circuit, the SHA-3 family of algorithms is assumed quantum-safe.

For this reason, our work only focuses on PKC and Shor's algorithm. According to PKC, there is an increasing interest in Adiabatic Quantum Computation and Quantum Annealing that might represent a threat since these are promising approaches to solving NP-hard problems. An example worth mentioning is the work on speeding up factoring large integers with quantum SAT solvers presented in [17].

2.2.2 Quantum Computing positive impact on cybersecurity

Beyond QTs' negative aspects and impact on cybersecurity, many benefits to cybersecurity should be briefly discussed. First, Adiabatic Quantum Computation and, in particular, Quantum Annealing, as already mentioned, play an essential role in solving NP-hard problems, commonly optimisation problems. Many branches of cybersecurity beyond cryptography, directly or indirectly, leverage this class of problems to counteract potential threats and design, implement, and optimise cyber defences.

A tangible example is the work presented in chapter 6, where Quantum Annealing is used to optimise the allocation of virtual security functions. Another interesting example, considering the role and the possible application of bloom filters in network security [18], is the quantum version of bloom filters. This has interesting applications in cybersecurity for privacy-preserving issues [19]. Moreover, it is also worth mentioning the work related to QUBO reformulations for the Maximum k-Colourable Subgraph problem (MkCS) [20] and its applications to cybersecurity. An example of application is the case of fault diagnosis for multiprocessor systems [21]. Obtaining a QUBO formulation paves the way towards solving these problems using a quantum annealer, as demonstrated for a different problem in chapter 6. Last but not least, Quantum Machine Learning (QML) techniques can leverage either the classical gate/circuit model or quantum annealers to mitigate, through intrusion detection, Distributed Denial of Service (DDoS) attacks [22, 23].

2.3 Quantum Error Correction

Noise, decoherence time, and other factors depending on the specific hardware technology adopted may cause errors that affect the computation results in terms of the final state of the physical qubits. For this reason, it is pivotal to have efficient Quantum Error Correction (QEC) techniques whose objective is to create robust logical qubits out of several fragile physical qubits. The author of [24] proposes an introduction to QEC and a presentation of the basic techniques available. QEC techniques may also vary depending on the specific hardware and the topology of the quantum processor. As for the classical case, the notion of *code distance* is the minimum Hamming distance, i.e., the minimum distance between any two codewords or the number of bits one must flip to obtain another codeword. The reader can find in subsection 2.3.1 a brief presentation of the Heavy Hexagon Code proposed and implemented by IBM. This presentation helps understand some of the estimations provided in chapter 7.

2.3.1 Heavy Hexagon Code

The heavy hexagon code [25] is a surface/Bacon-Shor code [26] mapped into a hexagonal lattice that includes all the ancilla qubits required for fault-tolerant error correction. IBM proposed this code and has planned to adopt it in its devices' current and future versions. Figure 2.1 shows how $3/5$ of the qubits presents a degree (number of edges adjacent to that node/qubit) equal to 2, and the other $2/5$ has a degree equal to 3. IBM intends to use cross-resonance gates, considering that they achieved a gate fidelity of around 99%. A low degree value allows mitigating frequency collision, and crosstalk [27]. As a remark for the reader, certain quantum gates, such as the controlled phase shift, require qubit frequency to be different from all its neighbours and chosen from an available set. In order to minimise frequency collisions, this code outputs a set of only three frequencies present in the device.

The following equation [25] gives the total number of qubits n_p for a given code distance d:

$$n_p = \frac{5d^2 - 2d - 1}{2} \tag{2.1}$$

As a reminder for the reader, this equation will be relevant for the resource estimation in chapter 7. The code distance will strongly influence the logical error rate as it depends on the probability that $\lfloor (d-1)/2 \rfloor$ errors co-occur in the underlying physical qubits.

The logical error rate should not exceed a threshold conditional on the desired circuit's depth; otherwise, the final result of a quantum circuit execution may report inconsistent data.

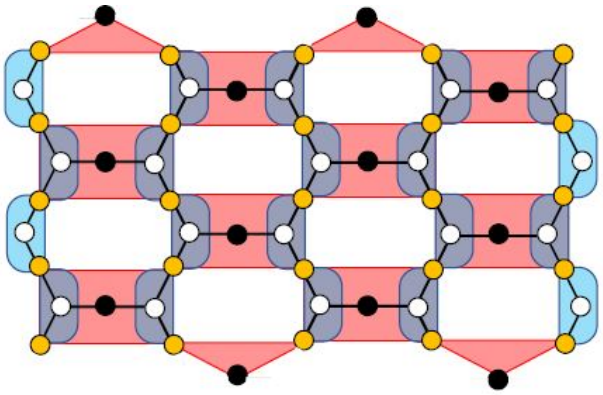

Fig. 2.1 Heavy hexagon code with d = 5 for one logical qubit. Yellow vertices are the data qubits, white vertices represent the flag qubits, and dark vertices describe ancillae to measure bit-flip errors (red zones) and sign-flip errors (blue zones). Source: [25]

From this starting point, one can decide the value of the code distance and the number of required physical qubits. As an example for the reader, considering a depth of 10^6, a width of 50, a Physical Error Rate (PER) of 10^{-4}, and a required fidelity of the quantum circuit of 90%, the final error probability is given by the equation:

$$\rho \simeq \frac{1-0.9}{\text{depth}} = 10^{-7} \tag{2.2}$$

Choosing a code distance $d = 7$ and according to Equation 2.1, the whole circuit would require 115 physical qubits for each logical one for a total of 5750. Correcting three errors for each logical qubit at each step, one can estimate the error probability on each logical qubit $\rho_{L,d=7}$ using Binomial distribution:

$$\rho_{L,d=7} = 1 - \sum_{k=0}^{3} \binom{115}{k} (\text{PER})^k (1-\text{PER})^{115-k} = 6.85 \cdot 10^{-10} \tag{2.3}$$

The probability of error at each step $\rho_{d=7}$ is therefore:

$$\rho_{d=7} = 1 - (1-\rho_{L,d=7})^{50} = 3.42 \cdot 10^{-8} \tag{2.4}$$

That code distance is sufficient considering that $\rho_{d=7} < \rho$. Several IBM quantum processors, such as Falcon, Hummingbird, and Montreal already adopt this QEC approach.

2.4 Quantum communication and networks

Quantum communication is a field that leverages quantum mechanics principles to transfer information, typically encoding it in quantum states. Quantum communication nowadays, even if it mainly refers to the technologies related to Quantum Key Distribution (QKD), includes several branches of research that study how certain principles of quantum mechanics, such as entanglement can improve classical communication and cryptographic techniques. As an example for the reader, in the following, there is a list of various branches of quantum communication (QKD will be discussed in detail in section 2.5):

- **Quantum Teleportation:** this technique allows transferring an unknown quantum state among two distant particles by using an entanglement pair, projective measurements and two classical bits.
- **Quantum secret sharing:** is the quantum version of the classic secret sharing, where a secret is shared among many actors. Only when a minimum number of them agree to cooperate the final secret is revealed. It has been adopted in various techniques, particularly in quantum money and distributed quantum computing.
- **Quantum secure direct communication (QSDC):** even if this branch has not yet consistent, practical application, this is one of the most active in quantum communication beyond QKD. QSDC allows secure communication directly over a quantum channel, contrary to QKD, without exchanging a secret in advance [28].
- **Quantum identity authentication (QIA):** quantum communication needs identity authentication. This is typically implemented, as in the QKD case, with classical techniques allowing to avoid attacks such as the man-in-the-middle ones. QIA includes many approaches [28] leveraging quantum resources that aim at reaching unconditionally secure schemes. One class of particular interest is the quantum-based Physical Unclonable Functions (PUFs), e.g., PUFs challenged using quantum states.
- **Quantum signature:** this branch proposes quantum-resistant digital signature schemes alternative to the classical one [28].
- **Quantum bit commitment:** bit commitment is a fundamental cryptographic protocol used to implement schemes, such as coin tosses, zero-knowledge proofs, and electronic voting. A bit commitment protocol is usually divided into two phases: commit and reveal. Alice sends Bob information reflecting the bit value she selected, without exposing it, during the commit phase. Afterwards, Alice reveals the bit value she

picked to Bob during the unveiling phase. Bob utilises Alice's previously supplied information to determine whether Alice is speaking the truth or lying. Unconditionally secure Quantum bit commitment schemes have been proved impossible to achieve, but there are potential advantages over the classical counterpart [28].

- **Quantum coin flipping:** is the quantum version of the classical coin tossing primitive where two players that distrust each other have to agree on a random bit by leveraging message exchanges between them. Quantum coin flipping as a primitive allows the building of much more complex protocols, such as the Quantum Byzantine agreement [28].

- **Quantum oblivious transfer:** is a quantum version of the classical oblivious transfer, where messages are sent from a sender to one or more receivers so that the sender remains unaware of if one or more messages have been transferred while other messages are kept private from the receivers. There exist versions of oblivious transfer with multiple senders. Quantum oblivious transfer cannot be implemented with unconditional security, i.e., by only relying on quantum physics laws [28].

- **Secure multi-party quantum computation**: is the quantum version of the classical secure multi-party quantum computation, where two or more participants want to compute a publicly known function on their private data without disclosing their inputs. Compared to the classical version, the quantum has shown to be more robust in terms of security and eavesdropper detection.

Another essential concept is the one of "quantum networks". A quantum network is a network with nodes capable of exchanging quantum information, generally encoded as qubits, leveraging a quantum channel. In practical use cases, those networks need to interact with classical networks to implement actual protocols. This leads to a hybrid scenario where both classical and quantum networks shall coexist. A really neat view of the quantum network's evolution is available from the authors of [29], where it is explained how the long-term goal for quantum communication is to build the so-called Quantum Internet, which may intensively cooperate with the classical one. This roadmap also provides a discussion of the intermediate stage of this evolution as well as a description of the possible application and protocol for each stage.

Currently, quantum networks, from a practical perspective, mainly involve the implementation of QKD networks. Nevertheless, there are many tools to simulate quantum networks beyond QKD and with different purposes. Among the others, the most relevant are; SimulaQron [30], NetSquid [31], QuISP [32], QuNetSim [33], and SeQUeNCe [34]. For some

of them, there is a brief comparison against one of the solutions discussed in this thesis, the QKD simulator.

2.5 Quantum Key Distribution

QKD is a technique which allows leveraging quantum phenomena and principles, such as entanglement and the no-cloning principles, to build cryptographic protocols able to reach Information-Theoretic Security [35]. In particular, the task that allows achieving is the secure exchange of cryptographic keys among parties. Commonly QKD requires two channels on which parties can exchange information: a quantum channel where parties can send and receive quantum information (typically encoded as photons); a classical authenticated channel where they can exchange additional information, e.g., data to coordinate the post-processing of the keys or for authentication purposes.

Typically, one can divide QKD systems into Discrete-Variable QKD (DV-QKD) and Continuous-Variable QKD (CV-QKD). The first type performs encoding and decoding leveraging qubits or other quantum systems with finite-dimensional Hilbert space. Because of this, one can also call it qubit-based QKD. CV-QKD systems encode information in quadratures of the quantised electromagnetic field and decode it using coherent detections [36]. Such detection methods are valuable in modern QKD apparatuses as they are compatible with existing telecom equipment and show elevated detection efficiencies without requiring cooling [9]. As a remark for the reader, in this thesis work, one can refer only to a subset of DV-QKD systems for the QKD simulator part (section 3.4) for the remainder of the discussions, they can apply to both classes since QKD is treated as a black box. Examples of DV-QKD protocols are BB84, E91, SARG04, BBM92, COW [35], and T12 [37], where COW belongs to a particular class, known as distributed phase-reference pulse [38].

Another helpful classification may differentiate them depending on the particular adopted scheme: prepare-and-measure, entanglement-based, or Measurement-Device-Independent QKD (MDI-QKD). As for the first scheme, we have two entities, Alice and Bob, one preparing the quantum information encoded as qubits and the other receiving it through a quantum channel and performing opportune measurements. Other schemes usually involve another actor, such as Charlie. Charlie is a third-party module which acts as an untrusted entangled-pairs sender in entanglement-based protocols and an untrusted receiver in MDI-QKD.

Several companies commercialise actual QKD systems, such as ID Quantique[8] and Toshiba[9]. They are mainly point-to-point QKD devices (e.g., Cerberis XG), and in the Toshiba case, it relies on a decoy-state version of BB84.

The author of [39, 40] proposes a composable security definition for QKD, which is rigorously articulated and widely accepted. Many composable security proofs are known [35, 9, 41, 42] for DV-QKD and CV-QKD. A remark for the reader is that QKD protocols assume authentic classic channels in their security proofs. According to the definition of composable security, one can utilise composable Information-Theoretic Security (ITS) authentication for the classical channel with 2-universal hash functions [43] or adopt an efficient authentication scheme employing 2-almost strongly universal hash functions. This approach provides the final security parameter ϵ by the composability statement of the respective security proofs of QKD, and ITS authentication [44]. As a remark for the reader, ϵ indicates that the final key is epsilon-close to the ideal one.

Another critical remark is that in section 3.4, the reader finds out that in our work, we introduced other authentication methods whose security proofs have not been proven composable in strictly theoretical terms. However, so far, there are no known or possible attacks to break them and they appear to be more flexible applied to modern or software-defined infrastructures. One example might be the combination of QKD and PQC, where adopting PQC for authentication, no shared secret is required for the parties at the very beginning of their configuration. An elegant perspective would also be the possibility of leveraging a Post-Quantum Public-Key Infrastructure (PKI) [45].

Another essential issue to introduce is the long-distance QKD exchanges. Point-to-Point (PTP) QKD works fine under a certain distance range (e.g., usually a hundred kilometres), and this range depends on the specific classes, the scheme and the protocols chosen. Clearly, many factors lead to this loss in efficiency, e.g., attenuation of the laser beam. To overcome this issue, trusted repeaters have been introduced. These devices act as trusted relays (intermediate node of a QKD network in subsection 4.3.2), being able to transfer the cryptographic material, e.g., using a store and forward strategy [46]. Clearly, there is a significant impact on security because all the intermediate nodes have to be trusted, and this is not a trivial assumption in large networks. An alternative strategy, which is not yet practically implementable in current commercial systems, is to use quantum repeaters. These repeaters use entanglement swapping to entangle particles far away from each other, which allows for verifying the presence of an eavesdropper from the endpoint's perspective.

[8] https://www.idquantique.com
[9] https://www.toshiba.co.jp/qkd/en/products.htm

Table 2.1 BB84 protocol steps. (source:[49])

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

As another remark for the reader, QKD systems may be susceptible to side-channel attacks, and a particular category of attacks called quantum attacks, which involve leveraging the vulnerabilities of the imperfection of the physical implementation. The authors of [35] presented a comprehensive list of these possible attacks and possible countermeasures.

Finally, some relatively recent proposals and experimental developments in the field of QKD can be found in [47], where the authors presented the Twin-Field QKD (TF-QKD). TF-QKD is based on a measurement-device-independent QKD (MDI-QKD) scheme [48] and is remarkable because, as the basic MDI-QKD, it preserves the advantage of removing all detector side-channels and also offers a novel approach to overcome the rate-distance limit of QKD. TF-QKD is pivotal for designing future long-distance QKD systems.

2.5.1 BB84 protocol

The authors of [49] proposed the BB84 protocol which is discussed in this section and summarized in Table 2.1.

BB84 depends on the no-cloning theorem, leading to the result that an eavesdropper on a quantum channel can be detected if he performs measures on this channel. This action will introduce errors that the involved parties can identify. BB84, like other QKD protocols, needs a quantum and a classical channel to be run.

The first step for BB84 is encoding a string of qubits taken from a random string of classical bits. The encoding process requires the sender, Alice, to randomly choose a string of bases representing the reference frame one uses for the measurement. After this step, Alice

sends the encoded quantum states to Bob, who randomly chooses other bases to measure the qubits and perform that measurement.

More in detail, as reported in Table 2.1, the chosen bases are either "Rectilinear" (R) or "Diagonal" (D). This convention arrives from a real use case connected to the linear polarization of photons, which can be used as a mean to encode these quantum states. Rectilinear means using the basis of vertical (↕) and horizontal (↔), corresponding to the polarization angles of 0° and 90°. Diagonal means using the basis of (↗) and (↖), corresponding to the polarization angles of 45° and 135°. Four possible encoded states arise in BB84: ↔ and ↗ for 0 in Rectilinear and Diagonal bases, and ↕ and ↖ for 1 in Rectilinear and Diagonal bases. Bob retrieves a raw version of the final key by measuring the qubits sent by Alice.

After these steps, classical post-processing actions have to be performed. Alice and Bob share data about their bases and cancel out the bits corresponding to the non-matching bases in the Key Sifting step. Leaving out non-idealities, such as noise on the quantum channel, the presence of Eve can be detected by sacrificing part of the exchanged key, sharing that information publicly and measuring a metric called Quantum Bit Error Rate (QBER), which indicates the error introduced by the eavesdropper in this case. Assuming that all the qubits were measured independently by Eve, we would get a QBER of 25%. In a real scenario, one needs to consider both noise and Eve acting simultaneously on the quantum channels and strategies beyond the simple intercept-resend attack. For this reason, classical Error Correction (EC) techniques are applied as post-processing procedures and Privacy Amplification (PA), in turn, to correct errors and decrease the quantity of information Eve retrieves from the quantum channel. One can refer to this last step as Key Distillation.

2.5.2 E91 protocol

The author of [50] proposed E91, which can be defined as an entanglement-based protocol. E91 leverages a source of spin-1/2 particles in a singlet state, maximally entangled and anti-correlated:

$$\phi = \frac{1}{\sqrt{2}}(|\uparrow_A\downarrow_B\rangle - |\downarrow_A\uparrow_B\rangle) \tag{2.5}$$

For clarity, one can observe that if particle A is in the state $|0\rangle$, particle B will be in $|1\rangle$ and vice versa. This coordination is beyond any classical equivalent, i.e., when a measurement is applied to one of the particles, the other will assume the opposite state even if these are farther away from each other.

Once generated, Alice and Bob receive one particle each, and they proceed with the measurement phase, which relies on bases expressed by unit vectors a_i and b_j where $i, j = 1, 2, 3$. Considering an x-y-z reference frame and assuming the particles travel according to the z direction, then a_i and b_j lie on the x-y plane and can 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 $\{\phi_1^b = 45°, \phi_2^b = 90°, \phi_3^b = 135°\}$.

Alice and Bob's analysers can be distinguished by the superscripts a and b. The choice of the orientation angle has to be performed randomly, and the probability of selecting a particular value is $1/3$. Equation 2.6 describes Alice and Bob's measurement correlation adopting as bases a_i and b_j.

$$\begin{aligned} E(a_i, b_j) = & P_{++}(a_i, b_j) + P_{--}(a_i, b_j) \\ & - P_{+-}(a_i, b_j) - P_{-+}(a_i, b_j) \end{aligned} \tag{2.6}$$

$P_{\pm\pm}(a_i, b_j)$ represents the joint probability of obtaining a ± 1 (+1 for $|0\rangle$, -1 for $|1\rangle$) along a_i and b_j. When Alice and Bob choose the same orientation (bases), as for (a_2, b_1) and $(a_3.b_2)$, one can observe a total anticorrelation in the results: $E(a_2, a_1) = E(a_3, b_2) = -1$. When the orientation is different among them, one can calculate the quantity Equation 2.7, which represents the sum of all correlation coefficients. Such a correlation is one of the CHSH inequality, one form of Bell's inequalities.

$$S = E(a_1, b_1) - E(a_1, b_3) + E(a_3, b_1) - E(a_3, b_3) \tag{2.7}$$

For maximally entangled particles, according to quantum mechanics, this correlation value has to be equal to Equation 2.8, which is also known as Tsirelson's bound.

$$S = -2\sqrt{2} \tag{2.8}$$

Disobeying the CHSH inequality (Equation 2.9), which provides classical correlation boundaries, demonstrates that the system exhibits a quantum correlation and that the two particles are entangled.

$$|S| \leq 2 \tag{2.9}$$

The authors of [50] also demonstrated that for every eavesdropping strategy and direction of Eve's measurements, the inequality 2.10 holds. Violating the bound in Equation 2.8 demonstrates that Bell's inequalities can be adopted as a valuable means to check an eavesdropper's presence.

$$-\sqrt{2} \leq S \leq \sqrt{2} \tag{2.10}$$

After the measurement phase ends, Alice and Bob exchange data about the bases (orientations). They can leverage those data to separate the resulting bit string into two sets: one with all compatible choices and another with the reminder of all other measurements. The final secret key is obtained using the first set, and the correlation for Eve's detection is calculated using the second one. As mentioned before, the detection through correlation is related to the bound in Equation 2.8, and, for practical purposes, one can establish a threshold to verify whether the result is close to that bound.

2.5.3 ETSI QKD GS

The European Telecommunications Standards Institute (ETSI) started the standardisation process of QKD for many aspects, from the low-level devices for the PTP QKD to the design of the software stack needed to integrate those systems in modern infrastructures. The specification group dedicated to this standardisation is the ETSI ISG QKD [10]. One can also count additional efforts from different standardisation bodies such as CEN-CENELEC and ITU, among others [51]. Regardless of the particular specification, the ETSI QKD standardisation is relatively immature and still needs efforts to define better the interaction with modern infrastructure and clear architecture that can be used as a guideline for integrating QKD systems. One favourable aspect is that this standardisation process has accelerated during the past years because of the competing standard bodies proposing their specifications. The first interesting contribution of this work (section 3.1) aims to target that integration and analyse practical aspects that might be crucial to evolving the current standards.

Relevant to the core section of this thesis work, the reader can find in the following the principal ETSI QKD specifications adopted as references:

- *ETSI GS QKD 004 V2.1.1* [52]: namely "QKD; Application Interfaces", this specification proposes a general software interface to the physical device;

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

- *ETSI GS QKD 014 V1.1.1* [53]: namely "QKD; Protocol and data format of REST-based key delivery API", this specification proposes a high-level interface from the security applications willing to use QKD-produced key material and the management systems of the QKD networks, which in turn contacts the QKD devices with the previous interface;
- *ETSI GS QKD 015 V1.1.1* [54]: namely "QKD; Control Interface for Software-Defined Networks", this specification represents an attempt to include *Software-Defined Quantum Key Distribution* (SD-QKD) in the standardisation promises. This promising approach combines the flexibility of Software-Defined Networking with the high level of protection guaranteed by QKD systems.

2.5.4 Standardisation efforts beyond ETSI

Beyond the standardisation efforts from ETSI (subsection 2.5.3), promoter of the Industry Specification Group on QKD (ETSI ISG-QKD) and the Technical Committee Cyber Security Working Group on Quantum-Safe Cryptography (ETSI TC CYBER WG QSC), other standardisation bodies are working on specifications on QTs, Quantum Networks, Quantum Cryptography, and QKD. The following are the primary bodies involved in the standardisation process and their contribution:

- *CEN-CENELEC*: is a standardisation body of the EU and EFTA member states, promoter of the Focus Group on Quantum Technologies (FGQT)[11]. It counts 150 participants, and the main objective is to ensure interaction between relevant stakeholders to standardise QTs. This group actively contribute to the Quantum Flagship initiative.
- *ISO/IEC*: the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC), that operate at a global level, contribute through the ISO/IEC JTC 1/SC 27 "IT Security"[12], which stands for Joint Technical Committee 1 of ISO and IEC and Sub-Committee 27. This is involved in standardising IT security, cybersecurity, and privacy protection. QKD is included in the Working Group (WG3), which also works on ISO/IEC 15408 "Common Criteria for Information Technology Security Evaluation (CC)". The standards ISO/IEC 23837-1 and 23837-2 are dedicated to QKD security as special applications of the Common Criteria.
- *ITU*: the International Telecommunications Union (ITU) is a body of the United Nations. It promotes the standardisation of QKD and QTs through the Study Groups ITU-T/SG

[11] https://www.cencenelec.eu/areas-of-work/cen-cenelec-topics/quantum-technologies/
[12] https://www.iso.org/committee/45306.html

13[13], ITU-T/SG 17[14], and the Focus Group FG-QIT4N[15]. The Working items related to QKD are prefixed with "Y.QKDN".

- *IETF*: the Internet Engineering Task Force (IETF) and the Internet Research Task Force (IRTF) contribute through two research groups: the Quantum Internet Research Group (QIRG)[16], focusing on the general architecture of the prospective Quantum Internet, and the Crypto Forum Research Group (CFRG)[17], discussing cryptographic mechanisms.

- *IEEE*: the IEEE Standards Association (SA) contributes through the IEEE SA QuantumComm, Software-Defined Quantum Communication[18], promoting the definition of a common classical interface for quantum communication devices.

- *GSMA*: the Groupe Speciale Mobile Association (GSMA) also started research on the threat and challenges of QTs through the GSMA Internet Group[19].

2.6 Software-Defined Infrastructures

Software-Defined Infrastructures (SDI) are a fundamental building block for almost all modern computation paradigms and network technologies such as Cloud-, Edge-, Fog-computing, Network Functions Virtualization (NFV), and Internet of Things (IoT). SDI allows the creation, management and scaling of an IT infrastructure with minimum human intervention by embedding the logic beyond operations to build and maintain the infrastructure in the software.

A remark for the reader is that currently, a large portion of the distributed applications and services available for a final end-user runs leveraging the concepts of microservice and container. The first is an approach to divide a classical monolithic application into several and more simple software components that can be managed as independent processes and can communicate among each other using a classical TCP/IP network, among other strategies. The second is an isolated environment in which those software components can run, leveraging specific capabilities granted by the kernel of operating systems. Using this

[13]https://www.itu.int/en/ITU-T/about/groups/Pages/sg13.aspx
[14]https://www.itu.int/en/ITU-T/about/groups/Pages/sg17.aspx
[15]https://www.itu.int/en/ITU-T/focusgroups/qit4n/Pages/default.aspx
[16]https://irtf.org/qirg
[17]https://irtf.org/cfrg
[18]https://standards.ieee.org/ieee/1913/11105/
[19]https://www.gsma.com/aboutus/workinggroups/internet-group-3

approach, sometimes informally addressed as cloud-native, for developing and operating distributed software benefits all the above paradigms. Several platforms and technologies are available for managing these entities' creation and life-cycle management. Probably, the current de facto standard in terms of orchestration platform is Kubernetes (section 4.1).

An essential aspect of SDI is security in terms of software and networking. The first is not thoroughly addressed in this thesis work because the focus is more on cryptographic aspects that are more relevant to network security and communication. Nevertheless, both are pivotal for SDI and are often deeply tied to each other. QC, in this regard, jeopardises most of the high-level security protocols adopted by SDI to build secure communication among their physical and virtual components and applications. Examples of these protocols for the reader are Transport Layer Security (TLS), Internet Key Exchange (IKE) for Virtual Private Networks (VPNs), and Secure Shell Protocol (SSH).

Because of this, integrating quantum-resistant mechanisms in SDI, for instance, by introducing QKD, represents a non-negotiable priority for the evolution of these technologies.

2.6.1 Network Function Virtualisation

This section briefly analyses NFV as it is one of the most recent approaches for building highly flexible and scalable network infrastructures, and telecommunication providers are gradually adopting it. As a remark for the reader, NFV is also relevant for the work presented in chapter 6, where it is discussed a general optimisation problem related to this paradigm in a Security-as-a-Service (SECaaS) scenario. SECaaS is a service model oriented to network security and it provides an end user with highly configurable and scalable security services. One can think of SECaaS as the synthesis of the current most advanced technologies in terms of network infrastructures based on NFV and complex security scenarios, where not only the enforcement of rules and configuration on security functions (e.g., firewall) is enough, but also the detection and the immediate response to an attack are pivotal to assure a consistent security service. Typically Artificial Intelligence and Machine Learning techniques are also introduced to improve and facilitate the detection phase.

Beyond the opening about the motivation for adopting NFV, one can find the roots of his introduction in the practical necessity of the telco providers to go beyond the traditional implementation of network and security functions. Indeed, these functions were usually implemented as a strict combination of special-purpose software and hardware, a strategy that led to the impossibility of cooperation among different vendors' devices, a lack of standardisation of the software for diverse apparatuses, and a massive reduction of the infrastructure's

scalability. Because of this, NFV introduced a decoupling between hardware and software so that general-purpose commodity servers can be adopted as standard hardware, on top of which arbitrary software-based network functions, namely Virtual Network Functions (VNFs), can be run. NFV can also leverage Software-Defined Networking (SDN), a similar paradigm, but in this case, related to the networking and the approach of decoupling the data plane (low-level traffic forwarding) and the control plane (logic for managing the rules to forward the traffic). This is possible by using an SDN controller, which is a central entity that knows the network topology, can run software applications to optimise the logic of traffic flows against chosen metrics, and can enforce the derived policies/rules to the low-level devices through a standard interface (e.g, using OpenFlow protocol).

This is relevant because, with those two paradigms combined, it is possible to manage end-to-end communication between two endpoints, implementing the network functions as VNFs (e.g., firewall, IDS) and connecting them using SDN. Clearly, this requires another level of orchestration and management which is described in the ETSI specification on NFV [55]. The same approach can be simply extended for network security, where VNFs are network security functions, and one obtains the infrastructural base for implementing the SECaaS service model.

Finally, NFV can be related to quantum technologies in different ways. The most trivial is considering the impact of quantum technologies and Shor's algorithm on this paradigm and the repercussion of SECaaS. Another one, which is the core of this thesis work, is related to the countermeasures provided by the Quantum Cryptography techniques, which have to be integrated with SDI and, as a result, in NFV. The third, less obvious, more general, but equally relevant, considers quantum-based optimisation techniques (e.g., using Quantum Annealing) to improve NFV infrastructures with an indirect benefit for security through the SECaaS optimisation.

2.7 Quantum Annealing

Quantum Annealing (QA) is a metaheuristic that can be adopted for solving optimisation problems and leverages the adiabatic theorem [56]. From this theorem, one can assert that a quantum state will stay in its ground state if the Hamiltonian of that system changes slowly enough. These changes are clearly correlated to the gap between the ground state and the first exited state (minimum gap) [56].

One way this can be used for computation is to prepare a quantum system in the ground state of a trivial initial Hamiltonian and change the Hamiltonian slowly to a complex one [56]. Because of this, the system will remain in the ground state and effectively solve the Hamiltonian. As mentioned before, the adiabatic theorem and, in particular, the model known as adiabatic quantum computing is equivalent to the quantum circuit: it can simulate any quantum circuit and is a universal quantum computing paradigm.

Being capable of modelling an optimisation problem as an evolution from an initial known Hamiltonian to a final hamiltonian allows one to solve - not necessarily in polynomial time - complex problems such as in the NP-hard and NP-complete classes (the solution is the ground state of the final or target Hamiltonian).

A possible method of formulating an optimisation problem suitable for a quantum annealer is using a Quadratic Unconstrained Binary Optimisation (QUBO) formulation [56].

D-Wave Systems is a company producing quantum annealer processors which is widely adopted for testing in both academia and industry. The reader can find in the following the general step for solving an optimisation problem with a D-Wave quantum annealer (adopted in chapter 6):

- **QUBO formulation and Graph representation:** this formulation is widely adopted for describing optimisation problems for quantum annealers. It might be an unconstrained optimisation, or constraints can be introduced in the form of penalty and weighted using lagrangian multipliers (chapter 6). This formulation is then converted to a graph in which nodes are binary variables and links the interaction among them.
- **Minor-embedding:** mapping the QUBO formulation on the QPU require an embedding process that can be itself an optimisation problem and for which classic metaheuristic exist. In this process, several physical qubits are used to represent logical variables, and the coupling between them represents the interaction among variables.
- **Programming:** in this phase, one configures the annealer parameters to solve the optimisation problem, such as the weight of the single qubits and the strength of the interaction between them.
- **Initialisation:** initialisation of the known trivial Hamiltonian.
- **Annealing process:** in this phase, there is the transition from the initial to the final Hamiltonian. The optimisation problem is solved.

- **Readout:** following the annealing process, the qubits seen as a quantum state are in a superposition of eigenstates, each representing a possible minimum of the final Hamiltonian. Here the readout of the solution takes place, and the result is stored classically.
- **Resampling:** due to the heuristic nature of the quantum annealing, the annealing and readout phases are repeated to pick the best candidate as the solution.

Chapter 3

Quantum Key Distribution in Software-Defined Infrastructures

This Chapter discusses the adoption of QKD in Software-Defined infrastructures, presenting a complete software stack, namely Quantum Software Stack (QSS), compliant with QKD ETSI standards and capable of easing the integration of QKD in this domain. The low-level layer of this stack offers a QKD simulator to provide a fast and consistent alternative to expensive QKD physical devices during testing. This simulator, even supporting different backend extensions, is based on IBM's Qiskit toolkit. This work has been published in the IEEE Access international journal [57].

3.0.1 Methodology

This section briefly summarises the methodology adopted for integrating QKD in SDI and the work on the QKD simulation. The clear objective of the work is to fill the gap between the advanced capabilities of the current SDIs and the rigid constraints of QKD protocols and devices. Moreover, the QKD simulation is required to test both key exchanges between nodes within a QKD network and specific attack scenarios. This work required two iterations, followed by a first and a second software release. The first has been a joint iteration regarding the QSS and QKD simulator, as described in this chapter (chapter 3). The second iteration, described in chapters 4 and 5, involved the QSS and QKD simulator separately. More in detail, the first iteration comprised the following activities:

1. analysis of the QKD background, techniques, protocols, and ongoing standardisation efforts.

2. Study of SDIs and definition of a threat model.
3. Design of the preliminary application, breaking it down into several components and well-defined interfaces.
4. Selection of the technologies and strategies to adopt for the development according to the state-of-the-art technological landscape.
5. Implementation of the first prototype with essential functionalities such as PTP key exchange, basic QKD simulation support, limited orchestration support, key management, key storage and public channel authentication.
6. Testing PTP exchange capabilities, particularly regarding the QKD simulator and attack simulations adopting appropriate metrics such as key rate and QBER.

The second iteration for the QSS involved:

1. code refactoring to support an asynchronous approach.
2. Support for routing on large QKD networks.
3. Support for Multi-Hop (MH) exchange with trusted repeaters.
4. Orchestration capability and deployment on cloud-native infrastructures.
5. Testing of end-to-end performance and bottlenecks.

The second iteration for the QKD simulator involved:

1. support for large networks.
2. Code refactoring to support an asynchronous approach.
3. Automatic QKD network deployment and reconfiguration.
4. Testing to show the improvement of PTP exchange performance.

3.0.2 General SDI quantum threat model

The majority of PKC techniques lie on the premise that certain mathematical problems are computationally unmanageable for modern classical computers (e.g. prime number factorisation in RSA). Quantum computing disproves this premise, rendering such issues "simple," i.e. reducing their mathematical complexity to polynomial (for instance, Shor's method may be used to attack RSA). Less revolutionary, but nevertheless a major risk-increasing factor for

symmetric-key cryptography, is the power of QC to study a 2^n-dimensional solution space in $2^{(n/2)}$ steps.

An SDI consists of several virtualisation actors and agile deployment components (such as hypervisors, orchestrators, and containers) that interact in a complex form and may be attacked (both in terms of single components and the communication channels among them). The majority of these attacks' defences rely on authentication and confidentiality countermeasures, which are often built directly or indirectly on PKC and PKI. Assuming QC is accessible, no SDI can depend on conventional PKC since private keys based on standard methods would become insecure and susceptible. The raison d'être of the QKD is to offer Information-Theoretic Security to the key distribution process, allowing for strong security even after the introduction of real quantum computers.

Due to the intrinsic nature of quantum cryptography, a QKD based on quantum cryptography is especially susceptible to Denial-of-Service (DoS) attacks. If the error rate of an exchanged key exceeds a certain threshold across an ideal channel, this indicates an eavesdropping action. A comprehensive security technique consists of discarding the key and re-executing the exchange. Thus, it would suffice for an attacker to "read" the transaction to refuse service. While some degree of DoS possibility is accepted in nearly every real-world scenario (e.g., any local wireless communication can be drastically reduced by relatively inexpensive and easy-to-obtain WiFi jammers), a robust architecture must address this aspect with the appropriate countermeasures, such as the presence of Intrusion Detection and Prevention systems to identify and isolate the attack source.

Even though the adoption of QKD guarantees Information-Theoretic Security in theory, implementations might contradict theoretical assumptions; hence, many developed QKD systems include security vulnerabilities. The proposed architecture is independent of the actual implementation of quantum devices. Therefore, while we do not cover actual device issues, we recommend that effective security implementations back theoretical key strength. Specifically, there is a huge amount of work analysing actual device vulnerabilities and how to handle realistic defects to eliminate physical side channels [35].

In addition to the issues regarding cryptography, a software-defined infrastructure still faces typical key management and usage problems. Common key vulnerabilities include insufficient entropy (the key is too short and/or was generated by a poor RNG), inappropriate reuse, both in terms of use in multiple scenarios and encrypting a large amount of data (since it compromises forward secrecy and increases the likelihood of brute-force attacks and key-relevant information leakage), and 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). In our approach, we assume the availability of either a True Random Number Generator (TRNG) or a Quantum Random Number Generator (QRNG) capable of producing random numbers with a high entropy source. Similar to the preceding point, a real implementation of these modules may have security vulnerabilities; however, this is outside the scope of our investigation. In our architecture, key reuse is handled with flexibility. Due to the fact that we present several possibilities based on the situation, a suitable security policy should be suggested to ensure an adequate lifespan for the generated keys, as is the case in modern organisations. Operating systems and applications may include implementation defects, but this is outside our analysis's scope.

Even with strong QKD, a key may be compromised (e.g., owing to an implementation flaw), hence a process to destroy it (i.e., safely erase the key and its traces beyond recovery) must be supplied. This last component might be difficult because it clashes with the necessity to document the key lifespan in order to detect potential security breaches, implement revocation and destruction options, and conduct forensics investigations. The implementation of QKD for generating the master secret to generate cryptographic keys has no effect on this element, which is governed by the organization's established regulations.

Key access must be restricted on a need-to-use basis and reinforced by separation-of-roles-based access control in order to prevent unauthorised access to the key vault (e.g., an entity that uses a key should not be the entity that stores that key). Secure backup and recovery methods must ensure key recoverability after unintentional loss. Best practice recommends internal custody of keys (or service at a comparable degree of security) maintained by central key storage technology. The Quantum Key Server, a flexible component that may adopt numerous configurations to comply with organisational security requirements, such as log level and approved QKD methods, manages this key feature of our design. It can also leverage different current safe secret management systems (e.g, Vault, in the concrete testbed described in this work).

3.1 Quantum Software Stack architecture

Our architecture's primary objective is to enable the implementation of QKD in Software-Defined infrastructures (e.g., cloud environments). Creating the software stack presented in Figure 3.1 advances this objective. Four logical levels and components represent the issue of key exchange between two cloud instances or services:

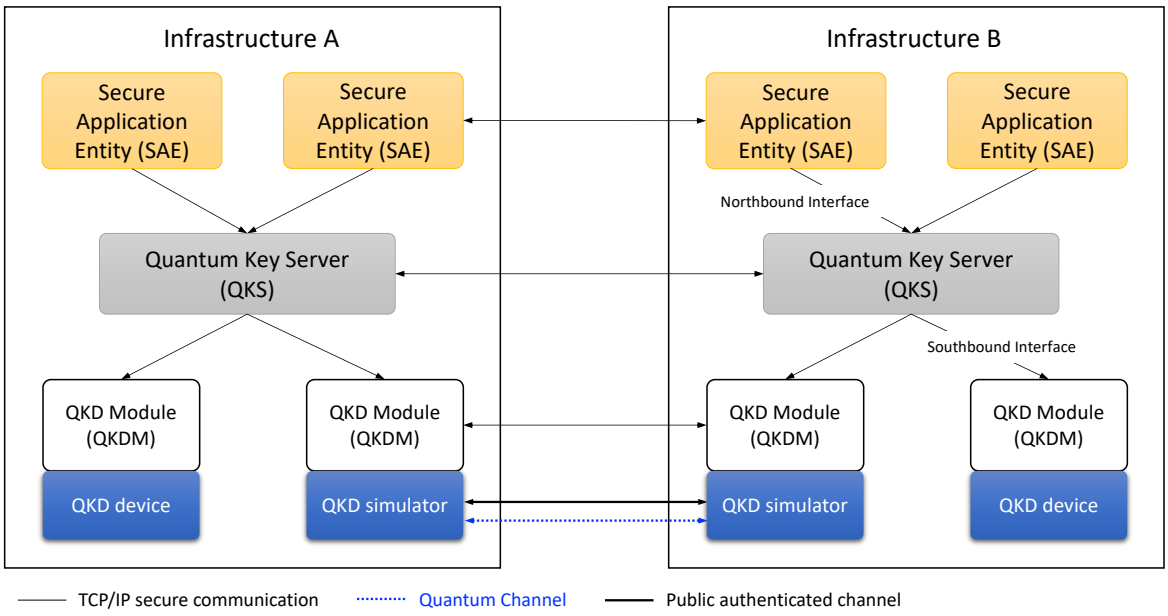

Fig. 3.1 QSS high-level architecture.

- **QKD device:** the physical QKD device that can execute QKD protocols on a real or simulated quantum channel. Since we built and implemented a QKD simulator that could be incorporated into our software stack, the current level may be also referred to as the "QKD simulator".
- **QKD Module (QKDM):** an abstraction of the low-level QKD device and offers a common interface for communicating with various devices. Regardless of the technology used to execute the QKD, further monitoring, management, and use capabilities are offered for the underlying devices.
- **Quantum Key Server (QKS):** at this level, QKD management across several nodes occurs. Specifically, coordination between distinct QKSs is essential to set up the key exchange process across dispersed infrastructures. QKS offers an interface for high-level security applications that need cryptographic keys and identifies the optimal route to take in situations involving several nodes between the two ends of the key exchange.
- **Secure Application Entity (SAE):** this is the highest level and represents the security applications willing to use the QKD for certain objectives (i.e., to set up a VPN).

The presented architectural framework adheres to ETSI standards [53]. Considering a point-to-point QKD connection, as seen in Figure 3.1, two infrastructures may be outfitted

with QKD devices able to exchange keys at a certain pace. These devices must share a quantum and a classical authorised channel to execute a QKD protocol. In our approach, the QKD device (or simulator) represents the actual system, and the QKDM provides an abstraction for running it.

If we go to a more sophisticated situation in which a full QKD network is in place, then we must handle several facets of this network's coordination. In addition, we must offer methods to effectively store the exchanged keys, routing features to permit the exchange across all possible SAE combinations, and monitoring systems to track important QKD-system lifecycle events. Here, the QKS comes into action. Each infrastructure must expose at least one QKS, the central management unit of the QKD systems. SAEs operating atop the infrastructure can only see the QKS as a component, which from their viewpoint, delivers QKD-as-a-Service. In reality, they are capable of exchanging quantum keys with all other peers inside the QKD network.

The underlying QKD network architecture might change dependent on infrastructure decisions at a basic level. Switched QKD and trusted repeater networks are two standard implementations [46]. An SDN controller dynamically modifies the paths between endpoints (QKD devices) in a typical switched QKD network. A network with trusted repeaters generates an alternate chain of intermediary trusted nodes that must be visited to complete the exchange. Regardless of the topology, it is assumed that any device using the QKD protocol utilises both a quantum and a classical channel.

If physical devices enable many connections simultaneously (i.e., multiple quantum channels), then numerous QKDMs might be associated with the same device. The fundamental concept is to establish an abstraction in which a QKDM pair identifies a single key stream through a quantum channel. These key streams and QKDMs may be effectively managed by a QKS tasked with determining when a key exchange must begin, end, and when higher-level applications must obtain keys. In our design, we identified two key interfaces: the Northbound Interface and the Southbound Interface. In turn, they enable SAEs-QKS and QKS-QKDMs communication. In sections 3.2, 3.3, and 3.4, we describe the subcomponents of the software stack in further depth. In subsection 3.5.3, we discuss the interplay between them in full.

3.1.1 Use case scenarios

This section discusses the possible uses of our software stack as well as its underlying assumptions and constraints. Currently, our software stack consists of Docker containers deployed on an SDI using Docker Compose. Our development is a cloud-native application

that can run on minimal infrastructures (i.e., edge-, fog- computing and IoT scenarios). This enables the infrastructure to operate QKD as a cloud service needing on-demand keys (from a security application perspective).

The end-user of this service is obviously an SAE inside the infrastructure. Almost certainly, a security application needs the QKD keys to establish a secure channel with another SAE in the same QKD network. This procedure may use protocols like TLS and IKE [58]. These protocols must be altered in order to be improved by QKD, and our technology might ease this process by offering a controllable method of accessing the keys. Even in this instance, the arbitrary combination of these protocols with QKD must be handled carefully from a security standpoint (chapter 2).

Integration of our software stack with Infrastructure-as-a-Service (IaaS) platforms such as OpenStack or container orchestrators such as Kubernetes is an even more viable use case. Similar to OpenStack, this sort of platform already handles secrets, keys, and certificates using some of its unique services (such as Barbican[1]). In addition, OpenStack has VPN-as-a-Service and other tools for creating site-to-site VPNs with other data centres. These utilities are integrated with Barbican to facilitate the administration of the cryptographic data needed by the protocols (e.g., IKE). The integration of our software stack within the scope of these platforms and the provision of a means for these utilities to use QKD-exchanged keys directly might be a viable option. Due to the simplicity and adaptability of our system, this kind of integration is both viable and straightforward. In the following Sections 3.2, 3.3, and 3.4, the assumptions and constraints of the existing programme, including the available authentication procedures, are outlined. Regardless of the scale of the infrastructure and the unique context, our solution provides for a variety of connections and usages as long as we remain within the scope of SDIs.

3.2 QKD Module

This section begins discussing the QKD integration software stack for SDI. The QKD module is an abstraction inside an architecture that seeks to offer a standardised interface for quantum devices. As shown in section 3.4, the QKD node indicates the actual device (or simulator) that executes the QKD low-level exchange (e.g., ID Quantique Clavis[3]). The QKD simulator offers a consistent alternative to using actual devices for this task. QKD Module serves as a wrapper for the QKD node, exposing the standard API conforming with the ETSI QKD

[1] https://wiki.openstack.org/wiki/Barbican

standard [52], and offering high-level functionality to the higher layers, such as obtaining a key exchanged with low-level devices. The precise implementation and vendor-specific devices used do not affect the interaction between the QKS and the QKDM in our approach. Following is a description of the APIs that must be provided and the methods that must be implemented or altered to offer the anticipated quantum device functionality. We created a template of QKDM based on this method and supplied a GitHub repository that could be cloned to develop a new QKDM for a particular device. The concept is that each device or simulator has its own QKDM, but the work required to build it consists of overriding a handful of the methods shown below. QKDM has been created with contemporary cloud-native application development concepts and Docker container technologies in mind. This facilitates the incorporation of the QKDM component into cloud-native infrastructures (such as Kubernetes). Following subsection 3.5.1 contains a comprehensive explanation of the QKDM APIs.

3.2.1 Architecture

The QKDM architecture consists of four distinct submodules, as represented in Figure 3.2: *QKD node*, *Key manager*, *Sync interface*, and *Southbound interface*. In addition to integrating the QKD simulator into this architecture, which is shown as the QKD simulator in Figure 3.1, we also included the quantum device simulation component. This gives a comprehensive overview of the QKDM's potential interactions with the underlying components.

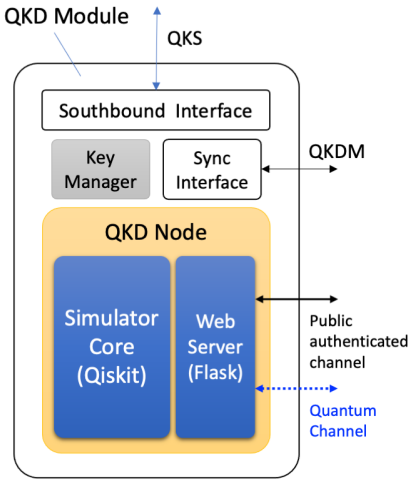

Fig. 3.2 QKD module architecture.

The QKD node represents the quantum device. This component is identical to the one described in section 3.4; it consists of a core component (for QKD protocol simulation)

and a REST interface for communicating with other QKD nodes during the exchange (i.e., for the key sifting process). In a real-world use case scenario, the communication between devices and the execution of protocols are reliant on the technology used. In this situation, the QKD node may be treated as a black box with a supplied interface. The Key Manager is the central component of the QKDM and is responsible for mapping requests from the Southbound Interface to the QKD node's instructions. It is also responsible for maintaining exchange-generated keys, often using resources given by the QKS (see Vault in section 3.3). The Southbound Interface is responsible for facilitating communication between QKS and QKDM. This interface exposes the ETSI GS QKD 004-proposed REST APIs, particularly the first three in Table 3.1. The Sync Interface is ultimately a series of REST APIs enabling the synchronisation of several QKDMs during the key exchange.

3.3 Quantum Key Server

QKDM is an independent component capable of controlling the process of key exchange inside a node, as defined in section 3.2. In theory, an infrastructure node needs just this module for point-to-point QKD integration. The QKDM may, in fact, store the acquired keys in a special secret engine supplied by HashiCorp Vault in addition to the exchange process. The only prerequisite is that the SAE has access to this engine and obtains a fresh key. This may not be possible in complicated infrastructures where separate SAEs need exchanging QKD keys with one or more distinct destinations. In order to centralise the administration of the exchange process, the key storage, the destination discovery, and the assessment of the quickest way to reach them, a software layer is necessary between SAEs and QKDMs. This component is the QKS.

ETSI [53] focuses on the ultimate security applications and anticipates a rigorous binding between two SAEs belonging to two distinct nodes. This implies that once a key exchange is necessary and communication between two QKDMs occurs, a key stream is formed, and only the two concerned SAEs have access to it in order to get keys. The Key Server is responsible for facilitating this binding and controlling the beginning and end of the key exchange procedure. Considering the complexity of software-defined infrastructures and their need for flexibility, our solution takes a new approach. Even though the majority of the logic governing the interactions between entities, interfaces, and a portion of the data model is preserved, we modify the function of the Key Server and create our QKS.

The QKS functions similarly to a SAE in that it is responsible for choosing whether or not a given stream should be generated and is the notional ultimate "endpoint" of all key

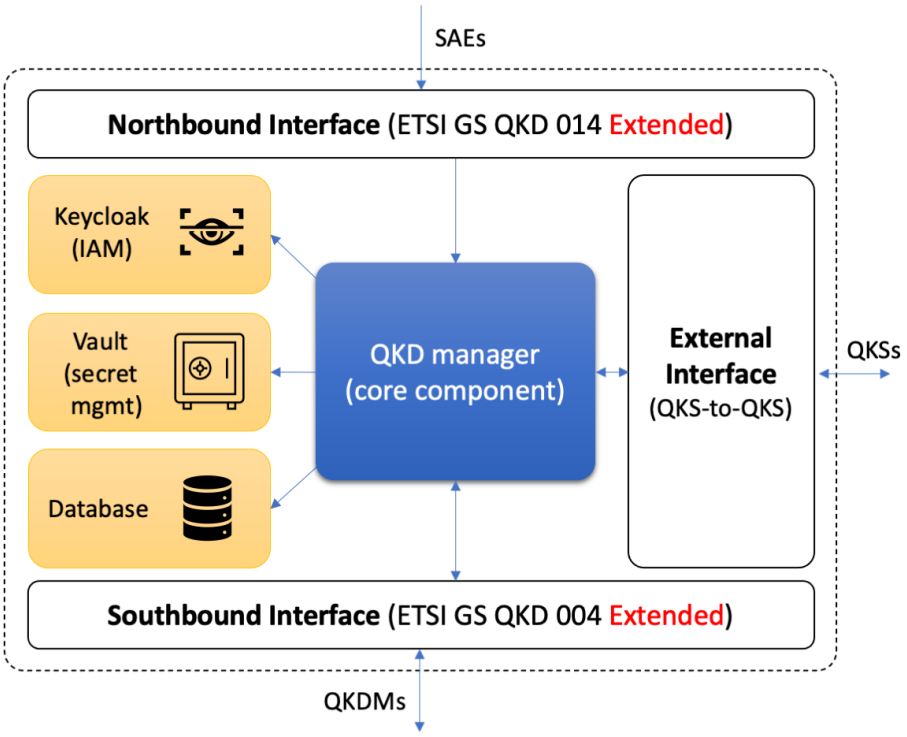

Fig. 3.3 Quantum Key Server architecture.

exchanges towards the node; however, it is also responsible for managing the exchange and coordinating with the other QKSs. In our approach, the QKS is a kind of middleware that gathers all the keys from all the destinations and distributes them to all the registered SAEs, irrespective of any binding that may exist between two particular SAEs. This enables the continual exchange of keys between QKDM PTP connections on QKD networks and the prospective usage of those keys for each SAE-to-SAE pair on the same physical link. This is a significant benefit for a virtualized environment in which keys may be readily distributed across many virtual instances.

The QKS was also built as a cloud-native application that could be seamlessly incorporated into a contemporary infrastructure environment. Noteworthy is its interface with the QKDM since the QKS might handle several QKD modules that must register with it prior to any operation. Before gaining access to the key server and requesting any key, the SAEs must also be authenticated and authorised. In subsection 3.5.3, the reader can find further information on the relationship between each component. Following is a description of the QKS architecture.

3.3.1 Architecture

In Figure 3.3, one can observe the overall QKS architecture, its three interfaces and four main components that are described in the following:

- **Northbound interface:** it supplies the SAEs with an expanded version of the ETSI API in order to query the QKS. This pertains mostly to the procedure of obtaining keys from SAEs.
- **Southbound interface:** it enables the QKS and QKDM to communicate with one another. According to section 3.2, the implementation efforts are mainly on qkdm. However, this interface is bidirectional, and QKS exposes a number of functions to service QKDM.
- **External interface:** it acts as a synchronisation interface among QKSs. It is essential for the exchange of data such as the KSID and routing information.
- **QKD manager:** component responsible for providing all fundamental functionalities, such as handling Northbound Interface requests and registering new QKDMs. This is also engaged in the process of key aggregation, which serves the top layer several keys of variable length.
- **Keycloak**[2] **(IAM):** extra component that is beneficial in situations involving a high number of SAEs and QKDMs. Keycloak offers IAM capabilities that enable the authentication and authorization of SAEs and QKDMs. This is beneficial for both the Northbound and Southbound Interfaces and may be simply extended to the External Interface.
- **Vault**[3] **(secret mgmt):** an HashiCorp solution offering a versatile and scalable method for handling secrets. This utility enables the development of distinct secret engines (one for each QKDM) in which to store the keys.
- **Database:** a component used to hold information about the whole QKD exchange process, registered QKDMs, and supported QKD protocols.

Keycloak and OpenID connect are used for the authentication and authorization of Southbound and Northbound interfaces. For the interaction between several QKSs, we have not

[2]https://www.keycloak.org
[3]https://www.vaultproject.io

yet developed these techniques. Although, we might simply expand the Keycloak solution also in the case of the External Interface. In a "quantum scenario", even for this interaction, the communication across a secure channel, within a QKD network, should be protected by a set of quantum-resistant algorithms and protocols. In the case of the authentication, we examined numerous alternatives that we would like to incorporate with our solution. First, we examined integrating post-quantum algorithms, such as SPHINCS$^+$ and NTRU, to set up a secure TLS channel among QKSs. In this case, Keycloak might still be easily adopted for authentication and authorisation. Also adopting TLS-PSK, which may be used with the QKD keys already created at the lower level, is an alternative method. This leverages the QKS acting as a SAE and demanding additional keys for the communication with other QKSs.

Even the connection between the multiple infrastructure layers (i.e., QKS and QKDM) must be protected. According to ETSI GS QKD 014 [53] that connection might depend on the conventional TLS v1.2 as a minimum requirement. We disagree, believing that quantum-resistant methods in a quantum environment should also care about communication inside the infrastructure. In addition, it is not required that the QKS needs to be placed on the same physical node as the quantum device, and this leaves room for various attack tactics. Even this case can be treated as the external interface.

In subsection 3.5.3, we explain more elements of the interplay between our solution's components. The QKS's code is accessible on GitHub[4]. This solution is entirely based on a cloud-native methodology and makes use of Docker technologies. Each subcomponent has been represented as a distinct Docker container. We further developed a Docker Compose[5] descriptor to allow easier deployment. In conclusion, we additionally discuss the QKS APIs in subsection 3.5.2.

3.4 QKD simulator

To continuously evaluate our solution and provide a novel approach to the modelling of QKD protocols, we built and tested the following QKD simulator. The present version of this software is in its infancy and offers the bare necessities for testing the BB84 and E91 QKD protocols. Even if the low-level simulation component must be improved, our design principles enable us to adapt and grow our simulator to include all the necessary features for a full-fledged QKD simulator. In theory, it may also be expanded to emulate protocols in the larger area of Quantum Networks that go beyond QKD.

[4]https://github.com/ignaziopedone/qkd-keyserver
[5]https://github.com/docker/compose

The design of the high-level QKD simulator is illustrated Figure 3.4. Each component has been developed as its own Docker container. This enables them to operate in lightweight virtual instances and communicate over traditional TCP/IP networks. Using Docker technology as *Container Runtime (CR)* offers a straightforward method for scaling the simulation over several infrastructure nodes. Considering this solution to be installed on a Kubernetes cluster expedites the testing of complicated QKD network situations. The reader may appreciate a summary of the QKD components:

- **QKD node**: a component representing the QKD device. This is one of the parties involved in the key exchange as defined by QKD (e.g., Alice, Bob). The QKD node is outfitted with software capable of simulating both the qubit encoding and quantum state operations. It provides mechanisms for serialising qubit-related data and exchanging information with other components over a TCP/IP network.
- **Quantum channel and Eve**: a particular QKD node designed to reproduce the specific effects of certain entities acting on a quantum channel (e.g., noise, eavesdroppers). It operates on the qubits encoded by QKD nodes using the particular representation.
- **Entanglement pairs generator**: a special node that provides pairs of maximally entangled qubits that are still compliant with the chosen representation. This is relevant within the context of protocols based on entanglement.
- **QKD Simulator Manager**: a component that implements a central management unit that collects simulation data and provides a convenient interface for initiating protocol simulations or configuring QKD nodes. Even though they are still in their infancy, both command-line and graphical interfaces have been offered.

We utilise Qiskit to simulate quantum encoding, manipulation, and measurement, as well as to recreate quantum phenomena (chapter 2). The fundamental concept is to use *State Vectors (SVs)* for the qubit encoding process and *Quantum Circuits* to simulate the development of these SVs in response to Alice, Bob, and Eve's interaction. In our scope, the first two act as parties participating in the key exchange, while the third functions as a quantum channel eavesdropper. We cover many features of our simulation platform regardless of the protocol utilised. Afterwards, we provide both the BB84 and E91 protocol implementations.

A first noteworthy similarity between the two protocols relates to the qubit encoding, which we exploited to generate an ensemble of qubits using the class `Statevector`. This class lets a qubit vector be initialised with particular values (e.g. $|00000\rangle$) and offers a

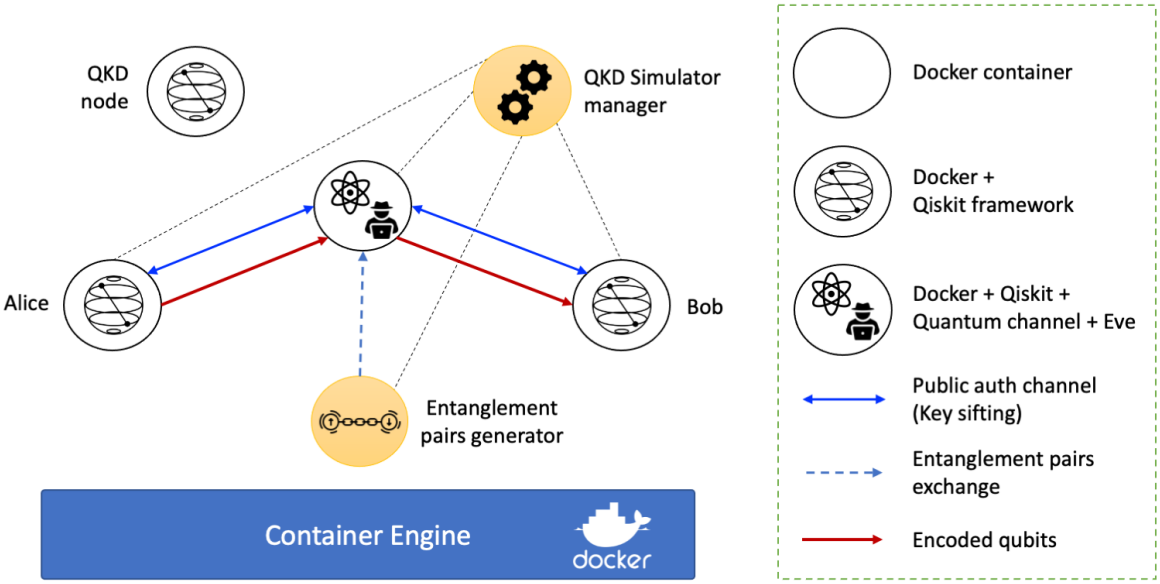

Fig. 3.4 QKD simulator high-level architecture.

method to evolve its quantum state given a certain quantum circuit. In addition, there exists a technique for measuring the final state. Only these three macro operations are necessary to emulate our QKD protocols.

Each component's capacity to communicate through a TCP/IP network for the exchange of quantum and classical data is also vital. The plan is to serialise SVs objects and transmit them over a traditional network using HTTP at the application layer. The Python package `pickle` for serialising objects was used for this purpose. After serialising an SV object, we may insert it into the body of an HTTP request and transmit it to another entity within a QKD network. This technique applies to any communications inside our simulated network that entail the exchange of quantum bits. In our approach, we use Flask[6] web servers to offer a REST-based API for peer-to-peer communication. The usage of HTTP and the Flask web service is optional and readily modifiable in the future, allowing for the implementation of a tailored application-level protocol.

As shown in Figure 3.4, any communication between participants (Alice and Bob) is mediated by an entity that represents both the quantum channel and a potential attacker. This is due to the fact that, directly on a TCP/IP network, we could not simulate the quantum effects occurring on transmission nor manipulate qubits in the manner of an attacker via a quantum channel. As a result, we suggested a new entity, Quantum Channel and Eve, capable of applying these effects on quantum bits. Without a doubt, this method might be expanded

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

by adding other intermediary nodes between Alice and Bob and allowing them to influence the qubits freely. The only feature that both QKD nodes and these special nodes' must have is a standard communication technique with the same qubit format.

In chapter 2, we stated that a public classical channel is required to share extra information during the QKD exchange process. This is valid for all the available protocols. This channel must be authenticated using a mechanism that is flexible and scalable enough to accommodate real-world distributed use case situations. We developed and implemented two distinct strategies: post-quantum algorithm SPHINCS$^+$ and AES with Galois/Counter Mode (AES-GCM) authenticated encryption. The first option involves signing communications sent via a public channel. This is a very flexible method since it does not need Alice and Bob to discuss secrets beforehand and simply requires knowledge of the counterpart's public key. The development of a new "Post-Quantum PKI" with post-quantum X.509 certificates provides a scalable solution to the authentication challenge. It still depends on computational assumptions, and post-quantum methods, notably SPHINCS$^+$, have not yet been standardised. Fortunately, authentication must be supplied only during the QKD exchange procedure, which means that no relevant information about the key could be retrieved from data stolen from the public channel, and no attack could be carried out using this information in the future.

The second method includes the use of an AES-GCM-secured encrypted and authenticated channel. This might be accomplished using a pre-shared key that is rotated using a portion of the key transferred during the QKD procedure. Under the premise that a certain key length is used, this approach is likewise quantum-resistant, but it requires initial pre-shared secrets and a priori understanding of those secrets among participants. Additionally, this method is susceptible to DoS attacks. In fact, if an attacker causes the exchange procedure to fail repeatedly, the parties will exhaust all pre-shared key material, resulting in a denial of service. This might be avoided with a backup technique whereby, in the event that the pre-shared key is compromised, a new secret could be derived using public-key cryptography.

The QKD simulator manager is responsible for overseeing and coordinating the exchange of keys. This module provides both a command line interface (CLI) and a graphical user interface (GUI) to initiate the exchange process between parties, define the settings of this exchange (e.g. protocol, key length, eavesdropper presence), and display the simulation results. This component might, in theory, serve as an orchestrator for the whole recreated QKD network. This implies that it might possibly add and set up additional nodes. The present version is still in its infancy and only supports two protocols for managing Alice and Bob's trade. Nevertheless, in light of the technologies that have been used, the simulator manager's aspirations for these elements are reasonable.

Included inside the QKD node is a module for simulating a QRNG. This enables the reproduction of the creation of random bits for the transferred key. The module's basic implementation consists of a circuit operating on an initialised $|0\rangle$ quantum register. The evolution of the circuit produces n qubits in the state $(|0\rangle + |1\rangle)/\sqrt{2} = |+\rangle$ when a set of Hadamard gates is applied to this register ($H^{\otimes n}$), where n is the number of qubits. If we measure all of these qubits in the computational basis, we will receive either 0 or 1 with a probability of 50 percent: this is the QRNG. Since this module regularly influences the performance of the whole system, we also accounted for the possibility of using a traditional pseudo-random number generator.

The last important factor is the expansion of our simulator to handle more protocols. The present version of our software offers the abstract class `qkd` with the following overridable methods:

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

Developing a new version of our simulator that supports different protocols by simply overriding those functions while retaining all communication-related and underlying Qiskit framework capabilities is feasible.

Moreover, we are not even bound to the simulation framework we use. Indeed, we might enhance the QKD node to enable several frameworks, such as those listed in section 3.7. Then, based on the protocol and simulation needs, the most appropriate libraries may be used for the particular implementation. For simplicity, we made available on Docker Hub[7] a Docker image including all the current software needed by the QKD node. This image might be used to install many QKD nodes and execute the simulation code for a particular protocol. This Docker image might be extended to allow the simulator to embrace more frameworks. Aside from this, it is permissible to use varied orchestration systems for deploying this image and generating a simulated QKD network of any design. We automated the deployment of our solution for this particular purpose using Docker Compose. Code and documentation for the QKD simulator are available on GitHub[8].

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

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

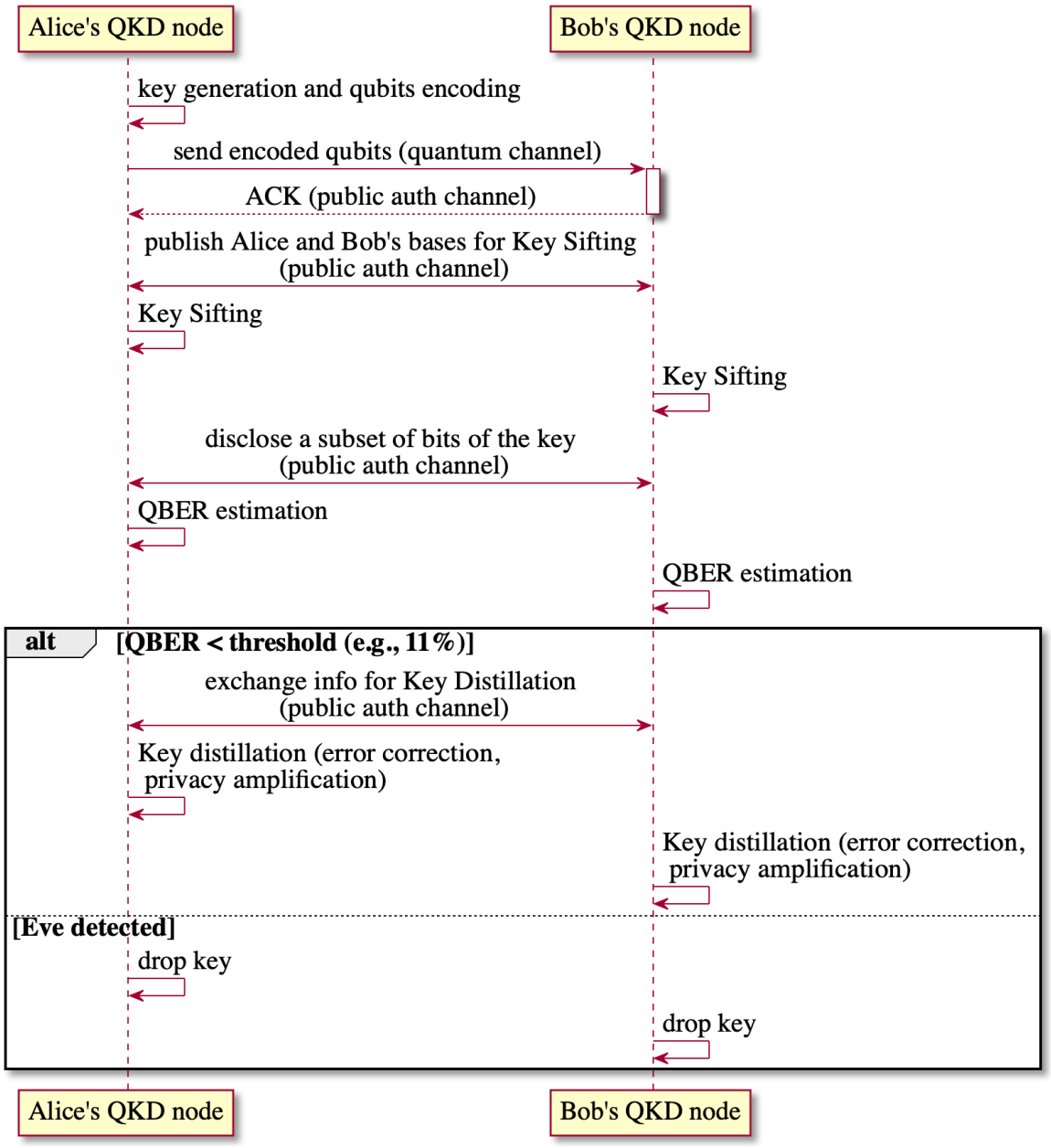

Fig. 3.5 BB84 simulation workflow.

3.4.1 BB84 implementation

In this subsection, one can find our Qiskit implementation of the BB84 protocol, starting from the scenario in Figure 3.4. Figure 3.5 shows the complete workflow of the BB84 simulation. As a reminder for the reader, section 2.5 presents an overall view of the BB84 protocol. Quantum channel and attack simulations require all traffic from Alice to Bob to pass through Eve's container, which is in charge of applying noise, measurement, and other possible effects from the external environment. Each container exposes a single interface on the overlay network provided by the CNI plugin and used by the CR. All traffic, regardless of the channel type (quantum vs classical), flows through this interface to make a node able to communicate.

A simulated key exchange starts only when one of the parties requires it; for simplicity, one can assume that Alice starts the communication. As a first step, Alice generates two sequences of random numbers: one corresponding to the key bits and the other to the sequence of randomly chosen bases. Given these two strings, the encoding process involves the creation of a train of Statevectors which has a fixed length equal to the size of the Qiskit register. This leads to a first important remark: the number of qubits required for a given key length (e.g., 256, 4096) is consistently larger than the one currently supported by the Qiskit local simulator or even the available quantum hardware. Because of this, the reader can appreciate that a mechanism to manage an arbitrary number of key bits is provided in our implementation. In particular, the train of Statevectors provides a straightforward method to divide the complexity of the task and apply the same operations iteratively on shorter quantum registers. Tests on different register sizes (section 3.6) allowed us to choose the best size-performance compromise for our experiments (5 qubits). This size can be heuristically adjusted depending on specific hardware testbeds and experiments.

Figure 3.6 depicts the high-level quantum circuit for BB84 with a quantum register initialised to $|00000\rangle$ in the computational basis (z-basis). As a remark for the reader, one can choose as a basis either the computational or the Hadamard (x-basis) one. Such bases are conjugate; namely, if one encodes a qubit in a z-basis and performs a measurement in an x-basis, he obtains a $1/2$ probability for a 0 or 1 outcome and vice versa. BB84 requires four distinct states, and since Qiskit initializes all states to 0, a bit-flip operation is required to obtain 1 (X gate). Once the string of qubits is set to the randomly chosen values, it is time to apply the correct basis. Given the second random string and the notion of Qiskit initialising all Statevectors in the computational basis, one can switch to the Hadamard one when required by the encoding by simply applying a Hadamard gate (H gate) to the specific qubits. As a result, one gets the four-state encoding in Figure 3.6: 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.

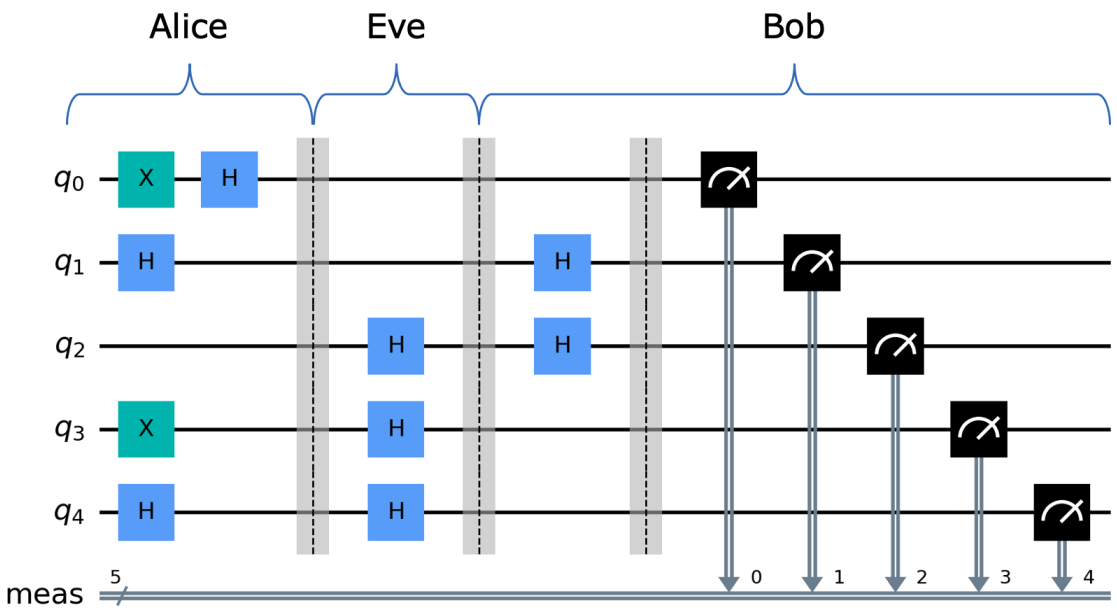

Fig. 3.6 BB84 quantum circuit with 5-qubit register.

After the encoding, Alice sends the SVs to Bob using the `pickle` serialisation process. Bob then chooses random bases and measures the received qubits with them. As a remark for the reader, choosing a basis is equivalent to applying a Hadamard gate. When Alice and Bob make the same choice, they get a valid qubit; otherwise, they may introduce an error with a total probability of $1/4$. This is the principle behind the detection of an eavesdropper (Eve). In Figure 3.6, the existence of a Hadamard gate for Eve indicates that she has chosen a wrong basis. Many approaches allow for simulating an intercept-resend attack in BB84: one can apply randomly chosen bases and measure the resulting states; in our approach, one considers an additional Hadamard gate when Eve chooses a basis different from Alice.

Once Bob gets his measurement results, Alice and Bob start the Key Sifting process by sharing their chosen bases and eliminating the related qubit when a mismatch happens. Figure 3.6 shows how only q_1 and q_3 remain: for q_1, Eve chooses the same basis as Alice, and no error should be detected (Eve cannot be detected); for q_3, Eve chooses a different basis, leaving room for a $1/2$ probability of detection. The intercept-resend attack is further discussed in section 3.6, providing also results of its simulation.

QBER estimation represents the subsequent step after the publication by Alice and Bob of a portion of the sifted key, in this specific case, half of it. QBER is a valuable metric for detecting Eve: in literature [59], a known threshold is 11%; in practice, it depends on the

key distillation process, i.e., adopting *Advantage Distillation* techniques [60] may enhance the performance up to 20%. Since non-idealities and quantum channel noise are not yet completely treated by the simulator, the only error measured comes from Eve. Given that all the key qubits have been attacked independently of each other, QBER should be around 25%. As a remark for the reader, the QBER threshold must consider error correction and privacy amplification. Moreover, both quantum channel noise and Eve's action introduce errors that may overlap, and we shall detect and correct them in the case of noise. This requires being careful in choosing an error correction code and suggests also adopting PA techniques to reduce the quantity of information gained by Eve and the number of usable keys. Error correction and privacy amplification are aspects of the simulator under development. In particular, Cascade protocol [61] for error correction and PA techniques based on Toeplitz universal hash functions [62] are great candidates to be included in the next version of the simulator, even though the general workflow already includes these two steps after the QBER estimation.

3.4.2 E91 implementation

E91 implementation is inspired by Qiskit community[9] and represents a primary prototype that must be enhanced in the coming version of the simulator. Figure 3.7 describes the quantum circuit adopted for the simulation. To get a singlet state (Equation 2.5), one can initialise q_0 and q_1 to $|1\rangle$, apply a Hadamard gate to the first qubit, and employ a *controlled NOT gate (C-NOT)* on the second qubit which is controlled by the first. The reader should notice that a dedicated distributed module to generate entanglement pairs has been provided. Once ready, this module sends the pair to the container acting as quantum channel, which is in charge of performing all the operations on the pair without sharing the representation with Alice and Bob. This is a mere implementation detail since it makes no sense to distribute the entanglement's simulation over different nodes realistically. Because of this, we demanded a central entity to perform all the operations on the initial state by taking only classical information from Alice and Bob, who get the final results after the computation. The forthcoming QKD simulator releases will probably provide more efficient solutions to this problem.

Looking at the details, after the pair generation, the central container performs measurements according to random choices provided by Alice and Bob (section 2.5). Alice and Bob get three choices each, in turn (A_j) and (B_j) that are represented as a quantum circuit in Figure 3.8:

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

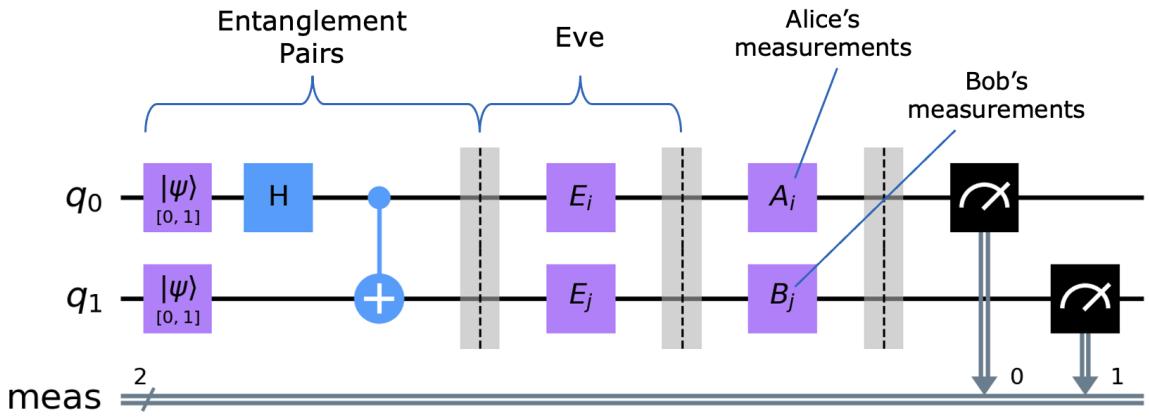

Fig. 3.7 E91 quantum circuit.

- 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 can perform the same measurements on both Alice and Bob's branches. Figure 3.7 does not consider Eve's measurements since they are implicit. Following the measurements, one can split the obtained bits into two groups (section 2.5): one for the final key and one for the CHSH inequality verification. One can compute the value in Equation 2.8, also using Equation 2.7 and Equation 2.6. If the value is close to the correlation (anti-correlation) desired, one can keep the final key; if not, the key must be discarded.

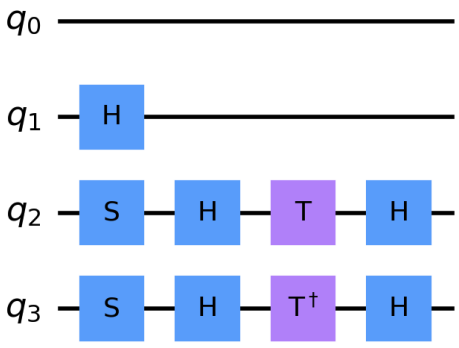

Fig. 3.8 E91 Alice and Bob's possible measurements.

Table 3.1 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	-

3.5 REST APIs and general workflow

This section provides an in-depth discussion of the implementation details, such as the overall workflow in a general use case and the list of the REST APIs exposed by the various interfaces. As a first remark for the reader, this data are also available in the open GitHub repository of the project (mentioned before) and the work published in IEEE access [57]. A second remark lies in the fact that the proposed APIs are an extension of the standard ones from ETSI, hence where one finds the corresponding ETSI method that indicates that those are ETSI-compliant in terms of interface.

3.5.1 QKDM interfaces

The southbound interface (QKDMs-QKS) is composed of the following calls:

- `/api/v1/qkdm/actions/open_connect`: creates a logical key stream between QKDMs, identified by a *Key Stream ID (KSID)* and starts the key exchange process among them.
- `/api/v1/qkdm/actions/close`: ends a key stream flow, preserving the already exchanged keys.
- `/api/v1/qkdm/actions/get_key`: retrieves keys from a key stream given a `KSID` and a `KIDs`, UUID_v4 identifiers acting as handles for the keys.
- `/api/v1/qkdm/actions/get_kids`: yields a set of `KIDs` to compose an aggregate key, given a `KSID` and the `key_info`, which are the characteristic of the preferred key.
- `/api/v1/qkdm/available_keys`: returns the number of available keys.
- `/api/v1/qkdm/actions/attach_to_server`: registers a QKDM to a QKS.

These are the calls for the Sync Interface:

Table 3.2 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	-

- /api/v1/qkdm/actions/stream_create: notifies the KSID to a peer QKDM.
- /api/v1/qkdm/actions/sync_KID: notifies the chosen KID to the peer after the exchange.

3.5.2 QKS interfaces

In the following the Northbound Interface (SAE-QKS):

- /api/v1/keys/{slave_SAE_ID}/status: returns to the master SAE, which starts the exchange, the status of a target slave SAE.
- /api/v1/keys/{slave_SAE_ID}/enc_keys: yields the keys requested by the master SAE. The slave QKS shall be informed to reserve those keys for the served SAE.
- /api/v1/keys/{master_SAE_ID}/dec_keys: returns the keys to the slave SAE.
- /api/v1/preferences: gets settings information, such as log levels, selected protocols, and timeouts.
- /api/v1/preferences/{preference_ID}: modifies current settings.
- /api/v1/information: returns QKS specific information, e.g., QKD supported devices.

As a note for the reader, registering a QKDM to a QKS implies giving the QKDM restricted access to the centralised database and Vault. Finally, the External Interface (QKS-QKS) provides these calls:

- /api/v1/saes/{slave_SAE_ID}: checks SAE reachability.

- `/api/v1/kids/actions/reserve_key`: reserves a key between two SAEs.
- `/api/v1/keys/actions/send_KSID`: forwards *KSID* among two QKSs.

3.5.3 General workflow

The general workflow can be divided into three phases: QKDM registration, key exchange initialisation, and key request from high-level applications. Figure 3.9 provides a view of these steps, considering an Alice-Bob key exchange setup.

Initially, Alice and Bob's QKDMs register to their QKSs by using a `register_module` call and providing information about the reachable destinations. QKSs set up the local environment granting the QKDMs restricted access to the centralised resources. As a note for the reader, only authorised QKDMs can perform such a request, i.e., they must be registered to Keycloak.

After the registration, QKDMs are able to exchange keys with each other and store them as secrets in Vault. As an optimisation, Vault also acts as a buffer continuously refilled with fresh key material even in the absence of requests. The QKS calls the `open_connect` on the QKD module to start the exchange, performs a second check on the destination and sends a `stream_create` to the other QKS, adding QoS-related information and a fresh `KSID`. This trigger a specular behaviour on the second site, which is then ready to accommodate the exchange. Since the requirement for aggregated keys is common, the reader can observe that most of the aggregation strategies can be implemented at the QKDM level.

Finally, Alice and Bob may request a key at the application level. Once the request hits the QKS, bearing additional information regarding the nature of the desired keys, the QKS asks for a set of KIDs. Once available, KIDs are reserved on the peer QKS through a `reseve_key` so that no other SAEs can claim them. The store-and-reservation process extends the standard, saving key material and adopting a robust sharing mechanism. When these preliminary operations are completed, a `get_key` is performed to propagate keys from the QKDM to the SAE through the QKS. Then, an aggregate of the key handles, namely Aggregate Key ID (AKID), is sent to the second SAE, which calls a `get_key_with_ID` to retrieve the key material.

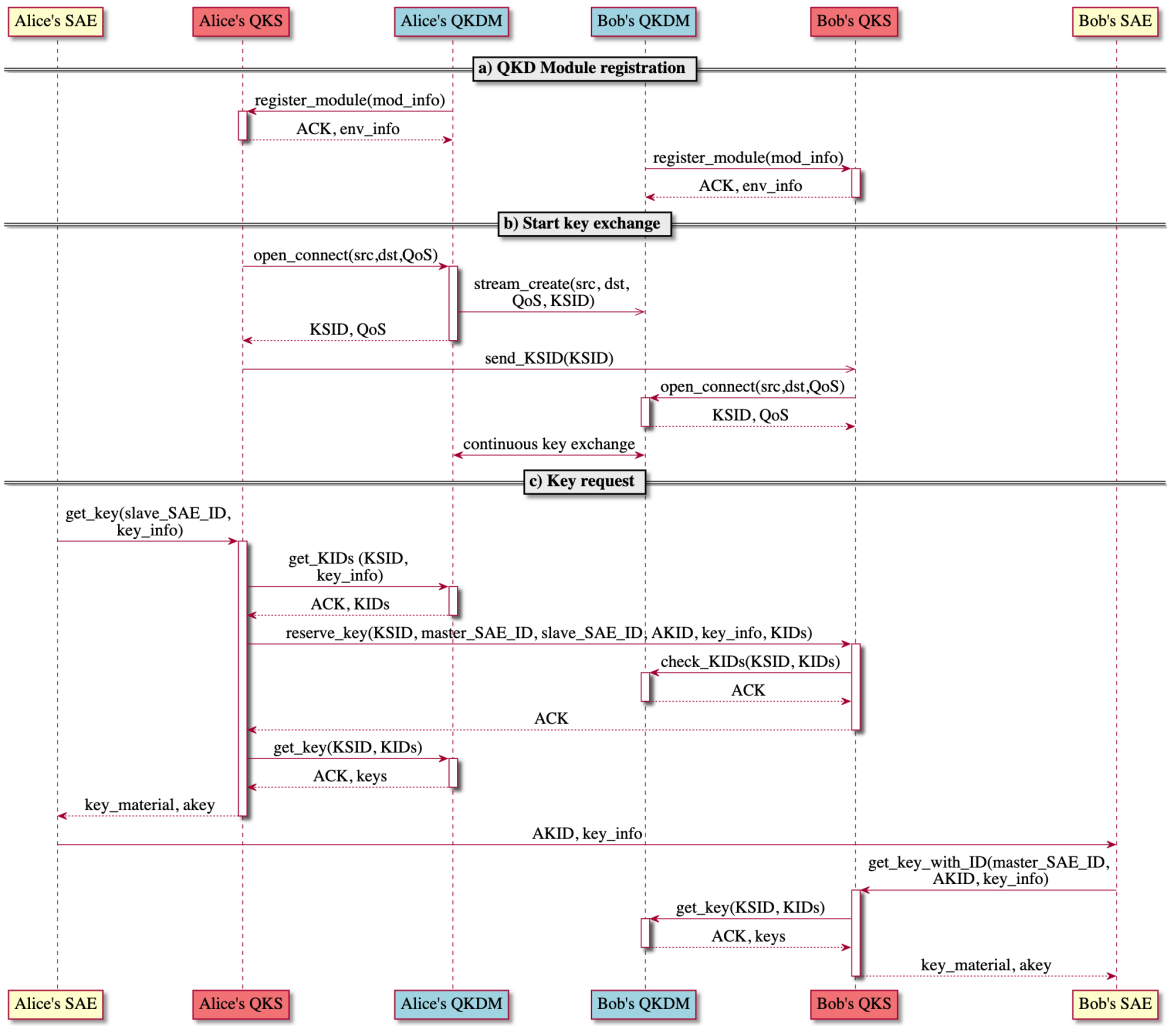

Fig. 3.9 Workflow of the complete solution: a) QKDM registration phase; b) key exchange; c) key request from a SAE.

3.6 QKD simulator Testing

The QSS version of this chapter is a preliminary version of the latest and more advanced software stack, presented in chapter 4. Nevertheless, the reader may appreciate all the underlying design principles which impact and sustain the following advanced version. Because of this, one can only find quantitative results about the QKD simulator in this section, leaving the chapter 4 to discuss the performance of the other components. All tests presented in this section have been performed considering: two bare-metal nodes with an Intel Core i5-5300U CPU @ 2.30 GHz, 16GB of RAM, an Ubuntu 20.04 LTS server Linux distribution, Docker Engine CE v20.10.1, and Docker Compose v1.27.4.

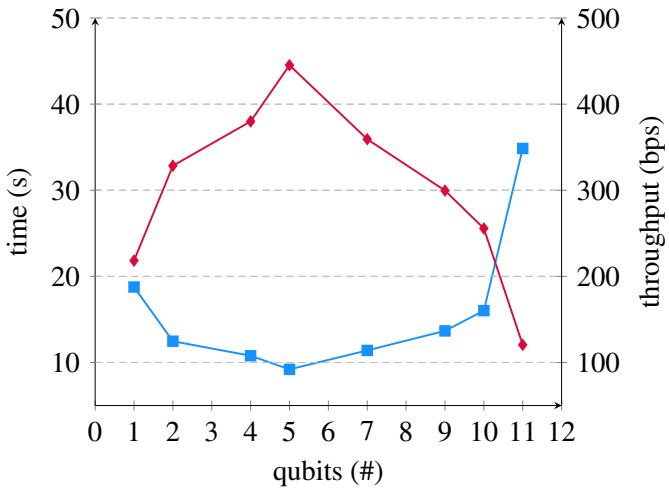

Fig. 3.10 Execution time for a 4096-bit key exchange depending on the qubit register size.

Table 3.3 BB84 key exchange rate and time. All values refer to a 4096-bit key exchange except the last one, which is calculated on a 2048-bit key exchange.

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 a note for the reader, QBER and throughput (bit-rate) of the exchanged keys are the metrics involved in all our tests. Figure 3.10 starts analysing the impact of the Qiskit quantum register size on the performance. The first result was to detect our experiments' optimum register size (n=5), which corresponds to a trade-off between performance and computational complexity. As a remark for the reader, IBM Q offers a remote quantum circuit simulator of up to 32 qubits; our local system can barely manage twelve qubits.

Assuming the optimum register size and aiming at measuring execution time and key rate, other experiments have been conducted. In particular, depending on the adoption of a QRNG and a specific authentication method, one can distinguish the following cases: QRNG and SPHINCS$^+$; SPHINCS$^+$ only; AES-GCM only; raw exchange without authentication and QRNG (baseline).

Figure 3.12 and 3.11 show how the key rate grows with the key size, reaching a plateau once the system resources are drained. Table 3.3 gives an overview of all configuration plateaus.

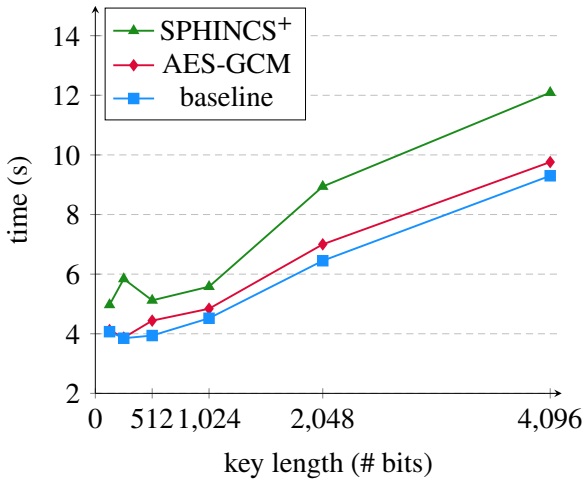

Fig. 3.11 BB84 key exchange time depending on the key length.

As remarks for the reader, QRNG simulation is time-consuming since one needs to simulate several quantum circuits to get every single key. Moreover, key generation can be considered an orthogonal problem. Because of this, QRNG has been excluded from the other tests after showing its impact on the performance. As the reader may expect, the baseline is the one showing the best performance, while adding authentication leads to a necessary overhead. AES-GCM performs better than SPHINCS$^+$ (SHAKE256-128f variant), which is another expected result since the large time required for signing and verifying, i.e., 275 ms and 11 ms on average.

Finally, Figure 3.13 shows the results - consistent with the theory - of a simulated intercept-resend attack. As a note for the reader, this is the simple version of the attack where all qubits are attacked independently.

In more detail, QBER is measured for different amounts of attacked qubits during an 8192-bit key exchange. As the number of the attacked qubits grows, the QBER increases, reaching the value of $1/4$ when all qubits are attacked. This result, in the absence of noise and under ideal conditions, follows the theory and proves the consistency of the simulator.

3.7 Related work comparison

The reader may appreciate a comparison of this work with the existing literature. One can divide this evaluation into two aspects: the integration of QKD in software-defined infrastructures (QSS) and the QKD simulator.

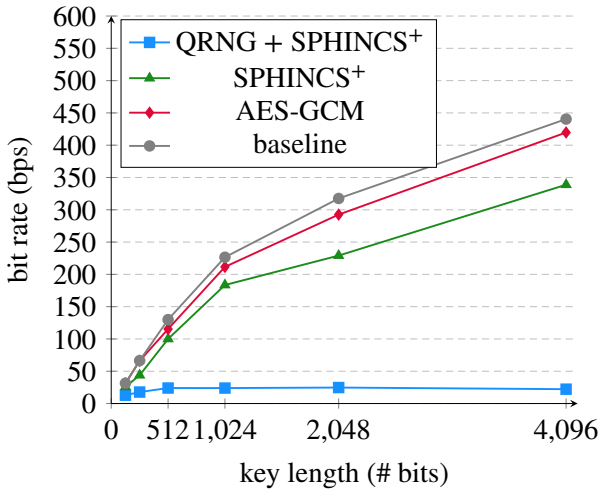

Fig. 3.12 BB84 key exchange bit rate depending on the key length.

As an example, a first point is explored in [63], where the authors give an intriguing perspective from the viewpoint of a telecommunications operator. They demonstrate QKD integration in current infrastructures by employing Software-Defined networking (SDN) and provide valuable ideas on actual use case situations. The authors of [64] present a method for combining QKD with SDN-controlled optical switches in an NFV context. Their study focuses primarily on reconfiguring the quantum channel, e.g., using programmable switches rather than the infrastructure's key management. Using SDN, Lopez et al. [65] demonstrate a viable QKD integration over a conventional telecommunications network. The authors of [66] describe the *SDQaaS* framework, which implements a QKD-as-a-Service (QaaS) strategy in which the QaaS functions are built inside an SDN controller.

These works use SDN to decouple the control and management plane from the data plane, i.e. key transmission, within QKD networks. This technique permits dynamic modifications in a QKD network, maximising the use of available quantum channels. Our approach is distinctive because it focuses on the software stack necessary inside an infrastructure to supply quantum keys to security applications. This is accomplished by creating a cloud-native application that can operate directly on the target infrastructure without requiring the deployment of SDN. Nonetheless, SDN might still be used to improve the utilisation of the quantum channel, as stated in [64] and to centralise QKD nodes management. Moreover, our system may be expanded by using the Software-Defined QKD (SD-QKD) method described in [54]. This entails that our QKS must interact with an SDN controller, e.g., through an SDN agent inside the infrastructure, to control the QKD modules. This modifies the configuration of the modules based on the key exchange requests and stores information on the participating apps. A more straightforward method might be readily implemented

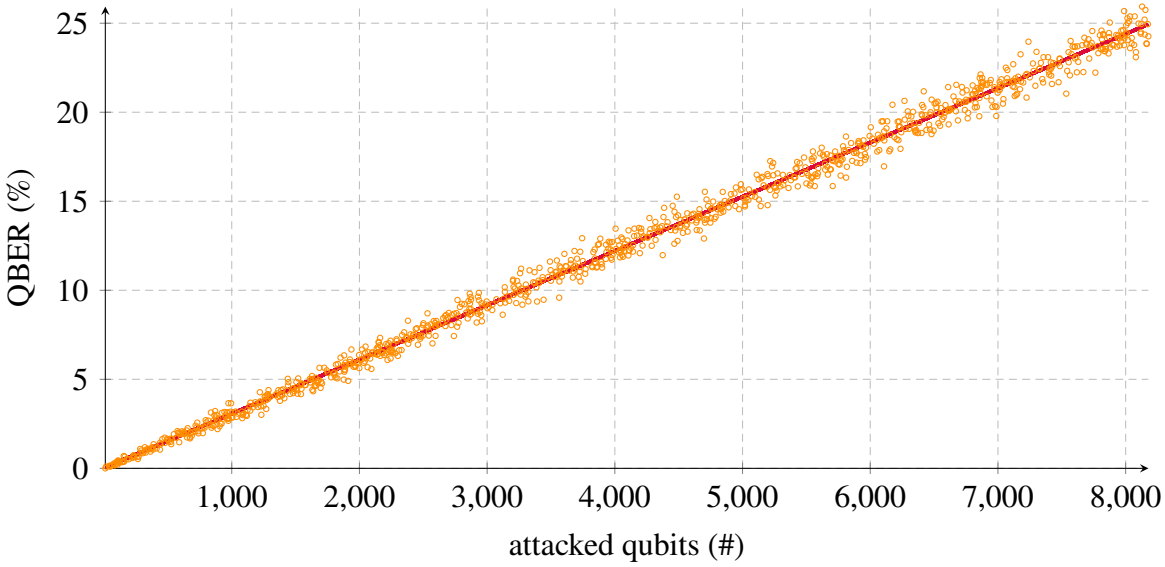

Fig. 3.13 BB84 intercept-resend attack: QBER estimation.

on restricted infrastructures, and for a variety of use cases. At the same time, SD-QKD functionality might be added to our solution.

Several publications [67, 68] specifically concentrate on modelling QKD protocols such as BB84 and B92 for the second part of QKD simulation. These works seek to capture certain protocols' characteristics while considering the imperfections of real implementations. Additionally, the prior studies establish a framework for simulating a key exchange. Comparisons with actual testbeds have been presented to demonstrate that their answers are consistent. In contrast, our technique is broader and seeks to replicate quantum nodes that can communicate via a simulated QKD network, independent of the protocols involved. In this regard, our approach is not dependent on a particular technology or framework; instead, it aims to offer a scalable, flexible framework for many use-case situations.

Other efforts [34, 33, 30, 31] approach the topic of Quantum Network simulation. These studies attempt to match various suggested abstracted quantum network designs [69, 29] and simulate a Quantum Internet scenario with several quantum nodes. This is a challenging endeavour since there is no consensus or standard on the development of these networks. *NetSquid* [31] is one of the most current and consistent techniques. This is a general discrete-event-based framework for modelling all elements of quantum networks and modular quantum computing systems, from the physical to the application level. In addition, the authors of [31] provide proof of processing nodes based on NV centres in diamond and repeaters based on atomic ensembles. In this instance, their solutions are more general than those offered by the QKD simulators. Instead of Quantum Cryptography, they concentrate on the quantum

network protocols required to establish a Quantum Internet. Our technology is unique since it simulates QKD networks and provides a testing tool for QKD in actual infrastructures. This does not rule out the idea of extending our approach into a more general simulator that could also use some of these frameworks to simulate Quantum Networks using a particular technology, e.g., NV centres in diamond.

Chapter 4

Quantum Software Stack improvements

This chapter discusses the evolution of the QSS towards a full-fledged and adaptable software stack for integrating QKD in SDI. In addition to several enhancements, the reader may also appreciate the presented use case, which involves the integration of QKD in a Kubernetes cluster. The presented work has been published in an international journal [70].

4.1 Kubernetes operators

Kubernetes is the de-facto standard for container orchestration and the management of complex distributed infrastructures. It is relevant to introduce the concept of operator to better understand the implication of using this objects to deploy, manage and integrate QSS in SDI. Within an infrastructure, physical or virtual nodes, applications, exposed services, and configurations are all treated as "resources". Physical or virtual nodes may be grouped together, forming one or many clusters. Nodes inside a cluster are designated as masters or workers according to their function. Masters are deputed to the management while workers run the user application workloads. For the sake of clarity, one can find in the following a very short summary of the Kubernetes resources:

- *Pod:* is the smallest deployable object in Kubernetes and abstracts the application workload that runs in practice by leveraging a container runtime such as Containerd, Podman or Cri-o.
- *Service:* allows exposing services running in the pods by load balancing requests over one or multiple replicas of the same service.

- *Deployment:* is a resource which allows aggregating and linking several other resources under the same descriptor to manage a specific application and automatically respond to change in the infrastructure state, e.g., the failure of a pod.
- *Secret:* is a resource that typically contains passwords, keys, certificates and other sensitive data used by an application.
- *Custom Resource (CR):* is a resource that can be defined to extend the standard Kubernetes API by declaring a custom descriptor that needs to be managed by a Custom Controller.
- *Custom Controller:* is an extension of the standard Kubernetes API and monitors a specific set of CRs, taking action whenever one of these CR is created or modified.

One of the guiding concepts of Kubernetes is the control loop, in which resources are defined, and controllers continually monitor them, taking appropriate action when necessary. The Operator design utilizes CRs and controllers to monitor and manage the life cycle of distributed applications by instantiating, updating, and customizing them. The ultimate objective of an Operator in Kubernetes is to imitate the behaviour of a human operator responsible for managing cluster resources.

4.2 Quantum Software Stack 1.0

For the reader's convenience, we provide a brief summary of the architecture of the first QSS version (Figure 4.1), so that the improvements to the QSS 2.0 appear clear and well organized.

QSS 1.0 was created using four distinct layers: the lowest one includes QKD devices and simulators; the layer immediately above treats QKDMs; then one can find the layer of the QKS; finally, on top, there are SAEs. QKDM-QKS communication takes place employing the southbound interface, while the northbound interface provides connectivity between QKS and SAEs, and the sync interface allows QKS to QKS coordination.

From the standpoint of QKDM, there is an ongoing flow of key material amongst peers. One can refer to this flow of keys as the KS. The QKS manages the KS's life cycle. Each QKS might have distinct QKDMs for managing multiple devices and thus reaching distinct destinations. This feature enables QKSs in a QKD network to operate as trusted repeaters. Additionally, mapping several QKDMs on the same physical device allows various flows

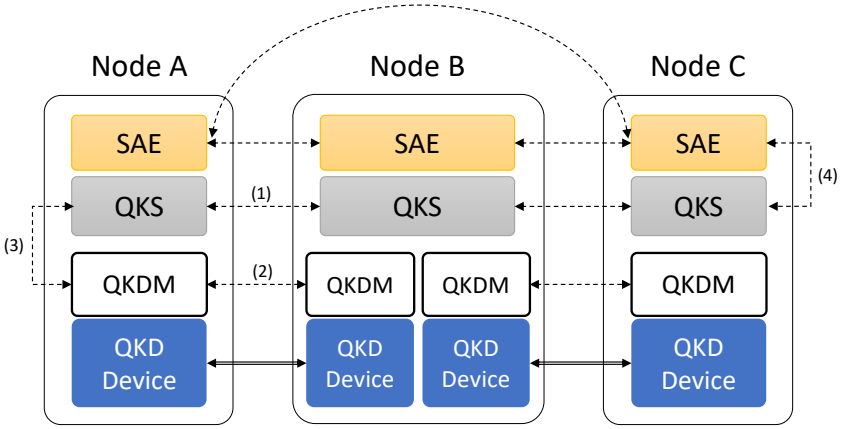

Fig. 4.1 Quantum Software Stack (QSS) presented [57]. Dashed lines represent TCP/IP secure connections. Thick, double lines indicate the combination of a classical and quantum channel for the QKD exchange. The numbers enumerate the main interfaces: (1) External Interface; (2) Sync Interface; (3) Southbound Interface; and (4) Northbound Interface.

when devices support multiple streams. As shown in Figure 4.1, a PTP exchange may occur between Node A and Node B, or we could access Node C from Node A by way of Node B, acting as a trusted repeater. This capability was not fully implemented in QKS 1.0.

QKS 1.0 was primarily responsible for fulfilling SAE key requests and informing QKDMs of where to store produced keys. When one SAE expects to securely connect with another SAE over the same QKD network, a Key Reservation Process (KRP) is necessary. When a source SAE (master or mSAE) contacts its QKS and requests key material, the latter provides the number of keys, their length, and the destination SAE (slave SAE or sSAE). Then, a KRP occurs between source and target QKSs. When a KRP is successful, the source QKS delivers the needed keys and their corresponding IDs. Afterwards, mSAE can exchange these IDs with sSAE, enabling it to obtain the duplicate keys through the QKS on-site.

A QKDM Registration Process (QRP) occurs with Keycloak when a QKDM registers with a QKS. As a reminder for the reader, QSS utilizes Keycloak as IAM, and the QKDM must be registered as a user in advance. Only then can the QKDM be configured by the system administrator, and the QRP may begin through the Southbound Interface.

4.3 Quantum Software Stack 2.0

This section presents the QSS 2.0 and underlines the differences with the original prototype in terms of architecture and programming approaches. Figure 4.2 depicts with dashed lines

the components introduced or modified from the primary version. All other components demanded minor changes.

QSS 2.0 introduces a paradigm shift from the synchronous multithreaded approach to a new asynchronous multiprocessing strategy that well reflects the choice regarding the programming language of the solution. Additionally, QSS 2.0 allows long-distance QKD exchanges by using trusted repeaters. Large QKD network scenarios are now treatable by this solution. The new software stack also includes a routing module, making possible the discovery of the QKD network topology and finding the best path over trusted repeaters when a long-distance exchange occurs.

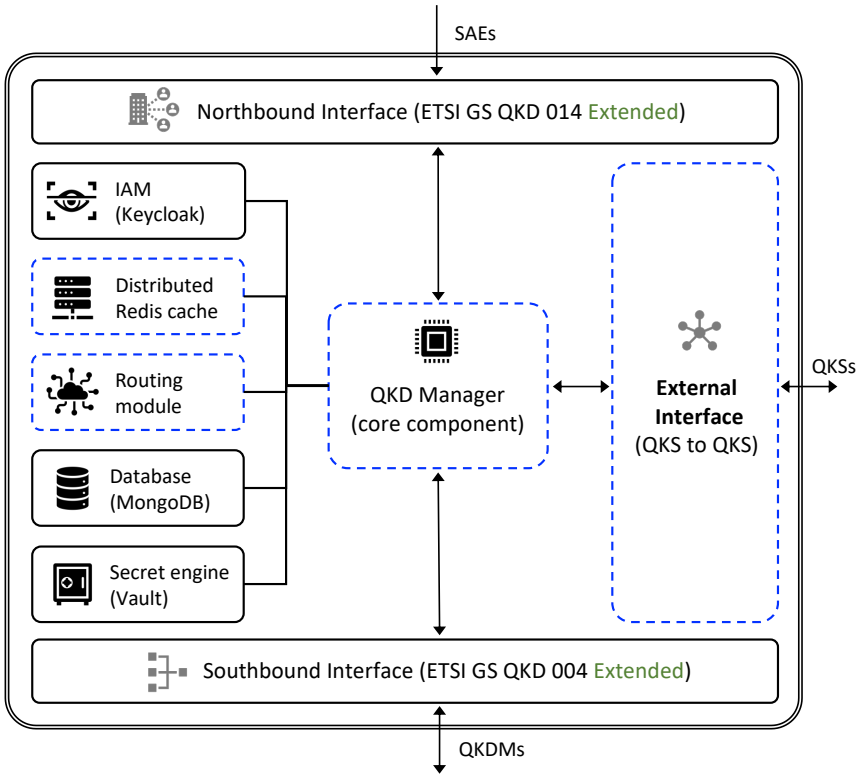

Fig. 4.2 QKS 2.0 architecture.

4.3.1 Asynchronous Approach

QKS 1.0 employed a multithreaded synchronous approach with a relevant limit since *CPython* interpreter restricts the execution of multiple threads at a time by exploiting the *Global Interpreter Lock (GIL)* [71]. Additionally, the QKD Manager inside the QKS is an I/O bound application, suggesting that it consumes consistent time waiting for results from

other applications. Because of this, the old synchronous approach negatively impacted the overall performances, requiring a completely new strategy for QKS 2.0 totally founded on an asynchronous pattern (`asyncio` Python library). Moreover, the QKD Manager exposes REST APIs leveraging Quart [72], and Hypercorn [73] to serve incoming requests. Both technologies support the `asyncio` library. All components, comprising Routing module, QKDM, and various clients for MongoDB, Vault, and Redis, have been adapted to support an asynchronous approach. As an essential remark, Hypercorn adoption allows scaling the application components vertically and horizontally, meaning that when the number of requests to the QKS grows, one can decide either to statically allocate additional workers to Hypercorn or dynamically allocate more replicas of the QKD Manager. Finally, as a cloud-native application, QSS can leverage Kubernetes to manage those scaling processes automatically.

4.3.2 Trusted Repeaters

Commercial devices for QKD are typically constrained to PTP exchanges and cannot exceed a hundred kilometres of range. Repeaters become necessary to communicate over a longer distance. The authors of [46] comment how the lack of quantum repeaters based on entanglement swapping in commercial devices leads to a requirement for trusted repeaters. These rely on the weak assumption that nodes within a QKD network are trusted, hence functional as relays to transmit key material.

QKS 2.0 introduces support for trusted repeaters. When a long-distance exchange is required, according to the available routing information, the best path for the exchange is selected, and multiple keys are reserved along the path nodes through a KRP. For key transmission, the "store and forward" approach has been adopted [74], but more sophisticated strategies can be added as a replacement or an alternative. As a remark for the reader, SAEs is completely agnostic with respect to the complexity of the multi-hop exchange: PTP and multi-hop require the exact same interaction between SAE and QKS.

4.3.3 Routing

The *Routing Module* (RM) provides traffic steering capabilities to the QKS and the entire QKD network: as long as a path between two topology nodes exists and key material is available, a key exchange is feasible among them. Each QKS includes an RM, which is able to share routing information with other RMs in the network.

As in [75, 76], the RM adopts as routing algorithm Dijkstra and implements a modified version of the *Open Shortest Path First (OSPF)* protocol [77]. *Link State Advertisements (LSAs)* are sent to update the routing information, e.g., exchange information about a SAE reachable from a specific QKS. Particular events trigger LSAs: registration or removal of a new SAE, opening of a new KS, and the expiration of the internal RM timer. A timer is indeed required to solve inconsistencies related to faliures of the signalling mechanism, i.e., a node becomes unreachable. Table 4.1 shows how RM data are structured in Redis.

Table 4.1 Description of the routing table entries in Redis.

Name	Type	Description
`SAE_ID`	String	name used to identify the destination SAE
`next_hop`	String	ID of the next QKS in the path to the destination SAE
`dest`	String	ID of the QKS of the destination SAE
`cost`	Integer	$C(t)$ for the path from source SAE to destination
`length`	Integer	number of nodes in the path to reach the destination

For message exchange RMs use TCP connections and a custom packet structure (Table 4.2). Two packet types are currently supported: "S" packets for the transmission of SAEs information and "K" packets for QKS related data. The cost function $C(t)$ for the routing algorithm is defined by Equation 4.1, where $\mathbb{P}$ is the set containing all the links of a chosen path.

$$C(t) = \sum_i K_i(t) \qquad \forall i \in \mathbb{P}. \tag{4.1}$$

The key load $K_i(t)$, namely the total cost of a link, depends on the available keys on the same link at time t and $t-1$. c_0 denotes the maximum cost when there is no key in the peer buffers (Equation 4.2). A second term adjusts this initial cost value, where K is the buffer size and $k_i(t)$ is the total number of keys at the time t; an increase of $k_i(t)$ decreases the key load. A third term is added to the equation, controlling possible rapid variations of $k_i(t)$. This is an effort to estimate additional cost contributions, such as congestion on a single connection, given that the only accessible metric information pertains to the available keys and no Quality-of-Service (QoS) parameters (e.g., error rate, jitter) are known. w_1 and w_2 enable balancing each term's contribution.

$$K_i(t) = c_0 - w_1 \cdot \frac{k_i(t)}{K} - w_2 \cdot \frac{k_i(t) - k_i(t-1)}{K}. \tag{4.2}$$

An essential remark for the reader is that the RM provides an independent routing management, where each QSS can act separately; there is no need to know the topology of the QKD network a priori and to leverage complex SDN-based scenarios. Alternative approaches [54], based on SDN, have their own benefits and flaws. They can lead to a consistent reduction of the traffic exchanged between peers (routing information), manage inconsistencies in the network faster, and adopt optimization based on multi-path routing. As flaws, one has a central point of failure that needs to be addressed with resilient and redundant resources, e.g., more than one SDN controller; moreover, a centralized managed entity must be carefully protected since retaining all topology data allows potential breaches to arm the QKD network in a more precise way, e.g., with DoS attacks. However, SDN approaches can be easily integrated into our solution as support for control plan management.

Table 4.2 Routing LSA packet structure.

Field Name	**Description**
`version`	protocol version
`type`	type of information, `S` for SAEs and `K` for QKDM links
`source`	QKS source ID, address and port
`routing`	source routing module address and port
`forwarder`	ID of the last QKS which forwarded the packet
`neighbors`	list of connected SAEs or active QKDM links (`type` field)
`timestamp`	packet creation time
`authentication`	data for authentication and integrity checks

4.4 QSS in a Kubernetes Cluster

Integrating the QSS in a Kubernetes Cluster allows enabling application workloads (pods) to access the QKD key material directly, treating it as a secret type resource. This approach enables extending the QKD usage to a wide range of use cases, considering that all modern computing paradigms (e.g., Cloud-, Edge-, Fog-, and Liquid computing), SDN, and NFV technologies massively use microservices and containers.

Master nodes in Kubernetes manage the resource type secret. Our idea of integration lies in the assumption of having this centralised secret management system, where each application inside a cluster can require QKD keys to communicate to an application belonging to another cluster. The QSS inside the control plane of a cluster act as this manager and

can be integrated as a Kubernetes operator. As a remark for the reader, from the final user's perspective, a key request is a simple resource creation (subsection 4.4.2), leaving the whole complexity of providing the key as a secret to the QSS operator. Once obtained through a common secret resource, the application workload can access the key.

4.4.1 Quantum Software Stack Operator

The QSS operator, based on the control loop pattern, relies on two CRs: `Sae` and `KeyRequest`. The first maps the creation of a logical SAE to serve a specific application, and the second models the event of a key request for a specific SAE pairs. The QSS operator manages the life cycle of these resources and deploys the software logic through a custom controller and the QSS software stack installed as a deployment resource type in the cluster. Figure 4.3 shows a PTP exchange between two clusters and subsection 4.4.2 provides further details. Once a `Sae` is applied beyond the logical representation of the resource, it registers to Keycloak and performs a series of health checks to ensure the possibility of operating over the QKD network. A `KeyRequest` creates a KRP at the QKS level and requires information on the keys' desired number and size.

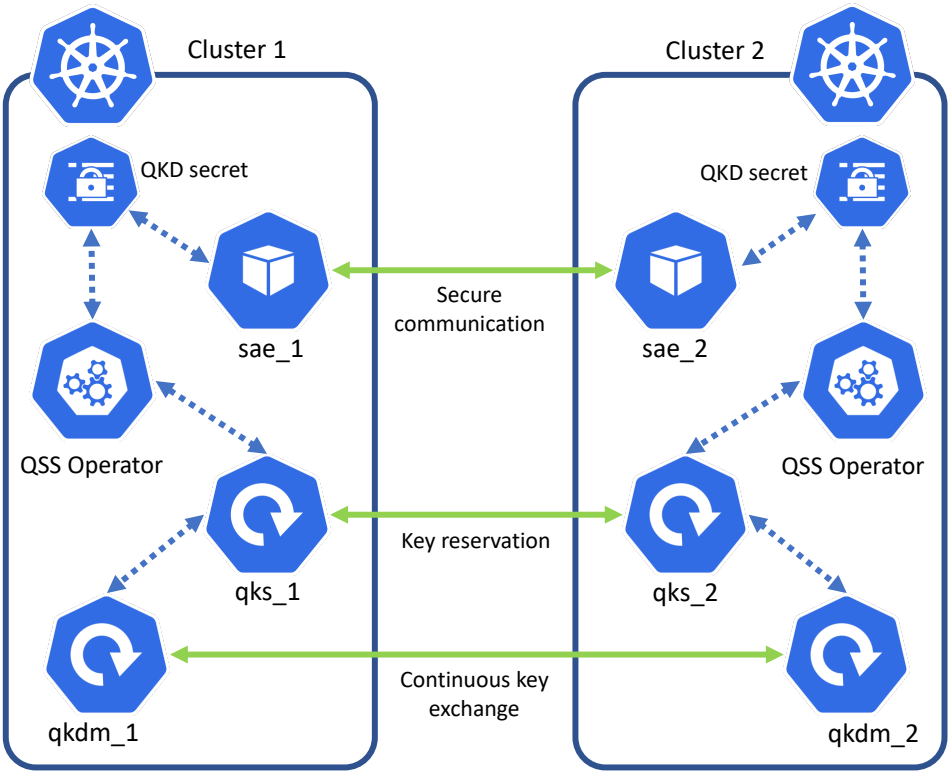

Fig. 4.3 PTP exchange among clusters.

After a `KeyRequest`, one expects to find a new secret in the namespace of the application workload. Figure 4.4 and 4.5 provides a template of `Sae` and `KeyRequest` YAML descriptors.

```yaml
apiVersion: "qks.controller/v1"
kind: Sae
metadata:
 name: sae1
spec:
 id: sae_1
 registration_auto: true
```

Fig. 4.4 `Sae` resource YAML descriptor.

```yaml
apiVersion: "qks.controller/v1"
kind: KeyRequest
metadata:
 name: request1
spec:
 number: 1
 size: 128
 master_SAE_ID: sae_1
 slave_SAE_ID: sae_2
```

Fig. 4.5 `KeyRequest` resource YAML descriptor.

4.4.2 Resource Creation and Key Exchange

This section analyses the specifics of resource production and the logic behind PTP and MH exchanges as a whole. Some assumptions must be made about the feasibility of registering SAEs and executing key requests. First, a QKD network and two or more Kubernetes clusters are required, as seen in Figure 4.3. In addition to the QKS deployment, each cluster must also have at least one QKDM deployed. Thirdly, we must instal and configure the QSS operator on each cluster to utilise the Kubernetes API instead of the QKS interface. The final assumption is that there is a constant interchange between the participating QKDMs; hence, all registration procedures and the formation of KSs have been handled as stated in Sections 4.3 standards and 4.3. Regardless of the kind of exchange, PTP or MH, all SAEs must be registered on the network using the technique outlined in Figure 4.6. A `Sae` resource may be created by an SAE administrator responsible for maintaining SAE resources in Kubernetes and requires QKD keys. This activity may also be handled without human participation by a custom operator. After creating the resource, the QSS operator recognises

the new `Sae` resource and begins registering it with Keycloak. Again, we have two options: manually register the SAE to Keycloak in advance, or allow the QSS operator to do this step (automatic registration). This option is selectable in the description shown in Figure 4.4. Once Keycloak has verified an SAE, the QSS Operator registers it with the QKS, and if the automated registration procedure is enabled, it acquires the QKS access credentials by generating a Kubernetes secret. Note that since the scope of the operator is global inside the cluster, an SAE may exist in a different namespace than the QKS.

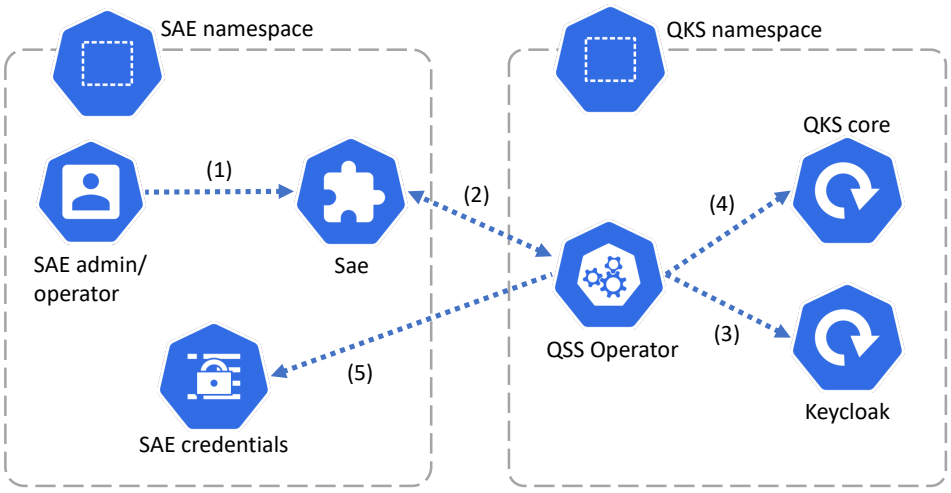

Fig. 4.6 SAE registration process. SAE admin/operator creates a resource of type Sae (1). QSS operator monitors resource creation (2). When it finds the new Sae it starts the registration to Keycloak (3) and the QKS core (4). At the end of the process it retrieves the SAE credentials (5).

When two registered SAEs want to exchange a key, it is necessary to generate a resource of type `KeyRequest`. In particular, the admin/operator is again responsible for applying the resource (Figure 4.5), supplying the key length, the number of needed keys, mSAE, and sSAE as arguments. The process in Figure 4.7 is identical to the previous scenario of SAE registration: the QSS operator detects the formation of the `KeyRequest` resource, authenticates the SAE using Keycloak, requests the key material from the QKS, which initiates the KRP and returns the key material to the QSS operator, and saves the final keys in a Kubernetes secret.

Regarding Figure 4.3, it is important to note that, given an SAE within Cluster 1 as the request initiator (or mSAE), the responder SAE (or sSAE) inside Cluster 2 must conduct another `KeyRequest` with the IDs of the keys it wishes to receive. This information might be sent through an authenticated channel, and this procedure could be automated using a Kubernetes custom operator. After this second request, all affected SAEs have access to a Kubernetes secret containing the QKD-generated key material.

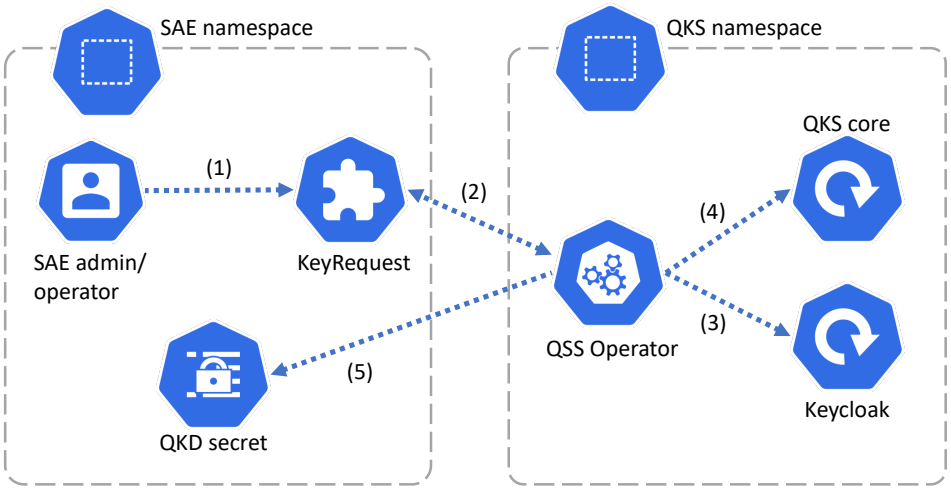

Fig. 4.7 Key request process. SAE admin/operator creates a resource of type KeyRequest (1). QSS operator monitors resource creation (2). When it finds the new KeyRequest it starts the authentication through Keycloak (3) and forward the request to the QKS (4). At the end of the process it retrieves the QKD secret (5).

The operator also enables MH exchanges, whose method is similar to the one described in Figure 4.1, where it is indicated that Node A and Node C may exchange keys utilising Node B as a Trusted Repeater. This conforms to what we described in subsection 4.3.2; hence, given a QKD network where clusters are the nodes, we may still do exchanges traversing those clusters and using QSS operators.

4.4.3 Applications to Suitable Use Cases

Using QKD [78] benefits security of several traditional applications. A central session initiation protocol (SIP) server may share a key with each of its clients, for instance, in a multi-user smartphone network with a star-type design. In this situation, QKD may provide these keys to symmetric cryptosystems, such as the Advanced Encryption Standard (AES), that are used to encrypt communication. Using QKD systems based on free-space optics technology to secure drone communications is another intriguing possibility [79]. In addition, QKD may be used in conjunction with other security protocols such as Internet Protocol Security (IPsec) and Transport Layer Security (TLS) in order to offer trustworthy key material. This strategy may be accomplished by modifying the latter protocols to use QKD keys through specific APIs. In several instances, it incorporates versions or extensions of these protocols (e.g., TLS-PSK).

There is a multitude of cloud-native apps that might directly or indirectly benefit from our approach. Indeed, security apps operating on a cluster, such as VPN endpoints, may need QKD keys directly from the Kubernetes Operator in order to create a VPN tunnel between two endpoints. Some applications may employ specialised microservices known as *service mesh* (e.g., Istio [80]) and delegate to them the administration of TLS connections and other security-related tasks. These objects can act as proxies and serve the upstream application while remaining unaware of the TLS connection. These microservices might benefit from our QKD-based TLS protocol security improvement solution.

From this wide variety of potential uses, blockchain technology represents a contemporary and compelling example. Blockchains currently serve various applications, from cryptocurrency to supply chain management. Due to the sensitive data held in a blockchain and the quantum danger impacting its underlying technologies, it seems appropriate to study quantum-resistant blockchains. The authors of [81–83] examine the security of blockchain technology, the implications of the arrival of quantum computing, and potential countermeasures from Post-Quantum and Quantum Cryptography. Similar to the preceding examples, QKD may increase the security of blockchain systems. The Hyperledger project [84], and in particular, the Hyperledger Fabric architecture, offers a clear example. Nodes inside a permissioned blockchain network interact with each other using the TLS protocol and algorithms such as RSA and ECDSA. The purpose of QKD for this application is to develop a quantum-resistant communication protocol between nodes and framework components, either based on an upgraded version of TLS or a new protocol. Regardless of the protocol's choice, our method might offer a consistent mechanism for integrating QKD even in this case. The Hyperledger Fabric framework may be readily deployed as nodes inside several Kubernetes clusters, needing no modifications to our present architecture.

Finally, it is essential to note that the QSS Operator has been built to adapt to various settings and situations. In particular, the QSS operators may function as a central entity in a large cluster of hundreds or thousands of nodes. In other cases, it can still be efficiently adopted with fixed resources, such as edge computing and IoT systems. K3s may help deploy an optimised Kubernetes cluster on a single node with constrained resources. These are the minimal requirements of our solution.

4.5 Testing

This section presents the evaluation of QSS 2.0 by using execution time and key rate as metrics. The reader can appreciate that both PTP and MH exchanges were considered, and also the performance of the introduced RM was collected.

The scenario adopted for all experiments involves three Kubernetes clusters, each one deployed on a separate physical node (master and worker on the same node) with the following specifics: CPU Intel Core i5-5300U 2.30 GHz, 16 GB of RAM DDR3 1600 MHz, Ubuntu 20.04.3 LTS, K3s version 1.22.5 and containerd v1.5.8 as the container runtime. The only exception is about the RM tests that were performed on a virtual machine with the following specifics: 16 vCPUs, 40 GB of RAM, Ubuntu 18.04.5 LTS, and Docker CE version 20.10.5. A remark for the reader is that, in this case, all nodes were simulated by Docker containers, and the metric was the average convergence time of the routing algorithm. This means that beyond the quantitative results, one wanted to asses the global trend of increasing the number of nodes in the QKD network.

4.5.1 Exchange time and key rate

The first test evaluated exchange time and key rate under different conditions, i.e., depending on the variation of the underlying QKD devices exchanges. Two main objectives were pivotal: understanding the plateau or maximum key rate available for a given architecture and specifics and comprehending how throttling the QKD devices' key rate may impact the overall exchange, e.g., showing blocking errors and system instability.

As a remark for the reader, we used a custom QKDM which simulated a key exchange at a given rate up to 2 Mbps. The first important result concerned Vault and its role as the bottleneck in the system. Considering that our solution in the standard configuration stores 128-bit keys and it takes an average of 5.6 ms to save one, the maximum key rate is around 22.8 Kbps. A simple solution to that would be to scale Vault horizontally since it is a cloud-native application itself. Despite a reasonable benefit, this solution does not solve the problem entirely and is resource-consuming. A second option would be to set another size for the stored key and move the complexity to the following aggregation step. The reason for a 128-bit size lies in the will to avoid key waste and the simplicity of aggregating keys for most of the available symmetric algorithms. For the rest of the tests, the reader can consider 22.8 Kbps the upper bound characterising the QKDM level.

Figure 4.8 shows the execution time of a `KeyRequest`, considering the QKS getKey operation time (highest impact) and the other operations performed by the QSS operator (QSS operator time in Figure 4.7). Varying the number of keys requested and the length of those keys, one can follow an increase in the `getKey` time despite the constant operator time. The `getKey` operation performs KRPs according to the number of keys requested and extracts the keys from Vault, values that clearly depend on the number and the specifics of the requests. Vault optimisation strategies are already clear, though the reader may appreciate more details about concurrency management. As shown in subsection 4.3.1, adopting the asynchronous framework already benefit this aspect. With a single Hypercorn worker and a single QKS replica, the throughput is around 56 Kbps (considering that the keys are already available in the buffer). Scaling this aspect is not an issue since one has two parameters, Hypercorn workers and QKS replicas, that can support reaching throughput far beyond the one reported here.

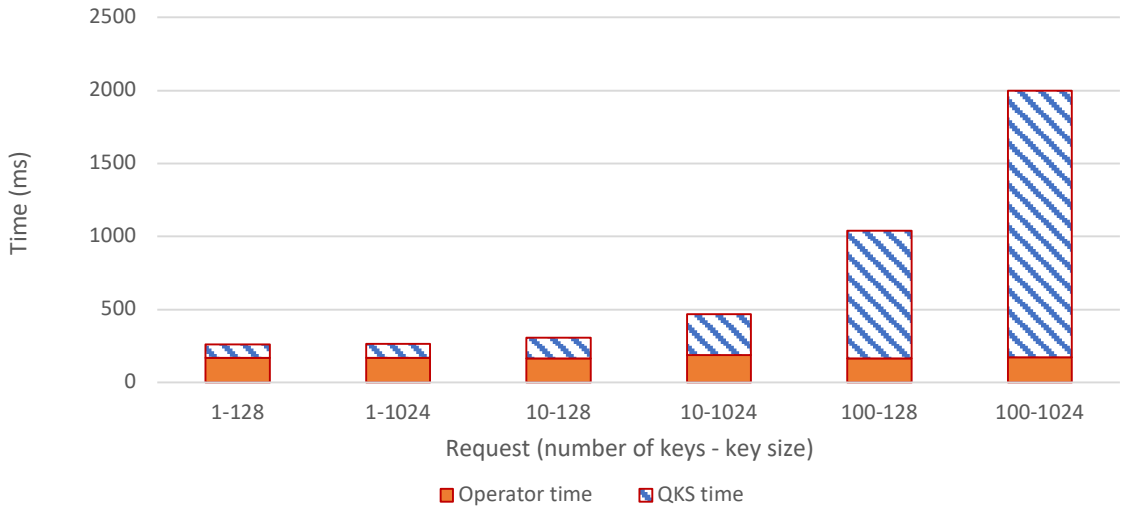

Fig. 4.8 Execution time to handle a `KeyRequest` resource type.

Figure 4.9 shows another test regarding the success rate of `KeyRequests` depending on the QKDM fixed exchange rate and key rate requested. The chosen reference values for the QKDM key rate were 2 Kbps and 1 Kbps. Clearly, in the first column, having a QKDM rate which is two times the request rate, leads to no 100% success. Even using the same QKDM rate leads to a small percentage of loss due to some internal delay in the propagation of the keys. Increasing the request rate beyond the QKDM rate leads to failure and retransmission in both cases. The interesting part is that dynamically varying these values shows how the

system can return to 100% success as soon as normal conditions are restored. Additionally, it notifies the layers above in case of loss and failure.

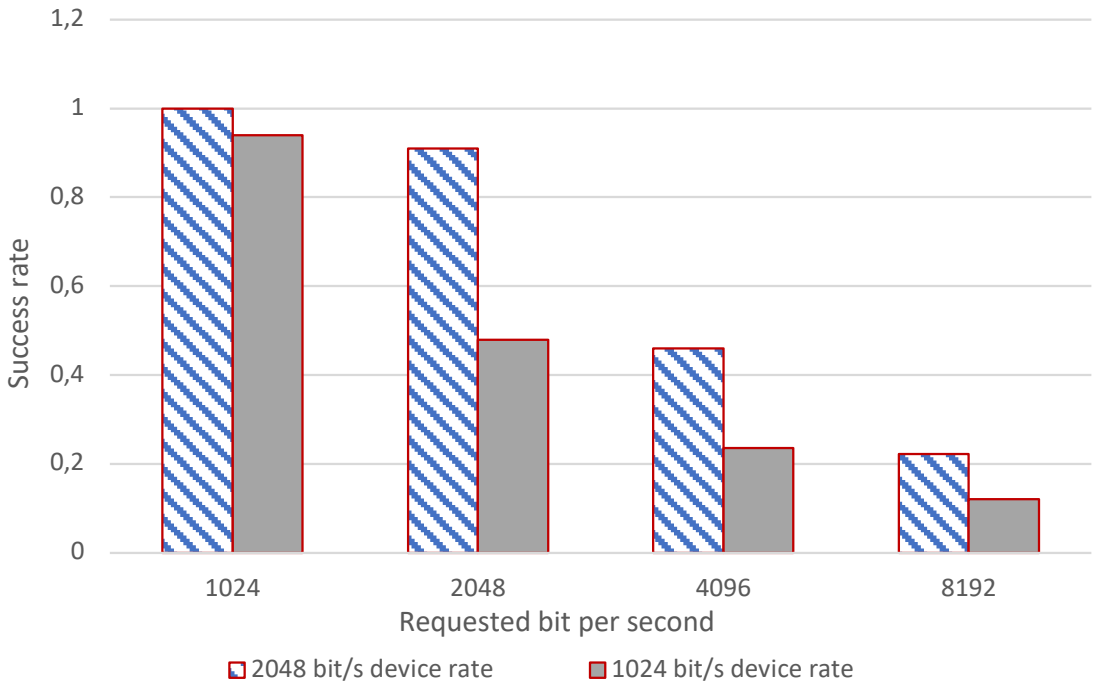

Fig. 4.9 Success rate of the getKey requests to the QKS varying the underlying exchange rate.

In the end, Figure 4.10 provides the results of an experiment performed on an MH scenario, where the cluster in the middle act as a trusted repeater. In particular, one can observe the values of `getKey` and `getKeyWithId` in a PTP and MH comparison. The expected yet interesting result shows how for the `getKey`, the execution time depends on the number of intermediate hops while for the `getKeyWithId` remains constant.

4.5.2 Routing

The last performed test involves the RM. In this specific case, there is no need for the whole stack or different physical nodes since the objective is to understand the trend of the average convergence time of the routing algorithm depending on the number of QKD nodes. Because of this, one can treat each QKD node as a Docker container that is provided with the RM code and communicates over a TCP/IP overlay network to reach other nodes. The experiment consisted in generating random events, such as SAEs and QKDMs registrations and removals,

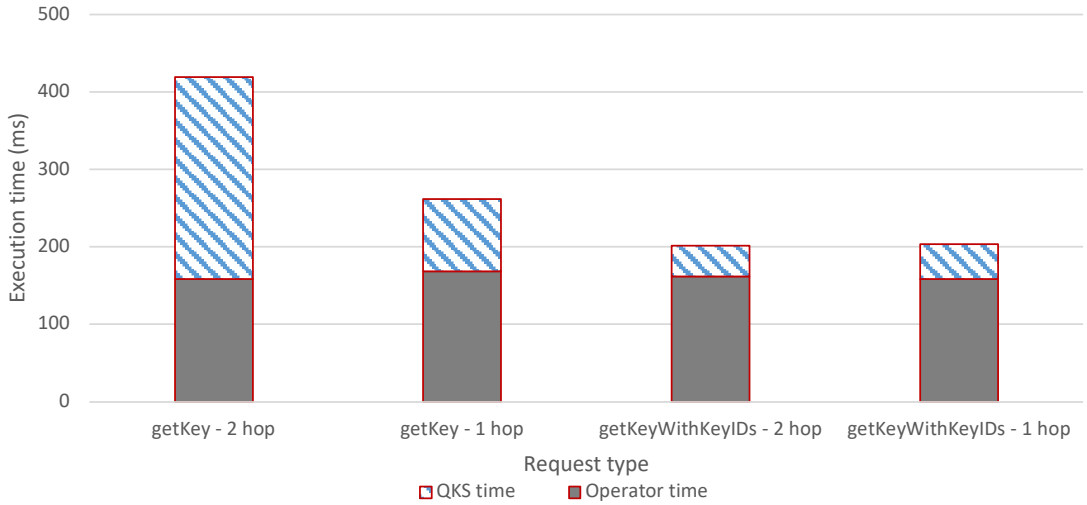

Fig. 4.10 Execution time of the getKey and getKeyWithId functions for Point-to-Point and Multi-hop exchanges.

with a time interval of 10 seconds and measuring the convergence time. The first result demonstrated that each node can correctly form the identical QKD network topology in his table for every topology represented by a connected graph and can contact all available nodes.

Figure 4.11 illustrates the average convergence time of the algorithm progressively increasing the number of QKD network nodes up to 50. An essential remark for the reader is that this test has been performed on virtual networks with relatively low latency and large bandwidth. A test over real large-scale networks may be affected by those parameters, leading to adjusting the expiration time of LSAs to preserve the convergence. Dijkstra has the following computational complexity: $\mathcal{O}(|E|+|V|\log|V|)$ where $|E|$ is the number of links and $|V|$ is the number of nodes. Large QKD networks may require support for different autonomous systems [77].

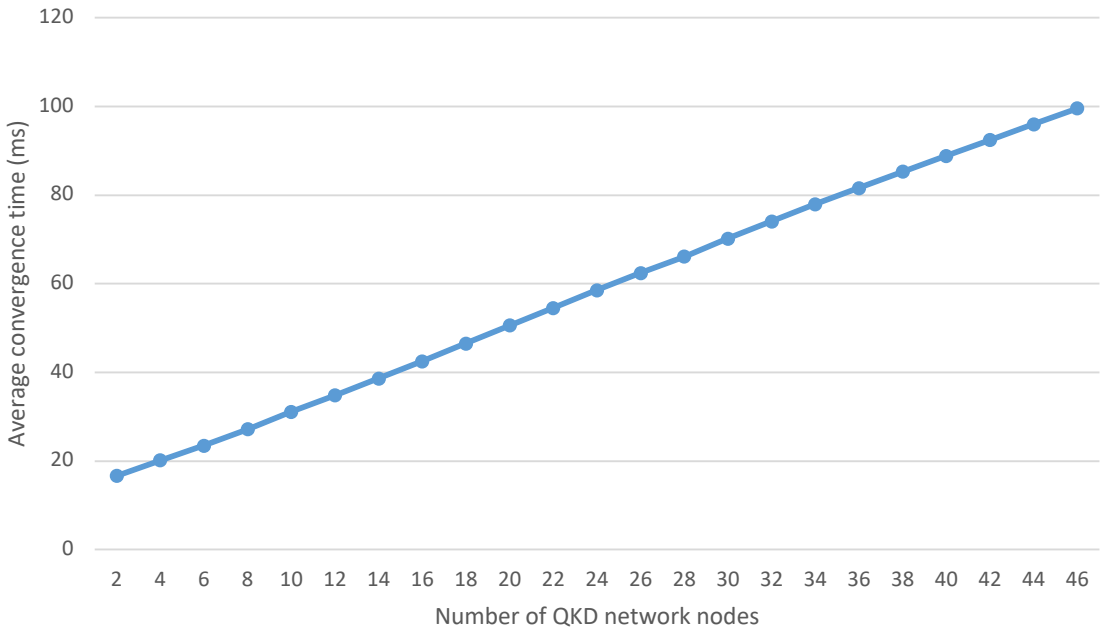

Fig. 4.11 Average convergence time of the routing algorithm depending on the number of nodes in the QKD network.

Chapter 5

QKD simulation over distributed infrastructures: QuaSi

This Chapter analyses Quantum Simulator (QuaSi), an evolution of the QKD simulator's prototype described in section 3.4. Despite the interesting model for the simulation involving a distributed cloud-native application, i.e., endpoints and channels as distinct and interchangeable modules, the first prototype of the QKD simulator presented several flaws. In particular, the initial idea was to focus on a PTP approach without considering the complexity of possible quantum network scenarios. Moreover, even promising high scalability, tools and strategies to orchestrate large network scenarios were not provided yet in the preliminary work [57]. Because of this and other sought performance improvements that the reader can appreciate in this chapter, QuaSi represents an enhancement and a more stable version of the QKD simulator. In the following sections, one can find a presentation of the overall advantages behind QuaSi, the overall architecture and the general overflow, and the final test and validation comparing its performance against its previous version.

5.1 Simulation process design and objectives

As depicted in Figure 5.1, the concept at the very ground of the simulation process does not change from the old QKD simulator. Alice, Bob, and, more generally, the endpoints are containerized processes that can communicate with each other over TCP/IP networks. The simulation of the attacks, as well as the characteristics of the quantum channel, can be included as an opportune transformation within a third containerized process (Eve) which can

filter out the traffic of both classical and quantum channels, hence simulating noise, quantum, and classical attacks.

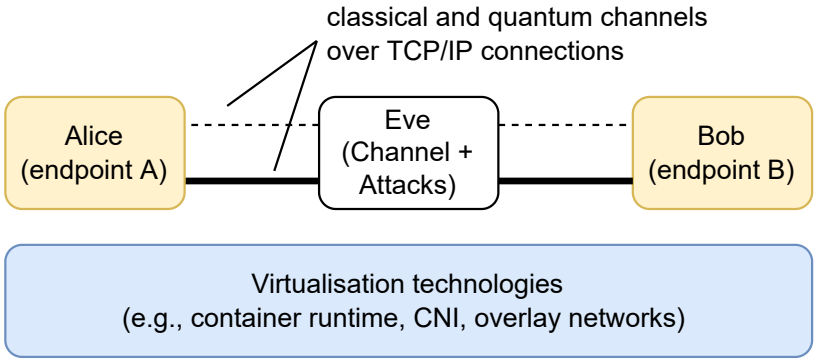

Fig. 5.1 Basic PTP simulation strategy within QuaSi.

As important remarks for the reader, one should point out that the very aim of this simulator at this stage is to provide an efficient tool to scale and manage the QKD simulation over a distributed infrastructure, not to provide theoretical or heuristic values of metrics such as the key rate, that even if possible would be just one of the many application of such a distributed simulator. There are many examples of Quantum network simulators (section 2.4) either working on low-level simulations based on a discrete event strategy or only focusing on a specific protocol, e.g. BB84 for the QKD. Our idea is different since we do not want to give a mere metric as a result but propose a framework that can be shaped as the requirements for a specific simulation, backend, and protocols arise.

As a first example, we have this distributed concept of endpoints because we do know that a direct application of the QKD simulation on a QKD network involves a direct interface with the upper layers, i.e. QKDMs, key managers, and security applications. Because of this, having a process, an entity that can directly expose an interface to the rest of the quantum stack, simplifies the interaction with the other components and leaves room to manage all the post-processing requirements for the QKD that can be developed as part of the distributed simulator at first and then reused in real use cases by replacing the simulation of the quantum part with real hardware.

Another important aspect is the possibility of simulating, with different backends, the low-level part by describing the protocol with the desired precision and model. We adopted - as in the case of the QKD simulator - Qiskit as the backend, which can be easily replaced with many alternatives. In particular, our initial simulation for DV-QKD is based on quantum circuits whose evolution can be computed over distributed processes (e.g., Alice, Bob, Eve). This

allows us to describe attacks as quantum circuits that can act on the opportune representation of the qubits object of the key exchange.

The reason why we are remarking on this is to inform the reader that our final objective is not to provide a more precise and technology-dependant simulation of specific protocols but to propose a general model on how to distribute this simulation, also considering the classical post-processing, a precise and flexible data model, and a practical framework on which it is possible to operate.

The only significant difference from the previous version in terms of simulation is related to the enhancement of the simulation performances. Computing quantum circuits in Qiskit is a tough task as the size of quantum registers grows and the operation over these qubits (represented as State Vectors or Density Matrices). In order to improve the simulation and considering that we used Python, a programming language, we adopted a multiprocess approach which allowed us to overcome the GIL limitations and measure a positive impact on the performance. Finally, the containers used for the simulation - that are open-source and available online - can be executed regardless of the specific container runtime adopted.

5.2 High-level Architecture

So far, the simulation approach has been discussed without considering the significant changes in terms of architecture and workflow of QuaSi with respect to the previous version of the simulator. In this section, one can find a description of the overall architecture of QuaSi and an analysis of the various component and their functionalities.

As depicted in Figure 5.2, QuaSi is based on a modular architecture where the more relevant elements are: a Kubernetes cluster, the Simulator Manager, and RabbitMQ. Kubernetes provides - as in the case of the software stack - a practical and flexible environment where one can orchestrate all the available nodes within the topology, endpoints and channels in the form of pods. Given a YAML representation of a network topology and the base container images representing the software logic of the various nodes, the idea is to leverage Kubernetes to deploy at once the whole topology and use specific software components to start, configure, and manage the simulation. These software components are the Simulator Manager, in charge of orchestrating the simulation and RabbitMQ, a message broker, in charge of handling and distributing messages across the nodes.

Starting from the infrastructure, everything is built on top of a Kubernetes cluster that can be scaled in terms of master and worker nodes as needed. This scaling impact neither the

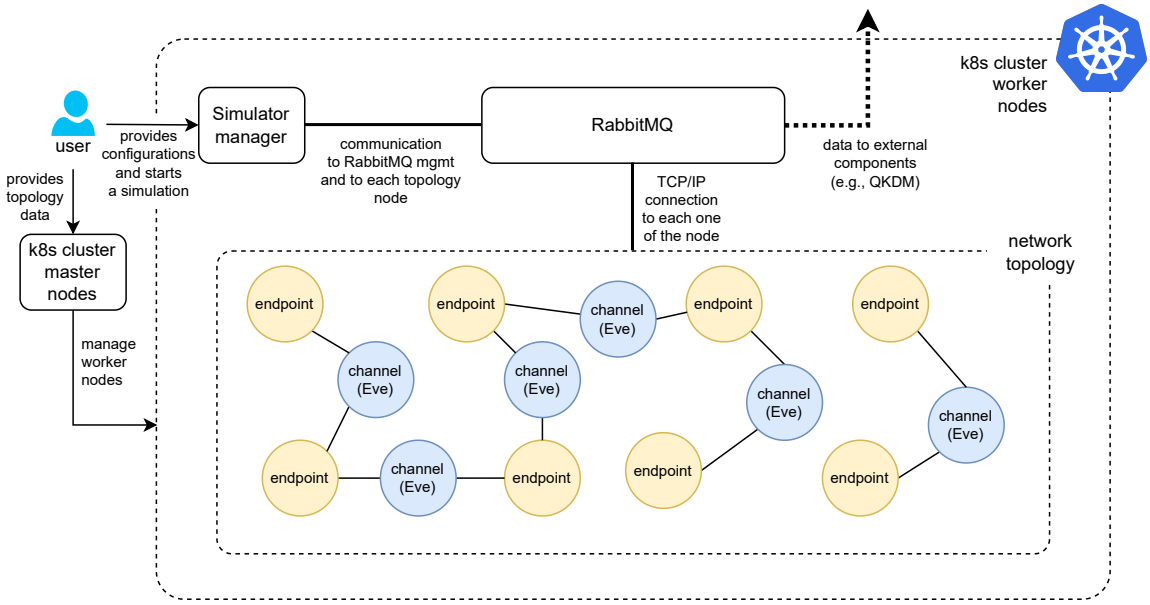

Fig. 5.2 High-level architecture of QuaSi.

logic nor the communication among topology nodes. As a remark, with topology nodes, we intend the nodes of the simulated network, e.g., the pods on the worker nodes. For master and worker nodes, we intend virtual or physical nodes on which the Kubernetes cluster is built. The additional services, Simulator Manager and RabbitMQ are deployed within the same cluster on the worker nodes in turn as a Deployment and a Stateful Set.

In more detail, the Simulation Manager is a software component which exposes a REST API towards the user of the simulator and, eventually high-level third-party application that wants directly use it (e.g., QSS). Through this REST API, it is possible to provide the simulator Manager with the deployed topology, start, stop, and manage the exchange process for the various QKD exchanges (Table 5.1). The simulator Manager, once instructed on the operational tasks, proceeds by injecting configuration in each node and starting the exchange processes by leveraging specific message queues on RabbitMQ for Simulator manager-nodes communication.

URL	**Access Method**
http://{Simulator URL}/getTopology	POST
http://{Simulator URL}/distributeKeys	POST
http://{Simulator URL}/startKeyExchange	POST
http://{Simulator URL}/begin	POST
http://{Simulator URL}/end	POST

Table 5.1 Simulator Manager REST API for end users and to third-party applications.

RabbitMQ, as anticipated, is used as the message broker for inter-node communication, connectivity to the Simulator Manager, and extra-cluster communication. In particular, each message among nodes passes through a specific queue, implementing an asynchronous communication which replaces the old inter-node REST API and benefits the performance of the overall systems since, by design, this simulator is an I/O intensive application, i.e., each node takes time to produce a valuable output for the next node. RabbitMQ exposes, upon a specific registration process with the Simulator manager, an external queue to retrieve the data from the simulation for each one of the nodes, i.e., key material exchanged during the simulation. As an example, a QKDM can be registered as a consumer for a specific topology node and receive key material as soon as it is produced.

Finally, we want also to underline a technical choice that supports the adoption of this model for a distributed simulation. As in the case of the QSS and its enhanced version and integration to a Kubernetes cluster, we developed a Kubernetes Operator for the Simulator Manager. Because of this, one has an automatic tool for deploying the basic component of the simulator, i.e. Simulator Manager, RabbitMQ, and all related additional resources. At the same time, the Simulator Manager can monitor Kubernetes CRs and provide changes to the infrastructure resulting in complete integration with the Kubernetes cluster itself. As an example, the Simulator Manager can monitor the creation of a custom resource (control-loop strategy) related to a network topology so that once they are instantiated through the Kubernetes control plane, certain operations can be triggered directly on the cluster, such as the RabbitMQ queues initialisation.

5.3 Data model and workflow

An important new element of QuaSi is the generic data model for inter-node communication. In order to validate messages, differentiate them, and reserve fields for peculiar additional information, we proposed the generic model in Table 5.2. Clearly, all the communication through RabbitMQ queues leverages the Advanced Message Queuing Protocol (AMQP).

Field	Description
Type	The type of the message
Source	The name of the entity that sent the message
Destination	The name of the entity the message is destined to
Metadata	Contains the configuration parameters of a Simulation
Body	What the source wants sent to the receiver

Table 5.2 Data model of inter-node message exchanges.

As for the workflow, one can summarise the whole QuaSi simulation in the following phases: deployment, initialisation, starting, and data collection. The first deployment step involves the user contacting the Kubernetes API and submitting the YAML file describing the desired topology. Afterwards, the same user, in the initialisation phase, interacts with the Simulator Manager API providing the same topology and the description of the required configuration. This allows for the same topology to change various settings and obtain various instances of the same simulation. Then, in the starting phase, the user asks the simulator manager to start the simulation of the exchanges for all the PTP links or a subset of them. In the final phase, the data collection, either the user or an external process (third-party application case) can collect data from specific RabbitMQ queues and retrieve statistics as well as key material exchanged.

5.4 Test and Validation

As a remark for the reader and before presenting the test and validation results, we want to report that the entire implementation of QuaSi is available on this GitHub repository[1] as open-source.

Beyond the intrinsic scalability of the cloud-native approach and the adoption of Kubernetes, relevant as quantitative results, one can measure the performance of the system in terms of key exchange time and key rate in a PTP scenario. This value can be then compared with the one related to the first version of the QKD simulator. In Figure 5.3, one can observe key exchange time and key rate depending on the key length, in the case of a single worker on both endpoints, the size of the Qiskit quantum register set to 5, and a single key exchange. One can also appreciate the difference between a raw exchange (w/o authentication) and an exchange using as the authentication method a digital signature using SPHINCS$^+$. Even in this case, due to the optimisation of the asynchronous approach and the message broker,

[1] https://github.com/ignaziopedone/QuaSi

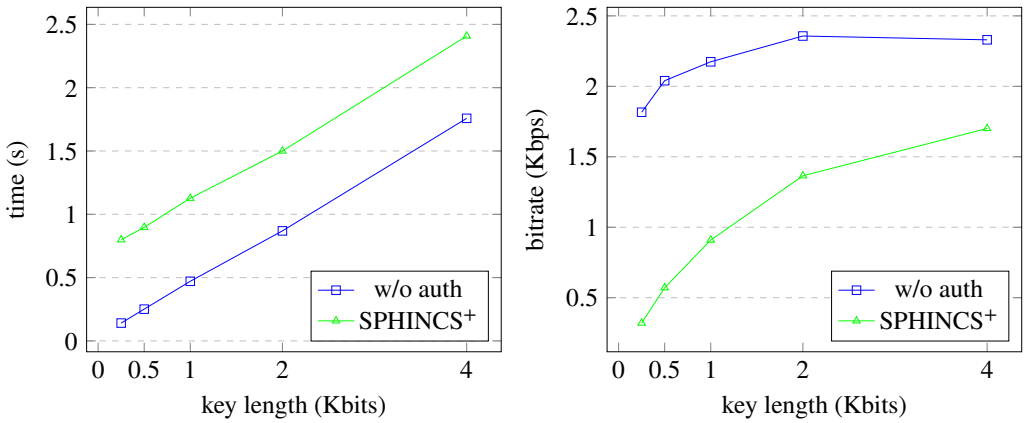

Fig. 5.3 Key exchange time and key rate depending on the key length for the base exchange without authentication and with SPHINCS$^+$. Quantum register size is set to 5, the number of exchanged keys to 1, and only 1 worker is available.

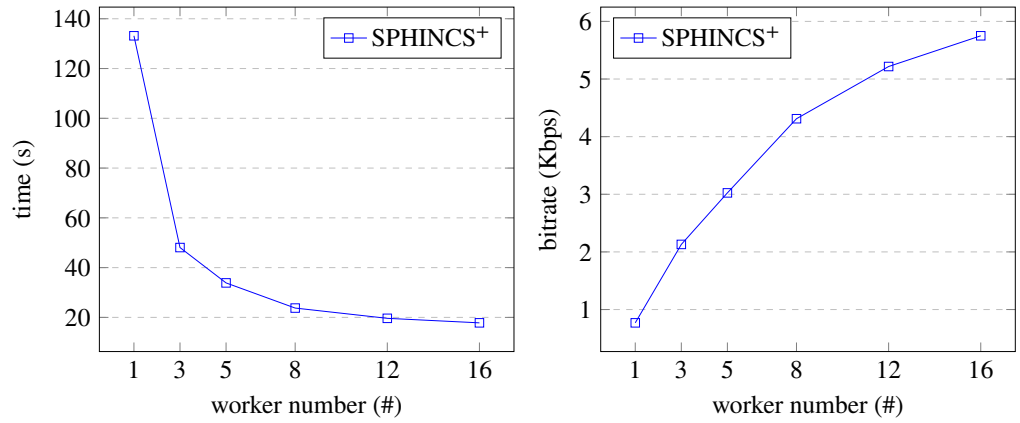

Fig. 5.4 Key exchange time and key rate depending on the number of available workers. Key length is set to 4096 bits, quantum register size to 5, and the number of exchanged keys is 25.

the performance is slightly increasing with respect to the classical QKD simulator. As we expected, signing messages with SPHINCS$^+$ introduces an overhead to the total time required for the key exchange. In Figure 5.4, one can observe the same metrics, but they depend instead on the number of available workers. In this case, the length of the key is set to 4096 bits, the quantum register size is set to 5, and the number of keys exchanged is 25. The reader can observe how the multiprocessing approach leads to a significant improvement in terms of performance (an order of magnitude) with respect to the classical QKD simulator. In this case, we only show the experiment using SPHINCS$^+$ for authentication. As a note for the reader, the variant of SPHINCS$^+$ used in these experiments is the same adopted for the test with the classical QKD simulator. As another remark, this last result leaves room for additional low-effort improvements since SPHINCS$^+$ is probably the worst-performing algorithm in

terms of signature per second (we are not considering the level of security and the robustness since the standardisation is still in progress). As an example, choosing CRYSTALS-Dilithium as the signature algorithm may dramatically improve the performance while maintaining a high level of security, according to the NIST scale.

5.5 Applications, extensions, and future work

An immediate application of QuaSi lies in the integration with the QSS. One can imagine a dedicated Kubernetes cluster as a core for simulating an extensive network of low-level PTP devices and providing the interactive results of this simulation to the QKDMs of the QSS instances running on top of the QKD network nodes lying on those low-level devices.

As for the extensions, perhaps the most interesting would be to add a feature to calculate the theoretical maximum key rate achievable for a given protocol and conditions of each PTP link. This can be obtained as a result of an optimisation problem whose complexity can be distributed among the very same topology nodes of the network.

Since the significant advancements of the simulator have been towards the infrastructural aspects, most of the required future efforts need to focus on the simulation part. Probably adding new backends, QKD protocols support as well as investigating protocols beyond QKD is a priority for the subsequent developments of this framework.

Chapter 6

Quantum Annealing to optimise Software-Defined Infrastructures

This chapter provides a QUBO formulation of a generic VNFEP. This formulation is tested using the D-Wave Advantage Quantum Annealer and, as an alternative, two classical solvers, based on Tabu search and Simulated Annealing. This work has been published in the proceedings of the third IEEE International Conference on Quantum Computing and Engineering (QCE22) [85].

6.1 Quantum Annealing-based SDI optimisation

Currently, the ever-expanding market for devices that need internet connectivity and the provision of sophisticated and diverse network services requires flexible and scalable infrastructures. SDI, as mentioned in chapter 2, can help with NFV technologies that may efficiently manage and orchestrate those services. As a reminder for the reader, NFV aims at creating network functionalities using software functions called VNFs that can be executed and allocated flexibly on general-purpose hardware. This eliminates the requirement for physical special-purpose equipment. Moving network function allocation from hardware to software substantially increases the network's adaptability and simplifies its scalability to meet ever-changing customers' requirements. VNFs can be developed to serve security purposes (e.g., Firewall, IDS, and NAT), as in the case of the SECaaS paradigm. Also, in this case, VNFs can be executed on standard physical resources, such as High Volume Servers (HVSs) and additional hardware if required, e.g., storage components.

A pivotal optimisation problem in this context, from a service provider's perspective, is the *VNF Embedding Problem* (VNFEP). A VNFEP corresponds to the problem of determining the best mapping of required logical resources against specific constraints and limits of the available physical resources. Clearly, an objective or multi-objective function has to be defined.

In general, the problem's complexity makes finding an efficient solution difficult since the solution space expands fast and full exploration is impracticable. As a compromise to lower the computing effort, earlier studies have developed heuristic ways to obtain high-quality solutions in an acceptable amount of time [86–89]. However, stochastic approaches do not ensure optimum solution convergence. Due to the nature of the domain, the optimal method for resolving a VNFEP is finding a suitable solution in a minimal period of time so that it can be utilised to adapt quickly to changes.

Quantum Annealers have demonstrated promising capabilities in tackling tasks such as graph partitioning [90, 91], Support Vector Machines [92], Restricted Boltzmann Machines [93, 94], feature selection [95, 96] and optimisation problems in particular QUBO. Optimisation problems are described using two principal models: Ising and QUBO (Quadratic Unconstrained Binary Optimisation). Formally, they are equivalent, but the first is more appropriate for describing problems in physics and the second for problems in computer science.

The main reason to adopt this technology for solving VNFEPs lies in the remarkable potential of Quantum Annealers to deal with NP-hard problems. As for the examples provided in the literature, VNFEPs can benefit from those solvers, even considering the current limitations of the available quantum processors.

6.1.1 VNF Embedding Problem

According to [97], the VNFEP involves mapping a collection of virtual resources (VNFs) to a substrate network in which general-purpose servers are represented as physical nodes. Each node has its attributes, resources, and computing capabilities. A node might be outfitted with specified CPU, RAM, and storage resources or contain specialised processing units like hardware accelerators.

In addition, the links connecting the various nodes are distinguished by particular figures of merit, such as bandwidth and latency (e.g., while having the same capacity, a direct fibre-optic link may have a much shorter delay than a satellite link). In addition to these physical limits, other criteria are sometimes imposed, such as the order in which VNFs within

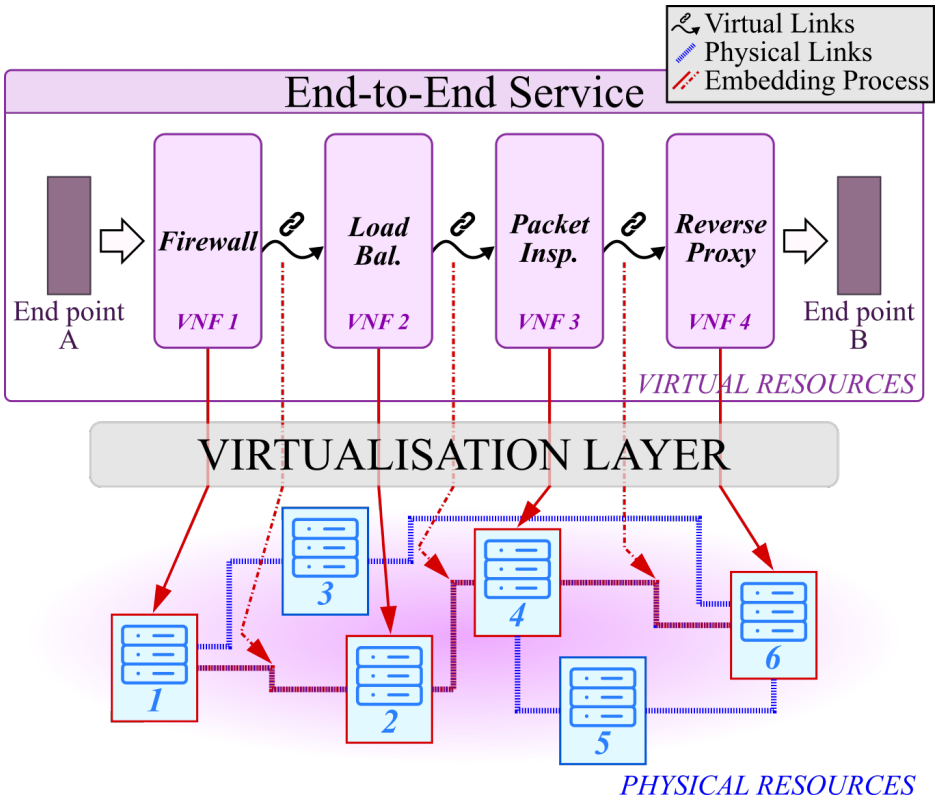

Fig. 6.1 VNF embedding of a SFC of 4 VNFs on a substrate network of 6 HVSs, or physical nodes.

a chain are assigned or restrictions on where VNFs may be hosted. These chains are typically addressed as Service Function Chains (SFCs) and combine elementary virtual functions to compose a more complex end-to-end network service. All of these factors must be addressed during VNF embedding in order to conduct an effective mapping. Given their physical connections, Figure 6.1 depicts an example of challenge mapping a chain of four VNFs into a substrate network with six physical resources.

6.2 Problem Formulation

This section describes the proposed QUBO formulation in terms of definitions of the binary variables, cost function, and constraints of the problem.

6.2.1 Substrate Network

The architecture of the substrate network is represented as an undirected graph G with nodes N representing the HVSs and edges E indicating the actual network channels linking them. Each server $i \in N$ in the network is capable of delivering distinct sorts of resources $w \in W$. The amount of a resource w accessible on a node i (e.g., CPUs, RAM, storage) is denoted by R_i^w, whereas C_i^w denotes its unitary utilisation cost. Each physical network connection $(i, j) \in E$ linking nodes $i \in N$ and $j \in N$ is similarly characterised by its bandwidth B_{ij} and the delay D_{ij} that it adds. Each link's bandwidth use has a separate unitary cost, denoted by C_{ij}.

C_{ij} and C_i^w might reflect monetary expenses associated with energy consumption, infrastructure provider, geographic location, and technology production.

6.2.2 Service Function Chains

Each problem specifies a collection of SFCs, denoted P, comprised of an ordered list of VNFs. Two characteristics distinguish each SFC $p \in P$: the maximum permissible delay for the traffic to go through the chain, denoted by D_p, and the maximum bandwidth it consumes, denoted by B_p. Each VNF q in the chain p needs a distinct set of resources $w \in W$ from the node on which it operates. R_{pq}^w represents the amount of each resource needed by q in the chain p. $q+1$ refers to the VNF next to q, while q_L refers to the last VNF in the chain to which q belongs.

6.2.3 Problem Variables

A solution to the VNFEP must specify which physical nodes host each VNF for each SFC. Using a QUBO formulation, binary variables shall represent the final solution to the optimisation problem. In this work's formulation, the selected binary variables have the form L_{pq}^{ij}. This variable equals one if the virtual link between VNFs q and $q+1$ of chain p is hosted by link (i, j) in the direction $i \to j$. This indicates the placement of VNF q on node i and VNF $q+1$ on node j. These variables were selected to prevent the introduction of terms of order greater than two while expressing the cost function and restrictions.

$$E_{cost} = \sum_{i,j\neq i}\sum_{p}\sum_{q\neq q_L}\sum_{w} L_{pq}^{ij}\cdot\left(R_{pq}^{w}\cdot C_{i}^{w} + R_{pq+1}^{w}\cdot C_{j}^{w}\cdot\delta_{p}(q+1)\right) + \sum_{i,j\neq i}\sum_{p}\sum_{q\neq q_L} L_{pq}^{ij}\cdot B_{p}\cdot C_{ij} \quad (6.1)$$

6.2.4 Cost Function

The quality of a VNFEP solution is quantified by a cost function that analyses the overall cost of node and link resource use required by the VNF embedding. The adopted cost function is presented in Equation 6.1 and is the sum of two terms.

The first term connects to the utilisation cost of allocated node resources needed by VNFs. If L_{pq}^{ij} equals one, the VNF q of SFC p is allocated on node i. Consequently, the utilisation cost of each resource w needed by q equals $R_{pq}^{w}\cdot C_{i}^{w}$, which is the product of the resource quantity and the unitary resource utilisation cost. This product alone does not account for allocating the last VNF q_L in each chain. When VNF q is the second-to-last element of chain p, and L_{pq}^{ij} equals one, this indicates that $q_L = q+1$ will be put on node j. Therefore, the cost of this allocation must be added to the objective function, $R_{pq+1}^{w}\cdot C_{j}^{w}$. Because of this, the Kronecker delta $\delta_p(q+1)$ equals one when $q+1$ is the last VNF of the SFC p.

The second term is associated with link bandwidth costs. If $L_{pq}^{ij} = 1$, the physical link (i,j) transports traffic between VNF q and VNF $q+1$ of SFC p. Since SFC p uses bandwidth B_p, the cost of bandwidth usage on link (i,j) for this traffic is $B_p\cdot C_{ij}$.

6.2.5 Constraints

Clearly, to obtain a valid solution, some conditions have to be met both in terms of allocation choices of the SFCs and resource limits of the physical infrastructure. These are the constraints defined for the proposed formulation:

- Allocation: a VNF shall be allocated exactly on one node; thus, as shown in Equation 6.2 it can be achieved by forcing to one the summation of physical links hosting the virtual link between two consecutive VNFs.

$$\forall p, q\neq q_L, \sum_{i,j\neq i} L_{pq}^{ij} = 1 \quad (6.2)$$

- Continuity: contiguous VNFs in an SFC must be allocated on nodes adjacent to the substrate network. This is obtained in Equation 6.3 by forcing to zero the difference

between the incoming and outgoing contiguous virtual links of the same SFC from a network node (j), i.e., for all VNFs other than the first and final in the chain, an incoming and outgoing link must always exist. Excluded are the nodes hosting the initial and final VNFs in the chain.

$$\forall j, p, q \neq q_L, \sum_{i \neq j} L_{pq}^{ij} - \sum_{k \neq j} L_{p,q+1}^{jk} = 0 \tag{6.3}$$

- Resources: resource consumption by VNFs hosted on a node cannot exceed the node's available resources. Given a node i and a resource w, in Equation 6.4, if $L_{pq}^{ij} = 1$, then node i is consuming amount R_{pq}^{w} of resource w requested by VNF q of SFC p. The aggregate of this contribution from all VNFs on the node must be less than or equal to the amount available, R_{i}^{w}. The second term is solely included to account for the resource consumption of the last VNF in the chain.

$$\forall i, w, \sum_{j \neq i} \sum_{p} \sum_{q \neq q_L} L_{pq}^{ij} \cdot R_{pq}^{w} + \sum_{j \neq i} \sum_{p} L_{pq_{L-1}}^{ji} \cdot R_{pq_L}^{w} \leq R_{i}^{w} \tag{6.4}$$

- Bandwidth: the bandwidth needed by VNFs assigned to a physical connection cannot exceed the bandwidth available on that link. Given a physical connection (i, j), Equation 6.5 constrains the necessary bandwidth from all SFCs transiting the link in both directions to be less than or equal to the available bandwidth.

$$\forall i, j < i, \sum_{p} \sum_{q \neq q_L} (L_{pq}^{ij} + L_{pq}^{ji}) \cdot B_p \leq B_{ij} \tag{6.5}$$

- Delay: the delay produced by the physical links cannot exceed the maximum delay permitted by the chain. Given a chain p, Equation 6.6 constrains the delay produced by the physical links traversed by p to be less than or equal to the maximum delay permitted for p.

$$\forall p, \sum_{i,j \neq i} \sum_{q \neq q_L} L_{pq}^{ij} \cdot D_{ij} \leq D_p \tag{6.6}$$

6.2.6 Full VNFEP QUBO Formulation

The comprehensive VNFEP QUBO formulation takes both costs and limitations into consideration. Since the QUBO formulation does not include hard constraints, each constraint

$v \in V$ must be connected with a penalty P_v applied to the cost function when a violation occurs.

According to the methods described in [98], the formulation of the penalty associated with restrictions has been completed. Therefore, the penalty for each equality condition is proportional to the distance squared from the target value. For inequality restrictions, we add slack variables.

Each penalty is after that weighted by its lagrangian multipliers λ_v and applied to the cost function as reported in Equation 6.1. Equation 6.7 describes the final formulation of the global optimisation problem.

$$E_{problem} = E_{cost} + \sum_{v \in V} \lambda_v \cdot P_v \tag{6.7}$$

6.2.7 Discretization and Slack Variables

Inequality constraints 6.4 on node resources, link resources (6.5) and delay (6.6) require the introduction of slack variables in order to be represented. This is a clever and simple workaround to the non-native capability of representing inequalities in the formulation. These slack variables have to assume values ranging from zero to the constraining limit, thus avoiding the exclusion of feasible solutions.

Since the variation interval of the slack terms is dependent on the unit of measure of the particular resource, the effect of the slack terms is discretised by multiplying them by a discretisation factor that is resource-dependent. This approach reduces the number of slack variables for resources whose ranges can be large due to the chosen unit of measure, e.g., the RAM amount represented in megabytes. It also allows for balancing the number of slack variables needed by the various constraints.

If discretisation is introduced, it must be followed by slack variables and constraining factors; otherwise, the formulation of the optimisation problem becomes incoherent. Even if the restriction is not breached, it may become hard to achieve zero energy penalty if the limitation holds.

6.3 Methodology

This section describes the approach adopted to validate the QUBO formulation for the VNFEP. As a reminder for the reader, the proposed formulation has been tested using a quantum annealer and two classical solvers. In the following, one can find a description of all the adopted solvers:

- **Quantum Annealing (QA):** a metaheuristic implemented employing a quantum device to find quantum states with low energy. For our analysis, D-Wave Advantage 4.1 Quantum Processing Unit (QPU) has been exploited.
- **Tabu Search:** a local search metaheuristic that only accepts inferior solutions if no previously discovered superior option is available [99].
- **Simulated Annealing (SA)**: a known metaheuristic which simulates thermal fluctuations [100]. SA undertakes a local search for better answers and stochastically accepts bad ones with a probability that relies on a decreasing temperature parameter at each step.

This decision of these three solvers is supported by the testing methods described below. One of the first concerns about using the QPU as a solver is the restricted number of available qubits. Moreover, given a QUBO formulation for a specific VNEFP (also addressed as a QUBO problem, where topology, SFCs, and parameters are known), an initial evaluation of the possibility of embedding this problem into the actual QPU has to be performed. Minor embedding [101] is a technique that allows converting the QUBO problem, generally represented as a graph, into an equivalent graph where there is a mapping between the binary variables of the problem and the available qubits and an additional mapping between links of variable and coupling of the physical qubits. Clearly, these mappings consider the physical properties of the annealer. As a note for the reader, this work leverages the `minorminer` library [1] to solve the minor embedding problem with a heuristic algorithm that executes in polynomial time.

Often a chain of several qubits may represent a QUBO variable. This is done, for instance, if the links between QUBO variables do not immediately fit on the quantum annealer hardware. Clearly, all qubits representing the same QUBO variable must have the same value for a solution to be consistent. This also implies that the number of qubits needed to solve a QUBO problem may be much more than the number of variables it contains.

[1] https://docs.ocean.dwavesys.com/projects/minorminer/en/latest/

In order to improve the clarity of the successive tests, the reader can find the list of the adopted metrics in the following:

- *Chain strength:* the chain strength allows controlling how strong the connection between qubits representing the same QUBO variable is[2].
- *Energy:* energy of the solution, i.e., the value calculated according to Equation 6.7 given the corresponding variable values. This metric is used to assess the quality of a solution; the lower, the better.
- *QUBO variables:* the number of QUBO variables required to formulate a specific VNFEP.
- *Qubits:* the number of qubits required by the Quantum Annealer to map the QUBO problem to the QPU, resulting from the minor embedding process.
- *Reads:* the number of solutions sampled by a solver. The higher the number of Reads, the more likely it is to find an optimal solution and the longer the time required by the solver is.

For what concerns the computational cost, the are these additional metrics:

- *Solver time:* the time required by a solver to find the desired number of solutions, i.e., Reads, as measured by the local client. Since the QPU is accessible via a cloud interface, the QPU solver time only accounts for the time required by the QPU, excluding the delay incurred to send the problem via the Internet.
- *QUBO time:* the time required for the QUBO formulation generation according to the formulation described in section 6.2 for a specific VNFEP.
- *QA time:* annealing time used by the QPU. As in the case of the number of Reads, it increases the probability of success in finding a good solution. Clearly, increasing these values increases the Solver time[3].

Obtaining an optimal solution, which corresponds to the lowest energy value, requires setting various parameters of the quantum annealer, such as the number of reads and the chain strength. Beyond finding the optimal solution to the same QUBO problem using all

[2]https://www.dwavesys.com/media/vsufwv1d/14-1041a-a_setting_the_chain_strength.pdf
[3]https://docs.dwavesys.com/docs/latest/c_qpu_annealing.html

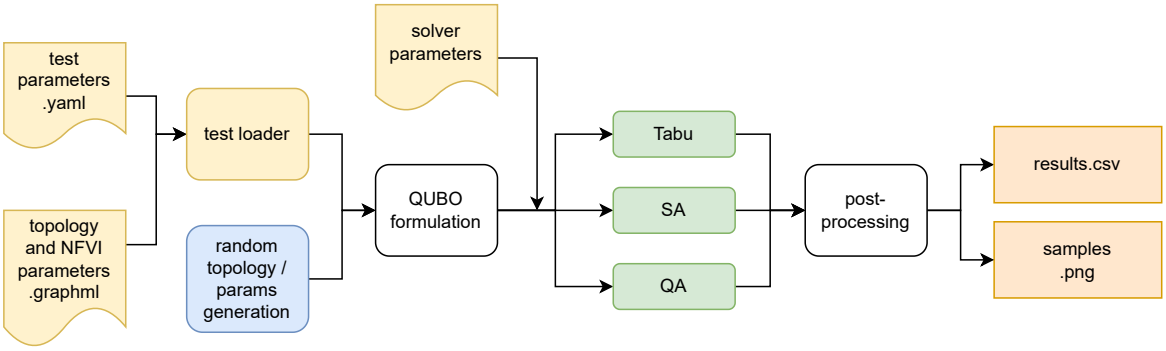

Fig. 6.2 High-level workflow of the testing process. An example of a solution is shown in Figure 6.8 and the additional data in Table 6.4.

Table 6.1 VNFs and constraint requirements for the SFCs.

SFC Name	VNFs	Bandwidth (Gbps)	Delay (ms)
`simple secaas`	NAT, Firewall	1	2
`medium secaas`	Firewall, IDS, Reverse proxy	2	3
`complex secaas`	NAT, Firewall, IDS, Reverse proxy, WAF	3	5

solvers, one can also compare the execution time to achieve the final result. In the case of the quantum annealer, this is one of its points of strength.

The number of SFCs and VNFs representing the demand in terms of allocation is the primary factor determining the optimisation problem's size and complexity. Another important aspect is the size of the topology in terms of the number of nodes and links as well as their associated costs.

Figure 6.2 shows a summary of the different steps of the testing process. The source code to replicate all the tests and manage the QUBO formulation is available on GitHub[4].

The QUBO formulation module provides an appropriate formulation given a specific VNFEP (i.e., topology and SFCs to be embedded) as input. This formulation is sent to the three solvers, adding information on the parameters for their configuration, e.g., the number of Reads and chain strength. The last step refines and saves the results coming from the solvers. As a remark for the reader, in this workflow, we also track the time for generating the QUBO formulation, the number of binary variables for the QUBO problem, and the number

[4]https://github.com/ignaziopedone/VNFQuantumOptimization

Table 6.2 Resources required by the available VNFs.

Resource	Firewall	IDS	NAT	Reverse proxy	WAF
CPUs	3	4	1	3	3
RAM (GB)	2	4	2	4	3

Table 6.3 Defined network topologies.

	net0	net1	net2	net3	net4	net5
Nodes (#)	4	5	5	10	8	7
Edges (#)	6	8	8	17	11	12

of target variables after the mapping (QPU qubits required). This allows us to present more consistent results in terms of execution time and better characterise the size of each specific problem.

As an example, one of the first tests studies the number of QUBO variables according to specific combinations of network topologies, SFCs, and VNFs. This analysis also allows determining which subset of networks and demands can fit on the QPU's limited number of qubits. The core test and validation can be divided into two parts, detailed in the following sections: the validation of the QUBO formulation using classical solvers and the tests on the quantum annealer comparing the results with the classical solvers' baseline.

6.3.1 QUBO formulation validation through classical solvers

As a part of the first core test, one can observe the analysis of the QUBO formulation in terms of the computational cost to generate the QUBO problem and the quality of the solution (i.e., level of energy) achieved using several networks of growing complexity.

Defining a sequence of topologies as Erds-Rényi graphs with four to thirty nodes and varying edge formation probability, the first core test solves different QUBO problems of increasing size (SFCs, VNFs) using classical solvers, i.e., Tabu, SA. To validate the constraints of the formulation, the VNFEPs have been solved without considering costs for the allocation of resources so that the expected energy of the optimal solution was zero. Clearly, in this case, the well-known conditions for the target VNFEPs allow for not breaking the constraints (Equation 6.1). After the preliminary validation of the constraints, cost components for the VNFEPs are introduced again so that one can calculate a baseline of results (optimal solutions) to have a comparison with the quantum annealer solver for the same VNFEPs.

6.3.2 Quantum Annealing Effectiveness

The second core test or analysis assesses the feasibility of adopting QA for VNFEPs and the advantages and disadvantages over the classical solvers.

All tests are conducted using small networks according to the current limitations of QA hardware. SECaaS is the use case scenario for these tests, assuming a service provider willing to optimise the allocation of VNFs on its infrastructure to grant on-demand security services to its customers. This provider has to offer those services to small and medium-sized enterprises and can choose between three different options: `simple secaas`, `medium secaas`, and `complex secaas`. Each one of these service options corresponds to an SFC that can be deployed on the provider infrastructure and contains a specific number of VNFs as depicted in Table 6.1. More details about the characteristics and resource requirements of those VNFs are available in Table 6.2.

As a remark for the reader, the QPU solver has three parameters that affect the quality of the solution, i.e., Reads, QA time and chain strength, that need to be selected. As in the previous case, parameter selection is optimised to find the best solution. The analysis has been performed in three steps: starting with a simplified formulation only using part of the constraints, then including costs, and finally considering the complete VNFEP formulation.

As mentioned before, minor embedding is necessary in the case of QA to map the QUBO problem on the QPU. As the VNFEP becomes more complex, this procedure requires more time. It is known that the quality of the minor embedding plays an important role in the quality of the solutions. Nevertheless, in our case, the full QUBO problem is highly connected (most of the quadratic terms are nonzero); thus, the usage of a precomputed embedding for a fully-connected QUBO problem of the desired number of variables and the target QPU is possible. The provider can also supply such an embedding scheme when he offers access to the QPU. Because of this, in this work, the minor embedding time is not analysed. A further comment for the reader is that with the technological advancement of the QPU and an increase in the number of available qubits, this problem can be automatically mitigated. section 6.4 provides a quantitative analysis of the results from the tests mentioned above. As already remarked, all the performed tests, adopted topologies, and in-depth analyses are publicly available as part of the online material.

6.4 Results and Discussion

This section describes the results of the tests presented in section 6.3.

6.4.1 QUBO formulation validation through classical solvers

As a reminder for the reader, this first group of tests has been performed over a set of randomly generated QUBO problems (from VNFEPs) of different sizes and complexity (depending on the number of nodes within the topology, SFCs, and VNFs). Given these inputs, the first results show the number of QUBO variables required to describe the problem. The parameters that impact the most are the number of SFCs and, in particular, the number of VNFs corresponding to the length of the individual SFC.

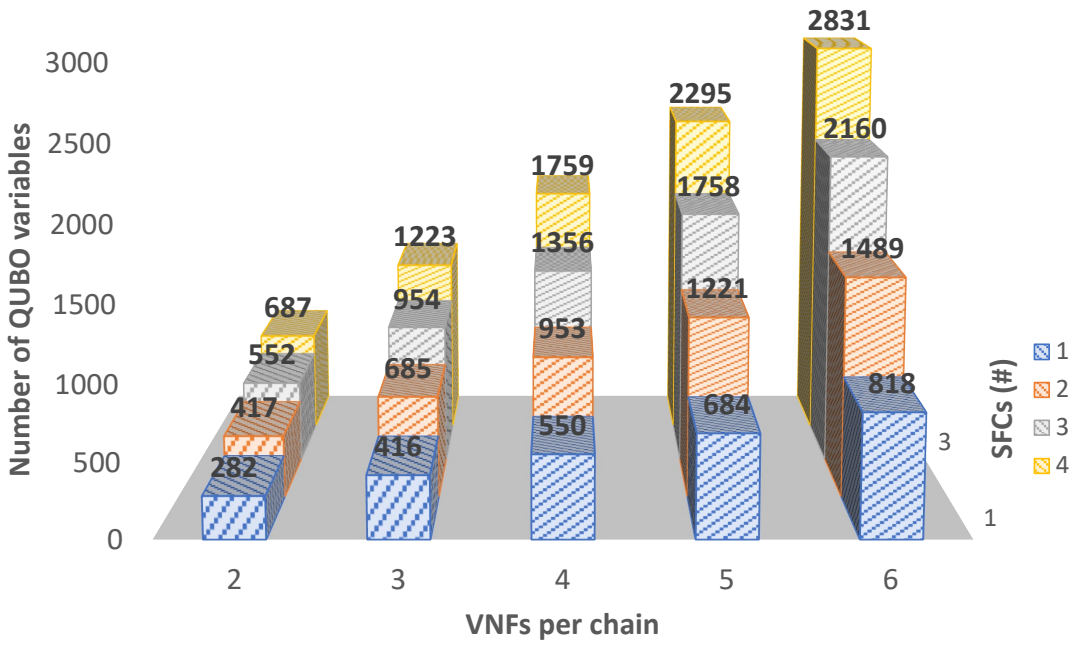

Fig. 6.3 QUBO variables obtained depending on the number of SFCs and VNFs per chain. Number of nodes is fixed at 20.

Figure 6.3 depicts the results of this calculation for a topology of 20 nodes (other parameters vary in two dimensions). Observing the already large number of variables (2831) required to represent 4 SFCs with 6 VNFs per chain allows assessing how fast the number of binary

variables grows. This result already makes mapping the problem to the QPU challenging since the maximum number of qubits for the current D-Wave architecture is 5000 and the minor embedding process usually requires far more qubits than the initial QUBO variables.

Figure 6.4 illustrates the time required to generate the QUBO problem depending on the number of QUBO variables and compares this data with the time needed by the solvers to find an optimal solution. As expected, QUBO problem time grows with the number of variables. From a different perspective, it is significant to observe how the latter time is similar to SA and Tabu times. Although several factors might influence this comparison, such as differences in how optimised their implementation is, this nonetheless indicates how crucial it is to account not only for the solver time but also for the computational cost required to generate the QUBO problem itself. This mirrors similar findings in [102].

Proceeding with other results, Figure 6.5 and Figure 6.6 measure the energy of the solution (also addressed as the quality of the solution) and the time needed by the solver to find that solution. In this case, those tests were conducted by using a 20-node topology and increasingly adding VNFs and SFCs. As the QUBO problem grows, the two metrics (energy and time) increase. The interesting result is that both solvers show similar results. Figure 6.6 also confirms the impact of the QUBO time on the total time required for solving the QUBO problem for different solvers.

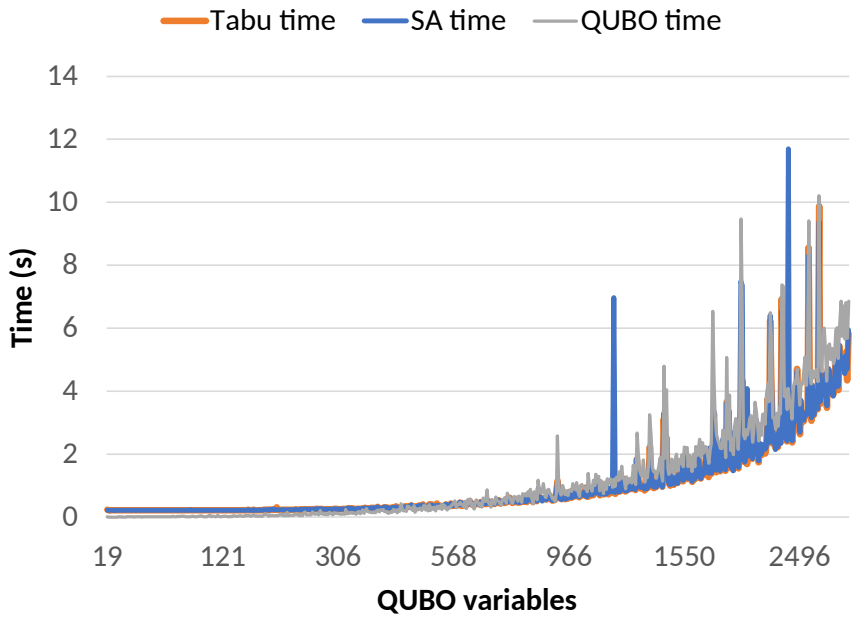

Fig. 6.4 Tabu Search and SA total time depending on the number of QUBO variables. This test does not apply costs to the objective function but only to the constraints.

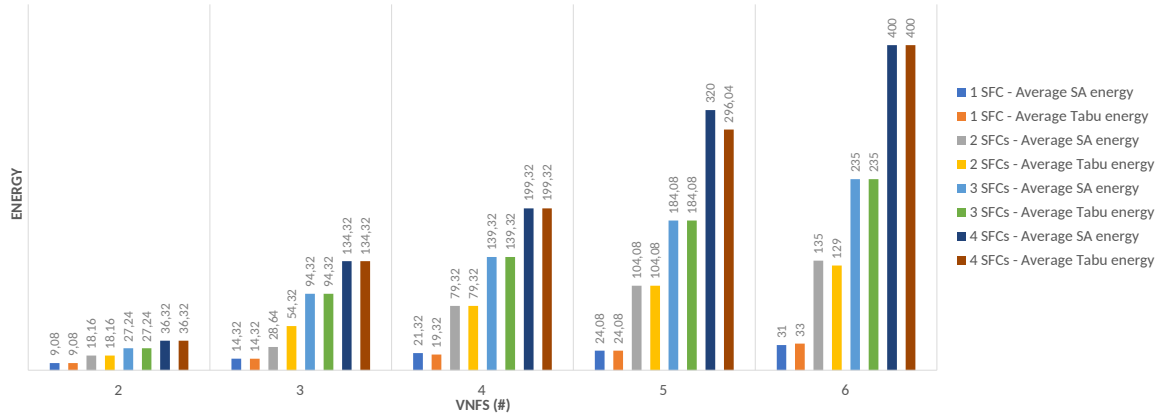

Fig. 6.5 Tabu Search and SA lowest energy values depending on the number of SFCs and VNFs per chain with a topology of 20 nodes.

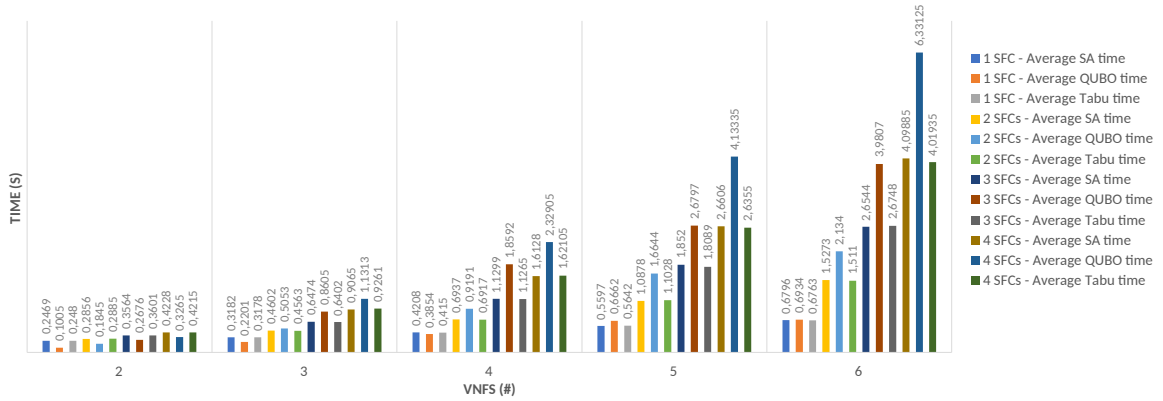

Fig. 6.6 Tabu Search and SA total time depending on the number of SFCs and VNFs per chain with a topology of 20 nodes.

6.4.2 Quantum Annealing Effectiveness

section 6.3 explains the nature of this other class of tests. The question one wants to address is whether the QA can efficiently solve VNFEPs problems and, in this case, what the advantage is over classical solvers. As a remark for the reader, the source code for replicating the tests, the possible settings, and further data are available online at the repository specified in section 6.3.

Among all the available online results, this section focuses on the `net3` topology, which contains as demand the allocation of two SFCs, `basic_secaas` and `medium_secaas` mentioned in Table 6.1.

Table 6.4 QPU vs classical solvers using `net3` with `simple` and `medium secaas`.

Problem	Solver	QUBO variables	Qubits	Solver time (s)	QUBO time (s)	Chain Strength	Lowest Energy	QA time (s)	Reads
Allocation and Continuity Constraints Only	SA	48	-	0,1261	0,0036	-	0	-	10^2
	Tabu	48	-	2,1120	0,0036	-	0	-	10^2
	QA	48	138	0,2033	0,0036	150	0	20	10^3
Allocation and Continuity Constraints with Costs	SA	48	-	0,1187	0,0036	-	33	-	10^2
	Tabu	48	-	2,1093	0,0036	-	33	-	10^2
	QA	48	138	0,0223	0,0036	150	33	20	10^2
Full VNFEP Formulation	SA	163	-	0,4716	0,0437	-	69	-	10^2
	Tabu	163	-	2,1797	0,0437	-	69	-	10^2
	QA	163	2194	0,0366	0,0437	150	204	40	10^2
	QA	163	2194	2,8495	0,0437	150	153	50	10^4

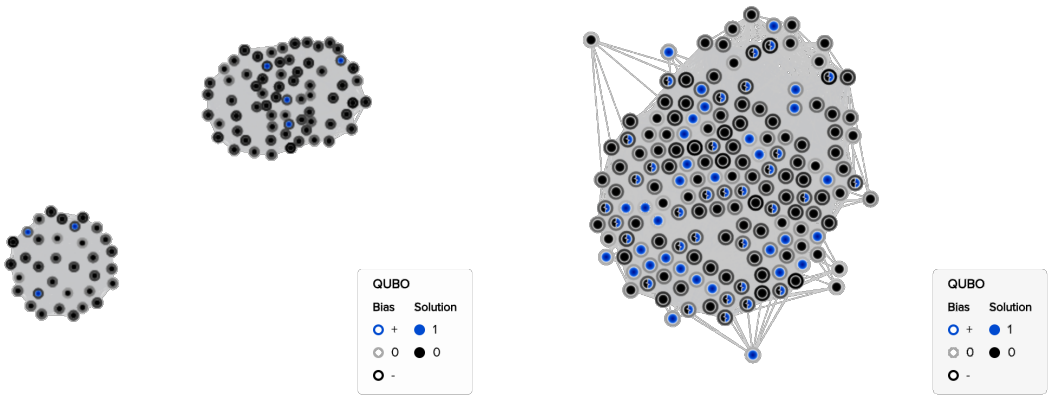

Fig. 6.7 Visualisation as a graph of the QUBO problem of Table 6.4. Binary variables are nodes; an edge exists between them if they appear in a quadratic term. On the left, the experiment only includes allocation and continuity constraints; on the right, the experiment with the complete VNFEP QUBO formulation.

For the first test, costs of the objective function were not applied. In addition, the considered QUBO formulation only contained allocation and continuity constraints (these constraints do not require slack variables). After the minor embedding, the QUBO problem counted 48 binary variables and 138 target qubits. This result is expected and mentioned in section 6.3, since the minor embedding allocates multiple qubits in a single chain which corresponds to a single QUBO variable. This process substantially increases the number of target variables (qubits).

Regarding the parameter setting of the QA, the chain strength is an essential factor. A high chain strength value may outweigh the other energy components and provide less optimal solutions. A value too small may cause the QA to produce inconsistent solutions where qubits connected to the same QUBO variable have different values. In practice, there is no ideal value for the chain strength; this value is usually tuned by performing different trials. The guideline followed is to determine this value as the largest absolute value in the problem's QUBO and then fine-tune it manually (from D-Wave whitepapers[5]). The selection of the lagrangian multipliers necessary to reflect as penalties the constraints demanded by the VNFEP formulation raises a similar problem; see section 6.2.

Table 6.4 summarises the meaningful findings. All quantum and classical solvers can find energy-zero solutions in the first instance of the test, only considering allocation and continuity constraints and excluding objective function costs. This demonstrates that all these constraints are met. When comparing solver times, SA is the quickest, while the traditional Tabu is the slowest. However, it should be emphasised that the QPU may examine ten times as many options as the SA while only taking two times as long.

The second experiment maintains the same constraints and adds the costs of the objective functions. The lowest energy in this situation will, by definition, be greater than zero. Once again, we can observe in Table 6.4 that all solvers find the same solution with an energy value of 33. QA is substantially quicker in this scenario than in the prior one, outperforming all traditional solutions. This is because QA could identify the optimal solution with fewer reads than in the prior instance. Figure 6.8 provides a visual representation of the solution, with colours denoting the allocation of the two SFCs among three physical nodes.

An observation for the reader is that the resultant QUBO problem is formed of a group of more minor independent problems, one for each SFC, when only allocation and continuity constraints are allowed. The `simple_secaas` and `medium_secaas` are visible on the left side of Figure 6.7 with the `simple_secaas` needing the fewest variables. When all costs and limitations are taken into account by the QUBO formulation, the same VNFEP is shown

[5]https://www.dwavesys.com/media/vsufwv1d/14-1041a-a_setting_the_chain_strength.pdf

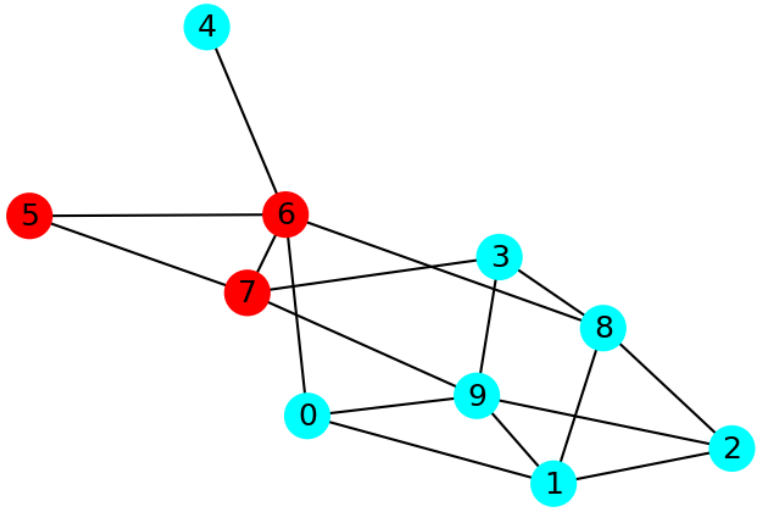

Fig. 6.8 Example of a solution from the experiment in Table 6.4. The VNFs of SFCs S0 and S1 are allocated on the red nodes inside the adopted network topology. The string at the bottom left describes the results of the embedding process and its energy. The two VNFs of S0 are allocated to nodes 6 and 7, and the three VNFs of S1 are to nodes 5, 6, and again 5. The two virtual links of S1 are both embedded in the same physical link (5-6).

on the right side of the same figure. The results show a densely linked network in which the various SFCs cannot be distinguished.

The main factor for the growing connectivity is the introduction of constraints on the node and links resources in the QUBO problem. This finding is crucial because it shows how such constraints significantly affect the number of links between QUBO variables, which raises the difficulty of embedding it on the QPU and the needed amount of qubits.

Table 6.4 also evaluates the whole QUBO formulation. The first clear result is the increased number of QUBO variables required to represent the problem and, consequently, the number of target variables (qubits). The necessity for slack variables to define inequality constraints on the resources has increased the number of QUBO variables. The required number of qubits for each QUBO variable jumps from 2.87 in the prior experiment to 13.46, which is a significant increase. This is most likely caused by the QUBO problem's extremely high connectivity (see Figure 6.7), which makes it much more challenging to ensure that all necessary physical connections are created on the QPU, resulting in the need for numerous additional qubit chains to represent every QUBO variable. QA is less effective than traditional solvers in terms of solution quality. The slight improvement that results from increasing the

Table 6.5 QPU vs classical solvers using `net5` and three `simple secaas` services.

Problem	Solver	QUBO variables	Qubits	Solver time (s)	QUBO time (s)	Chain Strength	Lowest Energy	QA time (s)	Reads
Allocation and Continuity Constraints Only	SA	72	-	0,1821	0,0021	-	0	-	10^2
	Tabu	72	-	2,1947	0,0021	-	0	-	10^2
	QA	72	250	0,0333	0,0021	50	0	50	10^2
Allocation and Continuity Constraints with Costs	SA	72	-	0,1862	0.0047	-	27	-	10^2
	Tabu	72	-	2,1319	0.0047	-	27	-	10^2
	QA	72	252	0,0299	0.0047	50	27	50	10^2
Full VNFEP Formulation	SA	128	-	0,3918	0.0355	-	27	-	10^2
	Tabu	128	-	2,1600	0.0355	-	27	-	10^2
	QA	128	1335	2,7550	0.0355	50	75	50	10^4

number of readings from 10^2 to 10^4 comes at the cost of a much longer QPU solver time. It can be observed that the QPU solution breaks the allocation constraints with 10^2 reads and 204 as the energy value.

The substantially greater connectivity of the QUBO problem, the longer qubit chains, and the need for several slack variables to describe the inequality restrictions are all potential causes of the QA's different effectiveness in this test. Long qubit chains have a detrimental effect on heavily linked problems, a well-known complication.

On the one hand, chains make the QA process more *rigid* since it is energetically more straightforward to change the state of a single qubit than a dozen linked ones at once. Furthermore, a solution is probably suboptimal or violates constraints if a chain breaks and the qubits have different values. Because of this, refining the suggested VNFEP formulation to reduce connection and hence increase effectiveness may be a future research work.

The analysis's findings hold for the other networks. The results of the same trials using the `net5` topology and allocating three `simple_secaas` services are shown in the Table 6.5. All solutions also satisfy the allocations and continuity restrictions in this example. When the cost is considered, all solutions provide the same lowest energy solution. QA is the quickest solver in this situation and can locate the best answer with 10^2 readings. The number of QUBO variables and the average number of qubits needed for each QUBO variable rises when the whole VNFEP formulation is again considered (from 3.47 to 10.42). Again, this causes QA to be less successful even with a large number of readings, 10^4.

Chapter 7

Quantum Offensive Security: the impact of Shor's Algorithm

As described in chapter 2, quantum computing poses a danger to current PKC schemes due to the quantum speedup that particular algorithms experience as a result of this new computing paradigm while solving mathematical problems such as prime factoring, DLP, and ECDLP.

This chapter tries to provide an in-depth analysis of Shor's algorithm, from which the quantum threat originates, and its function in prime factorisation. In addition, we explore varied efforts towards optimising the available quantum circuits to execute it, provide alternative implementations based on the Qiskit toolkit, and estimate the required resources to run it on practical hardware devices.

7.0.1 Methodology

This section briefly summarises the methodology used for the work on Shor's Algorithm. This work can be divided into two parts: Shor's algorithm applied to prime factoring (RSA) and applied to ECDLP (ECC).

The first part described in this chapter (chapter 7) involved:

1. analysis of the basic Shor's algorithm proposal.
2. Study of available practical implementations on state-of-the-art toolkits, executed on both simulators and real hardware.
3. Definition of the metrics to evaluate the analysed quantum algorithms and implementations (e.g., depth of the quantum circuit).

4. Resource estimation of the above implementations.
5. Analysis of the proposed enhancements to the basic implementations of Shor's algorithm available in the literature.
6. Implementation of both classical and literature-based enhancements using Qiskit toolkit and IBM Q simulator.
7. Porting of the implementations above from Qiskit to the Rigetti framework.
8. Testing and comparison of the standard and enhanced versions of Shor's algorithm using both Qiskit and Rigetti frameworks.

The second part is described in chapter 8 involved:

1. study of the application of Shor's algorithm to DLP and ECDLP.
2. Analysis of the proposed implementations in the literature.
3. Development in Qiskit of the basic quantum circuits and components to implement the proposed quantum algorithms.
4. Resource estimation for the proposed quantum algorithms and comparison with the results in chapter 7.
5. Testing of the implemented quantum circuits.

7.1 Shor's algorithm overview

Shor's algorithm, proposed by Peter Shor in 1994, allows for solving prime factoring in polynomial time. As an example, using Schoenhage-Strassen's algorithm for fast multiplication [103], we need a number of quantum gates of the order $\mathcal{O}(n^2 log(n) loglog(n))$, where n is the number of bits of the key. The most efficient classical method, called General Number Field Sieve (GNFS) [104], works sub-exponentially in the order of $\mathcal{O}(e^{1.9(log(n))^{1/3}(loglog(n))^{2/3}})$.

The idea behind solving integer factorisation using Shor's algorithm is divided into two parts: one classical, i.e., can run on a classical computer in polynomial time, and another quantum, i.e. it needs a quantum machine to be run in polynomial time. The first part derives from a more general classical algorithm to find the factors p and q of a biprime number N. Moreover, as a first step, this algorithm picks a random integer $x < N$ and computes $G = gcd(x, N)$, the greatest common divisor of x and N. If $G \neq 1$, then we have found a nontrivial divisor of N, and we do not need to continue with the procedure. If not, we can

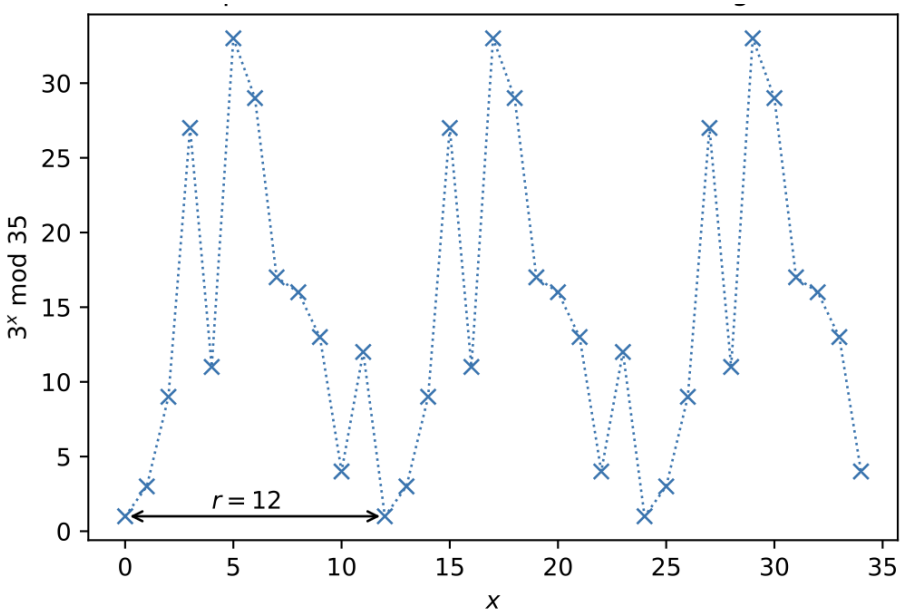

Fig. 7.1 Plot of the function $f(x) = 3^x \pmod{35}$. The period r is highlighted. Source: [2].

proceed by finding the period, or order, r of the function $x^r \pmod{N}$. In this case, if the period is even, we are able to compute the factors p and q with the Extended Euclidean Algorithm:

$$p = \gcd(x^{r/2} - 1, N) \qquad 1 = \gcd(x^{r/2} + 1, N) \tag{7.1}$$

We can easily observe that p is a non-trivial factor of N:

$$(x^{r/2} - 1)(x^{r/2} + 1) = x^r - 1 \equiv 0 \pmod{N} \tag{7.2}$$

If we either got an odd value of r or $x^{r/2} \equiv -1 \pmod{N}$, then the algorithm fails, and we need to start again with the first step.

Given a random number x, the aforementioned classical algorithm allows us to find a factor of N with probability $1 - 2^{1-k}$, where k represents the number of distinct prime factors of N. For a biprime N - as in the RSA algorithm case - the probability is equal to $1/2$. The step of the previous algorithm related to the period-finding problem is known to belong to the NP class. Because of this, Shor's algorithm comes into play allowing us to solve it in polynomial time. This corresponds to the quantum and central part of the algorithm.

Once described the general framework in which Shor's algorithm operates, subsections 7.1.1 and 7.1.2 give quantum computing preliminaries to appropriately describe the core of the algorithm in subsection 7.1.3.

7.1.1 Quantum Fourier Transform

The Quantum Fourier Transform (QFT) is the quantum implementation of the classical discrete Fourier transform over the amplitudes of the wavefunction and represents one of the building blocks of quantum algorithms such as Quantum Phase Estimation (QPE). The primary purpose of the QFT is to transform multi-qubit states between computational ($|0\rangle, |1\rangle$) and Hadamard ($|+\rangle, |-\rangle$) bases [2]:

$$QFT\,|x\rangle = |\widetilde{x}\rangle$$

Differently from what happens with computational basis, where numbers are stored in binary, in the Fourier or Hadamard basis, we follow the Equation 7.3. This means that as a number increases, we have different rotations on the X-axis on the various qubits adopted for the representation. As an example, if we assume 4 qubits, an increment of 1 will lead to a rotation of $\pi/8$ for qubit 0, $\pi/4$ for qubit 1, $\pi/2$ for qubit 1, and π for qubit 0.

$$|\tilde{x}\rangle = \frac{1}{\sqrt{N}} \bigotimes_{k=1}^{n} (|0\rangle + e^{\frac{2\pi i}{2^k} x}\,|1\rangle) \tag{7.3}$$

As depicted in Figure 7.2, the quantum circuit to implement QFT involves two types of gates: Hadamard and $CROT_k$. The latter can be defined as expressed in Equation 7.4.

$$CROT_k = \begin{bmatrix} I & 0 \\ 0 & UROT_k \end{bmatrix}, UROT_k = \begin{bmatrix} 1 & 0 \\ 0 & e^{\frac{2\pi i}{2^k}} \end{bmatrix} \Rightarrow CROT_k\,|1x_j\rangle = e^{\frac{2\pi i}{2^k} x_j}\,|1x_j\rangle \tag{7.4}$$

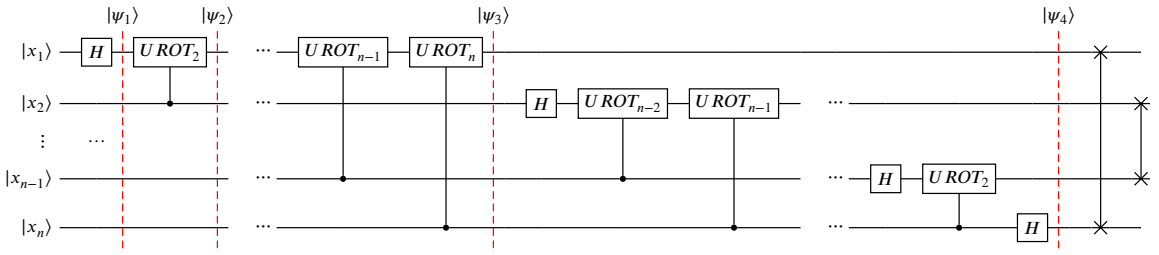

Fig. 7.2 QFT circuit (source: [2]).

In Equation 7.5, the development of the quantum state throughout various stages of the circuit is shown. This might be important for comprehending how quantum gates operate on the initial qubit register from the start to the final quantum state obtained.

$$
\begin{aligned}
|\psi_1\rangle &= \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2}x_1}|1\rangle] \otimes |x_2...x_n\rangle \\
|\psi_2\rangle &= \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1}|1\rangle] \otimes |x_2...x_n\rangle \\
|\psi_3\rangle &= \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^n}x_n + \frac{2\pi i}{2^{n-1}}x_{n-1} + ... + \frac{2\pi i}{2^2}x_2 + \frac{2\pi i}{2}x_1}|1\rangle] \otimes |x_2...x_n\rangle \\
&\implies |\psi_3\rangle = \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle] \otimes |x_2...x_n\rangle \\
|\psi_4\rangle &= \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^n}x}|1\rangle] \otimes \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^{n-1}}x}|1\rangle] \otimes ... \otimes \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2^2}x}|1\rangle] \\
&\otimes \frac{1}{\sqrt{2}}[|0\rangle + e^{\frac{2\pi i}{2}x}|1\rangle]
\end{aligned}
\tag{7.5}
$$

As final result, we observe the state $|\psi_4\rangle$, and to this state, we apply other swap gates in order to reverse the order of the factors (see the comparison with the state $|\psi_1\rangle$). The relevance of QFT and its inverse QFT† - in particular for algorithms such as Shor - lies in its capability of decoding information encoded in the phase of a quantum state and not directly available measuring the same state in the computational basis [3].

7.1.2 Quantum Phase Estimation

QPE is a crucial building block for several quantum algorithms; one of them is Shor. The purpose of QPE is to estimate the phase θ in the equation $U|\psi\rangle = e^{2\pi i\theta}|\psi\rangle$, where we have the eigenvector $|\psi\rangle$ and its eigenvalue $e^{2\pi i\theta}$. In Figure 7.3, we provide the high-level quantum circuit to implement QPE. From that figure, we observe that the phase of the U gates is encoded in the Hadamard basis of the t control qubits, a technique known as phase kickback [2]. Then, using QFT†, we are able to restore that information and measure θ in the computational basis.

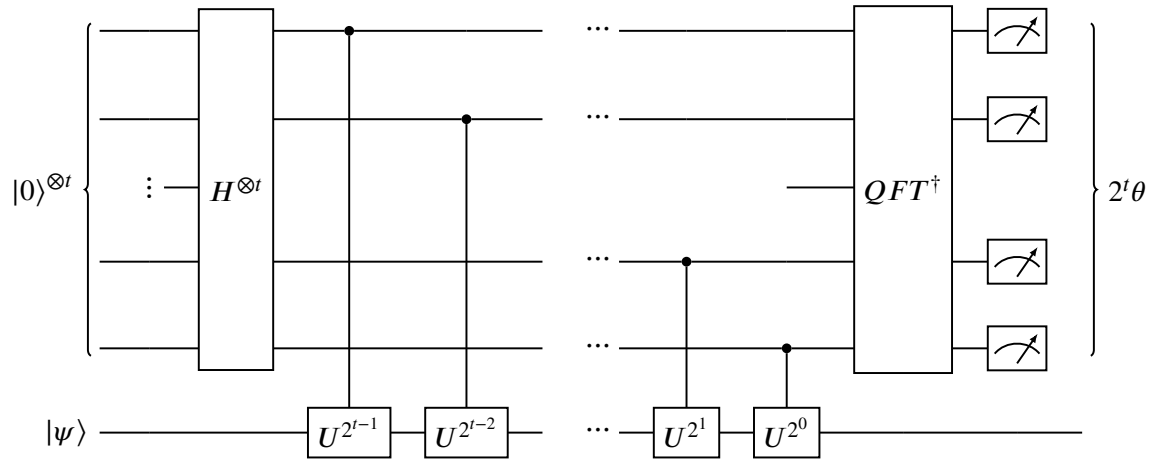

Fig. 7.3 QPE circuit (source: [2]).

The Equation 7.6 shows the results of the state related to the top register after the application of first the train of Hadamard gates - for obtaining the superposition of all possible states - and then the $2^t U$ gates. At this point, we get an equation of the same form as the one of the QFT, and one can effortlessly realise that applying $\text{QFT}^{\dagger}$ it is possible to retrieve $2^t\theta$ by simply measuring in the computational basis.

$$|\psi\rangle = \frac{1}{\sqrt{2^t}}(|0\rangle + e^{2\pi i\theta 2^{t-1}}|1\rangle) \otimes (|0\rangle + e^{2\pi i\theta 2^{t-2}}|1\rangle) \otimes \cdots \otimes (|0\rangle + e^{2\pi i\theta 2^{1}}|1\rangle) \otimes (|0\rangle + e^{2\pi i\theta 2^{0}}|1\rangle) \tag{7.6}$$

7.1.3 Shor's overall quantum circuit

In section 7.1, we discuss the classical algorithm for prime factoring and the need for a quantum algorithm to solve the period-finding problem. Here we combine the building blocks of Shor's algorithm and give the first implementation without focusing on specific optimisation.

The reader shall remember that the algorithm's starting point is the function $x^r \pmod N$ and the operation of finding its period r. To this end, Peter Shor proposed to apply the QPE to the unitary operator U defined in Equation 7.7.

$$U|x\rangle = |ax \pmod N\rangle \tag{7.7}$$

If one starts from state $|1\rangle$, k successive applications of the same operator lead to the state $|a^k \pmod N\rangle$. Referring to Figure 7.1, with $k = r$ steps, where r is the period of the function, the state return to $|1\rangle$. If we consider a superposition of all the eigenstates in this period, we obtain an eigenstate for the operator U:

$$|u_0\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |a^k \pmod N\rangle$$

This eigenstate has a limited interest since its eigenvalue is 1. Adding a different phase proportional to k for each basis state, we obtain the eigenstate:

$$|u_1\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-\frac{2\pi i k}{r}} |a^k \pmod N\rangle$$

The eigenvalue of the latter eigenstate is $e^{-(2\pi i k/r)}$. A further generalisation can be $|u_s\rangle = \frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} e^{-(2\pi i s k/r)} |a^k \pmod N\rangle$, where s is an integer such that $0 \leq s \leq r-1$. For this final eigenstate, f we sum up all the basis states, due to interference, all different phases cancel out leaving the state $|1\rangle$ alone:

$$\frac{1}{\sqrt{r}} \sum_{k=0}^{r-1} |u_s\rangle = |1\rangle$$

When the QPE is applied to U using the state $|1\rangle$, the result is the phase $\phi = s/r$. Using the continuous fraction expansion, it is feasible to get r beginning with ϕ. Beyond the general idea behind Shor's algorithm for factoring, in Figure 7.4, the reader can look at its possible implementation as a high-level quantum circuit.

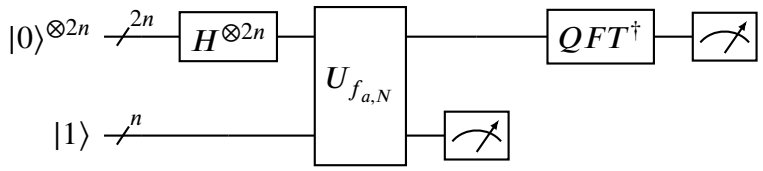

Fig. 7.4 High-level representation of the circuit for the Shor's algorithm. Source: [105]

The circuit starts with two quantum registers, one of length $2n$ and with the initial value set to $|0\rangle$ and the other of length n initialised to $|1\rangle$. The choice of the length of these registers depends, for the first, on an approximation to be able to retrieve the value of the period in the

final part of the algorithm after the estimation; the second register instead must contain the number N. The first operation - on the first register - is applying a train of Hadamard gates to obtain a superposition of all possible inputs. These correspond to all possible values of the period r and lead to the overall state:

$$|\psi\rangle = \frac{1}{2^n} \sum_{x=0}^{2^{2n}-1} |x\rangle\,|1\rangle$$

When we apply the oracle, i.e., the function that implements the modular exponentiation, we obtain the following state, where "q" is the power of 2 s.t. $n^2 \leq q \leq 2n^2$:

$$|\psi\rangle = \frac{1}{\sqrt{q}} \sum_{x=0}^{q-1} |x\rangle\,|a^x \pmod N\rangle$$

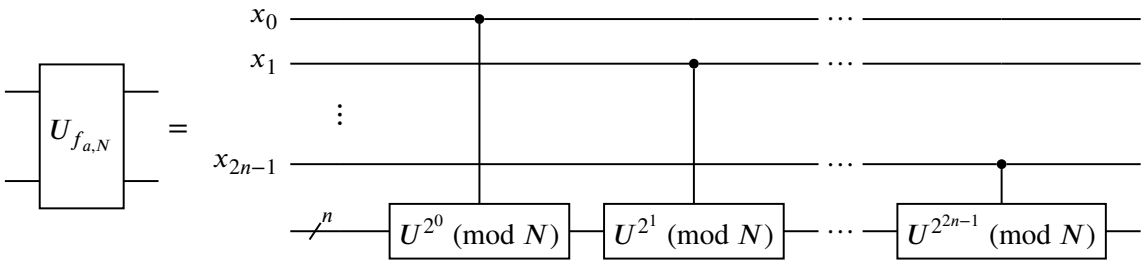

Fig. 7.5 Implementation of the oracle for the period finding function. Source: [105]

In Figure 7.5, there is a high-level design of the oracle $U_{f_{a,N}}$ whose implementation is treated in detail in section 7.2. As a reminder, this is a function $a^x \pmod N$ that can also be expressed as binary expansion. The next step involves applying the $QFT^\dagger$ to the first quantum register. This leads to the state:

$$|\psi\rangle = \frac{1}{q} \sum_{x=0}^{q-1} \sum_{c=0}^{q-1} e^{\frac{2\pi i x c}{q}} |c\rangle\,|a^x \pmod N\rangle$$

where $|c\rangle$ is the output of the first register. The last step of the quantum part is the final measurement and for this measurmement we can also compute the probability P of having a particular state $|c\rangle\,|a^k \pmod N\rangle$, where $0 \leq k \leq r$:

$$P(|c\rangle\,|a^k \pmod N\rangle) = \left|\frac{1}{q}\sum_{x\,:\,a^x\equiv a^k} e^{\frac{2\pi i x c}{q}}\right|^2 \overset{*}{=} \left|\frac{1}{q}\sum_{b}^{\lfloor\frac{(q-k-1)}{r}\rfloor} e^{\frac{2\pi i(br+k)c}{q}}\right|^2 \tag{7.8}$$
$$= \left|\frac{1}{q}\sum_{b}^{\lfloor\frac{(q-k-1)}{r}\rfloor} e^{\frac{2\pi i b r c}{q}}\right|^2$$

The sum of the above equation spans over all $x < q$ s.t. $a^x = a^k \pmod N$. Taking into account that r is the order of a, this sum can consider all x s.t. $x = k \pmod N$. The final result is reported in Equation 7.8, where x (after $\overset{*}{=}$) is written as $br + k$ ($b \in \mathbb{N}$). We can ignore the term $e^{(2\pi i k c/q)}$, since it can be excluded from the sum and it becomes 1. Peter Shor [106] proves that if rc is close to a multiple of q, then we get a large probability of measuring the corresponding states. Moreover, for a large n the probability is at least $1/3r^2$ if:

$$\left|\frac{c}{q} - \frac{d}{r}\right| \leq \frac{1}{2q}$$

Assuming that we measured the first register, we know the value of c/q. Considering that $q > n^2$ and $r < n$, there is at most one fraction d/r that satisfies the inequality Equation 7.9. We can obtain d/r by rounding the fraction c/q to the closest fraction with a denominator smaller than n. We can then classically found the value of this fraction in polynomial time using continuous fraction expansion [106] over c/q.

Shor also remarks that we have $\phi(r) \cdot r$ diverse states that allow to obtain a particular r. $\phi(r)$ is the Euler's totient function and the probability of obtaining a consistent value as output of the quantum circuit is $\phi(r)/3r$.

7.2 Shor's algorithm in-depth analysis

Relevant to the practical implementation of Shor's algorithm, there is the designing of the oracle, i.e. the core function of the algorithm, which is the most challenging part among all requirements. In this section, we discuss an in-depth view of the required elementary gates provided by [105]. In turn, we must define the elementary adder gate, the modular adder gate, and the controlled modular multiplier gate to build the controlled U_a gate, which is frequently used in the various steps of the quantum circuit.

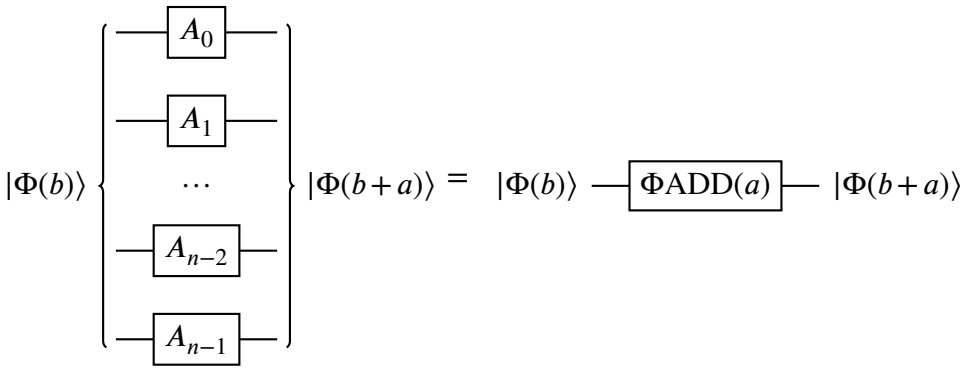

Fig. 7.6 Addition of a classical value a to a quantum register encoding b in the Fourier space. The A_i values are computed classically. Source: [105]

In Figure 7.6 we start by defining the adder gate which is in charge of adding a a classical random number smaller than N to a quantum register. This will be useful in the coming implementation of the aggregated gates. Two important remarks: the addition operation happens in the Hadamard basis so we need to be careful on adjusting the phase for each qubit (see Equation 7.9); in order to prevent overflow we apply the operation to an $n+1$ register.

$$A_i = \pi \sum_{j=0}^{i} a_j 2^{-(j-i)} \tag{7.9}$$

It is interesting to notice that applying the reverse of the $\Phi ADD(a)$ we obtain either $|b-a\rangle$ for $b \geq a$ or $|2^{n+1}-(a-b)\rangle$ for $b < a$. Clearly since we are in the Fourier basis, we need to apply a QFT before and a $QFT^\dagger$ after. The depth of the gate is 1. In the case of its controlled or double controlled version the depth becomes $n \cdot T$, where T is the depth of a Toffoli gate for a given architecture.

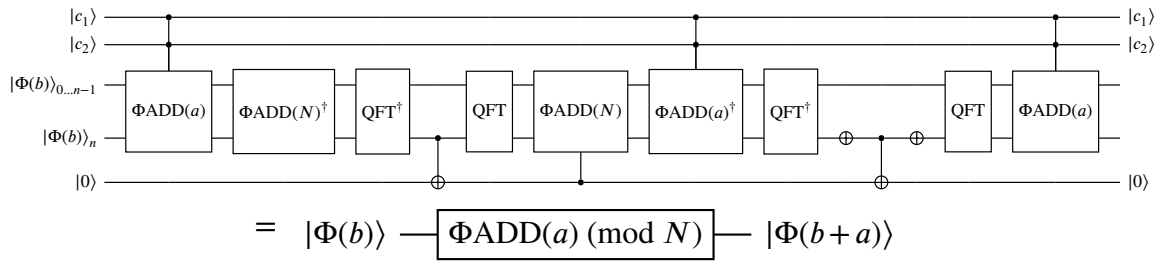

Fig. 7.7 Doubly controlled modular addition of a classical value a to a quantum register encoding b in the Fourier space modulo N. The result is $|\Phi(a+b) \text{ (mod } N)\rangle$ if $c_1 = c_2 = |1\rangle$, $|\Phi(b)\rangle$ otherwise. Source: [105]

In Figure 7.7 we have the modular adder gate which is in charge of performing the modular addition of a classical number a to a quantum register. In the previous gate we have that the register after the application of the QFT has its most significant bit (MSB) always to zero. In this circuit we can leverage this aspect since we know that the value of the MSB is $|1\rangle$ onfly if $a+b<N$, in fact, in this case we need to add N another time. It is easily understandable from Figure 7.7 that we can achieve this result by controlling the second $\Phi ADD(N)$ with the MSB of the previous sequence of operations. The number of qubits needed is $n+4$ while the depth (D) is:

$$\mathrm{D} = 4+4\cdot(2n+1)+1+3\cdot(n\cdot T) = 3n\cdot T+8n+9$$

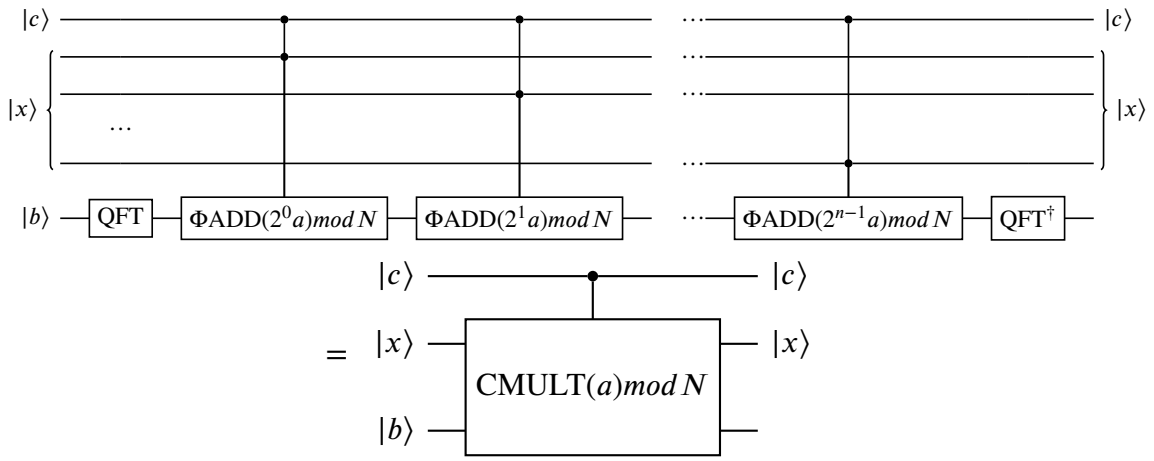

Fig. 7.8 Controlled modular multiplication of a classical value a times a quantum register encoding x modulo N, summing the result to a third register encoding the value b. The result is $|b+a\cdot x \pmod N\rangle$ if $c=|1\rangle$, $|b\rangle$ otherwise. Source: [105]

The final step before assembling the final gate is to build the controlled modular multiplier gate in Figure 7.8. The idea is to obtain $(ax) \pmod N$ by leveraging the identity:

$$(ax) \pmod N = (...((2^0 ax_0) \pmod N + 2^1 ax_1) \pmod N + ... + 2^{n-1} ax_{n-1}) \pmod N$$

The number of qubits are $2n+3$ and the depth of such a gate is:

$$\mathrm{D} = 2\cdot 2n + n\cdot(3n\cdot T+8n+9) = 3n^2\cdot T+8n^2+13n$$

The final unitary gate needed for the oracle in Figure 7.5 is depicted in Figure 7.9 and the number of qubits in this case are $2n+3$ and the depth:

$$\mathrm{D} = 2 \cdot (3n^2 \cdot T + 8n^2 + 13n) + (n \cdot T + 2) = 6n^2 \cdot T + 16n^2 + 26n + n \cdot T + 2$$

Considering the importance of the U_a gate, one can summarise the final action of this gate as $|c\rangle\,|x\rangle \rightarrow |c\rangle\,|a \cdot x \pmod N\rangle$ if $|c\rangle = |1\rangle$.

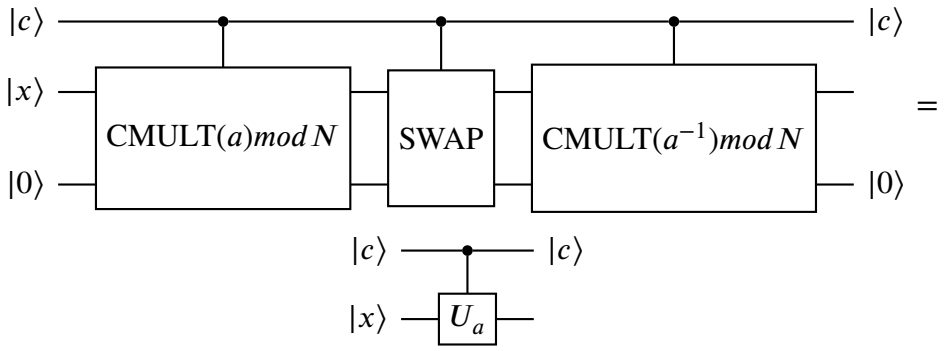

Fig. 7.9 Controlled U_a gate. The result is $|a \cdot x \pmod N\rangle$ if $c = |1\rangle$, $|x\rangle$ otherwise. Source: [105]

Looking at the Equation 7.10, the reader can easily observe that in order to vary the constant a^{2^i}, it is not necessary to apply U_a multiple times but just apply the $U_{a^{2^i}}$ calculating a^{2^i} classically.

$$a^i x \pmod N = (((a \cdot x \pmod N) \cdot x \pmod N)...) \cdot x \pmod N \tag{7.10}$$

According to this, the oracle requires $2n$ sequentially controlled $U_{a^{2^i}}$ gates, each controlled by a different qubit. The total width of the circuit, in this case, is $4n+2$, and the total depth will be the sum of the oracle, the Hadamard gate, and the $\mathrm{QFT}^\dagger$ applied to $2n$ qubits:

$$\mathrm{D} = 2n \cdot (6n^2 \cdot T + 16n^2 + 26n + n \cdot T + 2) + 1 + 2(2n) \simeq 12n^3 \cdot T + 32n^3$$

To summarise and make a helpful example, one can state that factorising a 2048-bit integer, i.e. in the RSA-2048 case, requires 8194 logical qubits. Additional but necessary consideration is that one, in order to have consistent logical qubits, needs to adopt QEC techniques with a code distance that allows managing the depth of the circuit with an acceptable error probability. This may result in a large number of physical qubits needed, as presented in the following sections. Last but not least, the topology of the quantum processor shall exhibit high connectivity to reduce the error rate of quantum gates, and the decoherence time of the qubits must be large enough to consider the depth of the circuit and the time needed to apply the quantum gates.

7.3 Shor's algorithm optimisation

In this section, we provide many optimisation techniques from the scientific literature. We developed a version of the first two, but the third is an essential case study due to the availability of a fairly current estimate of the resources required to factor RSA 2048.

7.3.1 Sequential QFT

Using the sequential version of the inverse QFT [107], we may minimise the number of qubits in our estimate, allowing us to utilise only one qubit for the register that will be measured: this lowers the width of the circuit from $4n+2$ to $2n+3$ without increasing its depth. However, a classically-controlled quantum gate is required, which IBM hardware lacks at present.

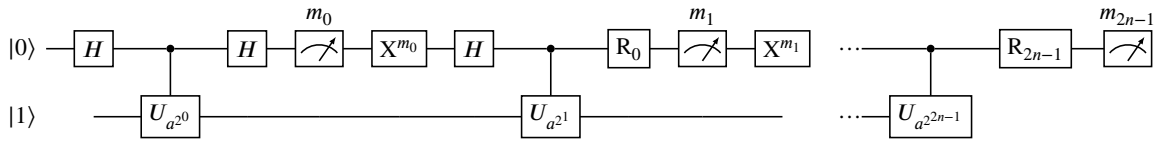

Fig. 7.10 Circuit for the Shor's algorithm with the sequential QFT. Source: [105]

This work provides a version of the circuit based on the aforementioned technique, despite the fact that we can only run it on a simulator. This version allows us to test the implementation by factoring larger numbers than the standard one considering that the number of required qubits restricts our tests' size. This technique is further discussed in [105]. In Figure 7.10, the reader can find a high-level design of the circuit. Additional remarks are that X^{m_0} are classically controlled CNOT gates, and when the value of m_i is one, the corresponding X^{m_i} is applied.

7.3.2 In-place addition

Another possible optimisation [108] permits to save one qubit using a dirty ancilla qubit instead of a clear one in the modular addition. Several dirty qubits are available when one performs the modular addition, such as the $|x\rangle$ register in Figure 7.8. Because of this, it is possible to run the circuit using just $2n+2$ qubits.

A modular addition (Figure 7.11) between a quantum register $|b\rangle$ and a classical value a compares $|b\rangle$ to $N-a$. When $b > N-a$, one removes $N-a$, otherwise adds a. The result is then compared to a in order to clear the ancillary qubit.

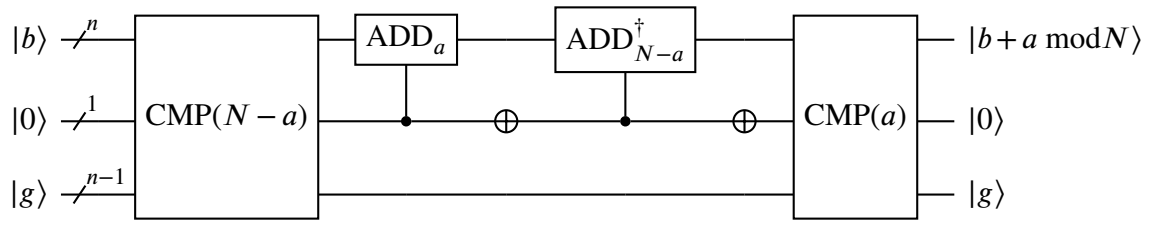

Fig. 7.11 Modular adder using $n-1$ dirty ancillae of register $|g\rangle$ and 1 clear qubit. Source: [109]

The comparator [109] requires a gate that computes the carry of the sum between a number encoded in a quantum register and a classical constant. This can be accomplished using $n-1$ dirty ancillae qubits, which are available at each iteration of the controlled modular adder in the x register represented in Figure 7.8.

To compute the final carry (Figure 7.12) of the addition between the number encoded in a register $|b\rangle$ and the classical value a, we must propagate the carry from each qubit to the last one. It is possible to toggle the qubit g_i of the auxiliary register when a carry from bit i to bit $i+1$ is required. If $b_i = a_i = 1$, g_i must toggle either $g_{i-1} = b_i = 1$ or $g_{i-1} = a_i = 1$. One can further preserve g_0, considering that it relies only on the condition $b_0 = a_0 = 1$, so a_0 can be replaced with g_0 when establishing the conditions for g_1.

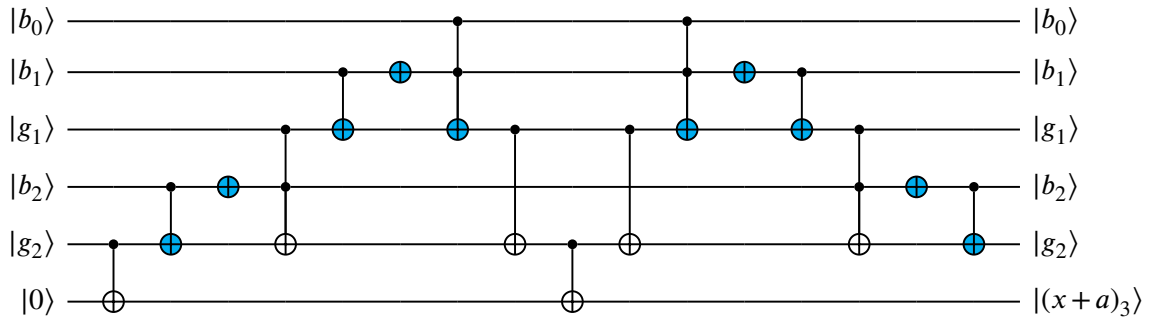

Fig. 7.12 Carry gate for $n = 3$. The blue gates are present if $a_i = 1$. In order to build the circuit for n qubits, we must copy the same three gates of the $g_1 a_2 g_2$ triple. Source: [109]

Another potential optimisation is based on not performing the addition in the Fourier space, such that we do not need all the QFTs of the modular addition circuit, although it does rely on Toffoli gates. In order to derive the constant adder, we need a gate that calculates the carry of the sum of a number stored in a quantum register and a classical constant utilising $n-1$ dirty ancillae qubits and a controlled incrementer. An incrementer may be generated for the controlled incrementer gate from a gate that performs the addition between two numbers

stored in two quantum registers of n qubits, one of which is composed of dirty auxiliary qubits:

$$|x\rangle|g\rangle \rightarrow |x-g\rangle|g\rangle \rightarrow |x-g\rangle|g'-1\rangle \rightarrow |x-g-g'-1\rangle|g'-1\rangle \rightarrow |x+1\rangle|g\rangle$$

The element $g'-1$ is the bit-wise complement of g and can be implemented by using n X gates. A controlled version of this gate may instead be implemented by controlling each gate and incrementing the register's least significant qubit using the controller. The authors of [110] propose a circuit to conduct the addition between two quantum registers by using no ancilla qubits and storing the carry of the operation (Figure 7.13).

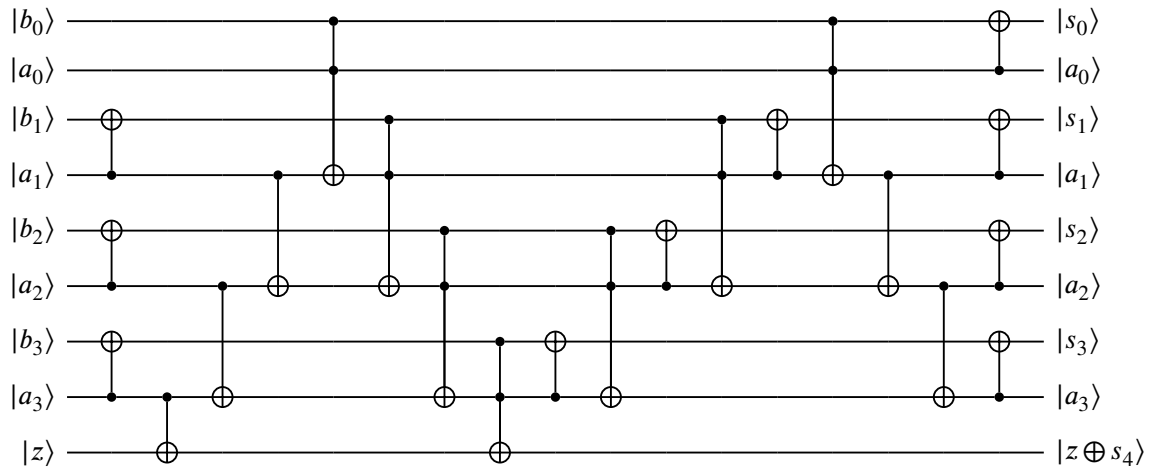

Fig. 7.13 Quantum circuit for the addition of two 4-bit registers. Source: [110].

A quantum circuit to perform the addition of a constant is depicted in Figure 7.14. The whole process begins by dividing the quantum register into two halves, the lower one, x_L and a_L, and the higher one, x_H and a_H). Afterwards, one can calculate the carry bit of $x_L + a_L$ leveraging the $|x_H\rangle$ register as ancillas. The final result controls the choice of whether incrementing $|x_H\rangle$. As a reminder, when the ancilla is set to $|1\rangle$, one needs to use one of them by adding another incrementer and inverting $|x_H\rangle$. Finally, the whole quantum circuit is recursively repeated for both halves.

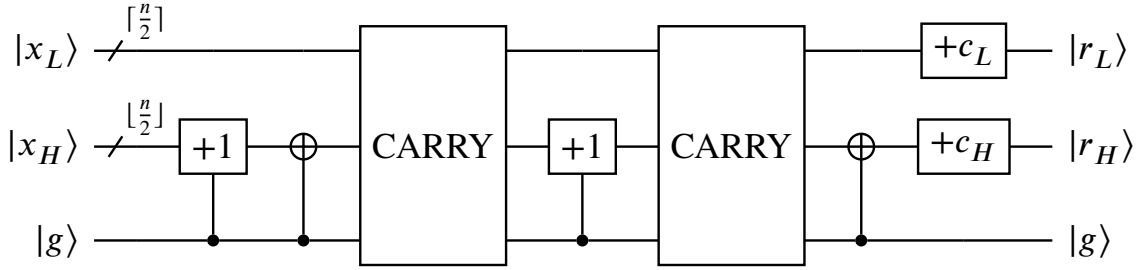

Fig. 7.14 Quantum circuit for adding a constant (a) to a quantum register. Source: [109]

The resulting gate is an essential building block for a modular adder with a Toffoli count of $8n\log_2 n + O(n)$ and considering the procedure presented in section 7.2, the total number of Toffoli in Shor's circuit is equal to $64n^3 \log_2 n + O(n^3)$.

7.3.3 Ekerå and Håstad's Algorithm

The authors of [111] proposed an interesting resource estimation work regarding Shor's algorithm for breaking RSA-2048. As a synthesis, the whole process would take 8 hours, utilising 20 million physical qubits and specific optimisation techniques. In this approach, variations of Shor's algorithm are utilised to minimise the number of multiplications, and windowed arithmetic is used to combine multiple of them. Specifically, the utilised circuit performs the DLP of the value y in the equation 7.11, where $g \in 1, ..., N-1$ is a random integer.

$$y = g^{N+1} \tag{7.11}$$

Applying this method, the discrete logarithm $d = \log_g y$ would be equal to $p+q \pmod r$, as reported in Equation 7.12. As a note for the reader, r divides $\phi(N) = (p-1)(q-1)$, which is the order of $\mathbb{Z}_N^*$ for Euler's totient theorem, and $pq+1 = p+q \pmod{\phi(N)}$.

$$d = N+1 = pq+1 = p+q \pmod r \tag{7.12}$$

Afterwards, the value d can be the starting point to determine, leveraging standard classical procedure, the roots of the equation:

$$p^2 - dp + N = 0 \tag{7.13}$$

The period of the function $f(e_1, e_2) = g^{e_1} y^{-e_2}$, where e_1 has a bit length of $n + O(1)$ and e_2 has a bit length of $n/2 + O(1)$ is then provided by the quantum circuit. Finally, the entire

exponent exhibits an asymptotic length of $1.5 \cdot n$, which is better than the $2n$ of the standard circuit.

An essential final remark is that the estimation of this study does not rely on the heavy hexagon code from IBM, which would require, with a code distance $d = 27$, a number of qubits around 25.5 million. They adopted instead a custom QEC technique which lowered that number to 20 million.

7.4 Summary on resource estimation

The reader may appreciate a summary and comparison of the different optimisation analysed for Shor's algorithm in this brief section. These considerations might also clarify the advantages, peculiarities, and constraints of the underlying approaches.

Looking back at section 7.2, we estimated for the classical quantum circuit a number of qubits required of $4n+2$ and a depth of $12n^3 \cdot T + 32n^3$, where T represents the depth of a Toffoli gate if this is not present as a native gate.

Version	**Logical qubits**	**Depth**	**# of Toffoli gates**
Sequential QFT [105]	$2n+3$	$144n^3 \lg(n) + O(n^2 \lg(n))$	$576n^3 \lg^2(n) + O(n^3 \lg(n))$
In-place addition [109]	$2n+2$	$52n^3 + O(n^2)$	$64n^3 \lg(n) + O(n^3)$
Ekerå and Håstad [111]	$3n + 0.002n \lg(n)$	$500n^2 + n^2 \lg(n)$	$0.3n^3 + 0.0005n^3 \lg(n)$

Table 7.1 Comparison among the different optimisation techinque presented.

Table 7.1 shows a summary of the same metrics for all the quantum circuits designed in the various studies that are reported in this work. As a remark, all the metrics have been calculated on the specific architecture and with the specific QEC techniques adopted in those studies. In section 7.5, the reader can find a link to the implementation of a Qiskit version of a subset of them as well as practical tests on the depth of the resulting quantum circuits.

Coming back to Table 7.1, one may observe that applying the sequential QFT allows reducing significantly the number of qubits required, while the in-place addition bears a small advantage of saving a single qubit and the third study needs even more qubit than the sequential QFT. The real differences here come when one adopts a specific QEC technique, as already mentioned for Ekerå.

Concerning the depth of the circuit, one can notice that the in-place addition brings some advantages over the sequential QFT; nevertheless, the Ekerå's set of optimisations leads to a better result.

7.5 Implementation and experimental results

As for the implementations carried out during this work, one can find them at the linked GitHub repository[1]. In particular, the reader can access a library for Shor's algorithm based on the Qiskit toolkit containing both the basic Aqua version and a set of functions that implement the various enhanced version presented in section 7.3.

In the following part of this section, one can also find the results of several tests conducted on these available variants to assess and better understand their peculiar characteristics. The metrics adopted for testing and validation are *execution time*, *depth*, and *quality* of the solution.

Since we selected the IBM Q 32-qubit simulator, the reader might presume that all tests were conducted using this simulator unless otherwise specified. Clearly, this affects all metrics, but particularly the first two: the execution time is dependent on the specific tools used to transpile the circuit, the available native gates, and the maximum decomposition of the circuit that is achievable on the hardware.

As a remark on why we chose to employ the simulator rather than quantum hardware primarily, one can easily comprehend that existing processors (e.g., 65-qubit Hummingbird, 127-qubit Eagle) require QEC approaches to provide consistent results. Because of this, the available logical qubits after applying a suitable distance coding were insufficient for executing Shor. The reader may independently examine the effects of noise in an experiment using the simulator described in subsection 7.5.2.

For the sake of clarity, in the following sections, we provide, in turn: a comparison between the standard version of Shor's algorithm and its enhanced versions, results regarding the impact of noise in quantum circuit execution for a specific architecture, and a comparison of the same quantum circuit executed on different simulators.

[1] https://github.com/ignaziopedone/shor-analysis

7.5.1 Validation of the Qiskit-based Shor's improvements

This section illustrates the comparison between the different Shor's algorithm implementations presented in this work and summarised in Table 7.2.

Description of the implementation	**# of required qubits**
Qiskit Aqua base version	$4n+2$
Toffoli-based	$4n+1$
Qiskit Aqua base version + sequential QFT	$2n+3$
Toffoli-based + sequential QFT	$2n+2$

Table 7.2 Summary of the available Shor's algorithm implementations.

As a reminder for the reader, one has a probability of at least $\phi(r)/3r$ to obtain a good value from the execution of an instance of Shor's algorithm. In this work, we define this probability as the quality of a specific solution. For some of the experiments, in fact, we compared the lower bound above with the probability of success measured in our experiments.

As already mentioned, our experiments have been conducted using the 32-bit IBM Q simulator, so the reader can expect certain constraints in terms of the size of the problem that we are able to solve. In this specific case, we run experiments using different semiprimes (N values) from 15 to 143, depending on the limits of the specific variant. A critical remark here is that we were not able to hit the theoretical limit of the maximum size of N for the 32-bit processor since other factors needed to be considered: the growing depth of the circuit and the time needed to execute the simulation classically which is also related to the number of shots or iteration of the algorithm. For instance, in these experiments, we were able to solve the standard QFT up to 6-bit moduli in a reasonable time and the sequential QFT up to 8-bit moduli. These numbers are clearly lower than the theoretical maximum, i.e. $2n+2$ for the sequential QFT.

The first experiment, depicted in Figure 7.15, shows how the growth in terms of the number of bits of the modulus influences the execution time. These results confirm the better performance of the sequential QFT against the classical one. In addition, one can also observe that the Toffoli-based version of the circuit generally provides an advantage that is more significant for the variant without the sequential QFT. The reader may also notice some differences in the trend of the various implementations with respect to the data presented in the Table 7.1. To further clarify this aspect, the depth of a Toffoli gate for a

specific architecture should be considered in the equation - when this is not native - and further optimisation for the specific architecture in the transpiring process.

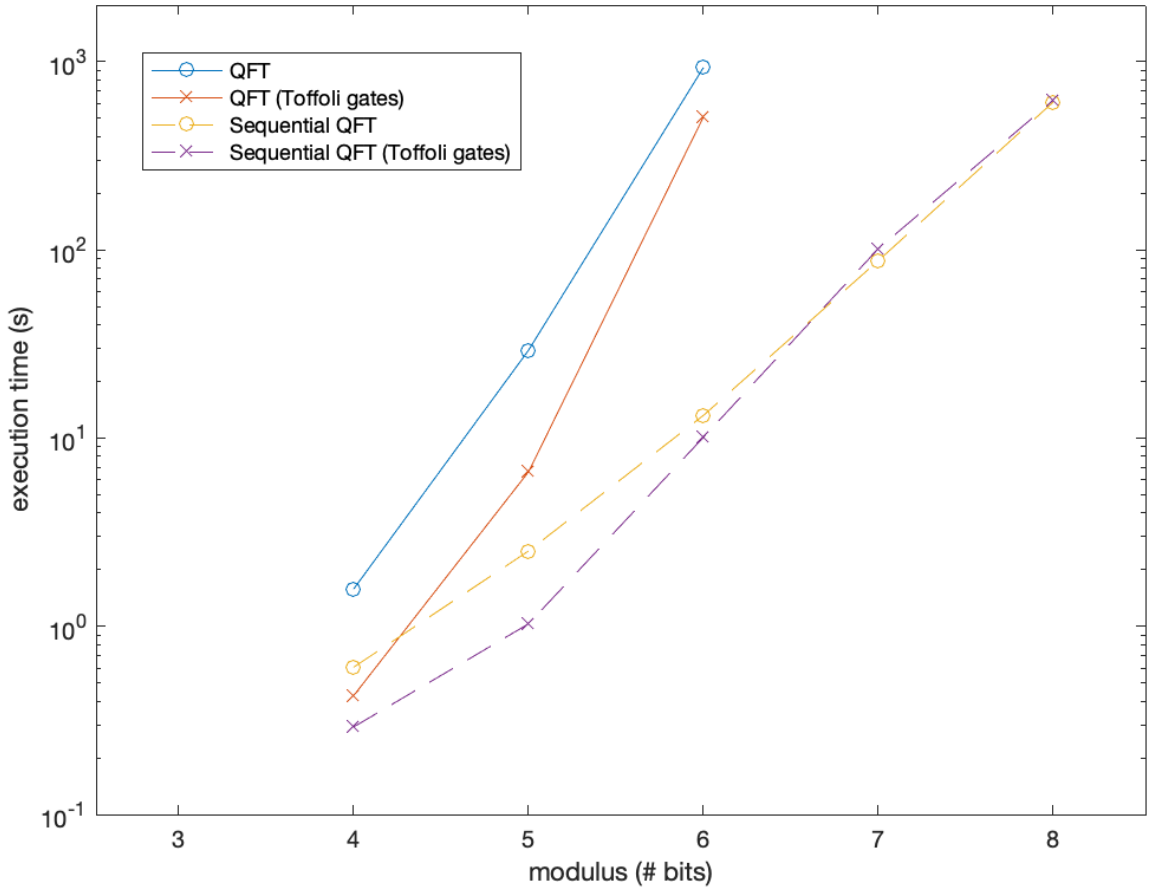

Fig. 7.15 Execution time for each variant using the 32-qubit IBM Q simulator and 100 shots.

Figure 7.16 shows another interesting metric during the very same experiment, the depth of the maximally decomposed circuit. This value corresponds to the depth of the quantum circuit calculated by decomposing it in native gates and taking into account all possible parallelisation. This calculus may be directly obtained by using the Qiskit `decompose` function.

According to the theoretical analysis, the depth of the sequential QFT grows faster than the one with the classical QFT due to possible parallelisations of quantum gates in this second case. The reader may also notice that the depth of the Toffoli-based versions is larger than the standard implementation. This once again depends on the Toffoli decomposition on the IBM architecture. Finally, one can observe that this gap from the Toffoli decomposition becomes narrower between the two sequential QFT variants: this happens because the

primary influence on the general depth depends on the lack of parallelisation rather than the decomposition of the Toffoli gates.

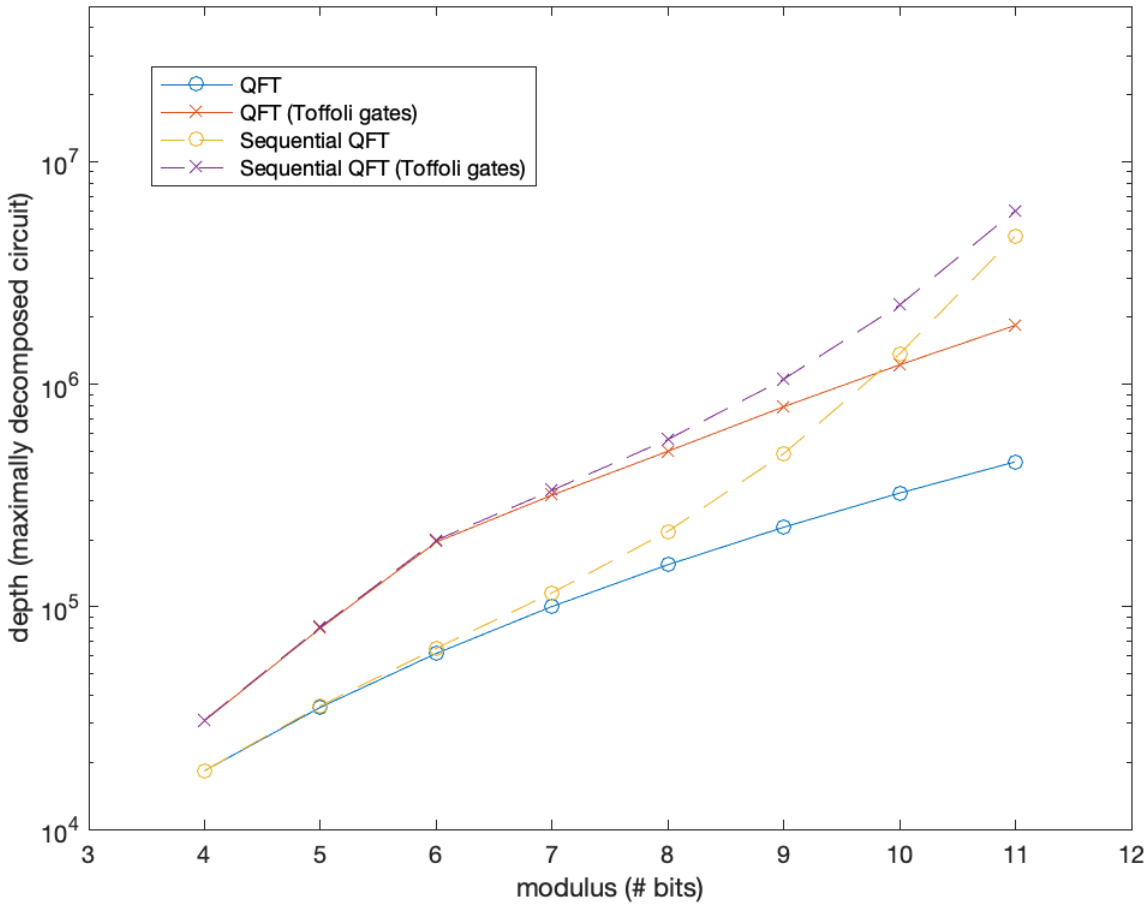

Fig. 7.16 Depth of the quantum circuits for each variant depending on the number of bits of the modulus and choosing $a = 2$.

In Figure 7.17, we can observe a neat peculiarity of the Toffoli-based versions. In particular, depending on the choice of a, the depth of the circuit may oscillate, a behaviour not arising in the standard version and caused by the carry gate.

Changing metric to the quality of the solution, one can observe in Figure 7.18 and Figure 7.19 the probability of success of a quantum circuit depending on the chosen value of a for all the analysed variants and $N = 15, 21, 33$.

For $N = 15$ and $N = 21$, the reader can observe that the two essential factors are the choice of a and the method adopted; in fact, these factors are the main reason for the variation of the success probability. Nevertheless, one can notice that all those values are greater than the expected value; thus, the experiment results are consistent for each case, and there are

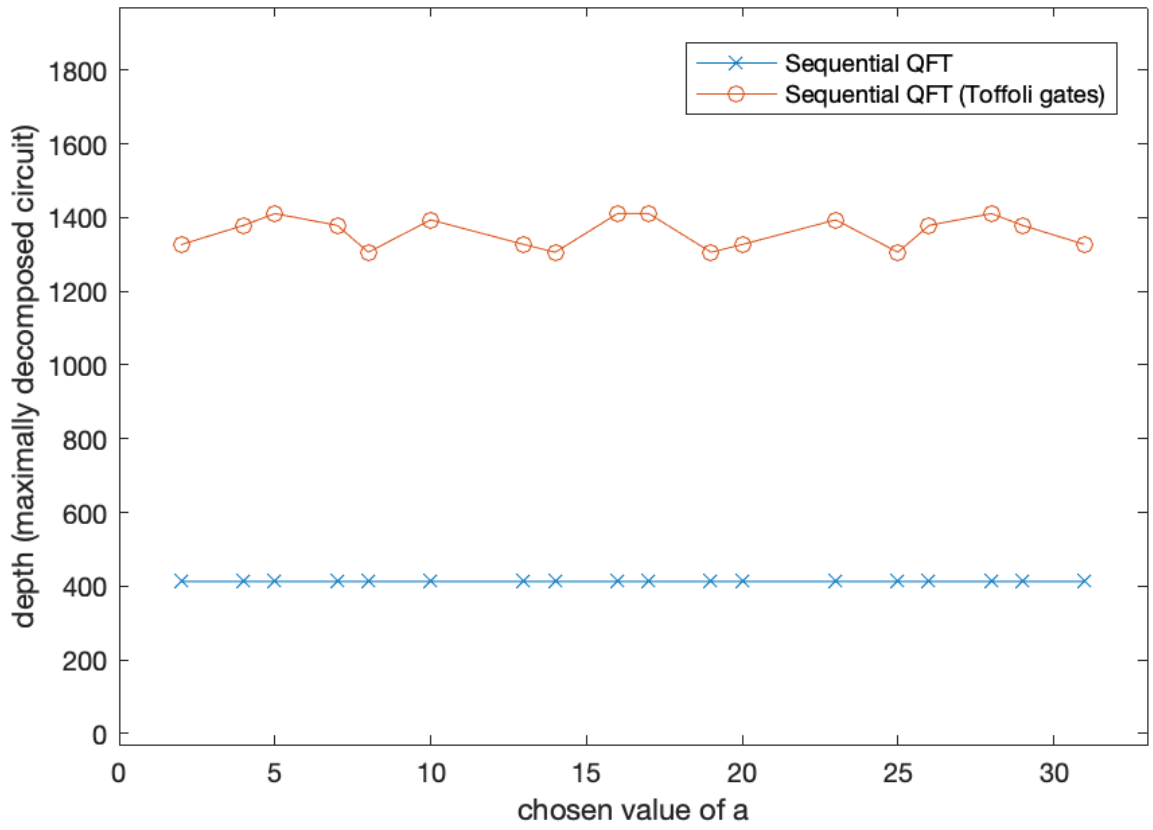

Fig. 7.17 Depth of the doubly controlled modular multiplication circuit with $N = 33$ and for different value of a.

no significant differences. As a remark for the reader, all values are subject to oscillations according to the number of shots set during the experiment: there is an explicit trade-off between precision and execution time.

As the modulus grows (e.g., N=33), the differences between the various methods arise. In particular, the sequential QFT version seems to be more stable than the classical one. In Figure 7.20, the reader can also notice that varying the modulus Toffoli-based method appears to react better to higher value with respect to the standard version.

7.5.2 Noise models

This section started with an important observation about the current limits in using real hardware to run Shor's algorithm for a sufficiently large modulus. Because of this, we adopted the IBM Q simulator. Nevertheless, understanding and evaluating the effect of noise

on the quantum circuit is essential to both support our choice and clarify some ideas to the reader.

The Qiskit toolkit allows applying different noise models to the quantum circuit executed through their simulator. We selected the Montreal noise model, related to the same processor, which offers 27 qubits and a QV of 128, and adopted it in two cases: a two-cyclic binary halving circuit (Figure 7.21) and the Toffoli-based version of Shor's algorithm implemented with the sequential QFT (Figure 7.22).

These tests show how the noise affects the final solution, even in a simple case like the one of the binary halving. In the specific case of Shor's algorithm, noise makes reading the results impractical and emphasises the need for QEC techniques supported by architecture with a more significant number of physical qubits.

7.5.3 Rigetti simulator

As a final test, the reader might be interested in whether these implementations are portable and the differences in the solution's quality. Because of this, we ported the code of the various Shor's algorithm implementations from Qiskit to *pyquil* in order to be executed on the Rigetti simulator [2]. A helpful tool to perform this porting is the QPS online converter[3]. An important remark is that the Rigetti simulator that we used is a local machine simulator (pure state QVM); thus, we had constraints in terms of execution time and performance with respect to the one from IBM.

The tests performed and depicted in Figure 7.23 involve the prime factoring of $N = 15$ using $a = 2$ with all the available implementation variants and $N = 21$ also using $a = 2$ but only with the sequential QFT implementation.

These results, rather than aiming at giving a solid comparison between the two architectures, want to demonstrate that it is indeed possible to migrate a large quantum circuit implementation relatively easily between two architectures preserving the consistent quality of the solutions. This sounds even more compelling, considering the great effort profused by different vendors in proposing new architectures with more qubits, larger values of QV, and more sophisticated QEC techniques.

[2] https://pyquil-docs.rigetti.com/en/v2.0.0/wavefunction_simulator.html
[3] https://quantum-circuit.com/qconvert

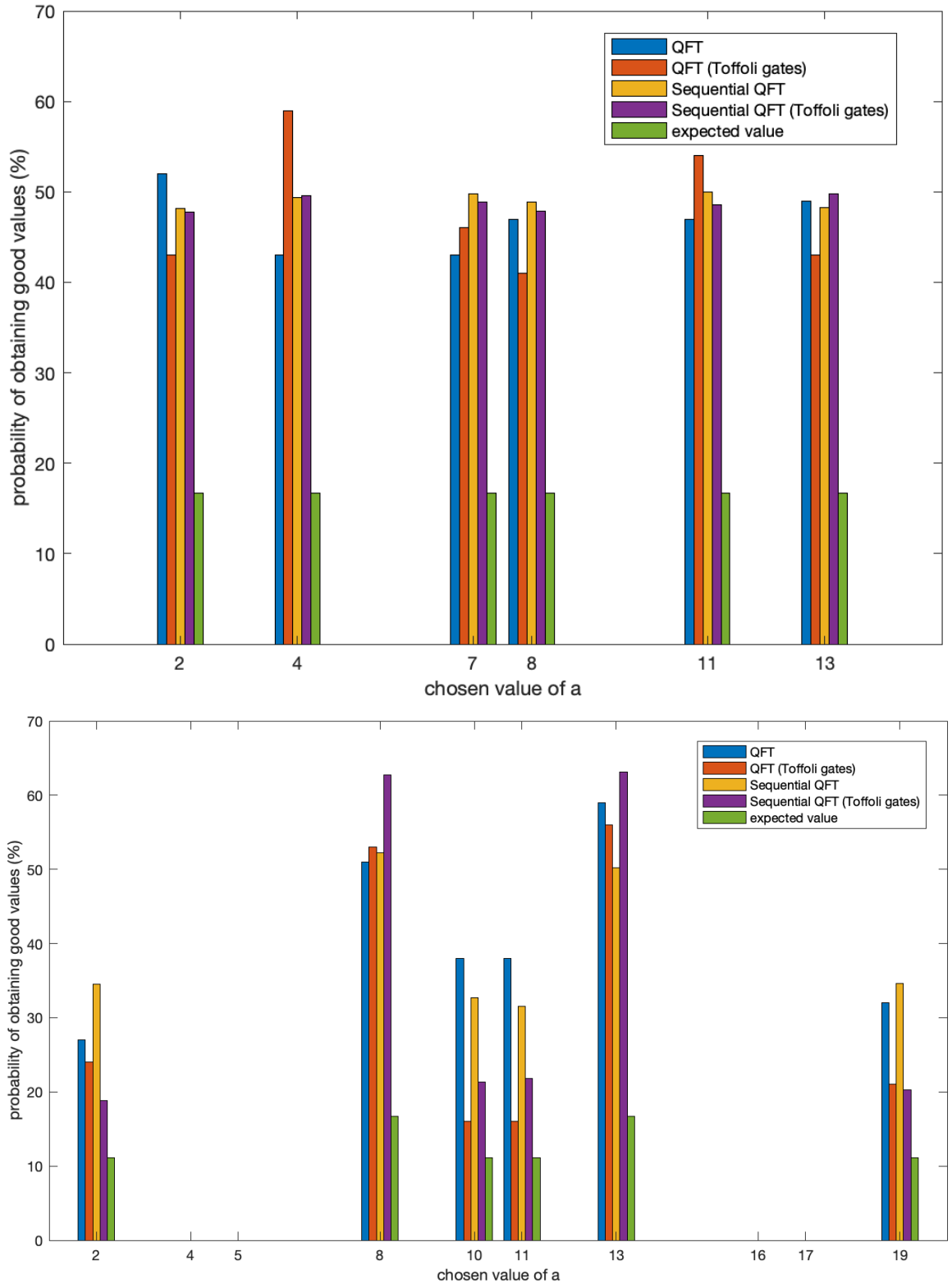

Fig. 7.18 Probability of success for the different variants against the minimum expected value varying the value of a. For the figure at the top N=15, while for the on at the bottom N=21.

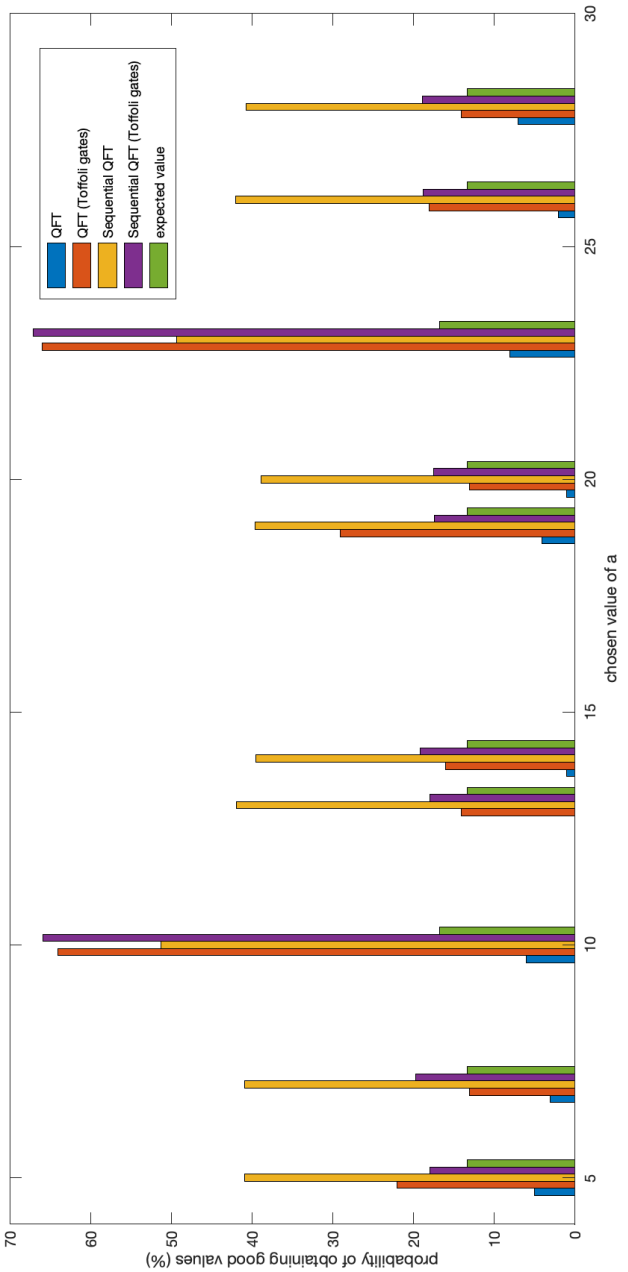

Fig. 7.19 Probability of success for the different variants against the minimum expected value varying the value of a and with N=33.

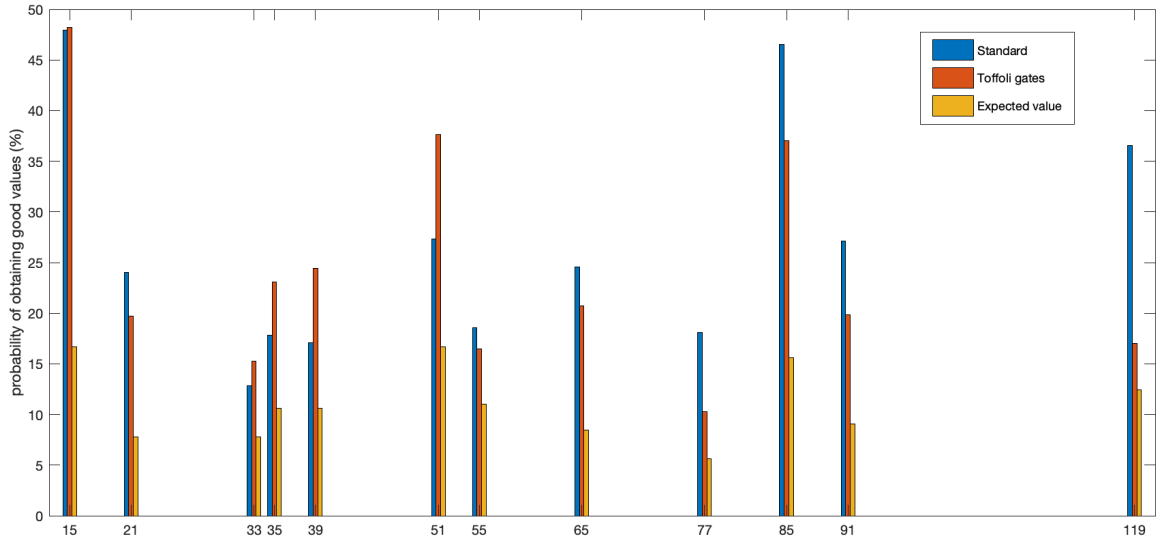

Fig. 7.20 Quality of the solution comparison between the standard and the Toffoli-based variants for different values of N.

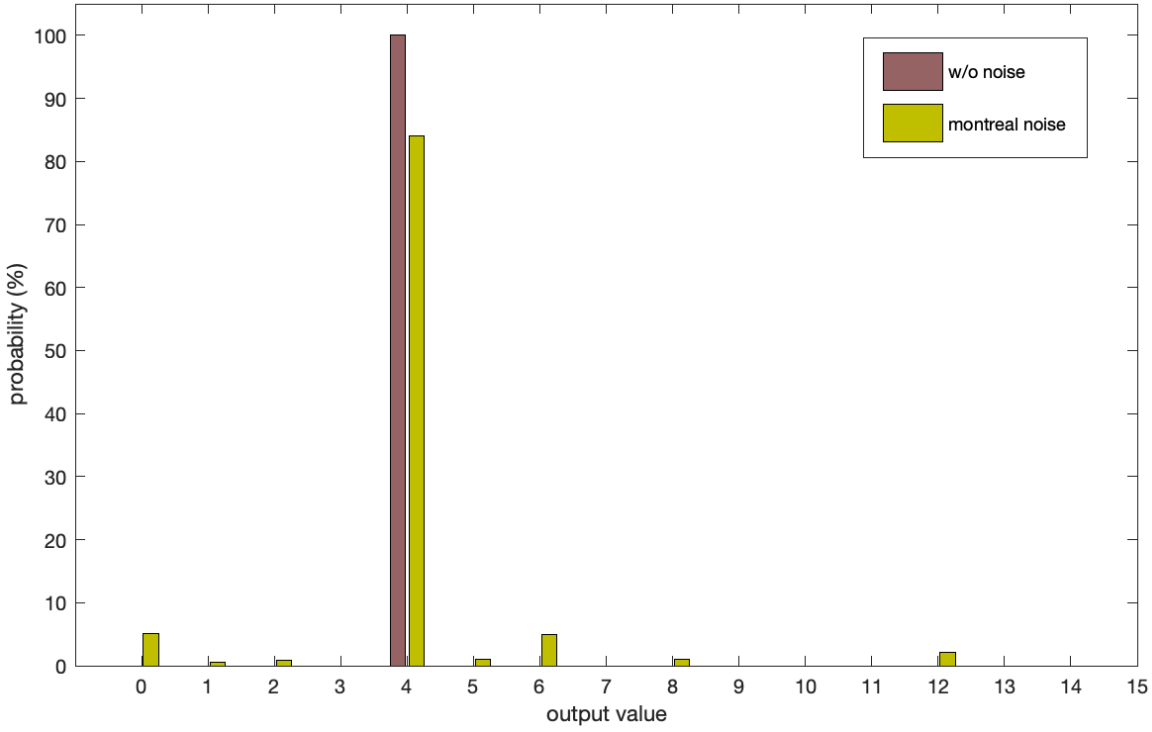

Fig. 7.21 probability of obtaining a certain value as the output of two binary halving gates with and without applying a montreal noise model.

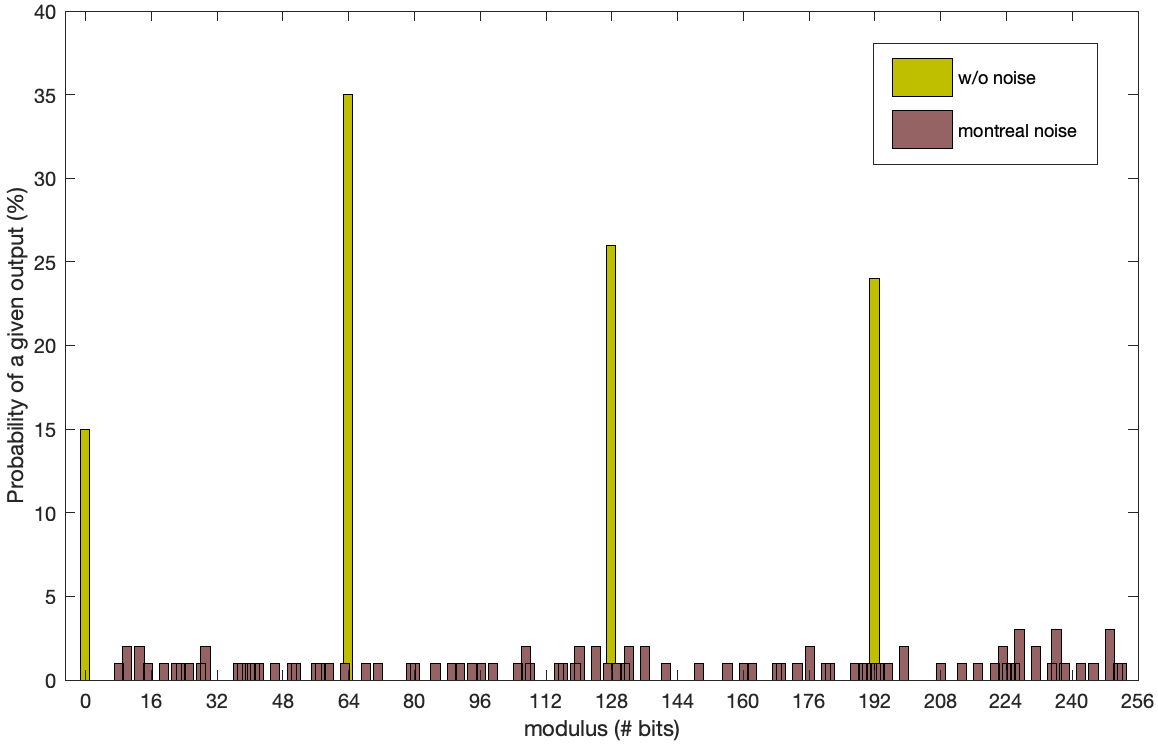

Fig. 7.22 probability of obtaining a certain value as the output of a whole Shor's algorithm quantum circuit with sequential QFT and the Toffoli-based method with and without applying a Montreal noise model.

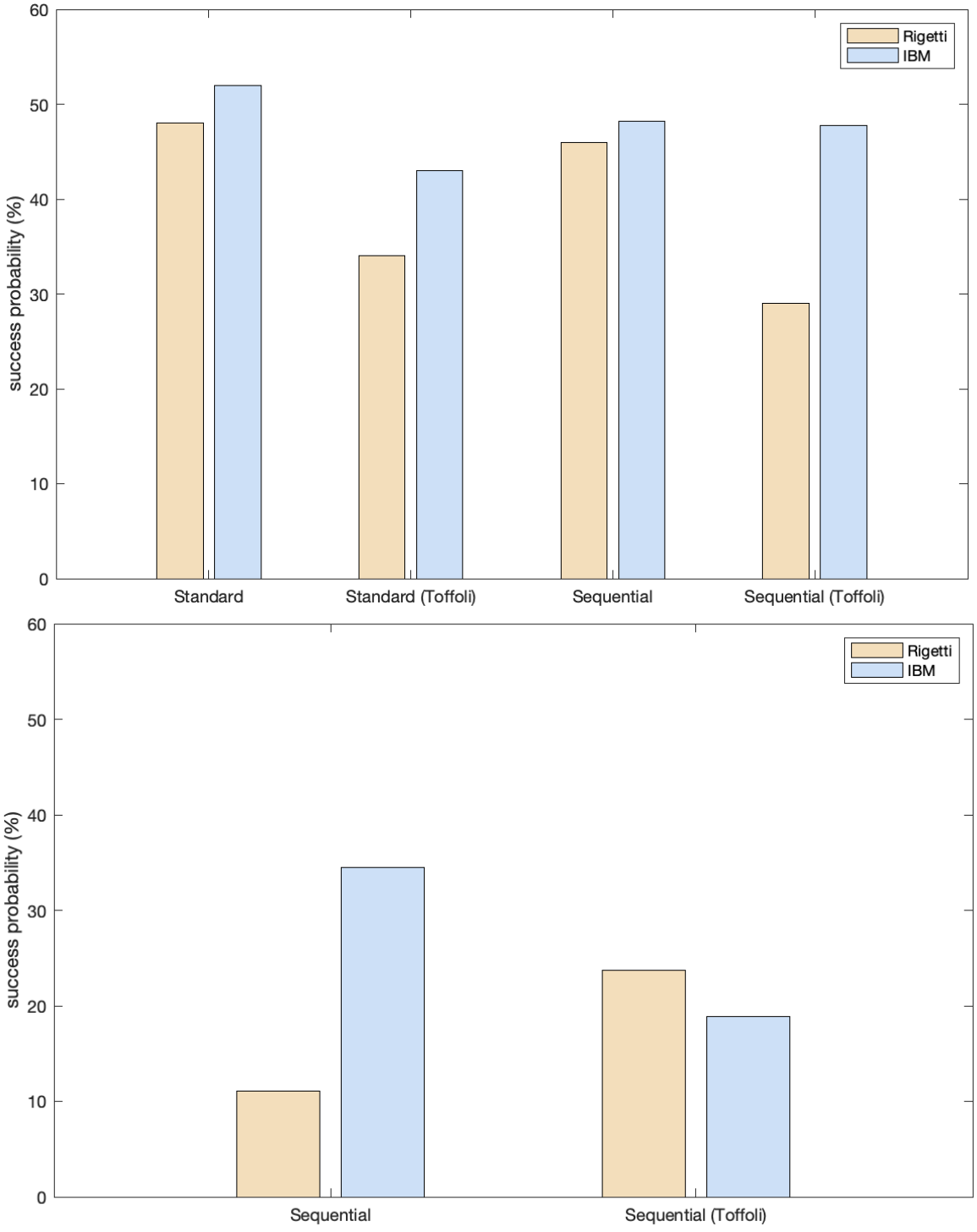

Fig. 7.23 Success probability comparison between the 32-qubit IBM Q simulator and the Rigetti QVM. At the top $N = 15$, while at the bottom $N = 20$.

Chapter 8

Shor's algorithm and Elliptic Curve Discrete Logarithm Problem

As for the prime factoring problem, the DLP is a building block of current public key algorithms and protocols such as Diffie-Hellman (DH) for key exchange. Shor's algorithm is able to break also the DLP. Even if current security protocols (e.g., TLS 1.3) are progressively abandoning DH in favour of Elliptic Curve Diffie-Hellman (ECDH), the latter can be broken in polynomial time by adapting Shor's algorithm for the DLP to the case of ECC.

In this chapter, an approach for designing a quantum algorithm to break ECC is presented as well as the description of all the steps needed from solving the DLP for multiplicative groups - as in the case of DH - to adapting it for the specific case of Elliptic curves.

An interesting outcome of this analysis is that in the case of ECC, we need fewer qubits and gates than in the prime factoring case to solve the underlying problem.

8.1 Quantum algorithm overview

The authors of [8] provided a remarkable overview of the generalisation of Shor's algorithms - both for factoring and DLP - and we adopted it for presenting the required circuit that must be adapted in the case of Elliptic Curves.

The final goal is to find the value d of $x \equiv g^d \pmod{p}$. While in the factoring algorithm, we work with subgroups of $\mathbb{Z}$ (group of integers), for the DLP we are interested in subgroups of $\mathbb{Z}^2$ that can be expressed as a linear combination of two vectors in $\mathbb{Z}^2$. This consideration leads the authors to view the DLP algorithm as a two-dimensional version of the factoring

algorithm. Considering the function:

$$f(a,b) = g^a x^b \pmod{p}$$

one can observe that it has two distinct periods in $\mathbb{Z}^2$, which are:

$$f(a+q,b) = f(a,b) \qquad \text{and} \qquad f(a+d,b-1) = f(a,b)$$

where q is the order of g. This reflection allows extending Shor's algorithm - described in section 7.1 - to the DLP case by adding a third register, applying Hadamard to the first two, and reformulating the oracle as follows:

$$\frac{1}{q}\sum_{a=0}^{q-1}\sum_{b=0}^{q-1}|a,b,1\rangle \Rightarrow \frac{1}{q}\sum_{a=0}^{q-1}\sum_{b=0}^{q-1}|a,b,g^a x^b\rangle$$

Assuming that $x \equiv g^d \pmod{p}$, one can leave the third register in the state $|g^{a+db}\rangle = |g^{a_0}\rangle$; thus, each b has only one solution and the two top registers transform as the following:

$$\frac{1}{\sqrt{q}}\sum_{b=0}^{q-1}|a_0 - db, b\rangle \tag{8.1}$$

Afterwards, the QFT applies to these two registers, leading state:

$$\frac{1}{q\sqrt{q}}\sum_{a',b'=0}^{q-1}\sum_{b=0}^{q-1} e^{\frac{2\pi i}{q}(a_0-db)a'} e^{\frac{2\pi i}{q}bb'}|a',b'\rangle$$

Considering that we look at conditions where $b' \equiv da' \pmod{q}$ and:

$$\sum_{b=0}^{q-1} e^{\frac{2\pi i}{q}(a_0-db)a'} e^{\frac{2\pi i}{q}bda'} = \sum_{b=0}^{q-1} e^{\frac{2\pi i}{q}a_0a'} = qe^{\frac{2\pi i}{q}a_0a'}$$

the resulting state becomes:

$$\frac{1}{\sqrt{q}}\sum_{a'=0}^{q-1} e^{\frac{2\pi i}{q}a_0a'}|a',b'\rangle \tag{8.2}$$

where $b' = da' \pmod{q}$. Now, if we measure the two top registers, we can retrieve d by performing the following:

$$d = b'(a')^{-1} \pmod{q} \tag{8.3}$$

According to [106], this circuit may extract an appropriate pair (a', b') with a probability $p/(240 \cdot q')$, where q' denotes the power of 2 such that $p < q' < 2p$. By evaluating this probability, we may determine that fewer circuit shots can be necessary if the modulus of the algorithm is near to q', the next power of 2. Nevertheless, since $p < 2q'$, we may lower-bound this probability, getting a minimal fixed probability of $1/480$.

In the instance of Elliptic Curves, modifying the oracle's technique enables this high-level circuit to solve the ECDLP. Notably, many elliptic curves now employed in cryptographic applications have moduli that are close to the next power of two: for example, *Curve25519* is defined over a prime field with modulus $2^{255} - 19$, therefore the chance of a quantum circuit producing a valid output is close to $1/240$.

Given that the literature about ECC is complete and extensive we assume that the reader is familiar with the concepts of elliptic curves over finite fields, generator, and operations over the curves, e.g., addition between two points. The authors of [8, 112] provide primers on ECC, useful to deeply understand the following concepts. Nevertheless, the reader can find all the mathematical elements in this chapter to understand the core of the work which lies within the analysis and the enhancement of the quantum algorithm. Suppose one has an elliptic curve following the equation:

$$y^2 = x^3 + ax + b \tag{8.4}$$

where a and b are constants such that $4a^3 + 27b^2 \neq 0$ and the addition operation between two points on the curve $(p_1 = (x_1, y_2), p_2 = (x_2, y_2))$ is defined as:

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3) = p_3 \tag{8.5}$$

where $x_3 = \lambda^2 - (x_1 + x_2), y_3 = \lambda(x_1 - x_3) - y_1$, and:

$$\lambda = \begin{cases} (y_2 - y_1)/(x_2 - x_1) & if\ p_1 \neq p_2 \\ (3{x_1}^2 + a)/(2y_1) & if\ p_1 = p_2 \end{cases} \tag{8.6}$$

To crack ECC (e.g., ECDH, ECDSA), determining the secret key d by knowing the curve parameters, generator G, and point $B = d \cdot G$, we must perform the following oracle transformation:

$$\frac{1}{2^n} \sum_{a=0}^{2^n-1} \sum_{b=0}^{2^n-1} |a, b, 1\rangle \Rightarrow \frac{1}{2^n} \sum_{a=0}^{2^n-1} \sum_{b=0}^{2^n-1} |a, b, aG + bB\rangle$$

Given that the required operation is a multiplication, we may use the same double-and-add approach as for the factoring circuit. However, because the addition is based on elliptic curve arithmetic, we must discover a reversible way to do it. We may apply the technique provided in [113] and reported in Algorithm 1 to calculate the new coordinates $(x3, y3)$ of Equation 8.7. Our main objective is to implement all the constituent gates and circuits so that one day the whole quantum circuit representing the algorithm can be tested on real hardware. Considering all the limitations regarding the current simulators and quantum hardware, we analysed and tested the building components. available in section 8.2.

Algorithm 1 reversible point addition procedure between a point stored in two quantum registers and a point considered to be a precomputed classical constant. The result of $p_1 + p_2$ is put in the register containing p_1 when $ctrl = 1$. All operations are conducted using modular arithmetic.

$x_1 \leftarrow x_1 - x_2$
if $ctrl = 1$ **then**
    $y_1 \leftarrow y_1 - y_2$
end if
$t_0 \leftarrow x_1^{-1}$
$\lambda \leftarrow y_1 \cdot t_0$
$y_1 \leftarrow y_1 \oplus \lambda \cdot x_1 \quad (=0)$
$t_0 \leftarrow t_0 \oplus x_1^{-1} \quad (=0)$
$t_0 \leftarrow \lambda^2$
if $ctrl = 1$ **then**
    $x_1 \leftarrow x_1 - t_0$
    $x_1 \leftarrow x_1 + 3x_2 \quad (= x_2 - x_3)$
end if
$t_0 \leftarrow t_0 \oplus \lambda^2 \quad (=0)$
$y_1 \leftarrow \lambda \cdot x_1$
$t_0 \leftarrow x_1^{-1}$
$\lambda \leftarrow \lambda \oplus t_0 \cdot y_1 \quad (=0)$
$t_0 \leftarrow t_0 \oplus x_1^{-1} \quad (=0)$
if $ctrl = 1$ **then**
    $x_1 \leftarrow$ modular negation of x_1
    $y_1 \leftarrow y_1 - y_2 \quad (= y_3)$
end if
$x_1 \leftarrow x_1 + x_2 \quad (= x_3)$

8.2 Underlying building blocks

Given our primary objective of implementing the whole quantum circuit to compute the DLP in the subgroup of an Elliptic Curve, we started from the work proposed by the authors of [113]. The latter works introduced the design of the circuit and its subcomponents. In this section, the reader can find an analysis of each one of the building blocks for that circuit to better understand the practical implementation referenced in subsection 8.4.1 and several improvements suggested in section 8.4.

8.2.1 Modular addition and doubling

Figure 8.1 shows the circuit to compute the addition modulo p. This circuit can be derived from the one described in subsection 7.3.2 by adding a gate for the comparison between quantum registers and using only one clear ancilla qubit and a dirty one. The comparison of the register $|x\rangle$ and $|y\rangle$ is straightforward since it is only made of subtraction and addition without the carry. For the first subtraction, the qubit that stores the result is toggled if $y < x$, then the addition without the carry restores the value $|y\rangle$ in the second register.

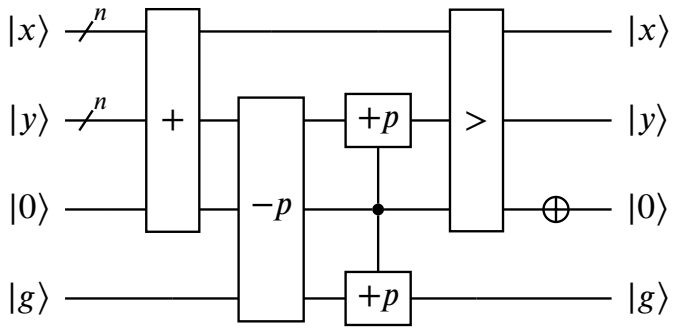

Fig. 8.1 Circuit for the modular addition between two numbers encoded in quantum registers. Source: [113]

An important feature of this circuit regards its controlled version, which is used in Algorithm 1: as done for previous circuits, we do not need to control every gate, but just the first (the adder) and the comparator. The circuit for the modular doubling, which is helpful for modular multiplication, follows the same principles but requires one less register and a simple gate that performs the bit shift of a quantum register, built using only qubit swaps.

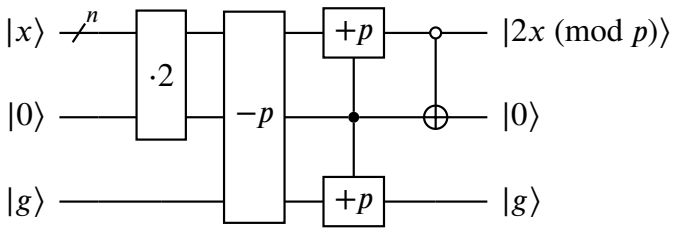

Fig. 8.2 Circuit for the modular doubling of a number encoded in a quantum register. Source: [113].

One can make the same assumption of the addition for the doubling in the case of the controlled version. Moreover, the latter can be obtained by managing the control of the only binary shift and the last CNOT. In addition, a CNOT gate must be added to restore the clear ancilla.

8.2.2 Modular multiplication

The modular multiplication circuit required for point addition stores the outcome of the operation of two integers encoded in two quantum registers in a third quantum register, necessitating at least $3n$ qubits to operate and allowing for two implementation options. The first follows the equation Figure 7.2, while the second uses Montgomery arithmetic [114] since numbers in Montgomery representation will also be employed in the modular inversion to produce the most efficient approach.

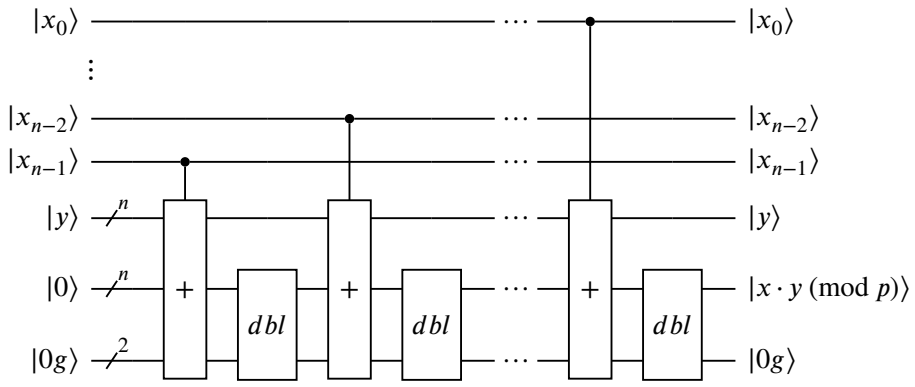

Fig. 8.3 Circuit for the modular multiplication between two numbers encoded in quantum registers. Source: [113]

Figure 8.3 shows the implementation of the first option. Moreover, it repeats the two gates seen in subsection 8.2.1. Figure 8.4 depicts the circuit for modular squaring of a number encoded in a quantum register derived from the previous one by controlling the additions and adding a clear ancilla.

As for the resource estimation, the first circuit requires $3n+2$ qubits and the Toffoli gate count is $32n^2\log_2(n) - 59.4n^2$; the second circuit $2n+3$ qubits and the same amount of Toffoli gates.

The second option, namely the Montgomery multiplication strategy, allows instead obtaining a depth-width trade-off. As a reminder for the reader, a number in Montgomery form is expressed as $aR \pmod p$ with R coprime and greater to p. Moreover, the Montgomery reduction algorithm computes $cR^{-1} \pmod p$ for an integer c given as input. Performing the multiplication of two integers in Montgomery form $aR \pmod p$ and $bR \pmod p$ and applying the reduction algorithm, one obtains $abR \pmod p$, namely the Montgomery form of the product between a and b. One can further simplify it by selecting $R = 2^n$ and obtaining the reduction with a series of binary shifts.

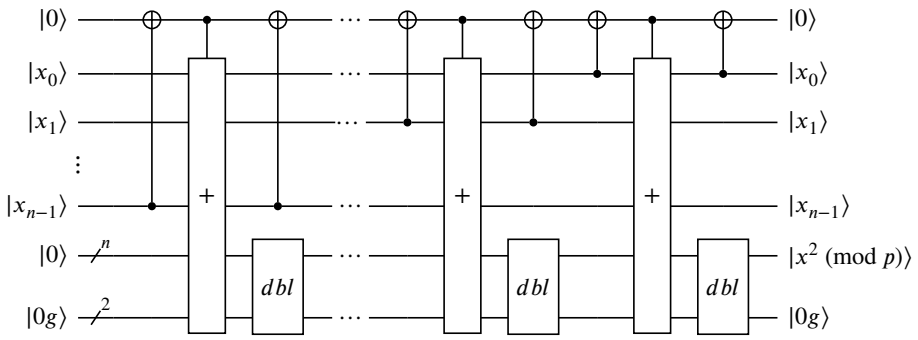

Fig. 8.4 Circuit for the modular squaring of a number encoded in quantum register. Source: [113]

Comparing these two approaches, the Mongomery strategy requires only one modular operation per round, while the other method two. This achievement is possible due to the absence of the modular doubling gate at the cost of a more significant number of required qubits to store intermediate results.

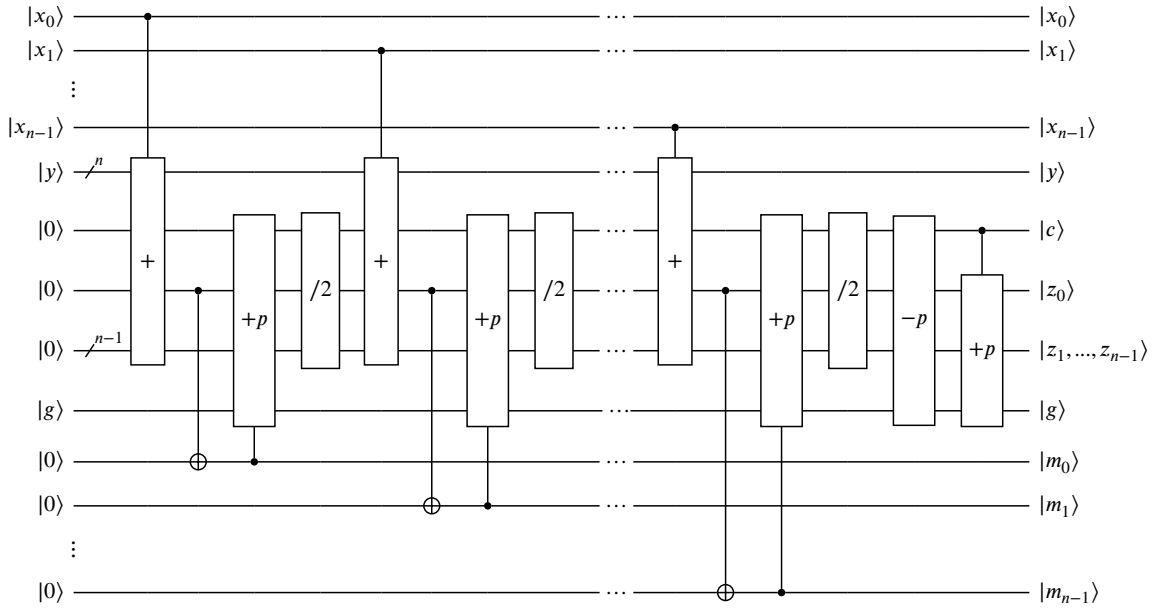

Fig. 8.5 Circuit for the Montgomery modular multiplication between two numbers in Montgomery representation encoded in quantum registers. Source: [113]

To better clarify the following depth analysis, the reader must know that once applied the circuit in Figure 8.5, the result is copied to another register allowing the calculation of its inverse and the reset operation of the ancilla. These operations can be executed in parallel.

Finally, Montgomery modular multiplication requires $5n+4$ qubits and its Toffoli count is $16n^2\log_2(n)-26.3n^2$. Giving the same consideration to the first approach, it requires $4n+5$ qubits.

8.2.3 Modular inversion

Assuming that modular inversion is the most resource-consuming operation, the choice of an optimised algorithm is essential. We adopted Kaliski's algorithm [115], which provides a procedure to invert a number in Montgomery representation with $R = 2^n$. As an example for the reader, the input $x2^n \pmod p$ produces $x^{-1}2^n \pmod p$ as output. As the first data for resource estimation, this algorithm requires five input registers to calculate the GCD of two numbers (x, p). Four registers are required to store intermediate results, while the last one is used as a counter.

Algorithm 2 shows how Kaliski's procedure is divided into two phases. During the first phase, one gets the value $x^{-1}2^k$ and needs to save the value of k for the second phase. Phase 1 has a while loop whose block is the quantum circuit represented in Figure 8.6. The whole high-level operations of Phase 1 for Kaliski's algorithm is instead depicted in Figure 8.7.

Algorithm 2 Kaliski's algorithm to calculate the pseudo inverse of x modulo p.

Phase 1
$u \leftarrow p$, $v \leftarrow x$, $r \leftarrow 0$, $s \leftarrow 1$, $k \leftarrow 0$
while $v > 0$ **do**
  if u is even **then**
    $u \leftarrow \frac{u}{2}$, $s \leftarrow 2s$
  else if v is even **then**
    $v \leftarrow \frac{v}{2}$, $r \leftarrow 2r$
  else if $u > v$ **then**
    $u \leftarrow \frac{u-v}{2}$, $r \leftarrow r+s$, $s = 2s$
  else
    $v \leftarrow \frac{v-u}{2}$, $s \leftarrow r+s$, $r = 2r$
  end if
  $k = k+1$
end while
if $r \geq p$ **then**
  $r \leftarrow r-p$
end if
return $r \leftarrow p-r$, k
Phase 2
for i=1 to k-n **do**
  if r is even **then**
    $r \leftarrow \frac{r}{2}$
  else
    $r \leftarrow \frac{r+p}{2}$
  end if
end for
return $x \leftarrow r$

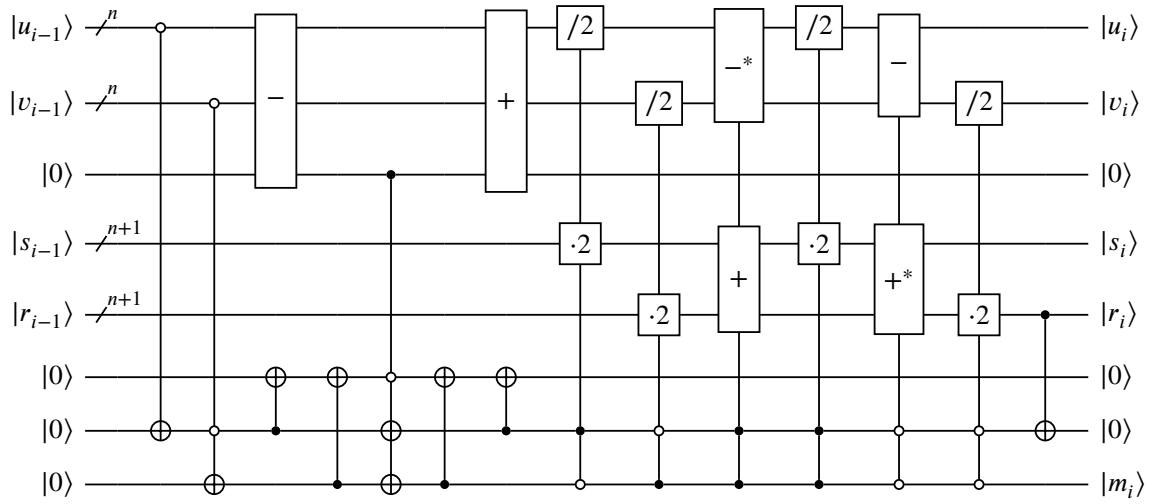

Fig. 8.6 Block for the implementation of the modular inverse circuit. The multi-wire gates store the output the register at the bottom, except for the cases with "*". Source: [113]

After $2n$ rounds, r stores the result and one can copy it to another register executing the inverse of the circuit and restoring the original values of the registers. A critical remark is that one needs to store also the value of $|k\rangle$ because its value is crucial to understand how many times to double the result. The total number of qubits needed for phase 1 becomes $7n+2l+7$, while the number of Toffoli gates is $32n^2 \log_2 n$.

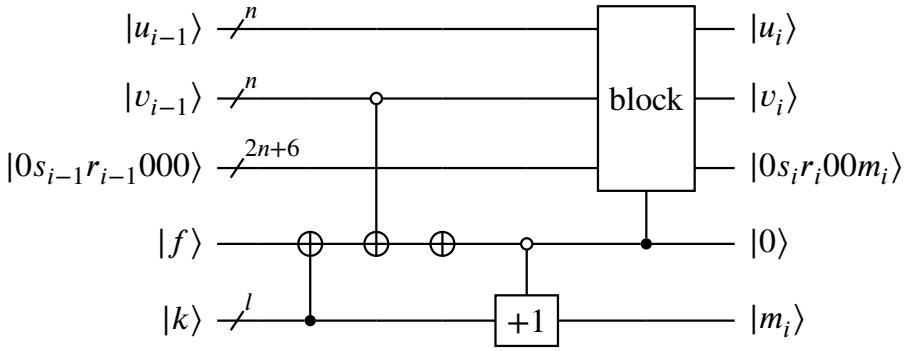

Fig. 8.7 Round of the modular inversion circuit implementing Kaliski's algorithm. In each of the $2n$ rounds, a different m_i qubit is used. The block gate is executed until v reaches 0, then k starts to be incremented. Source: [113]

ECDLP for curves modulo p			RSA with modulus N		
$\lceil \log_2 p \rceil$	#qubits	#Toffoli gates	$\lceil \log_2 N \rceil$	#qubits	#Toffoli gates
110	1014	$9,44 \cdot 10^9$	512	1026	$6,41 \cdot 10^{10}$
160	1466	$2,97 \cdot 10^{10}$	1024	2050	$5,81 \cdot 10^{11}$
224	2042	$8,43 \cdot 10^{10}$	2048	4098	$5,20 \cdot 10^{12}$
256	2330	$1,26 \cdot 10^{11}$	3072	6146	$1,86 \cdot 10^{13}$
384	3484	$4,52 \cdot 10^{11}$	7680	15362	$3,30 \cdot 10^{14}$

Table 8.1 Comparison between qubits needed to break ECDLP and RSA at similar security levels. The values for the RSA columns are derived by the version using the in-place adder. Source: [113]

8.3 Analysis of the required resources

An estimation of the number of qubits required by the Algorithm 1 needs to consider modular inversion, additional qubits for sequential QFT, and the register for storing the intermediate results. As for the depth, [113] provides an estimation of the Toffoli count for each point addition ($224n^2 \log_2 n + 2045n^2$) and for the whole circuit ($(448 \log_2 n + 4090)n^3$). The Mongomery variant can be adopted due to the available auxiliary qubits for modular multiplications.

An essential remark for the reader is that the presented gates work for all point additions except for adding a point to itself, to its negative, and to the point at infinity. The authors of [8] estimate that this occurs $4n/p$ times; therefore, this fidelity loss must be added to the probability of finding a suitable pair, which becomes $1/480$. Finally, the oracle's inputs need to be adapted in the presence of an indefinite point, a problem which is trivial since we may start anywhere on the curve without affecting the end phase.

A really neat summary about a final resource estimation is the one presented in Table 8.1, where the resources required to compute the ECDLP problem are compared with the ones to solve the factoring problem, i.e., the RSA case. This remarkable analysis shows how counterintuitively, breaking ECC is even easier than cracking RSA for a similar level of security.

8.4 Quantum algorithm improvements

In the literature, some studies allow computing ECDLP even using fewer resources than the ones presented in the previous section. The authors of [116] propose enhancements using pebbling, windowed arithmetic, and other peculiar optimisations. In addition, they provide

Q#-based implementations as utilities. The final estimation for ECDLP on a 256-bit curve - after the optimisations - reports a decrease in the number of qubits from 2338 to 2124; the Toffoli count decreases by a factor of 119; the depth by a factor of 54. Further optimisations are presented in the same study to lower depth at the cost of increasing the number of qubits, but they are out of the scope of this study.

Pebbling techniques reduce depth by feeding auxiliary qubits of one operation as input for the next. Our implementation referenced in the coming section uses it for the sixth step of Algorithm 1. In particular, the modular square of λ is stored in t_0, then used to compute $x_1 - t_0$ and restored to $|0\rangle$.

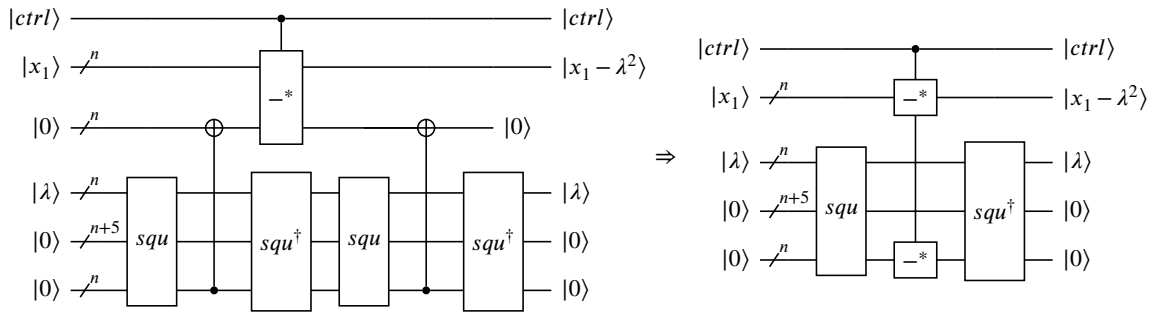

Fig. 8.8 Changes to the circuit for squaring-then-add using pebbling. The multi-wire gates store the output the register at the bottom, except for the cases with "*". The register with $n+5$ qubits collects all the other needed ancillae used. Source: [116]

The reader may notice that using Montgomery squaring as a black box, the quantum circuit in section 8.2 adds a non-required modular squaring, which may be excluded to proceed towards the more efficient version presented in Figure 8.8. Steps 5, 6, and 8 of Algorithm 2

Algorithm 3 Changes provided in [116] to the operations inside the while loop of Kaliski's algorithm 2.

$b_{swap} \leftarrow$ false
if u is even and v odd, or u and v odd and $u > v$ **then**
  swap u and v
  swap r and s
  $b_{swap} \leftarrow$true
end if
if u and v odd **then**
  $v \leftarrow v - u$
  $s \leftarrow r + s$
end if
$v \leftarrow \frac{v}{2}$
$r \leftarrow 2r$
if b_{swap} is true **then**
  swap u and v
  swap r and s
end if

may benefit from the same reasoning: pebbling allows avoiding the execution of a complete inversion circuit. Another improvement for the implementation in Figure 8.6 is provided in Algorithm 3. The latter uses swaps instead of doublings and halvings; since their cost is similar to the one of a binary shift, one fewer modular addition is required without adding any qubits.

The final estimation predicts an asymptotic number of $8n + 10.2\lfloor \log_2 n \rfloor - 1$ logical qubits for the low-width variant and a depth of $506n^3 - 1.84 \cdot 2^2 7$ using all the discussed enhancements.

8.4.1 Qiskit circuit implementation and contributions

The idea behind this work - as discussed at the beginning of the chapter - was to provide an in-depth analysis of the state of the art regarding the application of Shor to break ECC algorithms. This was relevant to perform an estimation on the required resources and a comparison with the effort required to break RSA and the factoring problem. Beyond the mere analysis, we also provided an implementation in Qiskit of all the elementary gates and circuits necessary to implement the whole Quantum Algorithm to break ECDLP up to Kaliski's algorithm. Unfortunately, we were not able to test the entire algorithm due to constraints in terms of resources (e.g., the number of qubits required).

Because of this, we implemented and tested the various components aggregating them in a library to be used and enhanced so that once a powerful enough simulator or quantum hardware is available, one can try the overall solution. In the meanwhile, the various components can be both improved and tested. The entire library source code is available on GitHub[1].

8.5 Tests on ECDLP quantum circuits

In this section, the reader can find the results of the tests performed on the Qiskit implementation of the ECDLP variant of Shor's algorithm. As mentioned, these tests have been performed on quantum subcircuits since executing them for the whole circuit is unfeasible, neither using simulation platforms nor available quantum hardware without efficient QEC techniques.

The first kind of tests conducted were functional tests, in which we validated the accuracy of our implementation by evaluating all the implemented circuits. Notably, a clever technique

[1] https://github.com/ignaziopedone/shor-analysis

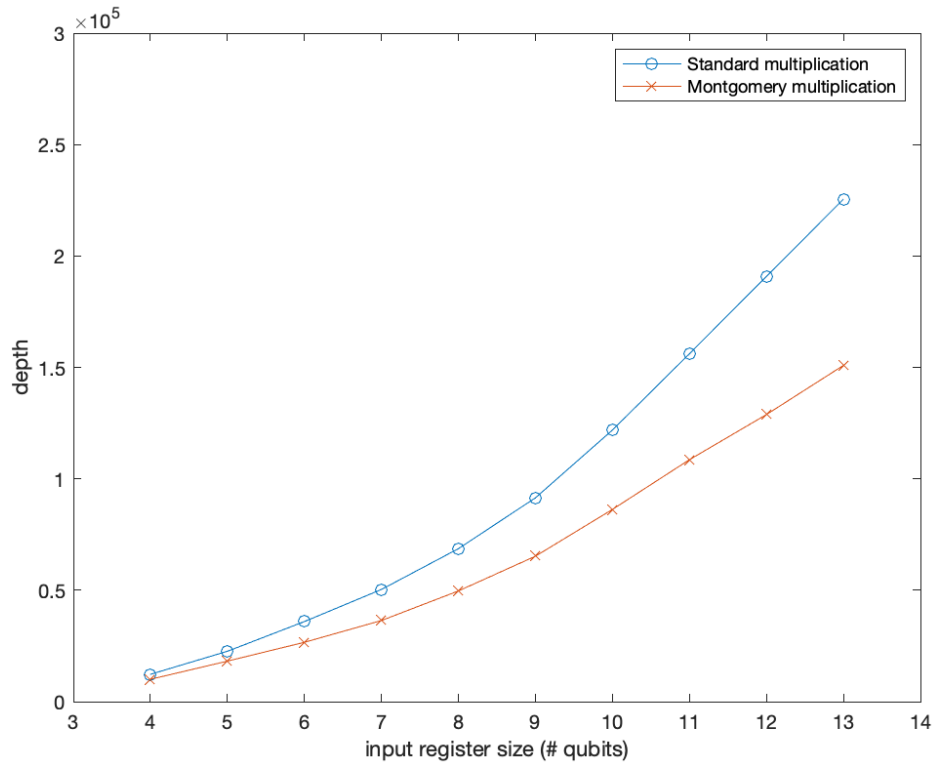

Fig. 8.9 Depth of the normal multiplication and of the Montgomery multiplication circuits (without the copy of the result and the decomputation of the qubits) for input registers with a qubit-size from 4 to 13.

to confirm the quality of the most delicate circuit, the modular inversion, was to perform the same experiment as in [113] (Fig. 8) and verify that the results for the identical inputs and conditions were the same using our improved version.

The second class of tests analyzed the depth of specific components, such as the quantum circuit for multiplication and the quantum circuit as a whole. Specifically, Figure 8.9 demonstrates how the Montgomery multiplication block is superior to the ordinary one. Instead, Figure 8.10 illustrates the benefit of employing the optimized version of the Ua gate as opposed to the standard version. Finally, in Figure 8.11, we examine the total benefit in terms of the depth of the whole circuit, and the depth gain is substantial.

For thoroughness, we must disclose that these tests were conducted using a heuristic estimate by summing the depth of the various circuit components. Clearly, on actual hardware, numerous other factors must be considered: the maximum potential circuit decomposition, the native quantum gates accessible, and further optimization.

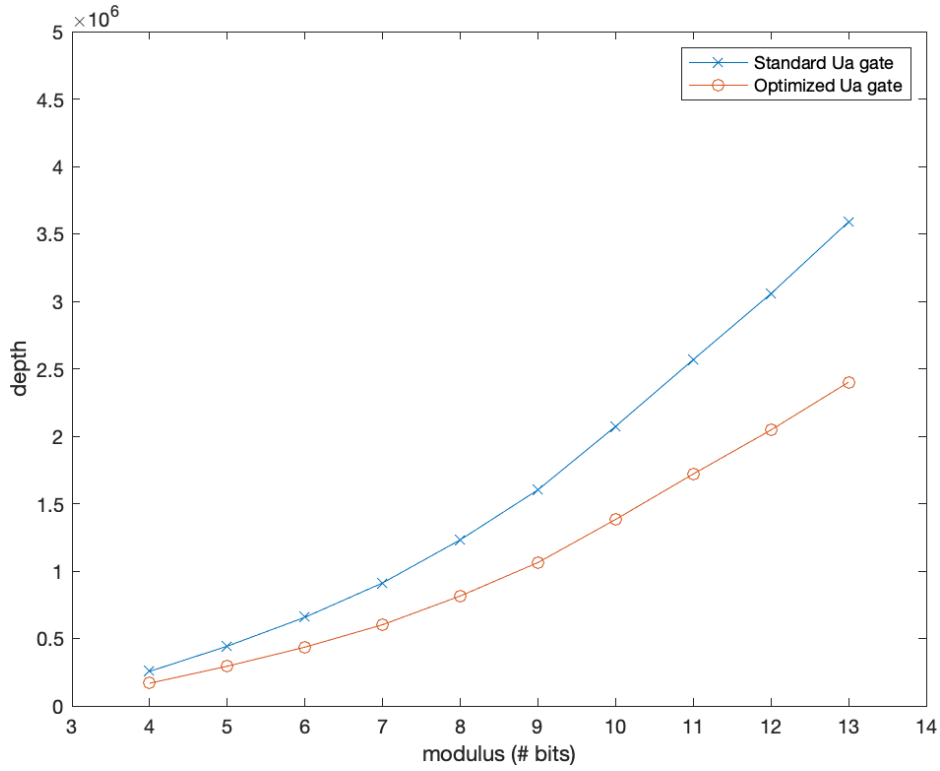

Fig. 8.10 Estimation of the depth for the single U_a gate using the basic and the optimized circuits for moduli that have bit-length from 4 to 13.

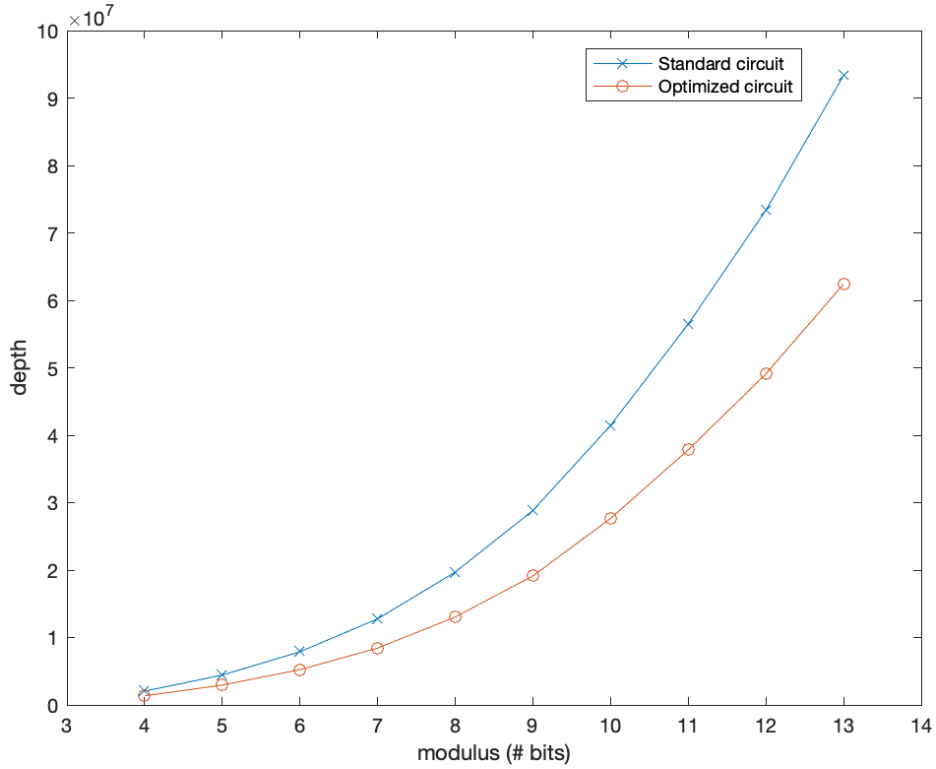

Fig. 8.11 Estimation of the depth for the whole circuit using the basic and the optimized circuits for moduli that have bit-length from 4 to 13.

Chapter 9

Conclusion

The objective of this thesis work was to present three main activities carried out during my PhD. As a summary for the reader, these activities are:

- design and implementation of the QKD integration in SDI. In particular, the presentation of the QSS and the QKD simulator with the respective improvements.
- Proposing a generic QUBO formulation for VNFEPs, which is particularly interesting for both SDI and cybersecurity. Tests have also been conducted using different solvers: D-Wave quantum annealer, Tabu Search, and Simulated Annealing.
- Analysing Shor's algorithm, its impact on classical cryptography, and possible optimisations. In addition, providing various implementations based on Qiskit, considering noise for some tests and different architectures such as the one from Rigetti. The analysis and the implementations have been conducted for both classic DLP and ECDLP.

A core result of this thesis is undoubtedly the development of the QSS and QKD simulators. These tools have been conceived as flexible software components that can be easily extended and adapted to any modern infrastructure. Scalability is a crucial point. All the presented software can easily scale horizontally on SDI to manage more extensive networks and strict requirements. The simulation part can be extended. For instance, new backends can be adopted as a toolkit, and even the logic behind the simulation can be changed, preserving the same infrastructural model. Future steps can involve looking beyond QKD and considering the simulation of generic Quantum Networks leveraging the same platform.

For the activity related to Quantum Annealing, the results achieved are quite interesting. They show how it is feasible using Quantum Annealing to solve VNFEPs, the advantage in

certain conditions over classical solvers, and the potential speed-up that they can have over large networks. Clearly, the immaturity of the current QPUs also poses limitations. Indeed, there is a limit to the current size of the networks that can be managed and the fragility of the current physical qubit implementations. Future work may study more extensive networks and how to optimise the formulation to fit the available QPUs. In particular, defining constraints for a QUBO formulation is a delicate task.

The last activity regarding Shor's algorithm is still a work in progress, with no publication submitted by the author. It is interesting, though, because it tries to capture a fundamental question: how far are we from the Q-Day? This is clearly from a high-level point of view and without considering all the possible alternatives in terms of hardware. Nevertheless, understanding the complexity of the task may help classical security experts to roughly estimate its advent. Future work may focus more on QEC in terms of more recent approaches and architectures based on other qubit technologies.

Finally, this work does not claim to be an exhaustive dissertation on all possible connections between QTs and cybersecurity, but it explores three aspects that are currently relevant and on top of which one can build several more studies and practical applications. As a final, personal comment for the reader, I must renew my profound belief that in the years to come QTs beyond QKD will become more and more central in cybersecurity.

References

[1] J. D. Whitfield, J. Yan, W. Wang, J. T. Heath, and B. Harrison. Quantum Computing 2022. *arXiv preprint arXiv:2201.09877*, 2022.

[2] A. Abbas et al. Learn Quantum Computation Using Qiskit, 2020. Available online: https://qiskit.org/textbook/.

[3] M. A. Nielsen and I. L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, USA, 10th edition, 2011. ISBN 1107002176.

[4] K. Azuma, S. E. Economou, D. Elkouss, P. Hilaire, L. Jiang, H. Lo, and I. Tzitrin. Quantum repeaters: From quantum networks to the quantum internet. *arXiv preprint arXiv:2212.10820*, 2022.

[5] P. W. Shor. Algorithms for quantum computation: discrete logarithms and factoring. In *Proceedings 35th annual symposium on foundations of computer science*, pages 124–134. IEEE, 1994. doi: 10.1109/SFCS.1994.365700.

[6] The IBM Quantum project. Available online: https://quantum-computing.ibm.com/.

[7] V. Mavroeidis, K. Vishi, M. D. Zych, and A. Jøsang. The impact of quantum computing on present cryptography. *arXiv preprint arXiv:1804.00200*, 2018.

[8] J. Proos and C. Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves. *arXiv preprint quant-ph/0301141*, 2003.

[9] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani, J. Pereira, M. Razavi, J. S. Shaari, M. Tomamichel, V. C. Usenko, G. Vallone, P. Villoresi, and P. Wallden. Advances in Quantum Cryptography. *Advances in Optics and Photonics*, 12(4):1012–1236, 2020. doi: 10.1364/AOP.361502.

[10] W. Castryck and T. Decru. An efficient key recovery attack on SIDH (preliminary version). *Cryptology ePrint Archive*, 2022.

[11] M. Mina and E. Simion. Information security in the quantum era. threats to modern cryptography: Grover's algorithm. Cryptology ePrint Archive, Paper 2021/1662, 2021. URL https://eprint.iacr.org/2021/1662. https://eprint.iacr.org/2021/1662.

[12] D. J. Bernstein. Grover vs. mceliece. In *Post-Quantum Cryptography: Third International Workshop, PQCrypto 2010, Darmstadt, Germany, May 25-28, 2010. Proceedings 3*, pages 73–80. Springer, 2010. doi: 10.1007/978-3-642-12929-2_6.

[13] M. Grassl, B. Langenberg, M. Roetteler, and R. Steinwandt. Applying Grover's algorithm to AES: quantum resource estimates. In *Post-Quantum Cryptography: 7th International Workshop, PQCrypto 2016, Fukuoka, Japan, February 24-26, 2016, Proceedings 7*, pages 29–43. Springer, 2016. doi: 10.1007/978-3-319-29360-8_3.

[14] S. Jaques, M. Naehrig, M. Roetteler, and F. Virdia. Implementing Grover oracles for quantum key search on AES and LowMC. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part II 30*, pages 280–310. Springer, 2020. doi: 10.1007/978-3-030-45724-2_10.

[15] M. Amy, O. Di Matteo, V. Gheorghiu, M. Mosca, A. Parent, and J. Schanck. Estimating the cost of generic quantum pre-image attacks on SHA-2 and SHA-3. In *Selected Areas in Cryptography–SAC 2016: 23rd International Conference, St. John's, NL, Canada, August 10-12, 2016, Revised Selected Papers*, pages 317–337. Springer, 2017. doi: 10.1007/978-3-319-69453-5_18.

[16] G. Brassard, P. Høyer, and A. Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN'98: Theoretical Informatics: Third Latin American Symposium Campinas, Brazil, April 20–24, 1998 Proceedings 3*, pages 163–169. Springer, 1998. doi: 10.1145/261342.261346.

[17] M. Mosca, J. M. V. Basso, and S. R. Verschoor. On speeding up factoring with quantum SAT solvers. *Scientific Reports*, 10(1):1–8, 2020. doi: 10.1038/s41598-020-71654-y.

[18] S. Geravand and M. Ahmadi. Bloom filter applications in network security: A state-of-the-art survey. *Computer Networks*, 57(18):4047–4064, 2013. ISSN 1389-1286. doi: 10.1016/j.comnet.2013.09.003.

[19] R. Shi. Quantum Bloom Filter and Its Applications. *IEEE Transactions on Quantum Engineering*, 2:1–11, 2021. doi: 10.1109/TQE.2021.3054623.

[20] R. Quintero, D. Bernal, T. Terlaky, and L. F. Zuluaga. Characterization of QUBO reformulations for the maximum k-colorable subgraph problem. *Quantum Information Processing*, 21(3):89, 2022. doi: 10.1007/s11128-022-03421-z.

[21] P. Berman and A. Pelc. Distributed probabilistic fault diagnosis for multiprocessor systems. In *[1990] Digest of Papers. Fault-Tolerant Computing: 20th International Symposium*, pages 340–346, 1990. doi: 10.1109/FTCS.1990.89383.

[22] E. D. Payares and J. C. Martinez-Santos. Quantum machine learning for intrusion detection of distributed denial of service attacks: a comparative overview. In *Quantum Computing, Communication, and Simulation*, volume 11699, page 116990B. International Society for Optics and Photonics, SPIE, 2021. doi: 10.1117/12.2593297.

[23] D. Said. Quantum Computing and Machine Learning for Cybersecurity: Distributed Denial of Service (DDoS) Attack Detection on Smart Micro-Grid. *Energies*, 16(8): 3572, 2023. doi: 10.3390/en16083572.

[24] T. A. Brun. Quantum Error Correction. *arXiv preprint arXiv:1910.03672*, 2019.

[25] C. Chamberland, G. Zhu, T. J. Yoder, J. B. Hertzberg, and A. W. Cross. Topological and subsystem codes on low-degree graphs with flag qubits. *Physical Review X*, 10(1): 011022, 2020. doi: 10.1103/PhysRevX.10.011022.

[26] D. Bacon and A. Casaccino. Quantum error correcting subsystem codes from two classical linear codes. *arXiv preprint quant-ph/0610088*, 2006.

[27] S. Sheldon, E. Magesan, J. M. Chow, and J. M. Gambetta. Procedure for systematically tuning up cross-talk in the cross-resonance gate. *Physical Review A*, 93(6):060302, 2016. doi: 10.1103/PhysRevA.93.060302.

[28] S. Zhang, Y. Chang, L. Yan, Z. Sheng, F. Yang, G. Han, Y. Huang, and J. Xia. Quantum communication networks and trust management: a survey. *Comput. Mater. Continua*, 61(3):1145–1174, 2019.

[29] S. Wehner, D. Elkouss, and R. Hanson. Quantum internet: a vision for the road ahead. *Science*, 362:1–9, October 2018. doi: 10.1126/science.aam9288.

[30] A. Dahlberg and S. Wehner. SimulaQron - A simulator for developing Quantum Internet software. *Quantum Science and Technology*, 4:1–15, September 2018. doi: 10.1088/2058-9565/aad56e.

[31] T. Coopmans, R. Knegjens, A. Dahlberg, D. Maier, L. Nijsten, J. de Oliveira Filho, M. Papendrecht, J. Rabbie, F. Rozpędek, M. Skrzypczyk, L. Wubben, W. de Jong, D. Podareanu, A. Torres-Knoop, D. Elkouss, and S. Wehner. Netsquid, a network simulator for quantum information using discrete events. *Communications Physics*, 4 (1):164, 2021. doi: 10.1038/s42005-021-00647-8.

[32] R. Satoh, M. Hajdušek, N. Benchasattabuse, S. Nagayama, K. Teramoto, T. Matsuo, S. A. Metwalli, T. Satoh, S. Suzuki, and R. Van Meter. Quisp: a Quantum Internet simulation package. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 353–364. IEEE, 2022. doi: 10.1109/QCE53715.2022.00056.

[33] S. Diadamo, J. Nötzel, B. Zanger, and M. M. Bese. Qunetsim: A software framework for quantum networks. *IEEE Transactions on Quantum Engineering*, 2:1–12, 2021. doi: 10.1109/TQE.2021.3092395.

[34] X. Wu, A. Kolar, J. Chung, D. Jin, T. Zhong, R. Kettimuthu, and M. Suchara. SeQUeNCe: a customizable discrete-event simulator of quantum networks. *Quantum Science and Technology*, 6(4):045027, 2021. doi: 10.1088/2058-9565/ac22f6.

[35] F. Xu, X. Ma, Q. Zhang, H. Lo, and J. Pan. Secure quantum key distribution with realistic devices. *Reviews of Modern Physics*, 92:025002–1–025002–60, May 2020. doi: 10.1103/RevModPhys.92.025002.

[36] C. Weedbrook, S. Pirandola, R. Garcia-Patron, N. J. Cerf, T. C. Ralph, J. H. Shapiro, and S. Lloyd. Gaussian quantum information. *Reviews of Modern Physics*, 84(2):621, 2012. doi: 10.1103/RevModPhys.84.621.

[37] M. Lucamarini, K. A. Patel, J. F. Dynes, B. Fröhlich, A. W. Sharpe, A. R. Dixon, Z. L. Yuan, R. V. Penty, and A. J. Shields. Efficient decoy-state quantum key distribution with quantified security. *Optics express*, 21(21):24550–24565, 2013. doi: 10.1364/OE.21.024550.

[38] V. Scarani, H. Bechmann-Pasquinucci, N. J. Cerf amd M. Dusek, N. Lutkenhaus, and M. Peev. The security of practical Quantum Key Distribution. *Reviews of modern physics*, 81(3):1301, 2009. doi: 10.1103/RevModPhys.81.1301.

[39] R. Renner and R. König. Universally composable privacy amplification against quantum adversaries. In *Theory of Cryptography Conference*, pages 407–425. Springer, 2005. doi: 10.1007/978-3-540-30576-7_22.

[40] R. Renner. *Security of Quantum Key Distribution*. PhD thesis, Swiss Federal Inst. Technol. Zürich, 2005.

[41] 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*, 97(5):052327, 2018. doi: 10.1103/PhysRevA.97.052327.

[42] V. Scarani and R. Renner. Quantum cryptography with finite resources: Unconditional security bound for discrete-variable protocols with one-way postprocessing. *Physical review letters*, 100(20):200501, 2008. doi: 10.1103/PhysRevLett.100.200501.

[43] Quantum Key Distribution (QKD); Security Proofs. Technical report, European Telecommunications Standards Institute (ETSI), December 2010.

[44] 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*, pages 282–285. Springer, 2015. doi: 10.1007/978-3-319-24315-3_29.

[45] P. Kampanakis, P. Panburana, E. Daw, and D. Van Geest. The Viability of Post-Quantum X. 509 Certificates. *IACR Cryptol. ePrint Arch.*, 2018:63, 2018.

[46] M. Mehic, M. Niemiec, S. Rass, J. Ma, M. Peev, A. Aguado, V. Martin, S. Schauer, A. Poppe, C. Pacher, and M. Voznak. Quantum key distribution: a networking perspective. *ACM Computing Surveys*, 53:1–41, September 2020. doi: 10.1145/3402192.

[47] M. Lucamarini, Z. L. Yuan, J. F. Dynes, and A. J. Shields. Overcoming the rate-distance limit of quantum key distribution without quantum repeaters. *Nature*, 557 (7705):400–403, 2018. doi: 10.1038/s41586-018-0066-6.

[48] Y. Liu, T. Chen, L. Wang, H. Liang, G. Shentu, J. Wang, K. Cui, H. Yin, N. Liu, L. Li, X. Ma, J. S. Pelc, M. M. Fejer, Q. Zhang, and J. Pan. Experimental measurement-device-independent quantum key distribution. *Physical review letters*, 111(13):130502, 2013. doi: 10.1103/PhysRevLett.111.130502.

[49] C. H. Bennett and G. Brassard. Quantum cryptography: public key distribution and coin tossing. *Theoretical Computer Science*, 560:7–11, December 2014. doi: 10.1016/j.tcs.2014.05.025.

[50] A. K. Ekert. Quantum cryptography based on Bell's theorem. *Physical Review Letters*, 67:661–663, August 1991. doi: 10.1103/PhysRevLett.67.661.

[51] M. Loeffler, C. Goroncy, T. Länger, A. Poppe, A. Neumann, M. Legré, I. Khan, C. Chunnilall, D. López, M. Lucamarini, A. Shields, E. Spigone, M. Ward, and V. Martin. Current Standardisation Landscape and existing Gaps in the Area of Quantum Key Distribution. *OPENQKD report*, 2021.

[52] Quantum Key Distribution (QKD); Application Interface. Technical report, European Telecommunications Standards Institute (ETSI), August 2020.

[53] Quantum Key Distribution (QKD); Protocol and data format of REST-based key delivery API. Technical report, European Telecommunications Standards Institute (ETSI), February 2019.

[54] Quantum Key Distribution (QKD); Control Interface for Software Defined Networks. Technical report, European Telecommunications Standards Institute (ETSI), March 2021.

[55] ETSI GS NFV specifications. Available online: https://www.etsi.org/committee/nfv.

[56] S. Yarkoni, E. Raponi, T. Bäck, and S. Schmitt. Quantum annealing for industry applications: Introduction and review. *Reports on Progress in Physics*, 85(10):104001, 2022. doi: 10.1088/1361-6633/ac8c54.

[57] I. Pedone, A. Atzeni, D. Canavese, and A. Lioy. Toward a Complete Software Stack to Integrate Quantum Key Distribution in a Cloud Environment. *IEEE Access*, 9: 115270–115291, 2021.

[58] Quantum Key Distribution (QKD); Use Cases. Technical report, European Telecommunications Standards Institute (ETSI), June 2010.

[59] P. W. Shor and J. Preskill. Simple proof of security of the BB84 quantum key distribution protocol. *Physical Review Letters*, 85:441–444, July 2000. doi: 10.1103/PhysRevLett.85.441.

[60] G. Murta, F. Rozpędek, J. Ribeiro, D. Elkouss, and S. Wehner. Key rates for quantum key distribution protocols with asymmetric noise. *Physical Review Letters*, 101: 062321–1–062321–10, June 2020. doi: 10.1103/PhysRevA.101.062321.

[61] M. Toyran, M. Toyran, and S. Öztürk. Optimized cascade protocol for efficient information reconciliation in Quantum Key Distribution systems. *Quantum Information & Computation*, 18(7-8):553–578, 2018.

[62] B. Tang, B. Liu, Y. Zhai, C. Wu, and W. Yu. High-speed and large-scale privacy amplification scheme for quantum key distribution. *Scientific reports*, 9(1):1–8, 2019. doi: 10.1038/s41598-019-50290-1.

[63] V. Lopez, A. Pastor, D. Lopez, A. Aguado, and V. Martin. Applying QKD to improve next-generation network infrastructures. In *European Conference on Networks and Communications*, pages 283–288, Valencia (Spain), June 18-21 2019. doi: 10.1109/EuCNC.2019.8802060.

[64] A. Aguado, E. Hugues-Salas, P. A. Haigh, J. Marhuenda, A. B. Price, P. Sibson, J. Kennard, C. Erven, J. G. Rarity, M. G. Thompson, A. Lord, R. Nejabati, and D. Simeonidou. 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*, pages 1–3, Dusseldorf (Germany), September 18-22 2016. doi: 10.1109/JLT.2016.2646921.

[65] D. R. Lopez, V. Martin, V. Lopez, F. de la Iglesia, A. Pastor, H. Brunner, A. Aguado, S. Bettelli, F. Fung, D. Hillerkuss, L. Comandar, D. Wang, A. Poppe, J. P. Brito, P. J. Salas, and M. Peev. Demonstration of Software Defined Network Services Utilizing Quantum Key Distribution Fully Integrated with Standard Telecommunication Network. *Quantum Reports*, 2(3):453–458, 2020. doi: 10.3390/quantum2030032.

[66] 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*, 27(5):6892–6909, 2019. doi: 10.1364/OE.27.006892.

[67] R. Chatterjee, K. Joarder, S. Chatterjee, B. C. Sanders, and U. Sinha. qkdSim, a Simulation Toolkit for Quantum Key Distribution Including Imperfections: Performance Analysis and Demonstration of the B92 Protocol Using Heralded Photons. *Physical Review Applied*, 14:024036, August 2020. doi: 10.1103/physrevapplied.14.024036.

[68] L. O. Mailloux, J. D. Morris, M. R. Grimaila, D. D. Hodson, D. R. Jacques, J. M. Colombi, C. V. Mclaughlin, and Jennifer A. Holes. A Modeling Framework for Studying Quantum Key Distribution System Implementation Nonidealities. *IEEE Access*, 3:110–130, February 2015. doi: 10.1109/access.2015.2399101.

[69] R. Van Meter. *Quantum networking*. 2014. ISBN 978-1848215375.

[70] I. Pedone and A. Lioy. Quantum Key Distribution in Kubernetes Clusters. *Future Internet*, 14(6):160, 2022. doi: 10.3390/fi14060160.

[71] Global Interpreter Lock. Available online: https://wiki.python.org/moin/GlobalInterpreterLock.

[72] Quart project. Available online: http://pgjones.gitlab.io/quart.

[73] Hypercorn project. Available online: https://pgjones.gitlab.io/hypercorn, .

[74] M. Peev, C. Pacher, R. Alléaume, C. Barreiro, J. Bouda, W. Boxleitner, T. Debuisschert, E. Diamanti, M. Dianati, J. F. Dynes, S. Fase, S. Fossier, M. Fürst, J-D Gautier, O. Gay, N. Gisin, P. Grangier, A. Happe, Y. Hasani, M. Hentschel, H. Hübel, G. Humer, T. Länger, M. Legré, R. Lieger, J. Lodewyck, T. Lorünser, N. Lütkenhaus, A. Marhold, T. Matyus, O. Maurhart, L. Monat, S. Nauerth, J-B Page, A. Poppe, E. Querasser, G. Ribordy, S. Robyr, L. Salvail, A. W. Sharpe, A. J. Shields, D. Stuck, M. Suda, C. Tamas, T. Themel, R. T. Thew, Y. Thoma, A. Treiber, P. Trinkler, R. Tualle-Brouri, F. Vannel, N. Walenta, H. Weier, H. Weinfurter, I. Wimberger, Z. L. Yuan, H. Zbinden, and A. Zeilinger. The SECOQC quantum key distribution network in Vienna. *New Journal of Physics*, 11(7):075001, 2009. doi: 10.1088/1367-2630/11/7/075001.

[75] C. Elliott, A. Colvin, D. Pearson, O. Pikalo, J. Schlafer, and H. Yeh. Current status of the DARPA quantum network. In *Quantum Information and computation III*, volume 5815, pages 138–149. International Society for Optics and Photonics, 2005.

[76] O. Maurhart. QKD networks based on Q3P. In *Applied Quantum Cryptography*, pages 151–171. Springer, 2010.

[77] J. Moy. OSPF Version 2. RFC 2328, April 1998. Available online: http://www.rfc-editor.org/rfc/rfc2328.txt.

[78] M. Sasaki. Quantum Key Distribution and Its Applications. *IEEE Security & Privacy*, 16(5):42–48, 2018. doi: 10.1109/MSP.2018.3761713.

[79] A. Conrad, S. Isaac, R. Cochran, D. Sanchez-Rosales, B. Wilens, A. Gutha, T. Rezaei, D. J. Gauthier, and P. Kwiat. Drone-based quantum key distribution: QKD. In *Free-Space Laser Communications XXXIII*, volume 11678, page 116780X. International Society for Optics and Photonics, 2021. doi: 10.1117/12.2582376.

[80] Istio project. Available online: https://istio.io.

[81] P. Zhang, L. Wang, W. Wang, K. Fu, and J. Wang. A blockchain system based on quantum-resistant digital signature. *Security and Communication Networks*, 2021, 2021. doi: 10.1155/2021/6671648.

[82] K. Ikeda. qBitcoin: a peer-to-peer quantum cash system. In *Science and Information Conference*, pages 763–771. Springer, 2018. doi: 10.1007/978-3-030-01174-1_58.

[83] M. Allende, D. López León, S. Cerón, A. Pareja, E. Pacheco, A. Leal, M. Da Silva, A. Pardo, D. Jones, D. J. Worrall, B. Merriman, J. Gilmore, N. Kitchener, and S. E. Venegas-Andraca. Quantum-resistance in blockchain networks. *arXiv preprint arXiv:2106.06640*, 2021.

[84] Hyperledger project. Available online: https://www.hyperledger.org, .

[85] P. Chiavassa, A. Marchesin, I. Pedone, M. Ferrari Dacrema, and P. Cremonesi. Virtual Network Function Embedding with Quantum Annealing. In *2022 IEEE International Conference on Quantum Computing and Engineering (QCE)*, pages 282–291. IEEE, 2022. doi: 10.1109/QCE53715.2022.00048.

[86] M. A. Abdelaal, G. A. Ebrahim., and W. A. Anis. Efficient Placement of Service Function Chains in Cloud Computing Environments. *Electronics*, 10(3):323, January 2021. doi: 10.3390/electronics10030323. Number: 3 Publisher: Multidisciplinary Digital Publishing Institute.

[87] W. Xuan, Z. Zhao, L. Fan, and Z. Han. Minimizing Delay in Network Function Visualization with Quantum Computing. *arXiv:2106.10707*, June 2021. arXiv: 2106.10707.

[88] B. Yi, X. Wang, and M. Huang. Optimised approach for VNF embedding in NFV. *IET Communications*, 12(20):2630–2638, 2018. ISSN 1751-8636. doi: 10.1049/iet-com.2018.5509. _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1049/iet-com.2018.5509.

[89] M. Asgarian, G. Mirjalily, and Z. Luo. Embedding Multicast Service Function Chains in NFV-Enabled Networks. *IEEE Communications Letters*, 25(4):1264–1268, April 2021. ISSN 1558-2558. doi: 10.1109/LCOMM.2020.3044894. Conference Name: IEEE Communications Letters.

[90] H. Ushijima-Mwesigwa, C. F. A. Negre, and S. M. Mniszewski. Graph Partitioning using Quantum Annealing on the D-Wave System. *In Proceedings of the Second International Workshop on Post Moores Era Supercomputing (PMES '17)*, abs/1705.03082, 2017.

[91] C. Bauckhage, N. Piatkowski, R. Sifa, D. Hecker, and S. Wrobel. A QUBO Formulation of the k-Medoids Problem. In *Proceedings of "Lernen, Wissen, Daten, Analysen"*, volume 2454 of *CEUR Workshop Proceedings*, pages 54–63, 2019.

[92] D. Willsch, M. Willsch, H. De Raedt, and K. Michielsen. Support vector machines on the D-Wave quantum annealer. *Comput. Phys. Commun.*, 248:107006, 2020. doi: 10.1016/j.cpc.2019.107006.

[93] S. H. Adachi and M. P. Henderson. Application of Quantum Annealing to Training of Deep Neural Networks. *CoRR*, abs/1510.06356, 2015.

[94] M. H. Amin, E. Andriyash, J. Rolfe, B. Kulchytskyy, and Roger Melko. Quantum boltzmann machine. *Physical Review X*, 8(2):021050, 2018. doi: 10.1103/PhysRevX.8.021050.

[95] M. Ferrari Dacrema, F. Moroni, R. Nembrini, N. Ferro, G. Faggioli, and P. Cremonesi. Towards Feature Selection for Ranking and Classification Exploiting Quantum Annealers. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, pages 2814–2824. ACM, 2022. doi: 10.1145/3477495.3531755.

[96] R. Nembrini, M. Ferrari Dacrema, and P. Cremonesi. Feature Selection for Recommender Systems with Quantum Computing. *Entropy*, 23(8):970, 2021. doi: 10.3390/e23080970.

[97] J. G. Herrera and J. F. Botero. Resource allocation in NFV: A comprehensive survey. *IEEE Transactions on Network and Service Management*, 13(3):518–532, 2016. doi: 10.1109/TNSM.2016.2598420.

[98] F. Glover, G. Kochenberger, and Y. Du. Quantum Bridge Analytics I: a tutorial on formulating and using QUBO models. *4OR*, 17, December 2019. doi: 10.1007/s10288-019-00424-y.

[99] G. Palubeckis. Multistart Tabu Search Strategies for the Unconstrained Binary Quadratic Optimization Problem. *Ann. Oper. Res.*, 131(1-4):259–282, 2004. doi: 10.1023/B:ANOR.0000039522.58036.68.

[100] S. Kirkpatrick, D. Gelatt, and M. P. Vecchi. Optimization by Simmulated Annealing. *Sci.*, 220(4598):671–680, 1983. doi: 10.1126/science.220.4598.671.

[101] V. Choi. Minor-embedding in adiabatic quantum computation: I. The parameter setting problem. *Quantum Inf. Process.*, 7(5):193–209, 2008. doi: 10.1007/s11128-008-0082-9.

[102] C. Carugno, M. Ferrari Dacrema, and P. Cremonesi. Evaluating the job shop scheduling problem on a D-wave quantum annealer. *Nature Scientific Reports*, 12(1):6539, Apr 2022. ISSN 2045-2322. doi: 10.1038/s41598-022-10169-0.

[103] A. Schönhageand and V. Strassen. Schnelle multiplikation grosser zahlen. *Computing*, 7(3):281–292, 1971.

[104] M. E. Briggs. *An introduction to the general number field sieve*. PhD thesis, Virginia Tech, 1998.

[105] S. Beauregard. Circuit for Shor's algorithm using 2n+ 3 qubits. *Quantum Information & Computation*, 3(2):175–185, 2003.

[106] P. W. Shor. Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer. *SIAM Review*, 41(2):303–332, 1999. doi: 10.1137/S0036144598347011.

[107] R. Griffiths B and C. Niu. Semiclassical Fourier transform for quantum computation. *Physical Review Letters*, 76(17):3228–3231, 1996.

[108] Y. Takahashi and N. Kunihiro. A quantum circuit for Shor's factoring algorithm using 2n+ 2 qubits. *Quantum Information & Computation*, 6(2):184–192, 2006.

[109] T. Häner, M. Roetteler, and K. M. Svore. Factoring using 2n+2 qubits with Toffoli based modular multiplication. *arXiv preprint arXiv:1611.07995*, 2016.

[110] Y. Takahashi, S. Tani, and N. Kunihiro. Quantum addition circuits and unbounded fan-out. *arXiv preprint arXiv:0910.2530*, 2009.

[111] C. Gidney and M. Ekerå. How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits. *Quantum*, 5:433, 2021.

[112] J. H. Silverman. *The arithmetic of elliptic curves*, volume 106. Springer, 2009.

[113] M. Roetteler, M. Naehrig, K. M. Svore, and K. Lauter. Quantum resource estimates for computing elliptic curve discrete logarithms. In *23rd International Conference on the Theory and Applications of Cryptology and Information Security*, pages 241–270, Hong Kong (China), December 3-7 2017. doi: 10.1007/978-3-319-70697-9_9.

[114] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of computation*, 44(170):519–521, 1985.

[115] B. S. Kaliski. The Montgomery inverse and its applications. *IEEE transactions on computers*, 44(8):1064–1065, 1995.

[116] T. Häner, S. Jaques, M. Naehrig, M. Roetteler, and M. Soeken. Improved quantum circuits for elliptic curve discrete logarithms. In *International Conference on Post-Quantum Cryptography*, pages 425–444. Springer, 2020.