From Weapon to Target: Quantum Computers Paradox

Mădălina Bolboceanu
mbolboceanu@bitdefender.com

Sorin Boloș
sorin.bolos@transilvania-quantum.com

Adrian Coleșa
acolesa@bitdefender.com

Andrei Kisari
akisari@bitdefender.com

Andrei Luțaș
vlutas@bitdefender.com

Dan Luțaș
dlutas@bitdefender.com

Radu Mărginean
radu.marginean@transilvania-quantum.com

Andrei Muntea
amuntea@bitdefender.com

Radu Portase
rportase@bitdefender.com

Miruna Roșca
mrosca@bitdefender.com

August 2, 2024

Executive Summary

The impact of quantum computing on the classical computing based cybersecurity has been discussed extensively over the past 30 years. This led to development of the so-called post-quantum cryptography. In the same time, relatively little attention has been paid to the security of quantum computers. This paper examines issues related to the *security of quantum computers and quantum computing process*. We investigated in this direction trying to *identify possible vulnerabilities and attack vectors in the most popular quantum computing infrastructures*.

We looked at the main quantum computer providers, like IBM and IonQ, and the different ways their resources could be used by end users. Furthermore, we analyzed the most popular quantum software development kits, like Qiskit, and the entire quantum programming workflow they imply.

The *result of our analysis is a set of threat models and attack vectors* on the different phases of the quantum programming workflow. For some identified attack vectors we *derived proof-of-concept attacks*.

In the realm of *classical attacks on* classical computing resources, in particular, on *quantum computing software stack*, including the cloud services that must be used to let end users access centralized quantum computers, we identified flaws regarding the way *authentication tokens* are managed, which make them vulnerable to being *stolen* and used to impersonate victim users. In the same class of attacks, we developed one that *corrupts quantum SDKs' packages* to transparently *tamper with end users' quantum circuits*, injecting attacker's circuits alongside victim's ones and making their presence invisible for the attacked user.

Regarding *quantum-based attacks on quantum processing units (QPUs)*, we implemented a couple of proof-of-concept attacks that try to (1) *exploit the quantum computers' qubit imperfect reset* to infer results of quantum circuits run before the attacker's circuit (2) *exploit the quantum computers' qubit imperfect reset* to affect the results of circuits run immediately after, and (3) evaluate impact of qubit *cross-talk*

effects in multi-tenant scenarios.

By our investigation, we want to *raise awareness* for both end users, to protect their data and computers while running quantum programs, and quantum computer providers, to protect their infrastructures against possible attacks.

We make publicly available the code of our experiments at `https://github.com/Transilvania-Quantum/quantum-computing-security-investigations`.

Contents

Executive Summary **1**

1 Introduction **6**

2 Quantum Computing Overview **9**

2.1 Quantum Bits, Gates, Circuits and Computers . . . 9
2.2 Quantum Computer Providers . . . 11
2.3 Open-Source Quantum Software Development Kits (SDKs) . . . 13
2.4 Quantum Programming Workflow . . . 15
2.5 Quantum Computers Today . . . 17

3 Threat Models **18**

3.1 Classical Attacks on Quantum Computing Software Stack . . . 20
3.2 Classical Attacks on Quantum Processing Units (QPUs) . . . 23
3.3 Quantum Attacks on Classical Computers . . . 24
3.3.1 Quantum Algorithms and Security on Internet . . . 24
3.3.2 Post-Quantum Cryptography . . . 25
3.3.3 The Transition to PQC . . . 25
3.4 Quantum Attacks on QPUs . . . 25

4 Attack Vectors **28**

4.1 Classical Attacks on Quantum Computing Software Stack . . . 28
4.1.1 Supply Chain Attacks . . . 28
4.1.2 Compromised Quantum User's Computer . . . 29
4.1.3 Untrusted Transpilers . . . 29
4.1.4 Plain-Text Authentication Tokens . . . 30
4.1.5 Man-in-the-Middle (MitM) . . . 30

4.1.6 DNS / IP Spoofing 30
4.1.7 Man-in-the-Browser (MitB) 31
4.1.8 Denial of Service (DoS) 31
4.1.9 Untrusted Quantum Providers 31
4.1.10 Untrusted Quantum Users 32
4.2 Classical Attacks on QPUs 32
4.2.1 Attacking QPU Calibration Using the Pulse API 32
4.2.2 Side-Channel Attacks 33
4.2.3 Scheduler Attacks 33
4.3 Quantum Attacks on Classical Computers 34
4.3.1 Quantum Algorithms 34
4.4 Quantum Attacks on QPUs 35
4.4.1 The $|11..1\rangle$ State Initialization Attack 35
4.4.2 Accessing Higher Energy States Attacks 37
4.4.3 Readout Attacks in Multi-tenant Environments 39
4.4.4 Readout Attacks in Single-tenant Environments 40
4.4.5 Cross-Talk Attacks 41
4.4.6 Shuttle Exploiting in Trapped-Ions Quantum Computers . . . 43

5 Research, Analysis and Experiments 45
5.1 Classical Attacks on Quantum Computing Software Stack 45
5.1.1 Attacking the API Authentication Tokens 45
5.1.2 Quantum Circuit Hidden Alteration 48
5.2 Classical Attacks on QPUs 50
5.2.1 Attacking QPU Calibration Using the Pulse API 50
5.3 Quantum Attacks on Classical Computers 51
5.4 Quantum Attacks on QPUs 51
5.4.1 Experiments on Qubit Reset Attacks 51
5.4.2 Fault Injection Attacks 57
5.4.3 Exploring the Potential for Cross-Talk Attacks 62

6 Reflections on Quantum Computer Related Security 66
6.1 Importance of Our Investigation 66
6.2 Attacks and Defenses 67
6.2.1 Classical Attacks on Quantum Computing Software Stack . . 67
6.2.2 Classical Attacks on QPUs 70
6.2.3 Quantum Attacks on Classical Computers 70

6.2.4 Quantum Attacks on QPUs 70

Conclusions **72**

Appendices **73**

1 Introduction

While the impact of quantum computing on the classical computing based cybersecurity has been discussed extensively over the past 30 years, relatively little attention has been paid to the security of quantum computers. This paper examines issues related to the *security of quantum computers and quantum computing process*. Our main objective is to shed light on the various attack vectors on quantum computing infrastructures.

Research into the field of quantum computing gained momentum in the 90s with the discovery of quantum algorithms that could solve some computational problems faster than classical computers are known to do [6, 29], [81, 80]. Today, quantum computers from various suppliers with up to hundreds of qubits are available to the public. Although quantum computers are unlikely to completely replace classical computers in the foreseeable future, specific applications of quantum computers are being investigated in several key areas. In chemistry and materials science, quantum computers will be used to simulate and calculate the properties of physical systems more precisely than classical computers ever could [39]. Promising applications are being investigated for large-scale optimization tasks in areas of practical interest such as transport, logistics, or finance [8]. Another area of research that has received a lot of attention in recent years is performing machine learning with the help of quantum computers [17]. In cybersecurity, quantum computers will one day be used to factorize large integers and solve discrete log problems that enable cracking classical cryptographic schemes like RSA, DSA, and elliptic curves [49].

Quantum computers use quantum mechanical effects like superposition, interference and entanglement to perform calculations, unlike classical computers, which operate within the framework of classical electricity and magnetism. To perform a meaningful task, such systems must be shielded from external influences long enough to execute a quantum algorithm. While quantum computers working at room temperature are being researched, most existing ones require isolated environments and special care and maintenance.

Consequently, quantum programs are executed on remote quantum computers, available to quantum software developers via cloud services. Some phases of their development workflow are performed as classical computation either on end users' computers or under user's account on various commercial cloud platforms. Classical data transfers also take place between end users and remote quantum computing infrastructures. This way of developing and running quantum programs could have important implications for the overall cybersecurity of the quantum computing process.

We investigated in this direction trying to *identify possible vulnerabilities and attack vectors in the most popular quantum computing infrastructures*. We looked at the main quantum computer providers, like IBM and IonQ, and the different ways their resources could be used by end users. Furthermore, we analyzed the most popular quantum software development kits, like Qiskit, and the entire quantum programming workflow they imply.

The *result of our analysis is a set of threat models and attack vectors* on the different phases of the quantum programming workflow. We *classified threats* relative to all possible combinations of computing resources an attacker could use and targets he could aim for: (1) classical attacks on classical computing resources, including in particular the quantum computing software stack of interest for us here; (2) classical attacks on quantum processing units (QPUs); (3) quantum attacks on classical computers; (4) quantum attacks on QPUs. We *defined threat models* for each such a class and tried to identify attack vectors for them. For some identified attack vectors we *derived proof-of-concept attacks*, like (1) stealing end user authentication tokens; (2) corrupting quantum SDKs' packages to transparently tamper with end users' quantum circuits; (3) exploiting the quantum computers' qubit imperfect reset to infer results of quantum circuits run before the attacker's circuit; (4) exploiting the qubit imperfect reset to affect the results of quantum circuits run immediately after; (5) evaluating impact of qubit cross-talk effects in multi-tenant scenarios. We also researched *developing new lattice-based post-quantum cryptographic schemes*.

By our investigation, we want to *raise awareness* and *provide guidance* for both end users, to protect their data and computers while running quantum programs, and quantum computer providers, to protect their infrastructures against possible attacks.

Our paper's is structured in the following way:

- Chapter 2 reviews the main quantum computer providers and the quantum programming workflow;
- Chapter 3 defines threat models against today's quantum computing infrastructures;
- Chapter 4 describes different attack vectors on the quantum programming workflow;
- Chapter 5 details theoretical research and experiments we conducted to validate the proposed threat models;
- Chapter 6 recommends some best practices in order to reduce exposure to the threats we identified;
- The last section concludes the paper.

2 Quantum Computing Overview

2.1 Quantum Bits, Gates, Circuits and Computers

Current computers, which we will refer to as *classical computers*, operate within the limits of classical physics, which assumes that a system can be in only one observable state at any given moment. Quantum physics, however, says that a system can be in a *superposition* of multiple classical states. Such a state is called a *quantum state* and can be expressed as a *vector of amplitudes* α_i, each corresponding to a classical state S_i, as if the system is in all those states at the same time. Equation 2.1 illustrates the formula for a quantum state. The amplitudes α_i are complex numbers.

$$|\Psi\rangle = \sum_{i=1}^{N} \alpha_i \, |S_i\rangle \tag{2.1}$$

Quantum computing applies concepts and knowledge from classical computer science while exploiting phenomena specific to quantum physics with the purpose of executing a computation. The devices build with such a purpose are called *quantum computers*.

In classical computers, the minimum unit of information is represented by a bit. A bit can have one of two possible values, 0 and 1, which usually correspond to the absence or presence of a voltage across an electrical circuit in a classical CPU. For quantum computers the minimum unit of information is called a *quantum bit* or *qubit*. Physically, a qubit is a system having two energy levels. In this case, the physical system can be either a two-state system provided to us by nature, such as the spin of an electron or an atomic nucleus, or a physical system engineered by humans, such as the subspace of the two lowest energy levels in a macroscopic quantum system like a superconducting circuit. Regardless of its physical realization, the state of a qubit is generally a quantum state. Using the *bra-ket notation* a qubit's basis states are noted $|0\rangle$ and $|1\rangle$, while its quantum state is a superposition of the two, like in Equation 2.2.

$$|\Psi\rangle = \alpha |0\rangle + \beta |1\rangle \tag{2.2}$$

Measuring a qubit always provides one bit of information, by returning one of the two basis states. In the conventional physics interpretation, when a quantum bit is measured, its state collapses randomly from a superposition state into one of the two available basis states: $|0\rangle$ or $|1\rangle$. $|\alpha|^2$ is the probability of measuring the state $|0\rangle$, while $|\beta|^2$ is the probability of measuring the state $|1\rangle$. The sum of probabilities must be one $|\alpha|^2 + |\beta|^2 = 1$. This is called the normalization condition.

In ket notation, the basis states for a two-qubit system are $|00\rangle$, $|10\rangle$, $|01\rangle$, $|11\rangle$. In general, the state of a system with N qubits has the dimension 2^N and its basis states can be represented either using N digits kets $|q_1 q_2 \ldots q_N\rangle$ where $q_i \in 0, 1$ or via column vectors with 2^N complex entries.

Qubits alone are useless for computing unless you can change their state. This could be done by applying a *quantum gate* to one or more qubits, which can be categorized as *1-qubit* (i.e. one-qubit) or *n-qubit* (i.e. multiple-qubit) gates, respectively. In physical terms, a quantum gate usually means that electromagnetic pulses are applied to the physical system that implements the qubits. An example of a one-qubit gate is the X gate used to transform the state $|0\rangle$ to $|1\rangle$ and vice versa.

The reason why quantum computers can perform different algorithms than classical computers are given by phenomena like superposition, interference, and entanglement. Superposition is a basic property of quantum physics and has already been mentioned. Interference makes it possible that at the end of a computation, the states corresponding to the answer we want to find will be enhanced while the probability of measuring other states is suppressed. Entanglement is difficult to explain in a few words, but it manifests itself as correlations among qubits that have no classical analogue. Entanglement between two qubits can be created using a Controlled-X gate.

Having a universal gate set at our disposal, we can express any quantum computation in terms of a *quantum circuit*, where gates are applied from left to right. While there are other ways to reason about quantum algorithms [98], in quantum computing practice a quantum circuit is the preferred abstraction used to describe how quantum gates are applied to qubits to perform a computation.

For a more thorough discussion of quantum computing the reader is referred to [88] for an elementary introduction and to [45] for an in depth treatment.

2.2 Quantum Computer Providers

Several quantum hardware producers have established themselves as key players in the quantum computing market. Many provide students, researchers, software developers and the public in general with access to their infrastructure, enabling users to upload and execute circuits. This access is typically facilitated through subscription models or free limited monthly usage allowances. Many of these companies have chosen to embrace the open-source community by making relevant parts of their software stack publicly accessible. This approach ensures that, for those interested to experiment with quantum computing, the intricate hardware-specific instructions are abstracted away, simplifying the utilization of quantum computing resources, but it comes with an associated risk as an attacker might be more familiar with the code and the framework and could find ways to exploit them with malicious intentions.

The information in this section reflects the state of affairs at the beginning of year 2024 and likely will become outdated relatively soon. The list of startups and companies that are developing quantum computers is growing fast. When trying to review this list, one needs some principles to organize and help make sense of it. A very natural criterion is to categorize the quantum hardware providers by the kind of technology used to build the quantum bits. Furthermore, it makes sense to start with the technologies that are more advanced at this moment and have attracted more users. We do not know which technology or technologies will prove most viable in the long run since each technology has its own advantages and disadvantages and scaling a quantum computer to hundreds of thousands or even millions of physical qubits is uncharted territory.

Superconducting quantum bits [34, 35] provide one of the most advanced technologies available today. A device with more than 1000 quantum bits has been unveiled recently by IBM [15]. Other companies using this technology are Google, Rigetti, Oxford Quantum Circuits, IQM, Amazon Web Services, Alice & Bob, Nord Quantique, Quantum Circuits Inc, SEEQC, and D-Wave. Today IBM offers public access via the IBM Quantum Platform to many of its quantum computers. Using Azure Quantum users can access devices from Rigetti and Quantum Circuits Inc. Using Amazon Braket users can access devices from Rigetti and Oxford Quantum

Circuits. IQM offers access to its systems via T-Systems Quantum Cloud. While selected users can submit jobs to Google Quantum Computing Service which offers access to Google's quantum devices, public access from Google is not available at the time of writing.

Devices based on trapped ion architectures are developed by companies like: IonQ, Quantinuum, AQT, Quantum Factory, Oxford Ionics and Eleqtron. At this moment IonQ, Quantinuum and AQT offer public access to their quantum computers and various simulators either directly (IonQ) or via cloud platforms like Amazon Braket (IonQ), Azure Quantum (IonQ, Quantinuum) or T-Systems Quantum Cloud (AQT). The number of qubits available at this moment is of the order of only tens of qubits, but devices built around such platform tend to have larger quantum volumes, which is a synthetic metric that characterize the overall performance of a Quantum Processing Unit (QPU) [22].

Photons can be generated having two different polarization states, which provides a very accessible method to implement a qubit. Quantum computers that use photons as qubits are in principle easier to scale but need sophisticated techniques for implementing two-qubit quantum gates because photons do not interact strongly with each other [82]. Xanadu develops QPUs taking advantage of technology named measurement-based quantum computing [12] while PSIQuantum uses a technique they name fusion-based quantum computing [5]. Such computers can be used to implement familiar gate-based quantum algorithms but can also be used to develop specialized algorithms [1]. Some other companies that develop quantum devices based on photonics technologies are Orca Computing, Quandela, Quix Quantum, and TundraSystems Global. Today, Xanadu offers public access to simulators and their quantum computers with up to a couple of hundred qubits via the Xanadu Quantum Cloud platform.

Another technology that has achieved promising results is based on neutral atoms, which uses lasers to cool and manipulate neutral atoms confined in optical traps [99]. A qubit is implemented by different energy states of the same atom and the prospects for scaling up such systems are promising. Similar to ions in trapped ions architectures, atoms can be shuttled, which enables implementation of direct two-qubit gates between any two qubits. This makes gates relatively slow, effect which is partially offset by longer coherence times. A few companies developing quantum computers using neutral atoms are PASQAL, Atom Computing,

[1]Both gate-based and measurement-based quantum computers are universal computers, meaning that they can be used to implement any arbitrary computation.

ColdQuanta, Nanofiber Quantum and QuEra Computing. Today, QuEra offers access to 256 qubit quantum devices via Amazon Braket while PASQAL offers access to 100 qubit quantum devices via Azure Quantum and PASQAL Cloud Services.

2.3 Open-Source Quantum Software Development Kits (SDKs)

Because the landscape of quantum computing software is evolving rapidly, we do not try to provide a comprehensive review here. We will restrict ourselves to discussing briefly several open-source SDKs that are popular today. Besides being general-purpose quantum programming frameworks, some of these provide additional libraries and support for approaching problems of practical interest with the help of quantum computers. These can be categorized into four broad application areas: (1) solving complex optimization problems, (2) simulating the properties of molecules and materials (3) machine learning and (4) finance. Many of those packages provide detailed tutorials, code samples and other kinds of support materials. These constitute valuable learning resources for anybody trying to learn the basics or to further expand their knowledge in quantum computing.

Qiskit [63] is a Python quantum SDK created by IBM and, according to the Unitary Fund's *State of Quantum Open Source Software 2023* survey [93] is the most popular open-source platform for quantum computing. It implements most of the functionalities needed for developing quantum programs and combines good feature coverage with excellent learning resources. At the moment of writing, IBM provides free public access to several 127 qubit quantum computers. Besides the general purpose software package Qiskit itself, there are specialized sub-packages in Qiskit for applications in chemistry and material science [66], machine learning [65], optimization problems [67] and finance [64]. Using Qiskit users can submit quantum programs to devices and quantum simulators provided by IBM, IonQ, Quantinuum, Rigetti, AQT, QuTech, and PASQAL.

The Qiskit framework started in 2017 with OpenQASM [21] which was initially intended to be a circuit description language. Since then, OpenQASM has become the de facto standard used for communicating circuits among libraries and quantum software packages published by different authors. In its latest version, OpenQASM 3.0 [20], besides describing quantum circuits, provides support for pulse based quantum programming and declaring fragments of classical code that run "near" the quantum computer. Here proximity is defined by the fact that the classical code is run by controllers physically near the QPU, but also because it can run within the coherence time of qubits. Thus, today the OpenQASM language has evolved into an intermediate representation of quantum code. Python code describing circuits is typically compiled into OpenQASM code before being compiled further into representations that can be executed on QPUs and their controllers.

Strawberry Fields [85] from Xanadu is an open-source cross-platform Python library for simulating and executing programs on quantum photonic hardware. Besides Strawberry Fields, Xanadu has created PennyLane which is an open-source platform targeting applications of quantum computing in machine learning and chemistry. PennyLane provides a set of plugins [50] that integrate the development environment with Qiskit, Amazon Bracket, Cirq and Microsoft SDK. Besides Xanadu devices users of these software packages can submit quantum jobs to IBM, IonQ, Quantinuum, Rigetti, AQT and Quantum Inspire, plus a number of quantum simulators.

Cirq [18] is a Python open-source SDK from Google Quantum AI. Google has also created TensorFlow Quantum [91] a library for hybrid quantum-classical machine learning and OpenFermion [48], an open-source package for doing chemistry using quantum computers. Cirq users can submit quantum programs directly to AQT, PASQAL, IonQ and Rigetti. Moreover they can send circuits to AQT, PASQAL, IonQ, Rigetti, and Quantinuum via Azure Quantum and access simulators from various companies.

Tket [95] is an open-source SDK from Quantinuum for the creation and execution of quantum programs. Being hardware-agnostic it provides extension modules that target multiple quantum platforms like Quantinuum, IBM, Rigetti, AQT or IQM. It also facilitates access to the Microsoft Azure and Amazon Braket cloud platforms. Additionally, it provides an extremely performant transpiler and optimizer for quantum circuits.

Forest SDK is an SDK created by Rigetti Computing. Part of this package is Pyquil [58] an open source Python library for quantum programming and Quil [70] a quantum instruction language that can be used to write programs that can be executed on the quantum devices and simulators from Rigetti.

Amazon Braket is a cloud based computing service from Amazon. Amazon supports the Amazon Braket SDK [2] an open source framework for quantum computing written in Python that enables access to devices and simulators from Oxford Quantum Circuits, Rigetti, IonQ and QuEra.

So far we have discussed mostly Python libraries, but there are programming languages dedicated to quantum computing. A representative example is Q# from Microsoft [60], a high-level, open-source quantum programming language. Programs written in Q# can be run via Azure Quantum on quantum computers from IonQ, Qunatinuum, PASQAL, Rigetti and QCI. Another feature of Q# worth mentioning is that it can be compiled into Quantum Intermediate Representation (QIR) [62], an LLVM [92] based intermediate representation for quantum code. Programs written in a language that targets QIR can be run on any quantum device that supports it.

2.4 Quantum Programming Workflow

In principle, a quantum program could be a piece of code written in a dedicated programming language like, for example, Q#. Today, in practice, a quantum program is most often represented by some module of code developed in Python using a dedicated Python SDK like those mentioned in Section 2.3. In general, such a Python module contains specifications in code for both classical and quantum computations. Quantum computations are laid out in the form of quantum circuits which can be written directly using the Python programming language, created using a dedicated quantum circuit description language like OpenQASM, or generated dynamically using some API provided by the Python SDK.

The classical computation runs locally either directly in a Jupyter Notebook on the user's PC or under the user's account on some cloud platform like Azure Quantum or Amazon Braket. It could contain simple code helping set up the circuits the quantum program needs or more elaborate code needed for hybrid quantum-classical routines like VQE [94], QAOA [9] or quantum assisted machine learning. The hybrid quantum-classical algorithms run iteratively where quantum circuits are sent to the quantum computing platform, processed remotely and results sent back to the classical client. After some extra processing, new circuits are generated and the cycle is repeated until a predefined condition is met and the quantum program terminates. What is transferred between the user and the quantum device are always quantum circuits and for this purpose, one uses a data exchange formats like JSON or the QPY serialization format [68]. Very schematically, everything a quantum computer is doing remotely, is to initialize the qubits to $|0\rangle$ state, execute the quantum circuits, measure the results for those qubits we want to measure, and report the results back to the user.

A generic quantum circuit cannot be run unmodified on a quantum computer. The reason is that quantum computers provide a limited set of physical gates, which may not coincide with the set of gates used to implement a quantum algorithm as a circuit. For such a set of physical gates one needs a minimum of several one-qubit gates and at least one two-qubit gate like for example the Controlled-X gate. Also in certain quantum computing architectures, the connectivity between qubits is limited which does not permit the implementation of two-qubit gates between arbitrary qubits. More complicated circuits containing chains of Swap gates must be used to entangle remote qubits. The process of rewriting a quantum circuit in a form suitable for execution on a quantum computer is called transpiling. This operation can use significant amounts of memory and take long processing times for larger circuits and is usually run locally by the user before the circuit is sent to the quantum computer. This is also the moment where the circuit is optimized. The primary target of optimization is to reduce the depth of circuits which implies optimizing gate layout and reducing the number of gates, especially the number of two qubit gates.

To understand the entire lifecycle of quantum code, we will take a closer look at the operations performed remotely on the quantum computer provider side. There, quantum circuits belonging to a job submitted by the user are received and wait in a queue until they are scheduled for execution. Most of the circuits are static but OpenQASM provides support for specifying simple routines of classical code that are executed "in real time", within the coherence time of the qubits, by the QPU controllers which could be for example FPGA devices. Next, the circuits are converted to specifications for electric pulse signals which may be microwave pulses in the case of superconducting qubits or laser pulses for trapped ions and neutral atoms. In principle, a pulse program constitute a virtual execution model. This virtual execution model is then compiled by a classical coprocessor into the instruction set architecture (ISA) of the underlying control hardware. Then the circuit is executed on the particular QPU chosen by the user when he submitted the job to the quantum provider. After the circuit execution has finished, the results are obtained by performing a quantum measurement on a subset or, on all the qubits. Because quantum algorithms are probabilistic, a circuit is run multiple times. The user specifies this as number of shots for his job. As far as we know, all shots run in sequence on the QPU. If a user specifies 1000 shots, 1000 circuits will run in series and are not interleaved with circuits from another user. After all shots have been run, the results for the job are aggregated and sent back to the user.

2.5 Quantum Computers Today

The quantum devices existing today are referred to as NISQ quantum computers, where NISQ stands for Noisy Intermediate-Scale Quantum [54]. One major feature which the current generation of computers does not support is quantum error correction. The lack of quantum error correction means that only limited depth circuits can be executed before the computation is overwhelmed by the inherent noise in the quantum gates. The errors for the best quantum gates today approach 0.1% for two-qubit quantum gates and are one order of magnitude smaller for single qubit quantum gates. In the future, when quantum error correction will become available, large depth quantum circuits will be run with an overall error that can be exponentially suppressed. In order to enable quantum error correction, hundreds or thousands of physical qubits will be needed to implement a logical qubit. An error corrected device having on the order of one million physical quantum bits and one thousand logical qubits will provide results that cannot be matched by the classical computational methods. While waiting for these machines to become a reality, error mitigation [13] is a technique that can in principle be used to extract useful results using the NISQ quantum computers existing today. Error mitigation improves the accuracy of results obtained using current devices with the price of running more circuits. This price is in principle exponential but, within a certain domain in circuit size and complexity, it can be used to obtain precise results as it has been proved in a recent landmark paper from IBM [33].

3 Threat Models

We try to identify in this chapter the possible cybersecurity threats that exist in a computing environment that provides access to quantum computers. As we saw in the previous chapters, the existing quantum computers are expensive and need specialized maintenance, so they are hosted by quantum hardware companies and are made accessible to end users via cloud services.

Quantum computers are needed in an attempt to provide better solutions to difficult computational problems than classical algorithms. Most of the problems that we know of, that can currently be addressed by quantum-assisted computation are fairly well-understood and relatively specialized. Notwithstanding, the field is constantly evolving, so new quantum computing algorithms could still be developed. In this context, keeping confidentiality of such new algorithms and their input data could be critical, especially in competitive domains like pharma or finance.

By sending to remote cloud services quantum circuits which model quantum algorithms and their input data, we expose them to different threats, like being stolen or tampered with. This could happen on communication channels while in transit, in cloud or even on quantum computers, if an attacker succeeds compromising one of those resources. In theory, they could be also stolen by untrusted cloud providers. Consequently, the problems the circuits solve could be inferred by the attacker.

On another perspective, malicious users could try to use quantum computers for malicious purposes. It is well known that in the post-quantum age, quantum computers could be used to break, at least partially, the classical cryptographic schemes, but we expect them to be used for wider malicious purposes also, like attacking the quantum circuits and other computations users may run on a quantum device.

In the arena of cybersecurity, classical and quantum computers interact at several contact interfaces, defining an emerging landscape of cyberattacks, which we try to classify in Figure 3.1.

Classical Attacks on Quantum Computing Software Stack	Classical Attacks on QPU
Quantum Attacks on Classical Computers	Quantum Attacks on QPU

Figure 3.1: The interplay between classical and quantum computing in the arena of cybersecurity

Classical attacks on quantum computing software stack, which is composed by *classical computing resources*, can target user input data or user quantum programs, which may contain sensitive intellectual property (IP) in the form of custom quantum algorithms. Furthermore, an attacker can alter the quantum circuits a user may wish to run or modify the number of shots or the maximum credits the user assigns to a particular computation. Such issues and others in the same category represent a high-impact attack surface on quantum computation, which is analyzed in Section 3.1, Section 4.1, and Section 5.1.

Classical attacks on quantum processing units (QPUs) can take the shape of specially designed pulses (for those quantum devices that support pulse API), which could be used, for example, to alter a quantum device's calibration. Another classical attack vector on a QPU may be provided by side channels like the radio frequency (RF) signal originating from the microwave pulses executed on a quantum device. This could be used to steal quantum circuits. Attacks of these types are discussed in Section 3.2, Section 4.2, and Section 5.2.

Quantum attacks on classical computers is currently understood as the possibility of using quantum computers to break classical cryptographic schemes, compromising digital signatures, or improving brute-force attacks on symmetric cryptography. It has been discussed in this form extensively in literature [49] and is one of the discoveries that contributed to the rise of interest in quantum computing [80, 29]. While such attacks are not yet viable using today's quantum computers, future-proofing security of communication on the internet may already be desirable for sensitive data like medical records, which need to be kept secret for a long time. A solution to this problem is so called *post-quantum cryptography*. We describe cybersecurity issues in this category in Section 3.3, Section 4.3, and Section 5.3.

Quantum attacks on QPUs are somewhat specialized and have a limited impact. However, what may seem surprising at first sight is that such attack vectors exist at all. They could take the form of a malicious quantum circuit running on a quantum computer, aiming to infer the results of other circuits run on the same quantum computer, influence the execution of other circuits, or reverse engineer the QPU. We discuss such attacks in Section 3.4, Section 4.4, and Section 5.4.

3.1 Classical Attacks on Quantum Computing Software Stack

In this section we try to identify different *types of attackers* that could target a quantum computing infrastructure and its environment. It is, also, important to understand what *advantages an attacker could take* from attacking such infrastructures and what *kinds of malicious actions* could perform on it.

Let us see, firstly, what an attacker could obtain by attacking a quantum computing infrastructure, which are the possible *targeted assets* and how an attacker could exploit them.

1. One obvious thing an attacker could do is to *steal confidential data* that lives in the quantum computing infrastructures. This could be, on one hand, *end-users' data*, like authentication credentials, quantum circuits, their parameters and results. The latter, in particular, are considered important *intellectual property* (IP). On the other hand, leaked data could belong to *quantum computer providers* and could consist in details about the quantum hardware's architecture and configuration. Such details could also be considered intellectual property or, even if not really confidential, could help the attacker develop other kinds of attacks.

2. Another advantage an attacker could get from quantum infrastructures is to

make *unauthorized use of quantum resources*. Running an attacker's quantum circuit on behalf of an unaware victim user, using her paid *cloud resources* and *quantum running time* is an example of this kind.

3. An attacker could be also interested in *tampering with an end user's circuits* or *influence their execution*. The purpose of such an attack is to affect, or even control, the results of victim user's circuits, indirectly *affecting her business decisions*.
4. Finally, an attacker could be pleased to only perform a *denial of service* (DoS) attack. This could make quantum resources unavailable to their users, affecting the business of both quantum computer providers and their users. Examples of this kind could be: (1) exhaust a user's cloud or quantum running time credit; (2) overload the quantum job queues maintained in cloud infrastructures, slowing down the advance of job schedulers and, indirectly, delaying indefinitely results awaited by quantum users; (3) decalibrate qubits or damage QPU of quantum computers, making them unavailable to quantum users.

In Section 2.4 we saw that different components of a quantum program are run in different places, i.e. on the end user's computer, in cloud or on quantum hardware. Data, e.g. quantum circuits and results, could be also stored, processed or in transit between those places. Let us see where the attacker could be positioned relative to that places, quantum programs' components and data and what malicious actions he could perform on them. We try to cover all possible *types of attackers* and corresponding *attack models*, even if some of them are not necessarily specific to quantum computing infrastructures, but could target any other kind of computing infrastructures.

1. The attacker could be an *anonymous user in internet*. In such a case, the attacker has no a priori advantage over any component of the quantum infrastructure, so could only try to target anyone of them using classical attack methods. For example, he could *target quantum users* through *social engineering attacks*, trying to trick them disclose confidential information or could simply *search for secret data publicly exposed* by mistake. Quantum users could be targeted indirectly, through *supply-chain attacks*, tricking them download and use malicious quantum SDKs. The attacker could also *target the cloud services* used to access quantum computers and exploit them if vulnerable

(e.g. brute force, broke or avoid flawed authentication, privilege escalation, steal information) as an unauthenticated or authenticated user, or once authenticated, *target the quantum hardware* (e.g. steal architectural details, decalibrate or perhaps even damage QPUs, steal or influence results of other user's quantum circuits).

2. A relatively small variation of the internet attacker is the *local network attacker*, i.e. an attacker in the same local network with a quantum computing infrastructure's user. In addition to the attack capabilities mentioned at item 1, an attacker of this type could also *monitor the network traffic* between the targeted user and quantum cloud services or could even try to *act as a man-in-the middle*. If the communication channel is not cryptographically protected, the attacker could steal confidential information or tamper with the transferred data.

3. Another kind of attacker could be even closer to an (unaware) quantum computing infrastructure's user, controlling in some way, partially or completely, the user's computer. Let us call such an attacker the *compromised user-system attacker*. There could be more variants of such an attacker, depending on how much power the attacker has over the compromised user's computer. For example, the attacker could act *remotely*, through malicious software installed on user's computer, or *physically*, having access to an unattended computer or a stolen one. When acting remotely, the attacker could run malicious code in an *unprivileged* or a *privilege process*. The malicious code, also, could provide the attacker a limited functionality (e.g. trying to escape installed security solutions) or a powerful one. Obviously, a compromised user-system attacker will mainly target user's data and processes, trying to steal confidential data or tamper with his victim's circuits and results.

4. The attacker could also be closer to the quantum cloud services or even to quantum computers, i.e. in a position that gives him some advantages over those resources. Let us called such an attacker *compromised-cloud attacker* and *compromised quantum-computer attacker*, respectively. Like a compromised user-system attacker, the compromised-cloud attacker could have more or less privileges over the cloud services, depending on which cloud services, resources or processes he controls, could have remote or physical access to cloud and quantum hardware, could be a sysadmin or just an outside intruder. However, an attacker like that has more power and attack

capabilities than the other types of attackers, and, also, could compromise more users and their data. An attacker with physical access to cloud or quantum hardware could perform even more sophisticated attacks, making really difficult to preserve the confidentiality and integrity of quantum user's data.

3.2 Classical Attacks on Quantum Processing Units (QPUs)

Some quantum hardware providers like those developing superconducting qubits expose programming APIs that help the user specify the implementation of quantum gates at the pulse level. This allows users to create pulse schedules tuned to the physical characteristics of each qubit and extract the maximum performance out of a QPU. However, this opens up the possibility that a user can leverage this feature to achieve unexpected results, like altering the calibration parameters of qubits in the case of superconducting quantum computers. Such an attack requires running long pulses at frequencies distanced from the qubit resonance frequency, which is a nonstandard usage of the pulse API and can be identified in software if the pulse schedule is scanned before execution. An attacker would not gain any direct benefits from executing such an attack, but could deteriorate the quality of results for other users. Since superconducting quantum computers are calibrated several times each day, the impact of such an attack is limited to a few hours. The Josephson junctions used to create a superconducting qubit are known to age. This leads to a degradation of the junction's performance. It is not clear to us if longer lasting effects of such an attack, like damage to the quantum bits, are possible.

Side-channel provide a different attack surface to quantum computing in a scenario where the attacker has direct or indirect access to quantum computer power usage. Monitoring the power usage of a computer is method that have been used for performing side-channel attacks in the case of classical processors [71]. For a quantum computer the most obvious vulnerability would be the power usage data from the QPU controllers. The target of such an attack would be to reverse engineer quantum circuits. In the case of superconducting quantum computers, this information can be corroborated with the RF signals emitted by the microwave pulses enacting the quantum gates. While gaining access to such information may be tempting for some attackers, the next logical step, namely extracting useful information about the problem approached by the user, is made more difficult by the fact that these circuits have already been transpiled and optimized.

A different instance of compromised quantum computer scenario would be a malicious user with access to the hardware provider infrastructure that may assign inferior hardware resources to the jobs sent by another user. There are various reasons why somebody would want to do that, like routing higher quality hardware for his own use. The victim does not control directly the QPU that actually runs his circuits. He only knows what the provider reports back to him. If the provider is under the control of the attacker, the information sent to the victim can be manipulated. There are mitigation measures that the victim can perform against such attacks which are discussed briefly in Subsection 4.2.3.

3.3 Quantum Attacks on Classical Computers

3.3.1 Quantum Algorithms and Security on Internet

The security of most of the modern cryptographic algorithms is based on the hardness of efficiently solving factorization, computing discrete logarithms or searching for a specific item. While Grover's impact on symmetric encryption schemes such as AES [56] could be easily countered by doubling the size of the keys, the impact of quantum algorithms on the RSA [72] public key encryption scheme and on the DSA [55] digital signature scheme is devastating. As a consequence, when quantum computers will become scalable, the security of our Internet connections will be completely broken.

When an attacker successfully breaches cryptographic schemes, they can unleash a wide range of threats, significantly undermining the confidentiality, integrity, and authenticity of digital communications and data. Having access to powerful quantum hardware means exactly that. When encryption can be broken, the attacker can intercept, read, and modify the messages between two parties without their knowledge, potentially leading to data breaches, eavesdropping, and information theft. If digital signatures are compromised, one can forge identities, creating or altering digital documents and transactions to impersonate legitimate users, thereby conducting fraudulent activities. Impersonation attacks, facilitated by breaking cryptographic authentication mechanisms, enable attackers to gain unauthorized access to systems, networks, and sensitive resources, posing as legitimate users. This can lead to a wide array of security breaches, including data theft, system sabotage, and unauthorized transactions, deeply impacting privacy, financial assets, and operational security.

3.3.2 Post-Quantum Cryptography

Post-Quantum Cryptography (PQC) is a modern branch of cryptography that focuses on developing algorithms that are believed to remain secure even against attacks implemented on quantum computers. The US National Institute of Standards and Technology (NIST), who has been actively involved in standardizing cryptographic algorithms for various purposes, initiated in 2016 a project known as the “Post-Quantum Cryptography Standardization” [46]. The goal of this project is to identify and standardize the most efficient cryptographic key encapsulation mechanisms and digital signatures that stay secure against quantum attacks. The security of most of the post-quantum proposals is based on the hardness of solving efficiently computational problems related to lattices, multivariate polynomial systems, codes or hash functions. The first set of post-quantum cryptographic standards is expected to be released during 2024 [47].

3.3.3 The Transition to PQC

Once the standardization process will be complete, organizations and industries will have to adopt new cryptographic algorithms to ensure the long-term security of their systems in the post-quantum era. The transition from current cryptographic algorithms to post-quantum algorithms is expected to take time, as it involves updating protocols, systems, and infrastructure. It is crucial for organizations to be aware of the potential risks posed by quantum computers and to plan for a smooth transition to post-quantum cryptographic solutions. NIST, in collaboration with other US agencies like the National Security Agency and the Cybersecurity and Infrastructure Security Agency, encourages early planning for the migration to post-quantum cryptographic standards. In pursuit of this goal, they have released a Quantum-Readiness Roadmap [69].

3.4 Quantum Attacks on QPUs

The research conducted to date on quantum-on-quantum attacks is based on assumptions about how quantum systems will be used in the future. Although quantum computers with over a thousand qubits exist today we do not yet know what usage patterns will prevail when quantum computers will become commercially mature. Accessing quantum computers is similar to the accessing high-performance computing resources: service providers host quantum systems and provide access to them as a service. This is indeed the case for almost all systems in operation today. Users typically submit a job containing quantum circuits to a cloud endpoint, and the service provider queues the job for asynchronous execution. The user can monitor the progress of the job and retrieve the results once the job is completed. Today queue times are rising and can be impractical as well as unpredictable. This is because the systems available for this kind of computation are scarce compared to the rising demand. For this reason, it is likely that service providers will move towards a multi-tenant, parallel execution model. This means that multiple users will be able to execute their jobs on the same quantum system at the same time. This will increase the utilization of the systems and reduce queue times. However, this also means that on quantum chips multiple circuits may run simultaneously. This significantly increases the potential for attacks. Almost all research on quantum attacks on QPUs assumes a multi-tenant environment [77, 23]. In this paper we also present new quantum attacks that can be carried out in a single-tennant scenario. For executing such an attack, the attacker and the victim must share some quantum resources or find a method to use one's quantum resources to influence the other's.

Many technologies compete to be the dominant one in the race to build a scalable quantum computing industry. From a theoretical point of view, to make good qubits, a technology must satisfy a set of criteria introduced by David DiVincenzo, now known as the DiVincenzo criteria [27]. From an economical point of view, the winning technology must not be prohibitively expensive to manufacture. From an engineering perspective, it must be scalable, reliable, and, if possible, reuse established technologies, like semiconductor processing techniques. We can add to all of these that the winning technology must be made secure and invulnerable to attacks. Not all technologies have been assessed from a security perspective. To the best of our knowledge, only superconducting qubits and trapped ions have been studied in this regard. It is likely that at this early stage of the quantum computing evolution, each technology has attacks specific to it.

In superconducting technology, qubits are made up of electrical circuits that are cooled down to temperatures close to absolute zero. The qubits are then controlled by microwave pulses. In these quantum chips, qubits are laid out on a single plane. This means there is limited connectivity between them. Each qubit can only be entangled with its direct neighbors. Another aspect of this technology is that the same physical qubits are reused for all quantum programs that run on the chip. This means qubits need to be reset to a reference state before they can be used again so that each program starts with a clean state [35]. These particularities of superconducting technology open up the quantum chip to attacks that are not possible on other technologies. For example an attacker could take advantage of unwanted interference between qubits either to inject noisy information or extract useful information [75]. This would be an attack based on crosstalk effects. Another form of attack on superconducting qubits could take advantage of the fact that the qubits are reused and the reset operation to the $|0\rangle$ state after each circuit run is critical for the correct operation of the system. If the reset operation is imperfect, residual information from the previous circuit could leak into the subsequent circuit. A malicious user could take advantage of this and either inject information into the circuit that runs after his [89] or read information from the previous circuit.

In trapped ion technology, qubits are made up of individual ions that are trapped in a vacuum chamber and manipulated using lasers. The qubits are grouped in ion traps in which all qubits connected to each other. This means that any qubit can be entangled with any other qubit. To perform entanglement between qubits in separate traps the ions are shuttled between them. The shuttle operation ads energy to the system and can lead to decreased fidelities on operations performed on all the qubits in the traps. Assuming the ions in the same trap can belong to different users, an attacker could take advantage of this and try to maximize shuttling such that a victim that owns qubits in the same traps will be affected [76].

4 Attack Vectors

4.1 Classical Attacks on Quantum Computing Software Stack

4.1.1 Supply Chain Attacks

Given that many open-source SDKs dedicated to quantum program development are actively evolving, they are typically installed within a Python virtual environment. This practice is adopted because these SDKs undergo frequent updates and modifications. To use an SDK the user must install it, which can be done using the following methods:

1. clone the repository that contains the SDK's code using a tool such as *Git*, if the SDK is public;
2. use the Pypi [57] platform to install the SDK using *pip* tool.

Pypi platform can be used by any user to upload Python packages. The uploaded package is not validated, therefore an attacker can upload malicious packages [84], which a victim can download later on. In our case, an attacker can implement and upload a malicious package that is supposed to be a plugin or a transpiler for an SDK. A user can install this package believing it to be a legitimate one, giving the attacker the possibility to collect sensitive data or even compromise the victim's computer. Attacks of this type are presented in [19], where the malicious package is used to collect data about the victim, execute commands on their machine or even track their activity.

4.1.2 Compromised Quantum User's Computer

This attack vector involves an adversary who is operating on the quantum user's computer and possesses the same privileges as the currently logged-in user. Consequently, the attacker benefits of read and write access to the source files of quantum SDKs. This means they can potentially manipulate or tamper with the source code, introducing malicious changes that could compromise the integrity and security of the quantum-related software on the user's machine. Alternatively, the attacker could attach to the processes running the quantum SDK, hooking certain functions of them and executing his own malicious code, tampering with the victim's quantum circuits.

4.1.3 Untrusted Transpilers

Today, there are several third-party transpilers, some very reputable like Pytket [59] or QBraid [61], and many others perhaps less known. Some of these software packages promise better performance in terms of the number of two-qubit gates and the depth of transpiled circuits compared to the compilation tools of standard quantum software toolkits such as Qiskit or Cirq. The use of untrusted third-party transpilers, might provide an attacker with the opportunity to steal circuits. Split compilation [78] and circuit obfuscation methods [87] have been proposed as protective measures against such attacks. Circuit obfuscation works by inserting dummy gates such as Controlled-X or Swap at strategically defined points in the circuit marked by barriers before the circuit is sent to a transpiler, and removing these gates either directly or by adding another Controlled-X or Swap gate at the known locations before the circuit is sent to a quantum computer. The use of barriers to mark the possible insertion points of dummy gates is an obvious limitation of this technique, even though this procedure is reported to increase the number of gates and the depth of the final transpiled circuit by only 5%. Another security issue with using such tools is the possibility of injecting Trojan attacks. Subsection 4.4.5 discusses how, in quantum computers shared by multiple tenants, crosstalk effects between qubits can be used to perform fault injection and even DOS attacks. Such observations are likely theoretical today, but it is possible that these attack vectors could become a legitimate security concern at some point in the future.

4.1.4 Plain-Text Authentication Tokens

Token based authentication protocol is used to verify the identity of a user that wants to connect to a cloud service. It replaces the authentication method based on user and password.

Some quantum providers use the token-based authentication. For user convenience reasons, quantum vendors implemented mechanisms to store such tokens, like in a:

1. predefined file-system location;
2. environment variable.

See Chapter 6.2.4 for a detailed description of the authentication mechanisms used by different quantum providers.

If the token is stored in plain-text, without any kind of encryption, in the absence of any protection mechanism, an attacker who has any type of access to the victim's machine can steal this token, without the need for privileged access rights. Alternatively, if the token is stored directly in the code, there is a high risk to be leaked when the code is uploaded to public repositories. The theft of an access token may lead to unauthorized access to critical and confidential information or services.

4.1.5 Man-in-the-Middle (MitM)

A MitM [7] attack consists in positioning the attacker between two entities that communicate over a network. This attack is common when the communication channel is not encrypted. The transmitted information between the user and the web service can be intercepted or altered.

In our case, when a user sends a quantum program to the cloud service, the attacker can tamper with the package that contains sensitive information. The collected information can constitute different forms of intellectual property or even credentials.

4.1.6 DNS / IP Spoofing

Domain Name System (DNS) is a protocol of the internet that provides a mechanism to resolve the human-readable name of a website to the corresponding IP address. DNS Spoofing attack aims to redirect the user web-request to a malicious IP address. The attackers can intercept the DNS requests that should resolve the IP address of a quantum hardware provider and sent back the IP address of a web-service controlled by them. In this way, every request sent by the user to a quantum provider will be redirected to that malicious address which can collect all the data sent by the user.

4.1.7 Man-in-the-Browser (MitB)

MiTB [30] attacks aim to infect a web-browser, with the intention of altering on-the-fly transactions made by the browser. The malware is installed as a plugin to the web-browser, thus being able to intercept the data that is sent with any request.

In the case of quantum providers, a MiTB attack can collect and change the user's circuits, credentials or any other sensitive information. An example of such an attack is described in Subsection 5.1.2.

4.1.8 Denial of Service (DoS)

A DoS attack aims to make a machine or a service inaccessible to its intended users. This attack involves flooding the target in the way that it triggers a crash or stops working properly. The cloud services have multiple points where a DoS attack may occur. One example is flooding the authentication APIs with request until it no longer responds to the users. This type of attack can cause material damage to quantum providers because their infrastructure no longer works for a period of time.

4.1.9 Untrusted Quantum Providers

If an attacker has compromised and controls the cloud services used to get access to quantum computers or is an insider attacker managing those services or the quantum computers themselves, he can access different information regarding quantum customers. Such information could be stored in the cloud, like user credentials, history of run circuits, obtained results etc., or could be about running jobs. Quantum circuits and their results could be considered intellectual property, therefore leaking or tampering with them may prejudice their owners.

4.1.10 Untrusted Quantum Users

From another perspective, considering the virtual attacker a quantum user and the victim a quantum provider, if the latter might want to keep the confidentiality of parts of their proprietary SDK run on quantum users' computers, the former might try to leak intellectual property. There could be, for instance, a special transpiler, whose design and functionality its owners want to keep confidential. In such a case, making it open-source is not an option anymore. Alternatively, providing the transpiler as a binary to be run on quantum users' computers still exposes it to reverse engineering.

A malicious quantum user, or a compromised one, as we noted above, could also tamper with the transpiled quantum circuits that are sent to be executed on the remote quantum computers. Such a circuit could be built with the purpose of tampering with the execution of other users' circuits, reverse engineering the quantum hardware or even trying to damage it. Next sections will detail about such kind of attacks.

4.2 Classical Attacks on QPUs

4.2.1 Attacking QPU Calibration Using the Pulse API

Superconducting qubits are characterized by a number of operating parameters. Some examples are the qubit resonance frequency which is given by the energy needed for the transition from $|0\rangle$ state to $|1\rangle$ state, the anharmonicity[1], qubit coherence times, cross-talk probabilities, gate and readout errors. Each qubit has slightly different parameters due to the fabrication process. Furthermore, these parameters drift in time due to environmental factors like temperature or magnetic fields whose values change slowly with time. Another source of variability is material defects propagating in the substrate of the superconducting circuit or even in the circuit itself. This can change parameters of the qubit materials like the dielectric constant [25].

Superconducting quantum computers are calibrated periodically to account for these variations. The calibration results are subsequently used by the quantum computer providers for building up pulse schedules that implement quantum

[1]The anharmonicity is given by the difference between the qubit resonance frequency and the energy difference between second and first excited state.

gates with maximum fidelity and minimum cross-talk effects. Ideally, these pulses do not have any effects other than changing the quantum state of the qubits. However, in real life this is only an approximation because real pulses also dump heat into the qubits [31]. The qubit dissipates energy [51] by coupling to various sources of noise like the readout resonator, equilibrium and non-equilibrium quasiparticles [16, 86], trapped vortices [96], two-level fluctuators [43] and other degrees of freedom in the qubit environment.

We speculate that activating some of these channels can be used to induce semi-permanent changes in a qubit like altering qubit calibration parameters, provided that off resonance microwave pulses with increased power and duration are applied to the qubits.

4.2.2 Side-Channel Attacks

Side-channel attacks use information extracted from computer hardware to infer information about the computation being run. In the context of quantum computation, a possible scenario is a rogue insider with access to the quantum computer enclosure using a device to intercept the radio frequency signals generated by the microwave pulses enacting the quantum gates. The insider attacker could rely on this information to reconstruct the circuits run by users. On superconducting architectures, the pulses executed for each qubit are tuned to the qubit's frequency which is different for each qubit. This makes it possible to distinguish pulses targeting different qubits. Quantum computer controllers' power usage might provide another attack surface that can be used to execute a similar exploit. Since circuits usually are being run in many shots, always in sequence, information collected from multiple runs can be consolidated to extract a cleaner signal.

A side channel attack could in principle be executed in the reverse direction as well, where the attacker might use a device to inject faults in the control or readout pulses generated by the quantum computer controllers as was suggested here: [100]. A thorough evaluation of various side-channel attacks relevant for NISQ computers' era can be found in [101].

4.2.3 Scheduler Attacks

Scheduler attacks are situations where a malicious entity on the quantum computer side allocates inferior quantum hardware resources to a client while the client user has no way of knowing what hardware resources have been assigned to him. Such an attack would result in the attacked side obtaining degraded or even incorrect results. In [52], quantum physically unclonable functions (QuPUF) are proposed as protection against such attacks. These work as a challenge/response mechanism by sending specially designed circuits to the quantum computer. The result of the execution of these circuits is influenced by the unique hardware characteristics of each quantum device, allowing a user to distinguish between different hardware devices.

4.3 Quantum Attacks on Classical Computers

While the available quantum hardware is not yet powerful enough, various algorithms have been already proposed and proven. When the hardware will become powerful enough in the future, all these algorithms can be leveraged by an attacker in order to challenge the nowadays cryptographic schemes. In this section we'll discuss some of them and include references for the reader.

4.3.1 Quantum Algorithms

- Simon's Algorithm, developed by Daniel Simon in 1997 [81], is a quantum algorithm that solves a specific problem exponentially faster than any known classical algorithm. It identifies a hidden binary string within a black box function, showcasing the potential of quantum computers to outperform classical ones for certain computational problems. Though Simon's Algorithm itself does not directly break cryptographic schemes, it laid important theoretical groundwork for subsequent algorithms like Shor's (listed next).

- Shor's Algorithm, introduced by Peter W. Shor in 1999 [80], represents a significant quantum computing breakthrough capable of factoring large integers. Shor's algorithm can solve these problems in polynomial time on a quantum computer, drastically reducing the complexity and time required to break cryptographic systems which use these methods, for example RSA.

- Grover's Algorithm, devised by Lov K. Grover in 1996 [29], offers a significant quantum computational advantage for searching unsorted datasets. Unlike classical algorithms, which require linear time to search through N items, Grover's algorithm can find a specific item in approximately square root of N steps, showcasing a quadratic speedup. This capability is particularly relevant to symmetric-key cryptography, where the security of algorithms like AES depends on the computational effort needed for brute-force attacks. Grover's Algorithm effectively halves the bit strength of symmetric keys, suggesting that key lengths may need to double to maintain equivalent security levels against quantum attacks.

4.4 Quantum Attacks on QPUs

4.4.1 The $|11..1\rangle$ State Initialization Attack

Understanding state preparation and measurement (SPAM) errors is essential for working effectively with a quantum computer. Most users take for granted that the qubits are initialized in a well-defined state before a circuit is executed, which is usually the $|0\rangle$ state. In practice, however, this is not always the case, and sometimes the actual state of certain qubits at the beginning of a computation is not $|0\rangle$. It is interesting to see what effect a bit flip has on a quantum algorithm. Most of the gates used in quantum computers today are imperfect, but small errors can be handled in different ways. Incoherent errors always creep in, but, on average, they cancel each other out, and their impact on the outcome of quantum computations is relatively suppressed. An important observation in this context is that incoherent errors are mainly caused by the interaction of the qubits with the environment, leading to the leaking of information into the environment, so the overall impact of these errors must be kept within certain limits. Otherwise, the computation is lost. Systematic, coherent errors, on the other hand, do not lead to a loss of coherence, but add up faster than incoherent errors. Coherent errors can be eliminated in considerable measure by correct qubit calibration and by careful tuning of the microwave pulses used to enact the quantum gates.

In contrast to the situation described above, in which the individual errors have small magnitudes, a qubit that is initialized in the state $|1\rangle$ instead of $|0\rangle$ can significantly influence the result of a quantum algorithm. Figure 4.1 shows the results of running the Grover algorithm on 5 qubits in three scenarios where (1) shows the results of running an ideal Grover algorithm, (2) shows the results of the same al-

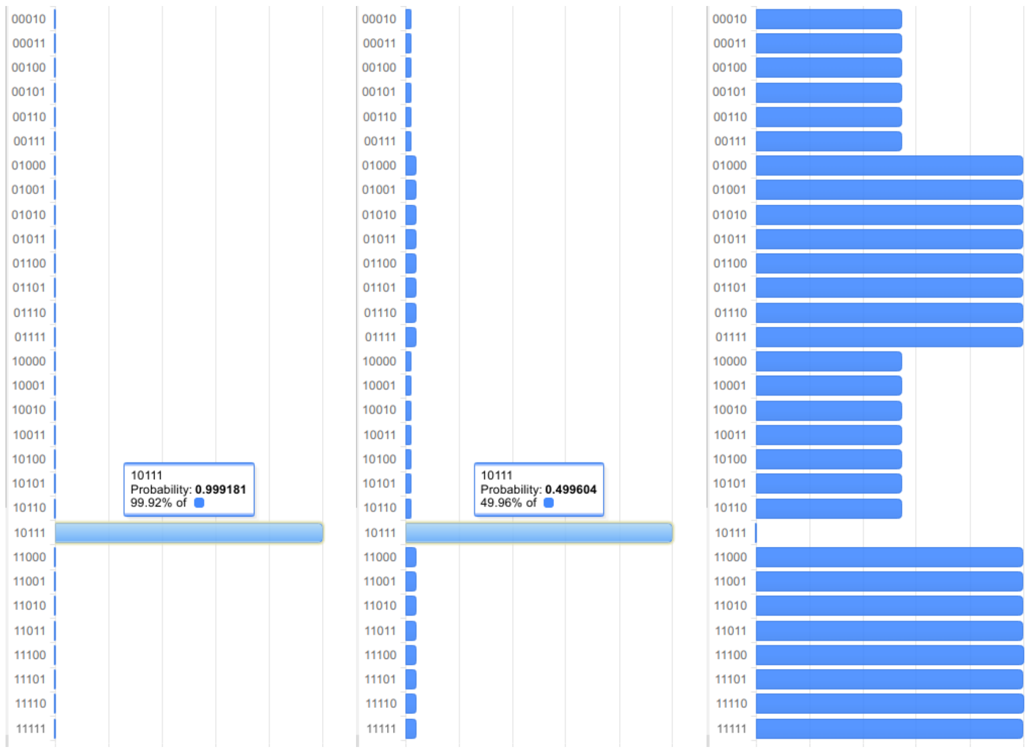


Figure 4.1: Results for Grover algorithm on 5 qubits in (1) the ideal case, (2) situation where a V gate is placed on second qubit before the Grover circuit and (3) the case where a X gate is placed on second qubit before the Grover circuit

gorithm where on the second qubit a V gate[2] is applied before applying Grover and (3) shows the results where an X gate is applied on second qubit before Grover, simulating the effect of an imperfect qubit reset error. In the ideal situation, the correct results are returned with 99% probability. In the second situation, the results are worse, but the state representing the correct result is returned with a sufficiently large probability (49%). This means that a user can run such a circuit and extract the right answer, given that he executes a slightly larger number of measurements. In the third case one can see that the computation is completely compromised.

Exploiting this phenomenon of failed qubit initialization, an attacker can attempt to compromise the results of a victim executing a job after him. The attacker must ensure that he leaves the quantum register in state $|11..1\rangle$ to maximize the likelihood of a qubit initialization failure. With a reasonable probability, the attacker can expect to influence only the first circuit executed after him. Since an average user runs thousands of shots, the impact of such an attack is limited. A very simple defense against such an attack would be to ignore the results of the first shot.

The ibm_kyoto quantum computer from IBM has 127 qubits. The estimated probability of at least one incorrect reset, given that in the previous run the state was $|11..1\rangle$, is extremely high. As quantum computers improve, the likelihood of an initialization error will likely decrease. However, the number of available qubits will also increase, making the overall probability that such an attack would be successful, large enough to be a legitimate concern.

4.4.2 Accessing Higher Energy States Attacks

The superconducting circuits used to build superconducting qubits have been nicknamed artificial atoms [97] because their energy spectra are qualitatively similar to the energy spectrum of an actual atom and, as with atoms, photons (in this case microwave photons) can be used to induce transitions between different energy levels. Since a quantum bit is, by definition, a physical system with two levels, implementing a quantum bit with superconducting circuits requires access to precisely two of these energy levels. Typically, the ground state and the first excited state are used for this purpose, with the ground state corresponding to state $|0\rangle$ and the first excited state corresponding to state $|1\rangle$. To enact a single qubit gate

[2]The V gate is the square root of the X gate.

such as the X-gate, which is needed to control transitions between the states $|0\rangle$ and $|1\rangle$, the qubit is irradiated with microwave radiation at the qubit resonance frequency. This is usually in the range of 4-8 GHz. The qubit frequency is the energy difference between the ground and first excited states. The microwave photons must trigger the transition of the qubit between the two lowest energy levels but must not excite higher energy states. In principle, this is possible due to the anharmonicity in the energy levels of the superconducting qubit. The magnitude of the anharmonicity is given by the difference between the energy required for the transition from the ground state to the first excited state and the energy needed for the transition between the first and the second excited state. This is usually in the range of 200–400 MHz [34]. While this situation is workable, it is not ideal because, while microwave pulses that activate the quantum gates can be engineered to target predominantly transitions between the ground state and the first excited state, they must be carefully designed to minimize the probability of accessing undesired higher energy states.

Pulses represent implementations of quantum gates as analog microwave signals. Although the pulse-level specifications are not unique to superconducting quantum bits, we will focus on this technology in this section.

A pulse schedule is a sequence of microwave pulses scheduled in time. The pulse specifies a microwave signal that oscillates at a particular frequency. In the case of pulses that activate individual qubit gates, the frequency is chosen to match the resonant frequency of the qubit. The phase of the microwave signal can be adjusted. For example, by varying the phase with $\pi/2$, a pulse implementing an X-gate can be transformed into a pulse implementing a Y-gate. The phase is also crucial because the Z-gates, are usually implemented exclusively in software by tracking and adjusting the phase changes [40]. The envelope describes the shape of the pulse, which changes more slowly. The total area under a pulse is directly related to the amount of rotation it performs. The optimal shape of the pulse is close to a Gaussian, but not identical [42, 28]. The smooth shape of the envelope is chosen to suppress coupling to higher harmonics and prevent the qubit from transitioning outside the computational sub-space.

An attacker could try to do the opposite and use a pulse that makes the transition to higher excited states probable. In principle, the second excited state can be excited directly by a pulse applied at the energy difference between the ground and second excited state. However, the frequency required for this is high and probably unavailable on most quantum platforms. Another method to access the second excited states is to start from state $|0\rangle$ apply a conventional X-gate first and then use the pulse API to apply a pulsed gate at the frequency corresponding to the energy difference between the first and second excited state. The frequency value can be taken from the calibration settings of the device or measured experimentally [14] by running certain circuits. The specific frequency required is less than the qubit resonant frequency by the amount of qubit anharmonicity and may or may not be available.

However, there is a more straightforward method to partially access the higher energy states that an attacker can easily exploit by performing pulses that couple to higher harmonics. One method is to execute a relatively long pulse at the qubit resonance frequency, having a rectangular shape. The effects of this attack are similar to the effects of the attack described in the previous section because it increases the probability that in the next shot some qubits are not initialized properly. Our experiments indicate that an attack using a rectangular long pulse is slightly more effective than the $|11..1\rangle$ state initialization attack, resulting in a significant probability that multiple qubits are initialized incorrectly. Other experiments where the second excited state is accessed directly are presented in Subsection 5.4.2. In principle, the attacker can expect to influence not only the first circuit run after him, but also subsequent shots. However, the likelihood for this to occur is exponentially suppressed. Again, a very simple defense against such an attack would be to ignore the results of the first few shots.

4.4.3 Readout Attacks in Multi-tenant Environments

In readout attacks, the attacker tries to find out the final state of a quantum circuit belonging to the victim. In other words, to steal the victim's measured result. If the attack is successful, this leaves the attacker with a value in binary format. Even if this value is the true result that the victim measured, this is not enough information to be useful. But if they can corroborate it with other information, obtained through other means, like what circuit was run, what was the input, what problem the victim is trying to solve, etc., then this value can be a very valuable piece of the puzzle.

There have been a couple of readout attacks studied so far, all on superconducting qubits. One such possible attack is based on a simultaneous multi-tenant scenario. This is the case where the QPU is shared by multiple users at the same time. This means different circuits are mapped to different qubits on the chip, and they can operate and be measured independent of other circuits running in parallel. In such a case, a readout attack was proposed by A. Ash-Saki and S. Ghosh where they use a qubit that is adjacent to the victim's qubits during measurement [75]. They take advantage of the fact that readout error probabilities are different for state $|1\rangle$ and $|0\rangle$ and that this probability correlates with the state of the adjacent qubits. In certain cases, they can infer the results of two adjacent qubits using a single attacker qubit.

The proposed solution that would fix this vulnerability is for the victim to apply a randomized set of qubit flips before measurement. An X gate would be applied, or not, on each of the output qubits right before measuring it. This operation can then be undone in the post-processing step. Since only the victim knows which qubits were flipped and which were not, the information obtained by the attacker becomes irrelevant.

4.4.4 Readout Attacks in Single-tenant Environments

Another type of readout attack that was studied takes advantage of an imperfect reset operation. This also applies only to superconducting qubits. As we mentioned previously, the qubits in a superconducting chip need to be reset after each run so the circuit that executes immediately after can start with all qubits in the reference $|0\rangle$ state. The reset operation is not always perfect in practice, and it is theorized that one can read the information leaked from the previous circuit run. In [41], the authors tried this with notable results. They added circuits that simply measure the allocated qubit and found that they can infer the state of the qubit in the previous run, even when the victim's qubit was prepared in a superposition. For this attack to work the malicious user must run his circuit immediately after the victim's circuit on every shot. As far as we know this is not the standard way of operation on current quantum devices.

There are some proposed solutions. One of them involves adding many reset operations or randomly varying the number of reset operations to make it more difficult for the attacker to create a model of the reset. Another solution is to detect attacker circuits before they are run on the QPU and flag them as potential threats.

In Subsection 5.4.1 we describe our own experiment for performing a readout attack on real-world quantum computing systems that are available today.

4.4.5 Cross-Talk Attacks

Crosstalk is a phenomenon where the application of control signals on one qubit causes unintended effects on another qubit. Uncontrolled residual couplings between qubits are a source of crosstalk errors in quantum computers. In the case of superconducting qubits, such errors are more likely to occur between neighboring qubits on the same chip, where, due to their proximity, it is difficult to operate one qubit without affecting the other to some degree. In general, crosstalk errors occur with varying strengths in all quantum computing platforms, regardless of technology, due to various physical effects [79]. For example, neighboring control lines can cause crosstalk between qubits. Crosstalk errors are systematic [83] and can be mitigated in various ways, such as by developing special pulse shapes and optimizing pulse scheduling [44, 26].

Using crosstalk effects to execute attacks on a quantum computer was first proposed in [3]. Taking advantage of the crosstalk effects, a malicious user could enact a fault injection attack. To carry out such an attack, two users must simultaneously share the same quantum computer. The attacker would try to induce crosstalk between his qubits and the qubits used by a potential victim, creating entanglement between the two circuits. Such an attack can even be upgraded to a denial-of-service (DOS) attack if the attacker, having achieved sufficient entanglement, measures his qubits and forces a collapse in the other user's qubits. Such a DOS attack is probabilistic and will not always work, but sometimes it will be successful.

While an attacker does not know in advance which qubits will be assigned to him, if he submits jobs with small enough circuits, he will increase the probability that the remaining idle qubits will be assigned to another user. An example of a circuit implementing such an attack is shown in Figure 4.2. By applying a Controlled-X gate, the attacker can create entanglement between his qubits and the victim's. Of course, applying a single gate has negligible effects, but the repeated application of the same gate can significantly increase the effectiveness of such an attack.

Instead of Controlled-X gates, an attacker could also use simple X gates. By placing an X gate on a qubit next to the victim's circuit, the attacker will trigger, with some small probability, a bit flip error on an adjacent qubit. The attack can then be amplified by using more X gates. For example, the attacker could run a circuit on one or two qubits containing a long block of X gates.

Crosstalk error rates are not typically reported by hardware vendors in their device characterization, but can be measured by running specially designed cir-

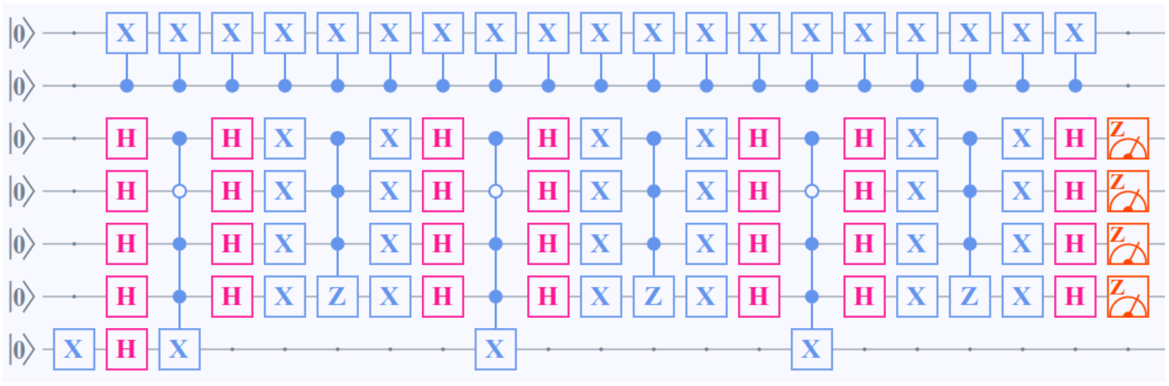


Figure 4.2: An example of a crosstalk attack on a circuit implementing the Grover search algorithm on 4 qubits. The bottom 5 qubits belong to the victim while the top two qubits are used to mount an attack. In order for this attack to work, there must be some non-negligible crosstalk effects between the second and the third qubit

cuits [79]. As quantum computers improve, the rate of crosstalk errors will likely decrease, making such an attack less effective. However, as the quantum devices improve, longer circuits will be able to run. As we already pointed out, this effect can be enhanced by the repeated application of a Controlled-X or X gate on the same qubits.

For a quantum device that supports pulse control [83], this attack vector can be substantially strengthened. One of the current methods to implement two-qubit gates for superconducting qubits is to apply a microwave pulse on one qubit at the resonance frequency of the other. This is the so-called cross resonance gate [35]. An attacker can use this principle by applying a pulse to one of his qubits at the resonance frequency of a qubit owned by another user. Our experiments performed on the quantum devices from IBM indicate that fairly long pulses can be applied, greatly enhancing such an attack.

A simple defense is to place a layer of unused qubits between two circuits allocated on the same chip. This will lower the crosstalk effects since the crosstalk rates are higher between neighboring qubits. However, this strategy is not perfect since crosstalk effects are not restricted to adjacent qubits. A more elaborate solution was proposed in [24] in the form of a quantum antivirus that would scan circuits submitted by the user, making it possible to identify potentially malicious circuits. The way this could work is to identify patterns of redundant gates replicated many times in a larger circuit and flag those circuits. In the same spirit, most of these patterns can be identified by a transpiler that ignores barrier and delay instructions and mergers all gates that can be merged logically using simple rules similar the rules implemented by the antivirus. If the transpiled circuit has a significantly smaller number of gates than the original circuit this could raise a flag prompting that this circuit is doing more than one thing (which should be performing a computation), and perhaps it should be run in isolation.

Analyzing a circuit after it has been allocated to a quantum chip and estimating the amount of crosstalk it induces on neighboring qubits is possible in principle if one knows the crosstalk rates between qubits. Again, such an analysis could be used to flag potentially dangerous circuits. Measuring crosstalk rates between two qubits is possible in principle but when the number of qubits increases the number of qubit pairs increases exponentially, so some heuristics should be used to reduce the number of qubit pairs that need to be considered.

4.4.6 Shuttle Exploiting in Trapped-Ions Quantum Computers

The vulnerabilities of trapped-ion QPUs have been studied significantly less than those of superconducting QPUs. However, a recent study [76] has demonstrated that the shuttling of ions in a trapped-ion QPU can be exploited to execute a fault injection attack. The shuttling of ions is a process wherein ions are moved from one trap to another while not altering the quantum state encoded in the qubits. An attacker can exploit this process to launch a fault injection attack on the qubits assigned to another user.

In trapped-ion systems, atoms like Yb (Ytterbium) or Ca (Calcium) are ionized and confined between electrodes using electromagnetic fields, hence the name "trapped-ion" quantum computer. States $|0\rangle$ and $|1\rangle$ are encoded into the internal states of the ions. The system features traps interconnected by a shuttle path that facilitates the shuttling of ions from one trap to another when necessary. Each trap has a capacity it can accommodate. Quantum gate operations on qubits/ions are executed using laser pulses. The attack discussed in this section relies on the fact that 2-qubit entangling gates can only be performed on ions located in the same trap. Therefore, executing gates between ions located in separate traps requires shuttling ions to co-locate them.

The attack model proposed in [76] is based on the observation that repeated shuttling operations add energy to the ions and decrease the gate operation fidelity on them. If the QPU is shared and multiple users are running circuits in parallel, an attacker can attempt to launch a fault injection attack by shuttling ions to the traps of the victim's qubits. The attacker can exploit the shuttling process to degrade the fidelity of gates executed on the victim's qubits.

This approach doesn't rely on direct manipulation of the qubits' states but rather on the physical process specific to trapped-ions quantum computing. Two methodologies are possible for generating programs that can launch such attacks. A systematic attack can utilize knowledge of the system's architectural policies to craft a targeted attack, exploiting specific weaknesses or inefficiencies. A random attack does not require prior system knowledge, trading off some level of attack potency for ease of execution.

One countermeasure to prevent this exploit involves the victim adding a sufficient number of dummy qubits to their program to fully occupy a trap, thus preventing an adversary's qubit from sharing the same trap and averting shuttle-induced fidelity degradation. The user applies virtual-Z gates to these dummy qubits to ensure the compiler allocates them without affecting the actual program since these gates have perfect fidelity and require no physical operation. However, this strategy introduces a trade-off between security and cost, as using more qubits could lead to higher charges from the quantum cloud provider.

Cloud providers can enforce a maximum shuttle limit for programs to prevent shuttle-exploiting attacks. If a program exceeds the maximum shuttle count, it can be scheduled to run in single-programming mode, essentially isolating it from other programs and eliminating the risk of affecting the fidelity of those programs. While this approach may reduce throughput due to the switch between multi-programming and single-programming modes, the loss can be offset by charging more for programs that require a high number of shuttles. This measure ensures that high-shuttle programs do not impact others, and the cloud provider does not incur losses.

5 Research, Analysis and Experiments

5.1 Classical Attacks on Quantum Computing Software Stack

In this section we studied attacks on a quantum computing software stack through a compromised user system. We developed a proof-of-concept (PoC) attack in code, which is accessible at [53]. This PoC demonstrates two basic classical attack vectors identified in the quantum programming ecosystem. In these scenarios, we assume the attacker is running on the local machine of the user, with the same privileges that the user has. Using this PoC, the attacker steals the user's circuits, its authentication token and takes advantages of the victim's resources like his quantum cloud credits. The scenarios we tested are detailed in the subsequent two subsections.

5.1.1 Attacking the API Authentication Tokens

In this scenario, we simulate an attacker who is already operating silently on the victim's machine, searching for different tokens in various predefined locations. An auth token is a unique identifier that allows a user or an application to access a resource without prompting for login credentials every time. In the quantum ecosystem, this token is a gateway to cloud infrastructure and user account, used to provide access to quantum computers. The main flow is illustrated in Figure 5.1.

The Python module responsible for token discovery is implemented in the *token_discovery.py* file, which is available in our public repository. According to our experiments, these authentication tokens are stored either directly on the disk or in the environment variables, as follows:

- *IBM Qiskit*

  - Saved on disk in a JSON file in "`default-ibm-quantum/token`" key. The path to the JSON file is:
    * *Windows*: `\\users\\<username>\\.qiskit\qiskit-ibm.json`
    * *Linux*: `$HOME/.qiskit/\qiskit-ibm.json`
  - Saved in environment variables:
    * `QISKIT_IBM_TOKEN`
    * `QISKIT_IBM_INSTANCE`
    * `QISKIT_IBM_CHANNEL`
- *IonQ*. Saved in environment variables:
  - `IONQ_API_KEY`
  - `IONQ_API_TOKEN`
  - `QISKIT_IONQ_API_TOKEN`

The malicious program is designed to search in predefined locations for potential access tokens. Upon discovering such tokens, they are leaked to a remote attacker' site, simulating a Command and Control (C2) server, represented by the *quantum _attacker_c2.py* module. In this scenario, the attacker attempts to steal

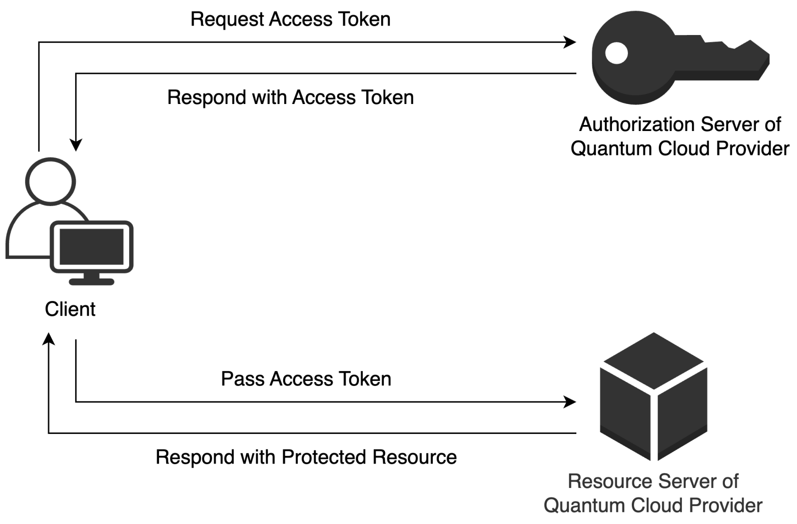

Request Access Token
Respond with Access Token
Authorization Server of Quantum Cloud Provider
Client
Pass Access Token
Respond with Protected Resource
Resource Server of Quantum Cloud Provider

Figure 5.1: Auth Token - Basic Flow

tokens and use them to access the victim's cloud account. For our PoC, we attempt getting from the victim's account the list of previously scheduled quantum programs (i.e. quantum jobs), to determine if we can access the victim's private resources. Specifically targeting the *IBM Qiskit* platform, the discovery process is straightforward. We utilize the *qiskit_ibm_provider* package, which provides APIs to query job history, and simply make the necessary API calls.

```
provider = qiskit_ibm_provider.IBMProvider(token=stolen_token)
jobs = provider.jobs()
for job in jobs:
    job_data = provider.retrieve_job(job.job_id())
    for circuit in job_data.circuits():
        circuit.draw(output="text")
```

Additional effort is required for IonQ, as there is no documented API to query a list of previously ran jobs. However, by examining the IonQ website, we discover that we can access the job history page. On the browser console, we observe a GET request to the following URL: `https://api.ionq.co/v0.3/jobs`. This allows us to manually craft a request to access the job history. We discovered private APIs (*make_path* and *_get_with_retry*) in *ionq_client.py* file from the *qiskit-ionq* package. Using these two APIs we manage to build the request path and authenticate with a valid token.

```
provider = qiskit_ionq.IonQProvider(token=stolen_token)
backend = provider.get_backend('ionq_simulator')
client = backend.client

req_path = client.make_path("jobs")
jobs = client._get_with_retry(req_path,
                              headers=client.api_headers).json()
```

To obtain the details of the circuits that have been run as part of the job, we repeat the same process that we used to discover the aforementioned GET request, and inspect the browser console again. In doing so, we notice another GET request to `https://api.ionq.co/v0.3/jobs/<job_id>/program`.

```
for job in jobs["jobs"]:
    job_id = job["id"]
```

```
req_path = client.make_path("jobs", job_id, "program")
program = client._get_with_retry(req_path,
                                 headers=client.api_headers)
print(json.dumps(program.json(), indent=4))
```

If the providers do not have any safeguards in place to protect against token leakage, once the attacker obtains the token, it can be used freely and could potentially cause additional costs for the victim.

A lesson learned from this is to ensure to adhere to best practices. Avoid storing sensitive data in plain text, particularly on the disk or, even more critically, within the program itself, as this can lead to accidental inclusion in a public repository. Instead, opt for storing such data in encrypted configuration files. Your program can decrypt these files when needed, providing an additional layer of security. In the event that your private token is accidentally exposed on a public repository, you should promptly revoke it through the cloud platform.

5.1.2 Quantum Circuit Hidden Alteration

An important resource needed to run quantum programs is the framework that helps the user designing, writing, and sending the circuit to the quantum cloud for execution. Most of these frameworks are open-source, thus an attacker can identify vulnerabilities more easily. For the PoC presented below we use IBM's open-source Qiskit framework. It is written in the Python programming language, therefore, it is provided as an installable Python package. Like other Python packages, Qiskit package could be installed in per-user package installation directory or in a virtual environment as recommended by Qiskit documentation [1]. Consequently, an attacker could access the package installation directory of a Qiskit user, supposing he succeeds running code with the same privileges as his victim. In our PoC we will consider the attack could run malicious code in one of the victim user's processes.

Considering the fact that anyone has access to the source code of the framework, one can observe that the user's circuits go through the function *execute_function.py* from the *execute* module. Thus, an attacker that tampers with this function can intercept and alter victim's circuits, and even inject his own circuits to be executed. In order for an attacker to replace or alter the file mentioned above, the

[1] `https://docs.quantum.ibm.com/start/install`

installation paths of the python packages must be discovered. To achieve this, all running python processes are monitored, as well as those that are going to be started while the malicious PoC process is running. This way, one can determine the location of the python packages, based on the path of the python executable and environmental variables. Once the location of the named packages is discovered, the original target package file is saved to be restored when needed, and the malicious process replaces it on the disk. Once the *execute_function.py* file is patched, every time the user submits a circuit to be executed, the attacker intercepts the circuit and sends it to a remote site, simulating a C2 server, which is represented by *quantum_attacker_c2.py* module. The corrupted package also gives the attacker the possibility to add other circuits to be executed. In this case, they are loaded from a predetermined directory. Adding circuits to be executed can represent an issue because the user might pay for the execution time on the IBM Quantum Platform, and the attacker would run his quantum circuits using the victim's paid resources. The malicious process runs for an indefinite time until it is stopped, and then the altered files are restored to their originals to leave no traces. All mentioned steps are implemented in the *patcher.py* module from the public repository.

On IBM Quantum Platform the user can inspect the state of the submitted circuits. If the submitted circuits are altered or new ones are added, this can be easily identified by the user through this platform. To hide these alterations from the user, a browser plugin can be implemented. In our case, we implement a plugin for the Firefox browser, so that when the web page is loaded, its content is altered in favor of the attacker, thus displaying only the user's original circuits. This plugin is executed only when the web page related to IBM platforms is accessed.

A solution to this problem is to provide a method to increase the security of the packets that are used by Qiskit. For this scenario, a security solution should be configured to monitor the Qiskit packages (i.e. files) and validate their integrity, so that if they are tried to be altered by an attacker, such an attempt could be blocked or the user warned of the ongoing attack.

Reader should note that this type of source-code alteration attack could be done in other SDK environments for other types of computing, not unique to Qiskit or quantum computing in particular.

5.2 Classical Attacks on QPUs

5.2.1 Attacking QPU Calibration Using the Pulse API

We have taken a brief look at several quantum devices from IBM that are freely available and provide pulse level controls like the 127-qubit device ibm_kyoto. Pulses are microwave signals characterized by an amplitude, a phase and a shape. The effect of a pulse when applied to a single qubit is to implement a rotation of the qubit state on the Bloch sphere [45] around an axis in the X-Y plane whose direction is controlled by the phase of the pulse. The angle of the rotation is proportional to the area under the pulse envelope. Our experiments indicate that the frequencies available on IBM Eagle processors range from roughly 10% less than the qubit resonance frequency to frequencies as large as 50% higher than the qubit resonance frequency. There is a limit on the maximum amplitude of the pulse. The overall length of a pulse can be much larger that the length needed for a typical implementation of an X or a Y gate. Our experiments indicate that pulses as long as 64 μs can be used to implement a single gate and 1400 such gates can be stringed together on each of the 127 qubits. Such a circuit is valid and runs without errors on ibm_kyoto. One single shot will have a runtime of 0.045 seconds but the number of shots for each job can be as large as 20000 which seems to indicate that pulses can be applied to the qubits in a QPU for relatively long times.

According to [32] spurious resonances in the photon loss spectrum in superconducting qubits may indicate couplings of the qubit with environmental degrees of freedom. We have executed a frequency sweep experiment on one qubit from ibm_osaka whose results are shown in Figure 5.2. The first peak on the left indicates the qubit resonance frequency. We do not understand the physical explanation for the second peak shown on the right, but it may be due to coupling to the readout resonator.

The IBM quantum devices are being calibrated once every several 2 - 4 hours. We considered that trying to run pulse programs with the purpose of affecting these calibration parameters would be a violation of user license terms, and we did not perform further experiments.

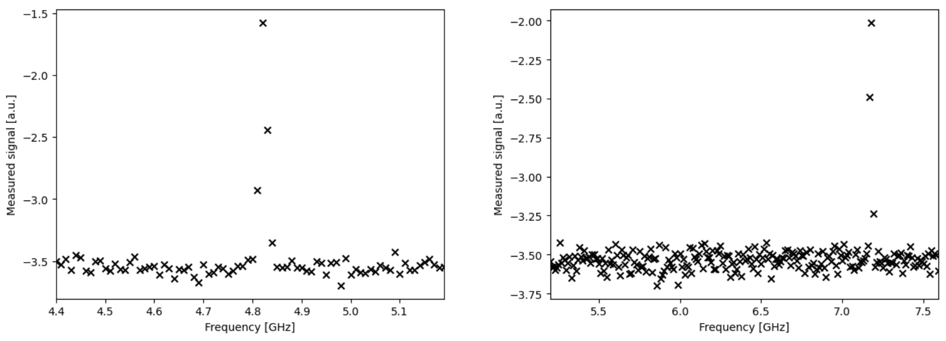


Figure 5.2: Frequency sweep results for one qubit from ibm_osaka in the 4.4 - 5.2 GHz range (left) and the 5.2 - 7.6 GHz range (right)

5.3 Quantum Attacks on Classical Computers

In the last years at Bitdefender we have strengthened our expertise in lattice-based cryptography, which is one of the most promising post-quantum solutions. We have developed standard cryptographic schemes [4] whose security is based on the hardness of new lattice problems [74] and lattice-based schemes with advanced functionalities and security requirements [36, 37, 1, 38], but also tested other such schemes in practice [90]. We have also worked on the mathematical hardness foundations of lattice-based solutions, by showing that the underlying problems of some NIST proposals are related [73, 11, 10]. Our results have been recognized and published at top crypto conferences and our researchers have been invited to talk about their work all over the world. We strongly believe that quantum computers will become a reality soon, and we are putting all our efforts into getting ready, both from a quantum and a post-quantum perspective.

5.4 Quantum Attacks on QPUs

5.4.1 Experiments on Qubit Reset Attacks

Almost all research about quantum attacks on QPUs published until the time of writing assumes some kind of multi-tenant environment [77, 23]. Our investigation in this section is centered around attacks on quantum systems that are possible with the technology available today where we do not assume multi-tenancy. We specifically focus on superconducting qubits.

In a readout attack, the attacker attempts to recover the results of a program executed by the victim without interfering directly with the victim's circuit. Here, the focus is on the last state of the qubits and the measurement operation. The goal is to let the victim perform the computation and then steal the end result.

In the superconducting technology qubits are reused and must be reset between consecutive shots of a circuit and between different circuits. The reset operation to the $|0\rangle$ state after each circuit run is critical for the correct operation of the system. If the reset operation is imperfect, residual information from the previous circuit could leak into the subsequent circuit. We explore the idea that the attacker would measure the qubit immediately after the victim runs his circuit and performs his measurements. The attacker assumes that the reset operation is not perfect and that information measured by the victim is not completely erased.

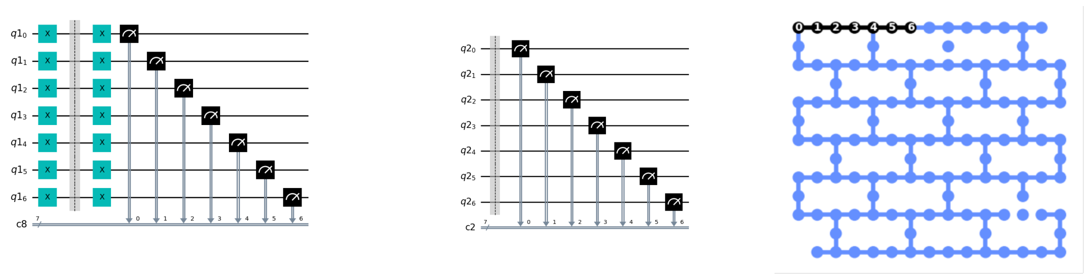
In our experiment, we aimed to test the effectiveness of the reset operation on ibm_osaka. We targeted the first seven qubits of the chip. We imagined a victim who runs a 7-qubit quantum circuit that ends with measurements on each qubit. The attacker has no prior knowledge about the victim's prepared state or circuit. He tries to recover the state measured by the victim by simply measuring the same qubits. In our attack model we assume that the victim runs their circuit multiple times and the attacker has the ability to consistently run a measurement circuit immediately after each shot of the victim's circuit.

IBM quantum chips are accessible through a cloud infrastructure. A user has the ability to submit a job of one or more circuits to the cloud, where it is queued for execution. Each job contains a set of circuits and a shot count. The circuits will be executed the number of times specified by the shot count. The results can be returned aggregated per result pattern or separately for each shot. If the user submits a list of circuits, then on each shot, all circuits will be executed. This means that on the first shot, the QPU runs all circuits, then moves on to the next shot where it runs all circuits again, and so on.

To perform our experiment, we needed to ensure the attacker's circuit, containing only measurements, is always run after the victim's circuit. For this we ran jobs with a list containing the victim and attacker circuits. This should have the desired outcome of the two circuits running in sequence for each shot and not be interrupted by other circuits. We have to note here that this requires a new assumption. We assume that the reset operation performed between separate circuits in the same job is the same as the one performed between jobs. For mapping our circuits on the physical devices, we performed transpilation and checked that the mapping was performed as expected for each circuit. This is important because we must be sure that the physical qubits measured by the attacker's circuit are the same ones prepared and measured by the victim. We also made sure that the qubit ordering was the same.

The backend we used, ibm_osaka, is a superconducting chip that has 127 physical qubits. The IBM system reports the last calibration time and some error parameters for each physical qubit. Of interest for our experiments is the probability of measuring 1 if the measured qubit is in state $|0\rangle$. We note this property for the first 7 qubits of the chip (qubits 0 to 6).

We start with a control setup where the victim's circuit ends in the $|0\rangle$ state for each qubit. To prepare this circuit, we apply an X gate to each qubit to flip all of them to state $|1\rangle$. We then apply a barrier to prevent the transpiler built into Qiskit from optimizing away the initial X gates. Then we apply another X gate on each qubit to bring them back to state $|0\rangle$. We apply another barrier, followed by measurement gates on all qubits. As a side note, the barrier has no role in circuit execution; it is used only at transpilation time and for visualization. The attacker's circuit consists only of measurement gates on all qubits.



(a) (b) (c)

Figure 5.3: Control experiment design (created using IBM Quantum) (a) Victim's circuit (b) Attacker's circuit (c) IBM Osaka backend with the qubits used highlighted

When we transpile the circuits we use a coupling map to ensure our circuit is mapped to qubits 0 to 6 on the chip. After transpilation, we wrap the two circuits in a list and submit them for execution, setting 19,000 shots. The backend will run the pair of circuits (the victim circuit followed by the attacker circuit) 19,000 times. On an ideal quantum computer, the result of executing a victim circuit should match the prepared '0000000' state in 100% of times. On a real QPU, we expect to see some results that differ from the prepared state. We go through the results to find all shots where the measured state for the victim was not ’0000000’. In these cases, we discard both the victim and attacker results. We then go through the remaining results for the attacker and count how many times each qubit was measured as ’1’. We compute the probability of measuring ’1’ for each qubit. Likewise, we perform this procedure 15 times.

In this control setup, there is no information in the victim’s circuit since the prepared state is the ground state. We expect the attacker’s probability of measuring ’1’ to match the probability of measuring ’1’ when the state is $|0\rangle$, as reported by IBM Q, for each qubit. We find this to be true within the range of shot noise as shown in Figure 5.8.

In the next phase of our experiment, we prepare the victim’s circuit such that it ends in the $|1\rangle$ state for each qubit. We do this by applying an X gate to each qubit followed by a barrier and measurement gates. In this case, the victim’s expected measured state is ’1111111’. Just like before, we follow this with an attacker circuit consisting of only measurement gates. We transpile, keeping the same coupling map, and submit the pair for execution 19,000 times. We filter out all results where the victim’s measured state was not ’1111111’. We go through the remaining results for the attacker’s measurements and count how many times each qubit was measured as ’1’. We compute the probability of measuring ’1’ for each qubit. Again, we perform this entire execution 15 times.

In this case, after the victim’s circuit ran, the qubits were left in an excited state. There is a possibility that the reset was imperfect and that some residual information was left in the qubits for the attacker to measure. We want to see how the attacker’s probability of measuring ’1’ in this case compares to the probability of measuring ’1’ when the victim’s state is $|0\rangle$ for each qubit. We plot the results in Figure 5.8. The figure shows the probability of the attacker measuring ’1’ for each qubit. For each qubit, we show 30 probability points. The 15 points in blue represent the probability of the attacker measuring ’1’ when the victim measured ’0’ for that qubit. The 15 points in red represent the probability of the attacker measuring ’1’ when the victim measured ’1’.

We can see that the reset operation efficiency is different for each qubit. We can also see that it is not correlated with the measurement error per qubit. For qubit 6, there is a consistently higher probability of the attacker measuring ’1’ when the victim measured ’1’ versus when the victim measured ’0’. This means that the attacker has a high chance of discriminating between the two possible states of

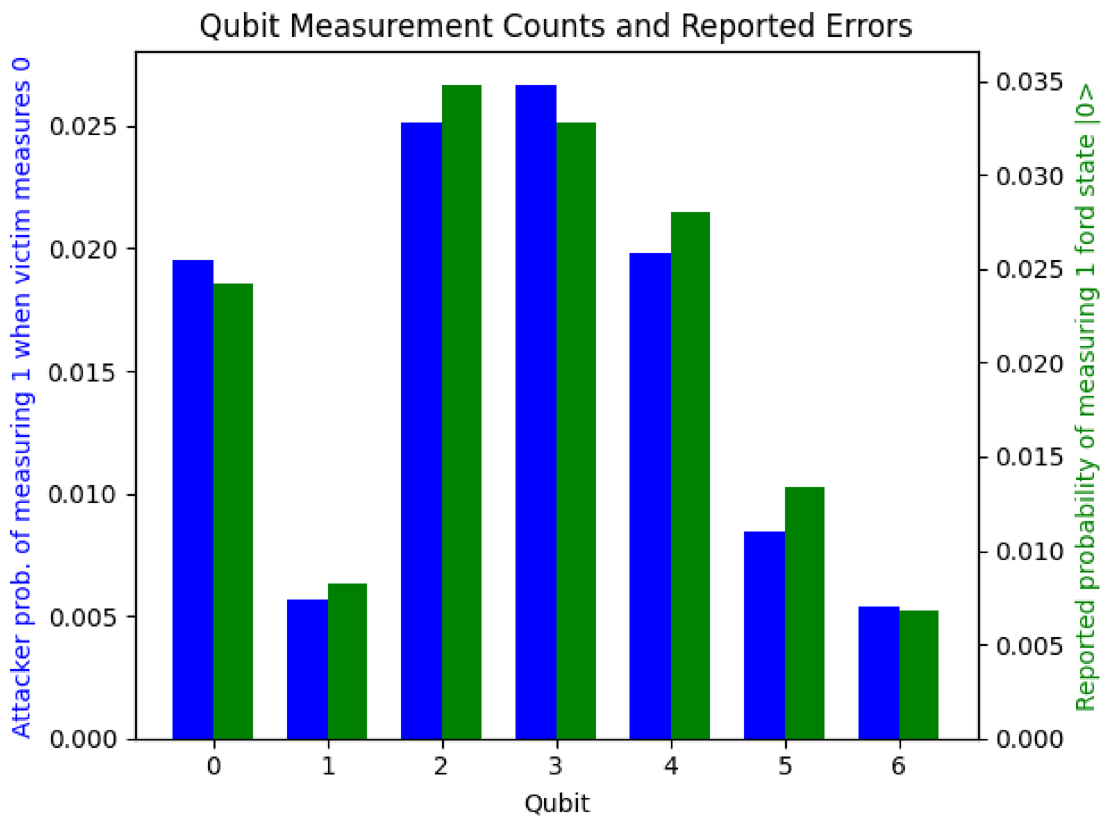
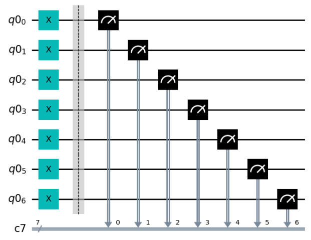
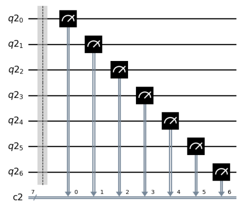


Figure 5.4

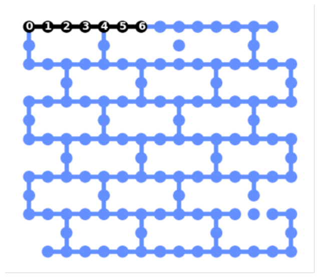
Figure 5.5: Probability of measuring ’1’ when state is $|0\rangle$, measured vs reported by IBM (created using IBM Quantum)



(a)



(b)



(c)

Figure 5.6: Information leak experiment (created using IBM Quantum) (a) Victim's circuit (b) Attacker's circuit (c) IBM Osaka backend with the qubits used highlighted

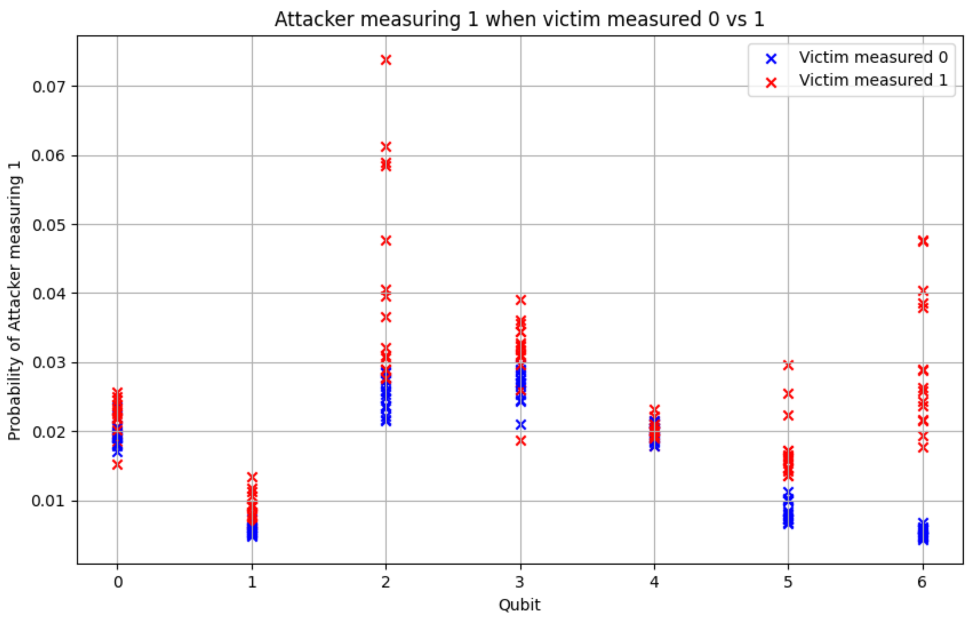


Figure 5.7

Figure 5.8: Attacker measuring '1' when victim measured '0' vs '1' (created using IBM Quantum)

the victim’s measurement. For qubit 2, the probability of success for the attacker is much lower since the two cases are less distinguishable. We can also note here that, in all cases, the probability of the attacker measuring ’1’ increased on average when the victim’s state was ’1’.

Based on these results, we try to create a model that can predict the victim’s state for each qubit. We repeat the above experiments, preparing the victim’s measured state in different patterns. We then again measure the attacker’s probability of getting ’1’ for each qubit. Furthermore, we set a threshold for each qubit. If the attacker’s probability of ’1’ is higher than the threshold, we assume the victim measured ’1’ for that qubit. If not, we assume the victim measured ’0’. The threshold was set as the highest blue point in the figure Figure 5.8 for each qubit.

We used 56 different patterns with an equal number of ’0s’ and ’1s’ for each qubit. The accuracy of the model is plotted in the table Table 5.1 Note that a random guess would yield an accuracy of 50%.

Qubit 0	Qubit 1	Qubit 2	Qubit 3	Qubit 4	Qubit 5	Qubit 6
57%	84%	64%	75%	66%	91%	100%

Table 5.1: Accuracy of the model for each qubit

As stated before we assumed that the reset operation performed between separate circuits in the same job is the same as the one performed between jobs. Though, after reviewing our paper, IBM stated that ‘reset’ function between jobs differs from the ‘reset’ function within jobs. The ‘reset’ function between jobs on IBM quantum systems is longer, with qubits becoming fully thermalized, making leakage of information between jobs impossible”. We think that further research is needed to clarify in what conditions this kind of attacks could still be performed.

5.4.2 Fault Injection Attacks

Having seen that information can indeed leak from one circuit to the next, there is another type of attack that can be performed on superconducting qubits. Each user that runs a circuit assumes that prior to the circuit beginning to execute, the qubits are in the ground state. If this assumption is incorrect, the results of running the circuit will be compromised. By leaving the qubits in an excited state, an attacker can influence the result of the next circuit. This would be a fault injection type of attack. Since we observed that, for all qubits, there is an increase in the probability of measuring ’1’ when the previous circuit ended in an excited state, this kind of attack is possible. For a discussion of the relevance and power of this scheme see Subsection 4.4.1.

We present here a new experiment similar to the one presented in the previous section, but where the roles are reversed. Here we envision an attacker who runs a circuit immediately *before* the victim’s circuit. The goal of the attacker is no longer to read the victim’s results, but rather to influence the state of the qubits at the start of the victim’s execution. To do this he will prepare his qubits in an excited state. If the attack is successful, the victim’s circuit will start execution with a set of qubits that are not all in the $|0\rangle$ state, and consequently, the results of running the circuit will be affected or in some cases completely compromised.

Similar to our previous experiment, we will use the first 7 qubits of ibm_osaka. We prepare two circuits that we bundle-up in a list and run them for a total of 19000 shots. This time the attacker’s circuit will run first, followed by the victim’s circuit in every shot. To maximize the probability of leaving the qubits in an excited state, we take advantage of the pulse library in Qiskit. We will create our own custom pulse that will excite the qubit to the second excited state. For compactness of notation we will label this as state $|2\rangle$.[2] We omit the fine-tuning of the frequency and amplitude for each qubit and calculate the expected frequency for the pulses from the backend properties reported by IBM. We use a generic amplitude for all qubits. An example of the pulse for one qubit is shown in Figure 5.9. We prepare a custom pulse for each qubit. We first apply an X gate to all qubits to bring them to state $|1\rangle$, then we apply the custom pulses to transition each qubit to the second excited state. After that, we added a barrier and measurement gates. This last step, of adding measurement gates, is not necessary and does not seem to influence the results significantly.

In order to find what is the state of the qubits when the victim’s circuit starts execution, the victim’s circuit consists of measurements gates placed each qubit. The attacker’s circuit, victim’s circuit, and layout of the qubits used on the backend are shown in Figure 5.10.

As in the previous section, we repeated the experiment 15 times. We then go through the results for the victim’s measurements and count how many times each qubit was measured as ’1’ right at the start of the circuit execution, and we calculate the probability of measuring ’1’ for each qubit.

[2] The reader should not confuse this with the more common two-qubit state $|10\rangle$ sometimes labeled in the same way

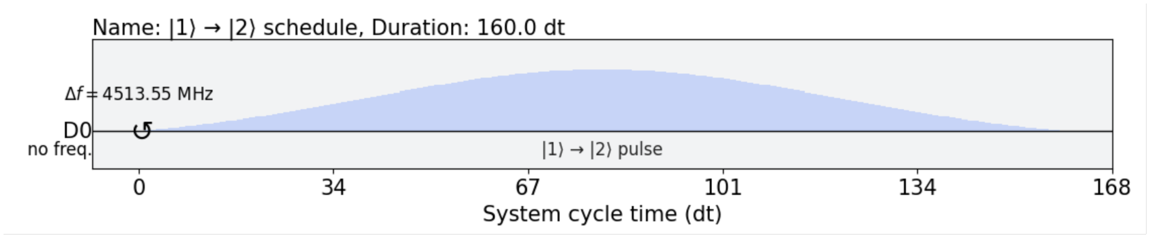


Figure 5.9: Custom pulse to excite a qubit from state $|1\rangle$ to the second excited state $|2\rangle$ (created using IBM Quantum)

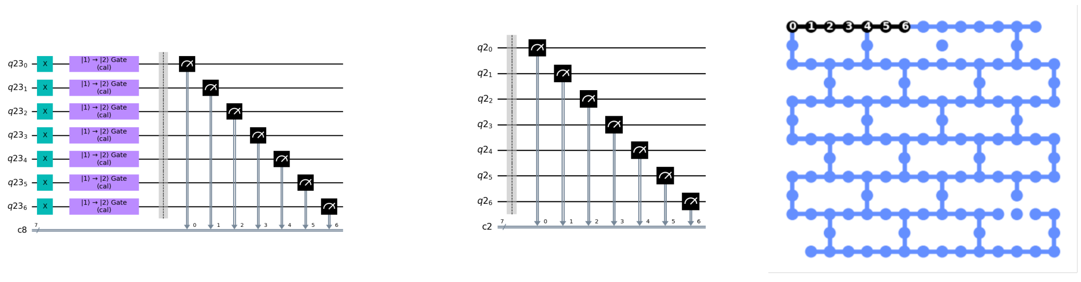


Figure 5.10: Fault injection experiment (created using IBM Quantum) (a) Attacker's circuit (b) Victim's circuit (c) IBM Osaka backend with used qubits highlighted

The results of the victim's measurements are shown in Figure 5.11. In blue and red are the data points obtained from the experiments presented in the previous section. They show how the qubit's starting state is affected when the previous run has prepared them in state $|0\rangle$ and $|1\rangle$ respectively. The green points show the probability of measuring '1' when the previous run has prepared the qubits in the second excited state. From analyzing the results, we can conclude that the probability of the circuit starting in a state different from $|0\rangle$ is increased dramatically when the attacker has prepared the qubits in the second excited state. For all qubits, the probability of starting in state $|1\rangle$ is relatively close to 50% which represents a random chance.

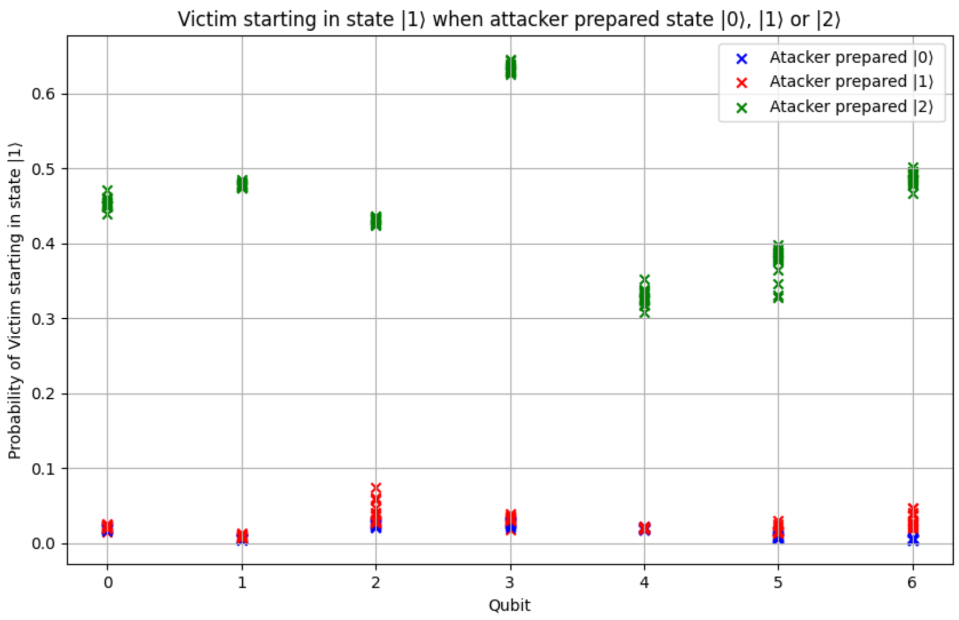


Figure 5.11: Probability of victim qubits starting in state $|1\rangle$ when the attacker has prepared the qubits in state $|0\rangle$, $|1\rangle$, or the second excited state $|2\rangle$ (created using IBM Quantum)

Having performed these experiments on seven qubits, we can scale up and see if the behavior we found generalizes to entire quantum chips. We ran the same experiments on all 127 qubits of IBM_Osaka and on all the qubits of IBM_Sherbrooke. The results are shown in Figures Figure 5.12 and Figure 5.13, respectively. From these two figures, we can draw the same conclusions we drew from our initial seven-qubit experiments: under the stated assumptions, information can leak from one execution to the next, and this is a vulnerability that can be exploited by an attacker either to read the victim's results or to influence the victim's executions. Moreover, this behavior is consistent across many qubits and across different chips. This indicates that more attention must be given to vulnerabilities that can arise from the design and manufacturing of quantum chips.

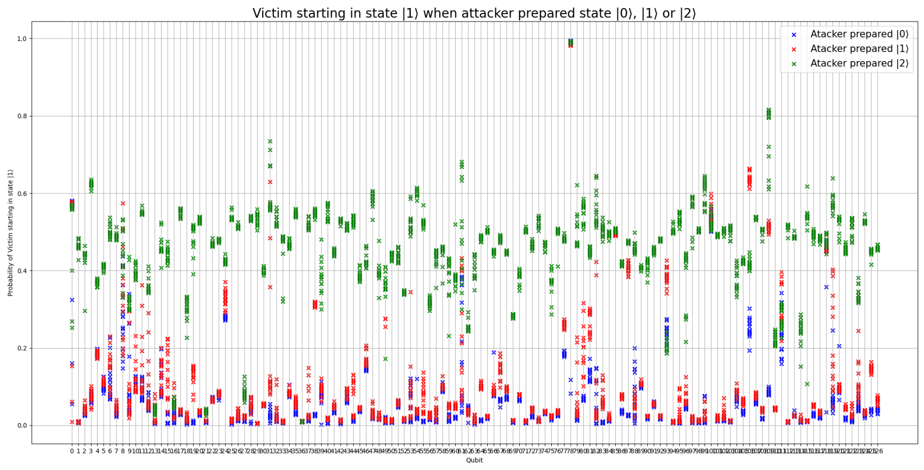


Figure 5.12: Probability of victim qubits starting in state $|1\rangle$ when the attacker has prepared the qubits in state $|0\rangle$, $|1\rangle$, or the second excited state $|2\rangle$ for all 127 qubits of IBM_Osaka (created using IBM Quantum)

As stated before we assumed that the reset operation performed between separate circuits in the same job is the same as the one performed between jobs. Though, after reviewing our paper, IBM stated that 'reset' function between jobs differs from the 'reset' function within jobs. The 'reset' function between jobs on IBM quantum systems is longer, with qubits becoming fully thermalized, making leakage of information between jobs impossible". We think that further research is needed to clarify in what conditions this kind of attacks could still be performed.

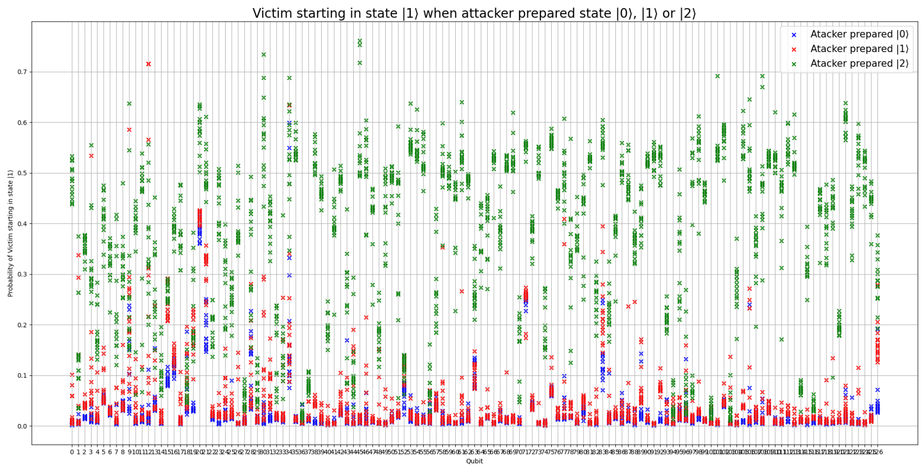


Figure 5.13: Probability of victim qubits starting in state $|1\rangle$ when the attacker has prepared the qubits in state $|0\rangle$, $|1\rangle$, or the second excited state $|2\rangle$ for all 127 qubits of IBM_Sherbrooke (created using IBM Quantum)

5.4.3 Exploring the Potential for Cross-Talk Attacks

Cross-talk attacks have been discussed in Subsection 4.4.5. To execute such an attack, the victim and the attacker must share a QPU at the same time. Today this scenario is not supported by the existing NISQ devices but could be plausible in the future when the number of qubits available in quantum computers will increase significantly. We will study here results of running the Bernstein-Vazirani algorithm. We start with the textbook algorithm as a reference. On the second step we add additional quantum gates on qubits adjacent to those used for implementing the reference algorithm and run the circuit again to study the effects of noise induced by the crosstalk effects.

A quantum circuit implementing this algorithm is shown in Figure 5.14 along with results obtained by executing it on the ibm_kyoto quantum device. Though note that IBM does not allow multiple users to work on a QPU simultaneously, making leakage between users impossible on today's systems. The two control gates implement a boolean function that returns the bit-wise product between an arbitrary 5-digit input number and a fixed binary 5-digit mask, which we'll call 's'. The purpose of the algorithm is to find the number 's' which, in the ideal case, can be computed by running the circuit one single time. In practice, due to noise inherent in current quantum computers, the experiment must be run multiple times and the majority outcome will indicate the correct result. Because we need to do many measurements as opposed to a single one, the quantum advantage is lost in this case, but this is less relevant for our discussion here. For this circuit, the correct result is the $|00011\rangle$ state which is a binary representation of the number 's'. The ibm_kyoto quantum device has 127 qubits, but we need only 6 qubits for implementing the Bernstein-Vazirani algorithm for 5-digit numbers.

Figure 5.15 shows on the left a subset of the ibm_kyoto qubit map that we used for this experiment, along with the qubit connectivity. The qubits used in the algorithm are indicated in green and the adjacent qubits where X gates have been added are indicated in red. To amplify the cross-talk effect, the additional X gates have been added on each layer in the circuit. In the same figure, count results from running this extended circuit are shown on the right. The measurement counts plot indicates a significant deterioration of the quality of the results where the probability for measuring the correct results is reduced by 35%.

Figure 5.16 shows on the left the layout for a similar attack where instead of X gates, Controlled-X gates are placed on adjacent qubits. Here the target qubit belonging to Controlled-X gate is indicated in red and the control qubit is indicated in blue. To amplify the cross-talk effect, the additional Controlled-X gates have been added on each layer of the circuit. Measurement count results are shown in the same figure on right. In this case, as well, the degradation of the results is substantial when compared to the results obtained from running the experiment with no induced crosstalk shown in Figure 5.14.

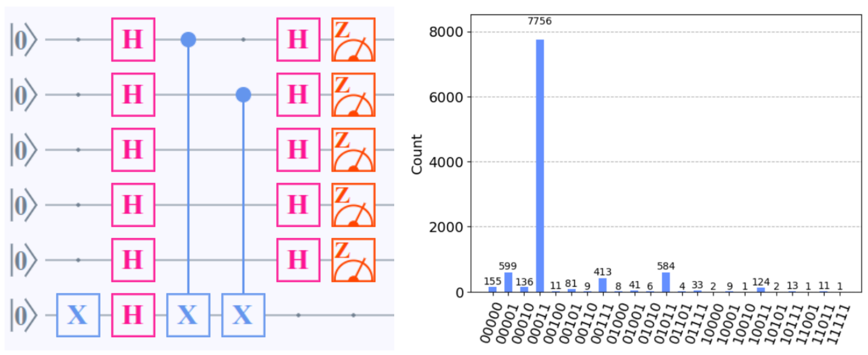


Figure 5.14: The Bernstein-Vazirani algorithm (left) and results from running it on ibm_kyoto (right). The plot on the right shows measurement results obtained from 10000 shots (created using IBM Quantum)

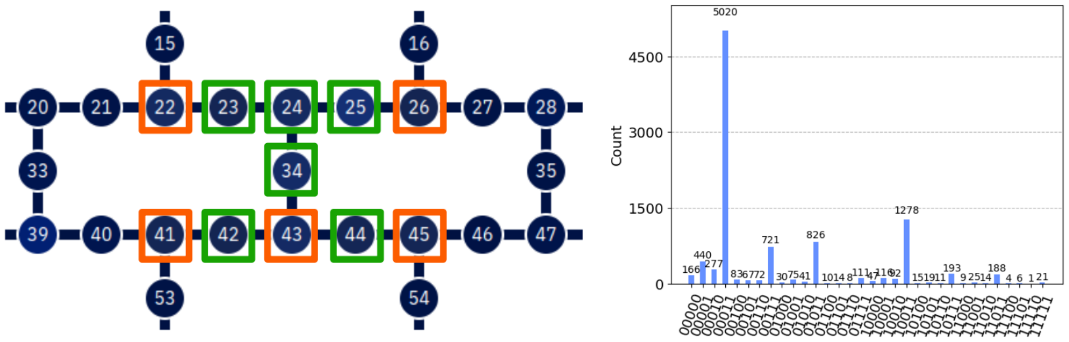


Figure 5.15: Cross-talk experiments results obtained by placing X gates on qubits adjacent to the qubits used in the algorithm. On the qubit connectivity map shown on the left, the algorithm qubits are indicated in green and the qubits where X gates were placed are indicated in red. The plot on the right shows measurement count results obtained from a 10000-shot experiment (created using IBM Quantum)

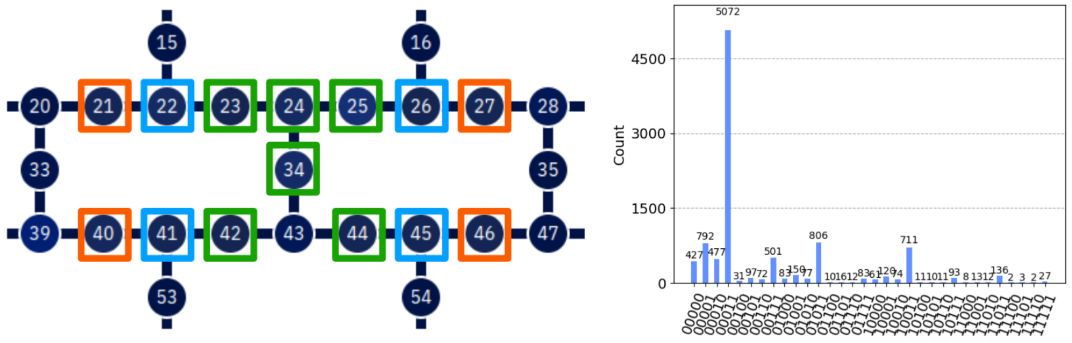


Figure 5.16: Cross-talk experiment results obtained by placing Controlled-X gates on qubits adjacent to the qubits used in the algorithm. On the qubit connectivity map shown on the left the algorithm qubits are indicated in green, the control qubits of the Controlled-X gates are indicated in blue while the target qubits of the Controlled-X gates are indicated in red. The plot on the right shows measurement count results obtained from a 10000-shot experiment (created using IBM Quantum)

6 Reflections on Quantum Computer Related Security

6.1 Importance of Our Investigation

In this section we want to analyze the findings of our work and their relevance. We would like to clarify whether they address realistic problems or just theoretical ones, who could they be useful for and in what context and conditions. Finally, if the security issues this paper reveals are important, we would like to see how they could be addressed and who should take care of this.

Firstly, let us note that even if the development of quantum computer technology and quantum algorithms relevant for and applicable to real-life problems is still an ongoing research, there has been tremendous progress in these fields in the last couple of years. It is conceivable that in the near future, to have quantum computers and quantum programs that will be used for commercial purposes. Not to mention the large community of quantum researchers, who are currently using the nowadays quantum computers. In this context, we think there is no doubt that looking at how secure quantum computers are and will be, and consequently, how secured their users are, is a very important aspect. Ignoring it could have critical consequences when quantum computers will be used for real-life relevant things and cyberattackers will have a great motivation to act against them. Of the same importance is the fact that quantum computers could be used one day to break classical cryptographic schemes currently in use. It is therefore mandatory for quantum providers and quantum users to be aware of the security challenges and risks the large-scale availability of powerful quantum computers will raise.

Like any other resource, it is of equal importance how secure quantum computers are for both their owners and their users. As we saw in previous sections, quantum computer developers have a great interest in keeping their technology confidential. Similarly, quantum computers' users want to keep their quantum algorithms and data confidential. In the same time, quantum providers must protect their resources and their clients.

During our investigation, we noted that, overall, the possible attacks on quantum computing infrastructures are of the same types as those targeting classical computing resources. There could be, thus, attacks like DoS, tampering with different operations and data, information leakage, escalation of privileges etc. Though, quantum computing infrastructures incorporate both classical and quantum computing components and technologies. This means that securing them implies a combination of both classical security mechanisms and quantum specific ones, depending on the type and particularities of the components that are the immediate target of an attack. This is why we tried to classify the possible threat models and attack vectors regarding quantum computing infrastructures, based on the classical and quantum combinations of both targeted components and the attack mechanisms and methods. What is important to note in our classification is that quantum computers could be both attack weapons and targeted resources. Similarly, from a defender's perspective, quantum computers could be both resources to secure and tools used to implement or improve security solutions. Let us review our classification, trying to emphasize main aspects we identified and possible solutions.

6.2 Attacks and Defenses

6.2.1 Classical Attacks on Quantum Computing Software Stack

In this area there could be any kind of classical attacks targeting classical computing resources. Because access to quantum computers is provided through cloud services and running a quantum program usually implies both the end user' computer and cloud resources, attacks could target each of the two and the communication channels between them. While most of such attacks (e.g. DoS, MitM, MitB, DNS spoofing etc.) and possible security solutions are well known, and not necessarily particular to quantum infrastructures, we will not review all on them. Still, we tried to identify what could make such classical attacks specific when targeting the quantum software development process. We mostly focused on end user computers, while not having needed privileges and rights to perform security assessments of cloud services and resources. We also considered communication channels between user computers and cloud to be secured by using classical cryptography, so protected against classical attacks.

One weakness we identified in several popular quantum SDKs was the way *cloud authentication tokens were managed*. Specifically, those tokens are often saved in plain text (in files or environment variables), without requiring an additional authentication factor or having no valability limit set. Consequently, if these tokens were stolen, the victims could easily be impersonated, allowing the attacker to use their cloud credits, access their quantum circuit history, or leak private data. Additionally, we sometimes found that the authentication tokens were placed directly in source code, meaning that if the source code became public, so would the tokens. *Well-known solutions* to this problem include creating short-lifetime tokens, requiring multifactor authentication when using them, storing tokens separately from source code, and prohibiting hard-coding of tokens.

Another vulnerability type we investigated was about the possibility of an end user to work with *corrupted quantum SDK packages*.

One way to get into troubles like this is having an *attacker running malicious code* on an end user's computer. In such a case, the attacker could tamper with the victim user's circuits to steal them, change them, change reported results, inject his own circuits etc. *Possible solutions* against such attacks would be to use a file integrity monitoring tool, configured to monitor and protect specifically the quantum SDK's files.

Another way for an attacker to compromise quantum SDK packages or files would be through *supply-chain attacks*. While this is a very general issue and not in control of an end-user security solution, it happens, most of the time when the user downloads SDK packages from *untrusted sources*. A *solution* against such an attack would be to restrict the user downloading files only from trusted sources, only *signed packages*, whose authenticity and integrity could be checked based on trusted certificates, and getting them through trusted channels.

A variant of using a trusted, yet attacker compromised quantum SDK, is to use a third party, yet *untrusted SDK*. In quantum software development context, this could happen if, for instance, a *third party transpiler* would be needed, to make special optimizations to a quantum circuit, not provided by the default transpiler included in the SDK. Such a transpiler could tamper undetectable with the user circuit, during the transpile phase. *Solutions* against such attacks could be monitoring the SDK's processes against deviated activity (like leaking info to suspicious sites) and obfuscating the quantum circuit provided to the untrusted transpiler. Still, these are complex problems requiring future research for a better protection.

An important aspect of quantum infrastructure security is about *protecting the intellectual property of quantum customers* in the context of *untrusted cloud and quantum providers*. This means that quantum users wants to run their quantum circuits on quantum computers provided remotely through cloud services, but also wants to keep their circuits and results confidential in spite of possible compromised or untrusted cloud and quantum computer providers. *Partial solutions* to such security problems are known to be, at least for classical computing, so-called trusted execution environments (TEEs). Intel SGX is an example of this kind. A TEE is launched and protected during its execution by using dedicated hardware support, such that it could be safely run in an untrusted environment, including all kind of privileged software (e.g. hypervisor, operating system) and users (e.g. sysadmins). A TEE's integrity can be attested remotely and during the attestation process a trusted (i.e. encrypted) channel is established between the TEE and its remote client. This way a quantum user could send her encrypted quantum circuits to the remote TEE, which can process and send them to the quantum computer to be run. While being in transit between user and TEE and processed in the TEE, the quantum circuits are protected. Though, it is not clear if this holds anymore when being sent to the quantum computer, while this means that they are out of the TEE not encrypted, in the untrusted environment. We could not evaluate how much confidential information, relative to the original circuit, an attacker controlling that environment is able to extract, because we had no detailed information regarding the final format of quantum circuits that are run on the quantum computers. Therefore, completely protecting the confidentiality of quantum circuits remains a challenge to be further researched.

From another perspective, quantum providers might want to keep the *confidentiality of their proprietary SDKs* run on *untrusted quantum users' computers*. This could be the case, for instance, for particular transpilers, whose functionality might want to be kept confidential and the transpiled circuits trusted in order to be run safely on quantum computers. Similarly to keeping confidentiality of quantum users' circuits, a possible *solution* to this kind of threat would be to run the SDK in a TEE, whose integrity could be attested remotely by quantum providers before being transferred the SDK. Even more, the resulted transpiled circuits would not leave the TEE unencrypted and unsigned, such that their integrity could also be checked by quantum providers before running them on quantum computers.

6.2.2 Classical Attacks on QPUs

We have presented in this paper several attacks on QPU, both classical and quantum attacks. Classical attacks on QPU could be side-channel attacks or attacks based on pulse level APIs. For side-channel attacks, insider access to quantum computer enclosure or power usage is needed, which makes preventing such attacks simple to implement, at least in principle. For pulse-based attacks, the quantum computer must expose a pulse API. While some quantum computer providers provide today this kind of access to help the user experiment and extract the most performance of current NISQ devices, it is unknown if similar access will be available on tomorrow's quantum computers. We did not provide evidence here that using pulses one can impact qubit calibration, to avoid breaking user license terms but, in our opinion this can likely be done. In such a case, the frequency and length of pulses available to the users should be restricted more than what is permitted today.

6.2.3 Quantum Attacks on Classical Computers

The time horizon for being feasible to execute quantum attacks on classical computer encryption schemes is probably longer than the next several years. However, attacks where data is collected today to be decrypted later are perfectly possible. In this context, the migration to quantum-resistant encryption schemes should be approached with high priority, at least for those applications where data is required to remain confidential for a longer time. This transition is a complex process and will not happen overnight. Accordingly, planning this transition should be started sooner rather than later.

6.2.4 Quantum Attacks on QPUs

Quantum attacks on QPU rely on the attacker and the victim sharing some quantum resources, or being able to use one's quantum resource to influence another. An example of the former would be the physical qubits that are being recycled for each shot in superconducting quantum computers and whose reset operation is imperfect. An example of the latter, unavoidable cross-talk between qubits can be used in scenarios where two users share the same QPU at the same time. The precision of resetting qubits will improve in the future but will probably never be perfect however, since on all quantum platforms users run their shots in sequence, defending against an attack involving imperfect qubit reset requires ignoring the results of the first two or three shots. This is a small price to pay because a typical user runs thousands of shots. Defending against cross-talk attacks could be implemented as an antivirus that scans against and identifies malicious circuits. Alternatively, a protection could perhaps be implemented in the architecture of quantum computers themselves.

Conclusions

This work examines the various issues present at the intersection of cybersecurity and quantum computing in the NISQ era, with a focus on security of quantum computers in particular.

We are approaching a phase in the development of quantum computers in which today's NISQ quantum computers will be replaced by a new generation of equipment, which will perform practical calculations that are not possible with existing classical computing. In addition with considerations like performance, scalability and price, the security of those machines becomes important because a technology with powerful applications like quantum computing will provide strong incentives for attackers.

In addition to reviewing existing work, we identify new vulnerabilities and attack vectors that make quantum computers susceptible to attacks. Besides the attack vectors which are shared with classical computer systems, quantum-specific attack vectors have been discussed. While we attempted a comprehensive approach when studying the security of quantum computing, the scope of our investigations was in part restricted by the need to respect the user licenses of those quantum computing companies that provided public access to their systems.

We thank IBM for providing feedback on this paper and salute their free public access policy to their quantum computers, which made our work possible. We also point out that many of our conclusions can apply to other quantum providers and quantum SDKs and even other cloud-based computational resources.

The purpose of our research is to raise awareness and provide guidance for both end users, on how to protect their data and computers while running quantum programs, and for quantum computer providers, on how to begin protecting their infrastructures against possible attacks. This work was a joint effort of researchers from Bitdefender (https://www.bitdefender.com/) and Transilvania Quantum (https://transilvania-quantum.com/).

Appendices

Detailed Analysis of Quantum Programming Workflows on Different Quantum SKS and Providers

Extending the general quantum programming workflow described in Section 2.4, in this section we analyze it in more technical details, using different quantum programming frameworks and also different quantum hardware providers.

Qiskit with IBM Quantum Provider

A quantum hardware provider that is used for this exemplification is IBM. To access their resources, two phases are necessary, one in which the user authenticates and one in which it sends the quantum programs. Figure 6.1 describes the authentication process.

IBM provides a python module, called `qiskit_ibm_provider` to facilitate the communication between the Qiskit framework, running on a quantum user's computer and the quantum provider cloud services. This module contains Python classes used for authentication, such as *Account*, *AccountManager*, as well as classes used to process the quantum programs, such as *IBMProvider*, *Backend*, *IBMJob*. Moreover, it offers the possibility to save the user's token on user's computer as local file in JSON format, such that at the next authentication it could be loaded automatically from that file.

To use the IBM's quantum hardware, the user must have an account with this provider. Once the account is activated, a unique *token* is assigned to that user, that will be used in the communication with the could services. When the user renews its token, the old one expires and can no longer be used.

The authentication phase begins in step 1 with a *GET /api/version* request. It

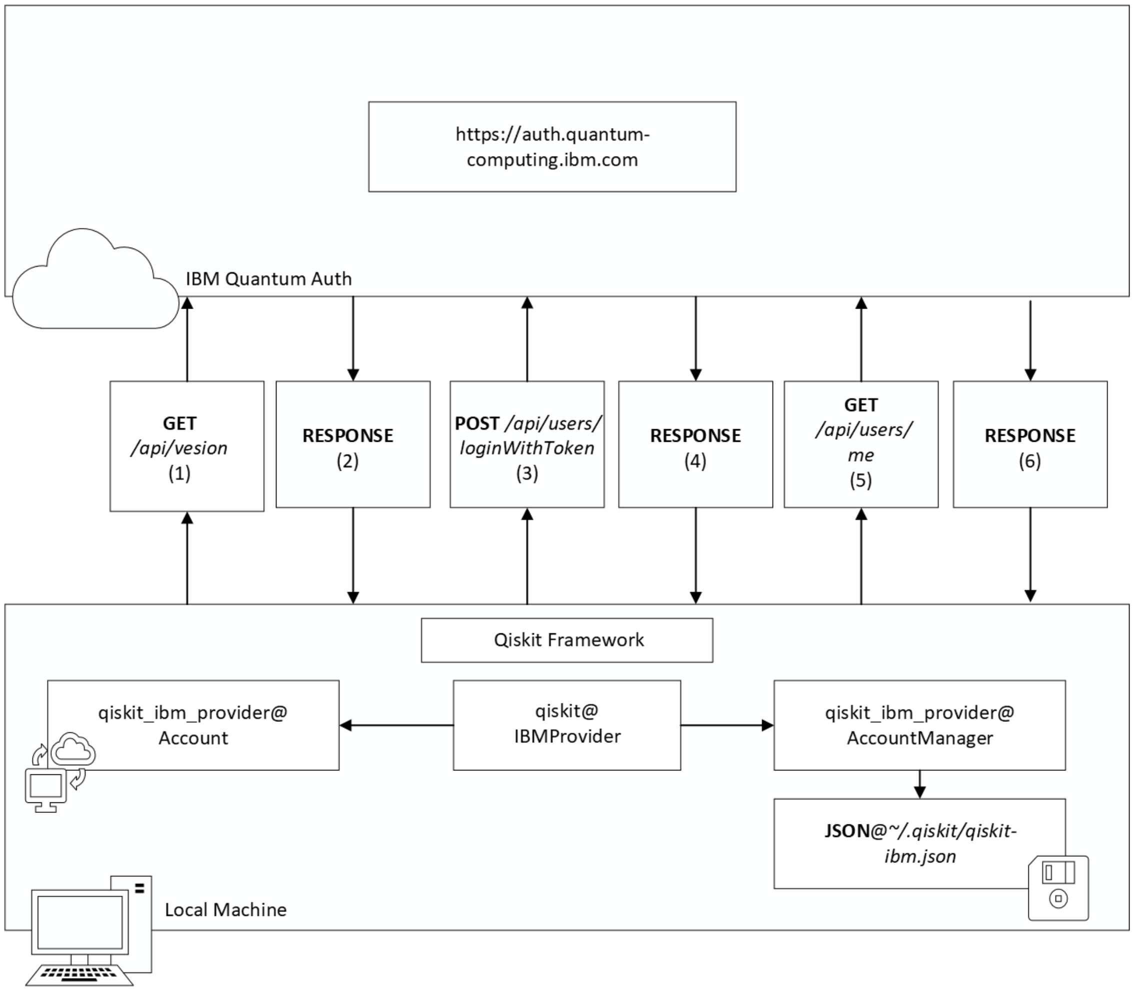

https://auth.quantum-computing.ibm.com
IBM Quantum Auth
GET /api/vesion (1)
RESPONSE (2)
POST /api/users/loginWithToken (3)
RESPONSE (4)
GET /api/users/me (5)
RESPONSE (6)
Qiskit Framework
qiskit_ibm_provider@Account
qiskit@IBMProvider
qiskit_ibm_provider@AccountManager
JSON@~/.qiskit/qiskit-ibm.json
Local Machine

Figure 6.1: Workflow of authenticating on IBM Quantum platform using Qiskit

fetches (step 2) the server version and checks the compatibility between the Qiskit framework and the cloud services. Once the checks are performed, authentication will be attempted using the previously mentioned token. This process is carried out in step 3 through a *POST /api/users/loginWithToken* request, which sends the user's token to the cloud services as illustrated in Listing 6.1.

Listing 6.1: API token for IBM Quantum Platform

```
{
  "apiToken":
  "e699de4537ab2b80d1263b09902e7fcce910b9b0050270b50ad469dc10d
  538ba6f6801cc9adba6adb0c23155cca799a6748111f550b0f78340dd0a
  daecbe65a0"
}
```

The response of this request (step 4) contains the authorization key, that is a random generated token, used until the session time expires, or the user logs in again using the IBMProvider object. It also contains other information about the created session as illustrated in Listing 6.2.

Listing 6.2: Authenticated user information for IBM Quantum

```
{
  "id": " jV0W5f3fuJY4mhIfikuj6HR5MRV862MlTKW87WiRSlUmhIfikuj6H",
  "ttl": 1209600,
  "created": "2023-12-11T11:12:19.927Z",
  "userId": "97369a485c752b06f8272309"
}
```

The Qiskit python module only knows in advance the authentication URL for IBM Quantum Platform. In order to find out the URL used to submit circuits, a *GET /api/user/me* request is sent (step 5). The response (step 6) contains information about the authenticated user, as well as the URL to the APIs where circuits can be submitted. A sample of such URLs could be seen in Listing 6.3.

Listing 6.3: URLs to submit circuits on IBM Quantum Platform

```
{
  "urls": {
    "http": "https://api.quantum.ibm.com/api",
    "ws": "wss://wss.quantum-computing.ibm.com/",
    "services": {
```

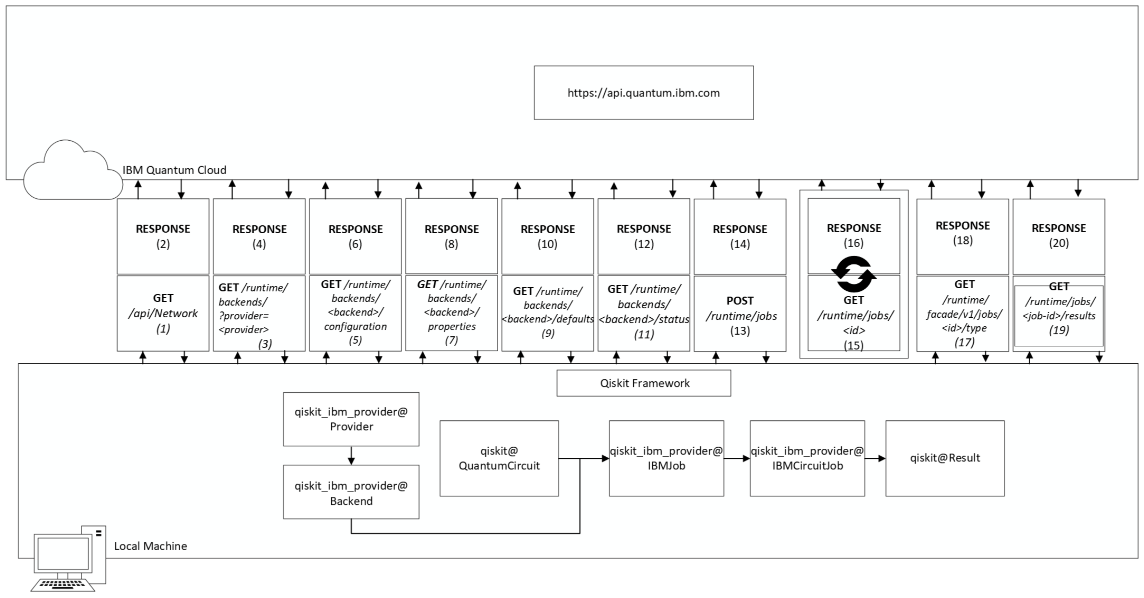

https://api.quantum.ibm.com
IBM Quantum Cloud
RESPONSE (2)
GET /api/Network (1)
RESPONSE (4)
GET /runtime/backends/?provider=<provider> (3)
RESPONSE (6)
GET /runtime/backends/<backend>/configuration (5)
RESPONSE (8)
GET /runtime/backends/<backend>/properties (7)
RESPONSE (10)
GET /runtime/backends/<backend>/defaults (9)
RESPONSE (12)
GET /runtime/backends/<backend>/status (11)
RESPONSE (14)
POST /runtime/jobs (13)
RESPONSE (16)
GET /runtime/jobs/<id> (15)
RESPONSE (18)
GET /runtime/facade/v1/jobs/<id>/type (17)
RESPONSE (20)
GET /runtime/jobs/<job-id>/results (19)
Qiskit Framework
qiskit_ibm_provider@Provider
qiskit_ibm_provider@Backend
qiskit@QuantumCircuit
qiskit_ibm_provider@IBMJob
qiskit_ibm_provider@IBMCircuitJob
qiskit@Result
Local Machine

Figure 6.2: Workflow of using IBM quantum computers with Qiskit

```
        "quantumLab": "https://notebooks.quantum-computing.ibm.com",
        "runtime": "https://api.quantum.ibm.com/runtime"
      }
    }
}
```

Once the authentication is performed successfully, a circuit can be sent to the IBM Cloud Services to be executed. Figure 6.2 shows the data flow that is transmitted between the framework and the cloud services.

IBM provides several quantum hardware on which the circuits can be executed, and these are available to a user according to the type of account, either paid or free. Using the REST APIs provided by IBM, the user can ask for a list of quantum systems that can be used, making a *GET /runtime/backends/provider=provider* request (step 3 in Figure 6.2). The response (step 4) contains a list of quantum computers as exemplified in Listing 6.4.

Listing 6.4: Quantum computers on IBM Quantum Platform

```
{
  "devices": [
    "simulator_extended_stabilizer",
    "simulator_mps",
    "simulator_statevector",
    "simulator_stabilizer",
    "ibm_brisbane",
    "ibm_kyoto",
    "ibm_osaka",
    "ibmq_qasm_simulator"
  ]
}
```

The transpilation process is done locally if the circuit is executed by IBM hardware. Therefore, for this process the hardware details of the chosen backend are required. In order to have these details, the framework makes few requests to build the *Backend* object, according to the specifications given by the cloud services. This process is done automatically by the framework, so that there is no difference for the user regardless of the chosen backend. For example, the following requests are sent when using the `ibmq_kyoto` quantum computer.

```
GET /runtime/backends/ibmq_kyoto/configuration (step 5)
GET /runtime/backends/ibmq_kyoto/properties    (step 7)
GET /runtime/backends/ibmq_kyoto/defaults      (step 9)
GET /runtime/backends/ibmq_kyoto/status        (step 11)
```

Once all the necessary information is gathered, the user's circuit can be transpiled, after which it can be sent to the backend. The submission of the circuit is made through a *POST /runtime/jobs* request (step 13), which will contain metadata used by the could services along with the circuit which is serialized using the QPY format. This could be seen in Listing 6.5.

Listing 6.5: Encoded circuit sent to IBM Quantum Platform

```
"circuits": [
{
    "__type__": "QuantumCircuit",
    "__value__":
      "eJwL9Az29gzhZJBkZoAAxkIG7jQGDiCLGYpBgAmKQYA9ObMouT
      SzRNfQyMBm4TI1Y5EYturaQkawcibGQgMGVMAIMyMZpiQZpxK4C
      BuUx8iAHYQauSeWpBaCmGlQIQ6YHFxAQtcl5Lci5wESTGPE0Izd
      NFTXoniE0TkC7jawkSRYz0SEZ9iR7cJimm9qYnFpEcTZLAICTo
      Yk8FqSNDBlAz13X8kAFIDAAFO0U="
```

```
  }
]
```

The response (step 14) to the previously made request contains the *id* of the created job that could be used to fetch the results. Listing 6.6 illustrates an example of this kind.

Listing 6.6: A job id on IBM Quantum Platform

```
{
	"id": "6hefkcey2i3nwydpoh4f",
	"backend": "ibmq_kyoto"
}
```

The remote job is created and placed in a waiting queue. Considering the fact that several users want to execute circuits using quantum hardware, the result will not be available immediately. As a consequence, the framework provides a method through which the user can monitor the status of a submitted job. It periodically checks the status of the job using a *GET /runtime/jobs/job-id* request (step 15) until it is marked as completed.

The results are also represented as a dictionary object as illustrated in Listing 6.7.

Listing 6.7: Results of a job run on IBM Quantum Platform

```
"data": {
  "counts": {
    "0x5": 9,
    "0x4": 17,
    "0x6": 17,
    "0x2": 17,
    "0x3": 20,
    "0x7": 13,
    "0x0": 19,
    "0x1": 16
  }
},
```

Qiskit with IonQ Quantum Provider

Figure 6.3 shows the data that is transmitted between the Qiskit framework, running on a quantum user's computer, cloud services from IonQ quantum provider. To facilitate the communication between these two, IonQ implements a python module, named `qiskit_ionq`, that contains vendor specific Python classes, such as *Provider*, *IonQBackend* and *IonQJob*.

To use the hardware from IonQ, an account at this vendor is required. Once the account is activated, the user can generate one or multiple *API-KEYs* that will be used to transmit data to and from the cloud services.

The communication between the Qiskit framework and the cloud services is made using *REST-API* requests to `https://api.ionq.co/v0.3`. The same address is used both for authentication and submitting quantum circuits. The authentication of the user is made through the field `"Authorization:  api Key $KEY"` in the header of each request, which is filled with the user's *API-KEY*.

The first step consists in sending a request to submit the circuit to the cloud services. The circuits that IonQ expects to receive must be in JSON format, while Qiskit has its own way of storing the information. To maintain compatibility, the *qiskit_ionq* module encodes the circuit from Qiskit format in the format required by IonQ, using the method described in Listing 6.8.

Listing 6.8: Convert a Qiskit circuit to a IonQ compatible dict

```python
def qiskit_to_ionq(
  circuit, backend, passed_args=None,
  extra_query_params=None, extra_metadata=None
):
```

The submitted circuit is represented in Listing 6.9.

Listing 6.9: Circuit initialization

```json
"circuit": [
  {
    "gate": "x",
    "targets": [ 1 ]
  },
  {
    "gate": "h",
    "targets": [ 0 ]
```

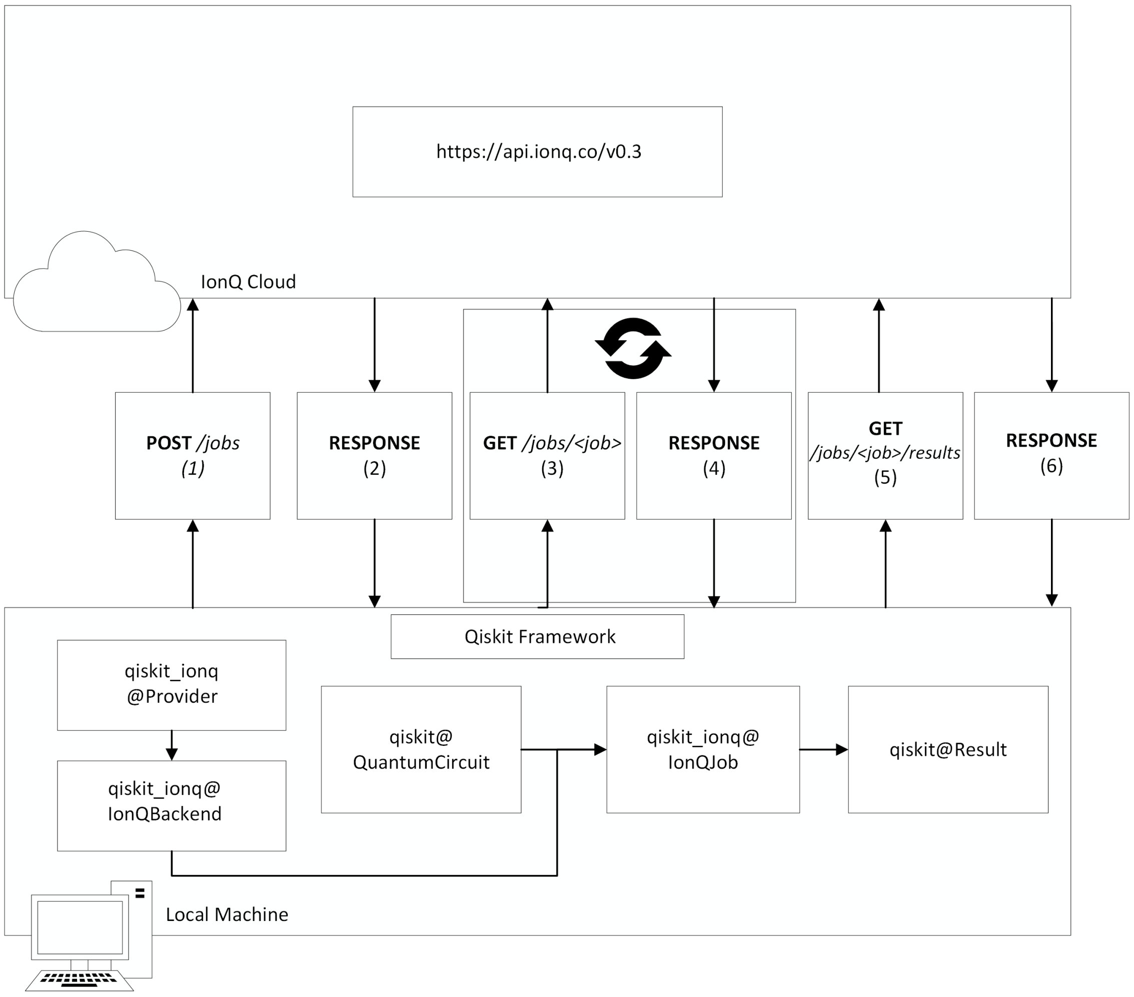


Figure 6.3: Workflow of using IonQ quantum computers with Qiskit

```
  },
  {
    "gate": "h",
    "targets": [ 1 ]
  },
  {
    "gate": "h",
    "targets": [ 2 ]
  },
  {
    "gate": "x",
    "targets": [ 1 ],
    "controls": [ 0 ]
  }
]
```

Moreover, some encoded metadata that provides additional information about the circuit is filled in the request. The encoding consists in converting the data that is in JSON format into a string, then a compression is applied using *gzip*, after which a *base64* encoding is applied. These steps are executed by method `compress_dict_to_metadata_string`.

The encoded string found in the request is illustrated in Listing 6.10.

Listing 6.10: Encoded string

```
"metadata": {
  "shots": "128",
  "sampler_seed": "None",
  "qiskit_header":
    "H4sIAMDlcWUC12OQQrCMBBFr1JmrZK2uPEqpQxJCDUwaZykXah4dyelGHU3vPY
     54QXIjpjpnikuHS9IcGJopGE96u0jtB6qQEzsir8ZOrEPJwPpkV78c23MHgm1yE
     2bcKU3DGCV0H4cS0QyR9LG0XeoJNvPtp7dNuEG1fb9syPjauNq4132sNvmvb6uoAAAA
        ↪ ="
}
```

The previously mentioned string is decoded by the cloud services in a JSON object as illustrated in Listing 6.11.

Listing 6.11: Decoded string as Json file

```
{
  "memory_slots": 3,
  "global_phase": 0.0,
  "n_qubits": 3,
  "name": "circuit-152",
  "creg_sizes": [
    [ "c0", 3 ]
  ],
  "clbit_labels": [
    [ "c0", 0 ],
    [ "c0", 1 ],
    [ "c0", 2 ]
  ],
  "qreg_sizes": [
        [ "q0", 3 ]
  ],
  "qubit_labels": [
    [ "q0", 0 ],
    [ "q0", 1 ],
    [ "q0", 2 ]
  ]
}
```

In step 2, a response with a JSON object is received (see Listing 6.12), which contains information about the job created, including its *id*. This job identifier can be used in the next steps.

Listing 6.12: Submitted job id and status

```
{
        "id": "e8d8028d-9494-4853-b782-3061f4b6b5c7",
        "status": "ready",
        "request": 5110880341
}
```

As previously mentioned, there is a method to wait for the execution of a job. It can be used to know when to move on to the next step. After the program is executed, the results can be fetched by the user using a *GET /jobs/job-id* (step 3) request.

Once the quantum program is executed, the response received in step 4 contains an additional field, which is the URL from where the user can fetch the results of the executed program. Listing 6.13 show and example of this kind.

Listing 6.13: URL where results can be obtained from

```
{
  "status": "completed",
  "results_url": "/v0.3/jobs/1caa40ca-0733-4c22-8ce9-01baf40f85ee/
      ↪ results"
}
```

Having this URL, the user can get the results by sending a *GET results_url* request (step 5). The response is a dictionary that contains the results, similar what can be seen in Listing 6.14.

Listing 6.14: Results of a circuit execution

```
{
  "0": 0.125000000,
  "1": 0.125000000,
  "2": 0.125000000,
  "3": 0.125000000,
  "4": 0.125000000,
  "5": 0.125000000,
  "6": 0.125000000,
  "7": 0.125000000
}
```

Cirq with IonQ Quantum Provider

An alternative to Qiskit is Cirq framework that will be explained using IonQ hardware. To use the Cirq framework for accessing the IonQ hardware from a quantum user's computer, a python module called *cirq_ionq* is required. The communication protocol is largely similar to the case of the Qiskit framework, the difference being the content of some packets that are transmitted between the two components, i.e. the user's computer and the quantum provider cloud services. Cirq uses different Python classes to abstract circuits, such as *Service*, *Ciruit*, *Job*, *Result*. Figure 6.4 shows the data flow between these two components.

The authentication is similar to the case of Qiskit, using the `"Authorization: api Key $KEY"` field in the header of each request. To submit a program, a *POST*

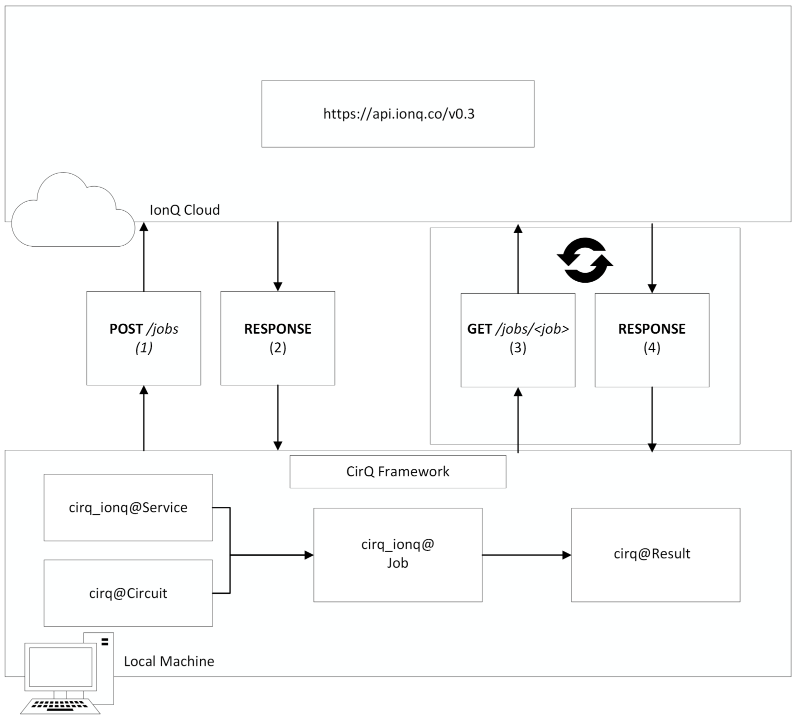


Figure 6.4: Workflow of using IonQ quantum computers with Cirq

/jobs request (step 1) is sent using a specific cloud API. That request contains the circuit in JSON format as illustrated in Listing 6.15.

Listing 6.15: Cirq circuit definition in JSON format

```
{
  "target": "qpu",
  "lang": "json",
  "body": {
    "gateset": "qis",
    "qubits": 2,
    "circuit": [{"gate": "v", "targets": [0]}, {"gate": "cnot", "control
        ↪ ": 0, "target": 1}]
  },
  "metadata": {"measurement0": "b\u001f0,1", "shots": "100"},
  "shots": "100"
}
```

The response (step 2) has the same format as on Qiskit, containing the *id* of the newly created job. The backend creates a job that will be put in a waiting queue, to be executed when scheduled based on some policy.

To get the result, periodic *GET /jobs/job-id* requests are made to check the status of the job. When the circuit is executed by the chosen backend, the status of the job will change, and the result is contained in the request made periodically by the framework. This is different from Qiskit, where the result must be taken from another URL. The result is represented as a dictionary object as illustrated in Listing 6.16.

Listing 6.16: Cirq circuit execution's results

```
"data": {
  "histogram": {
    "0": 0.500000000,
    "3": 0.500000000
  },
}
```

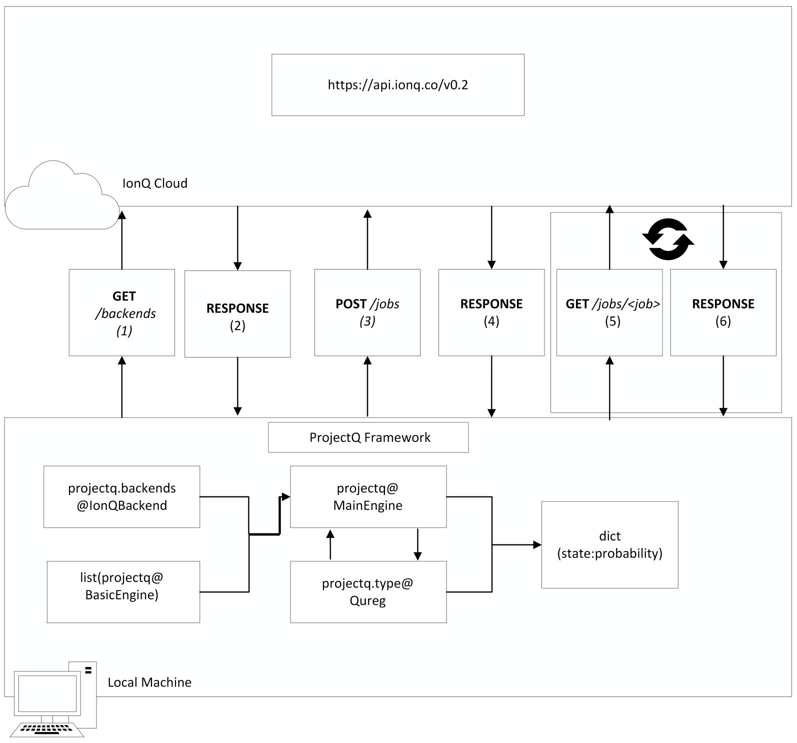


Figure 6.5: Workflow of using IonQ quantum computers with ProjectQ

ProjectQ with IonQ Quantum Provider

Another framework that can be used to execute quantum circuits on IonQ hardware is *ProjectQ*. A difference between it and *Cirq* is given by the objects used for data abstraction. The communication between the framework, running on quantum user's computer, and the cloud services is made using *REST-API* requests to `https://api.ionq.co/v0.2`. Figure 6.5 shows the data flow transmitted between these two components.

The framework can request a list of quantum computers that can be used, making a *GET /backends* request (step 1). The response (step 2) contains a list of quantum hardware as illustrated in Listing 6.17.

Listing 6.17: List of Ionq quantum computers for a ProjectQ program

```
{
  "backend": "qpu.harmony",
  "status": "available",
  "qubits": 11,
  "average_queue_time": 176025567,
  "last_updated": 1705500380,
  "has_access": false,
  "characterization_url": "/characterizations/b6dd937e-1803-44c3-a3e3
      ↪ -9215c8ff6b72",
  "degraded": false
}
```

The *POST /jobs* request in step 3 is used to submit a circuit. Similar to Cirq, the circuit is encoded in JSON format. The request has few framework specific fields as illustrated in Listing 6.18.

Listing 6.18: ProjectQ circuit submission request structure (only specific fields are shown, the others are simmilar to those in Listing 6.15)

```
"metadata": {
  "sdk": "ProjectQ",
  "meas_qubit_ids": "[0, 1]"
}
```

The response contains the *id* of the newly created job. Once the program is executed, the results are gathered using a *GET /jobs/job-id* request (step 5).

The results are represented as a dictionary of the form *state:probability* contained in the *histogram* field in the response, as illustrated in Listing 6.19.

Listing 6.19: Circuit initialization

```
{
  "histogram": {
    "0": 0.500000000,
    "3": 0.500000000
  }
}
```

Bibliography

[1] Shweta Agrawal et al. “Adaptive Simulation Security for Inner Product Functional Encryption”. In: *Public-Key Cryptography - PKC 2020*. Vol. 12110. 2020, pp. 34–64.

[2] *Amazon Braket SDK*. `https://github.com/amazon-braket/amazon-braket-sdk-python`. Accessed: 2024-02-01.

[3] Abdullah Ash-Saki, Mahabubul Alam, and Swaroop Ghosh. “Analysis of Crosstalk in NISQ Devices and Security Implications in Multi-programming Regime”. In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. 2020, pp. 25–30.

[4] Shi Bai et al. “MPSign: A Signature from Small-Secret Middle-Product Learning with Errors”. In: *Public-Key Cryptography - PKC 2020*. Vol. 12111. 2020, pp. 66–93.

[5] Sara Bartolucci et al. “Fusion-based quantum computation”. In: *Nature Communications* 14.1 (2023), p. 912.

[6] Ethan Bernstein and Umesh Vazirani. “Quantum Complexity Theory”. In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 1993, pp. 11–20.

[7] Bitdefender. *What is a Man-in-the-Middle attack (MiTM)?* `https://www.bitdefender.com/consumer/support/answer/49038/`. Accessed: 2024-02-07.

[8] Kostas Blekos et al. “A Review on Quantum Approximate Optimization Algorithm and its Variants”. In: *arXiv preprint arXiv:2306.09198* (2023).

[9] Kostas Blekos et al. “A review on quantum approximate optimization algorithm and its variants”. In: *Physics Reports* 1068 (2024), pp. 1–66.

[10] Madalina Bolboceanu, Zvika Brakerski, and Devika Sharma. “On Algebraic Embedding for Unstructured Lattices”. In: *IACR Cryptol. ePrint Arch.* (2021/053). URL: `https://eprint.iacr.org/2021/053`.

[11] Madalina Bolboceanu et al. “Order-LWE and the Hardness of Ring-LWE with Entropic Secrets”. In: *Advances in Cryptology - ASIACRYPT 2019*. Vol. 11922. Springer, 2019, pp. 91–120.

[12] Hans J Briegel et al. “Measurement-based quantum computation”. In: *Nature Physics* 5.1 (2009), pp. 19–26.

[13] Zhenyu Cai et al. “Quantum error mitigation”. In: *Reviews of Modern Physics* 95.4 (2023), p. 045005.

[14] *Calibrate Superconducting Qubits with Pulse*. `https://github.com/Qiskit/textbook/tree/main/notebooks/quantum-hardware-pulses`. Accessed: 2024-02-01.

[15] Davide Castelvecchi. “IBM releases first-ever 1,000-qubit quantum chip”. In: *Nature* 624.7991 (2023), pp. 238–238.

[16] Gianluigi Catelani et al. “Decoherence of superconducting qubits caused by quasiparticle tunneling”. In: *Physical Review B* 86.18 (2012), p. 184514.

[17] M Cerezo et al. “Challenges and opportunities in quantum machine learning”. In: *Nature Computational Science* 2.9 (2022), pp. 567–576.

[18] *Cirq*. `https://quantumai.google/cirq`. Accessed: 2024-02-01.

[19] Vlad CONSTANTINESCU. *Malicious PyPI Packages Bypass Firewall Restrictions via Cloudflare Tunnels*. `https://www.bitdefender.com/blog/hotforsecurity/malicious-pypi-packages-bypass-firewall-restrictions-via-cloudflare-tunnels/`. Accessed: 2024-02-07.

[20] Andrew Cross et al. “OpenQASM 3: A broader and deeper quantum assembly language”. In: *ACM Transactions on Quantum Computing* 3.3 (2022), pp. 1–50.

[21] Andrew W Cross et al. “Open quantum assembly language”. In: *arXiv preprint arXiv:1707.03429* (2017).

[22] Andrew W Cross et al. “Validating quantum computers using randomized model circuits”. In: *Physical Review A* 100.3 (2019), p. 032328.

[23] Sanjay Deshpande et al. “Towards an Antivirus for Quantum Computers”. In: *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. 2022, pp. 37–40. DOI: `10.1109/HOST54066.2022.9840181`.

[24] Sanjay Deshpande et al. “Towards an Antivirus for Quantum Computers”. In: *2022 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*. IEEE. 2022, pp. 37–40.

[25] Michel H Devoret, Andreas Wallraff, and John M Martinis. “Superconducting qubits: A short review”. In: *arXiv preprint cond-mat/0411174* (2004).

[26] Yongshan Ding et al. “Systematic crosstalk mitigation for superconducting qubits via frequency-aware compilation”. In: *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE. 2020, pp. 201–214.

[27] David P. DiVincenzo. “The Physical Implementation of Quantum Computation”. In: *Fortschritte der Physik* 48.9–11 (Sept. 2000), pp. 771–783. ISSN: 1521-3978. DOI: `10.1002/1521-3978(200009)48:9/11<771::aid-prop771>3.0.co;2-e`. URL: `http://dx.doi.org/10.1002/1521-3978(200009)48:9/11%3C771::AID-PROP771%3E3.0.CO;2-E`.

[28] Jay M Gambetta et al. “Analytic control methods for high-fidelity unitary operations in a weakly nonlinear oscillator”. In: *Physical Review A* 83.1 (2011), p. 012308.

[29] Lov K Grover. “A Fast Quantum Mechanical Algorithm for Database Search”. In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 1996, pp. 212–219.

[30] Philipp Gühring. “Concepts against Man-in-the-Browser Attacks”. In: *https://www2.futureware.*
(2007).

[31] E Gümüş et al. “Calorimetry of a phase slip in a Josephson junction”. In: *Nature Physics* 19.2 (2023), pp. 196–200.

[32] Eliot Kapit. “The upside of noise: engineered dissipation as a resource in superconducting circuits”. In: *Quantum Science and Technology* 2.3 (2017), p. 033002.

[33] Youngseok Kim et al. “Evidence for the utility of quantum computing before fault tolerance”. In: *Nature* 618.7965 (2023), pp. 500–505.

[34] Morten Kjaergaard et al. “Superconducting qubits: Current state of play”. In: *Annual Review of Condensed Matter Physics* 11 (2020), pp. 369–395.

[35] Philip Krantz et al. “A quantum engineer’s guide to superconducting qubits”. In: *Applied physics reviews* 6.2 (2019).

[36] Benoit Libert, Damien Stehlé, and Radu Titiu. “Adaptively Secure Distributed PRFs from LWE”. In: *Theory of Cryptography - 16th International Conference, TCC 2018*. Vol. 11240. 2018, pp. 391–421.

[37] Benoit Libert and Radu Titiu. “Multi-Client Functional Encryption for Linear Functions in the Standard Model from LWE”. In: *Advances in Cryptology - ASIACRYPT 2019*. Vol. 11923. 2019, pp. 520–551.

[38] Benoit Libert et al. “Simulation-Sound Arguments for LWE and Applications to KDM-CCA2 Security”. In: *Advances in Cryptology - ASIACRYPT 2020*. Vol. 12491. 2020, pp. 128–158.

[39] Sam McArdle et al. “Quantum Computational Chemistry”. In: *Reviews of Modern Physics* 92.1 (2020), p. 015003.

[40] David C McKay et al. “Efficient Z gates for quantum computing”. In: *Physical Review A* 96.2 (2017), p. 022330.

[41] Allen Mi, Shuwen Deng, and Jakub Szefer. “Securing Reset Operations in NISQ Quantum Computers”. In: Nov. 2022, pp. 2279–2293. DOI: `10.1145/3548606.3559380`.

[42] Felix Motzoi et al. “Simple pulses for elimination of leakage in weakly nonlinear qubits”. In: *Physical review letters* 103.11 (2009), p. 110501.

[43] Clemens Müller et al. “Interacting two-level defects as sources of fluctuating high-frequency noise in superconducting circuits”. In: *Physical Review B* 92.3 (2015), p. 035442.

[44] Prakash Murali et al. “Software mitigation of crosstalk on noisy intermediate-scale quantum computers”. In: *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*. 2020, pp. 1001–1016.

[45] Michael A Nielsen and Isaac L Chuang. *Quantum Computation and Quantum Information*. Cambridge university press, 2010.

[46] *NIST Post-Quantum Cryptography Standardization*. `https://csrc.nist.gov/projects/post-quantum-cryptography`. Accessed: 2024-02-07.

[47] *NIST to Standardize Encryption Algorithms That Can Resist Attack by Quantum Computers*. https://www.nist.gov/news-events/news/2023/08/nist-standardize-encryption-algorithms-can-resist-attack-quantum-computers. Accessed: 2024-02-07.

[48] *Open Fermion*. https://github.com/quantumlib/OpenFermion. Accessed: 2024-02-01.

[49] Ankita Pathare and Bharti Deshmukh. “Review on Cryptography Using Quantum Computing”. In: *International Journal for Modern Trends in Science and Technology* 8 (01 2022), pp. 141–146.

[50] *PennyLane Plugins*. https://pennylane.ai/plugins/. Accessed: 2024-02-01.

[51] Michael Peterer. “Experiments on multi-level superconducting qubits and coaxial circuit QED”. PhD thesis. University of Oxford, 2016.

[52] Koustubh Phalak et al. “Quantum PUF for Security and Trust in Quantum Computing”. In: *IEEE Journal on Emerging and Selected Topics in Circuits and Systems* 11.2 (2021), pp. 333–342.

[53] *POC for classical attacks.* https://github.com/Transilvania-Quantum/quantum-computing-security-investigations. Accessed: 2024-07-01.

[54] John Preskill. “Quantum computing in the NISQ era and beyond”. In: *Quantum* 2 (2018), p. 79.

[55] FIPS PUB. “Digital signature standard (DSS)”. In: *Fips pub* (2000), pp. 186–192.

[56] NIST FIPS Pub. “197: Advanced encryption standard (AES)”. In: *Federal information processing standards publication* 197.441 (2001), p. 0311.

[57] *pypi*. https://pypi.org/. Accessed: 2024-02-07.

[58] *Pyquil*. https://github.com/rigetti/pyquil. Accessed: 2024-02-01.

[59] *Pytket*. https://cqcl.github.io/tket/pytket/api/. Accessed: 2024-02-01.

[60] *Q Sharp*. https://learn.microsoft.com/en-us/azure/quantum/overview-what-is-qsharp-and-qdk. Accessed: 2024-02-01.

[61] *qBraid*. https://github.com/qBraid/qBraid. Accessed: 2024-02-01.

[62] *QIR Alliance*. https://www.qir-alliance.org/. Accessed: 2024-02-01.

[63] *Qiskit*. `https://github.com/Qiskit`. Accessed: 2024-02-01.

[64] *Qiskit Finance*. `https://github.com/qiskit-community/qiskit-finance`. Accessed: 2024-02-01.

[65] *Qiskit Machine Learing*. `https://github.com/qiskit-community/qiskit-machine-learning`. Accessed: 2024-02-01.

[66] *Qiskit Nature*. `https://github.com/qiskit-community/qiskit-nature`. Accessed: 2024-02-01.

[67] *Qiskit Optmization*. `https://github.com/qiskit-community/qiskit-optimization`. Accessed: 2024-02-01.

[68] *QPY Serialization Format*. `https://docs.quantum.ibm.com/api/qiskit/qpy`. Accessed: 2024-02-01.

[69] *Quantum-Readiness: Migration to Post-Quantum Cryptography*. `https://www.cisa.gov/resources-tools/resources/quantum-readiness-migration-post-quantum-cryptography`. Accessed: 2024-02-07.

[70] *Quil*. `https://github.com/quil-lang/quil`. Accessed: 2024-02-01.

[71] Mark Randolph and William Diehl. "Power side-channel attack analysis: A review of 20 years of study for the layman". In: *Cryptography* 4.2 (2020), p. 15.

[72] Ronald L Rivest, Adi Shamir, and Leonard Adleman. "A method for obtaining digital signatures and public-key cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[73] Miruna Rosca, Damien Stehlé, and Alexandre Wallet. "On the Ring-LWE and Polynomial-LWE Problems". In: *Advances in Cryptology - EUROCRYPT 2018*. Vol. 10820. 2018, pp. 146–173.

[74] Miruna Rosca et al. "Middle-Product Learning with Errors". In: *Advances in Cryptology - CRYPTO 2017*. Vol. 10403. 2017, pp. 283–297.

[75] Abdullah Ash Saki and Swaroop Ghosh. *Qubit Sensing: A New Attack Model for Multi-programming Quantum Computing*. 2021. arXiv: `2104.05899 [quant-ph]`.

[76] Abdullah Ash Saki, Rasit Onur Topaloglu, and Swaroop Ghosh. *Shuttle-Exploiting Attacks and Their Defenses in Trapped-Ion Quantum Computers*. 2021. arXiv: `2108.01054 [quant-ph]`.

[77] Abdullah Ash Saki et al. *A Survey and Tutorial on Security and Resilience of Quantum Computing*. 2021. arXiv: `2106.06081 [quant-ph]`.

[78] Abdullah Ash Saki et al. "Split Compilation for Security of Quantum Circuits". In: *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE. 2021, pp. 1–7.

[79] Mohan Sarovar et al. "Detecting Crosstalk Errors in Quantum Information Processors". In: *Quantum* 4 (2020), p. 321.

[80] Peter W Shor. "Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer". In: *SIAM review* 41.2 (1999), pp. 303–332.

[81] Daniel R Simon. "On the power of quantum computation". In: *SIAM journal on computing* 26.5 (1997), pp. 1474–1483.

[82] Sergei Slussarenko and Geoff J Pryde. "Photonic quantum information processing: A concise review". In: *Applied Physics Reviews* 6.4 (2019).

[83] Kaitlin N Smith et al. "Programming physical quantum systems with pulse-level control". In: *Frontiers in Physics* 10 (2022), p. 900099.

[84] Silviu Stahie. *Supply Chain Attack Detected in PyPI Library*. `https://www.bitdefender.com/blog/hotforsecurity/supply-chain-attack-detected-in-pypi-library/`. Accessed: 2024-02-07.

[85] *StrawberryFields*. `https://strawberryfields.ai/`. Accessed: 2024-02-01.

[86] L Sun et al. "Measurements of quasiparticle tunneling dynamics in a band-gap-engineered transmon qubit". In: *Physical review letters* 108.23 (2012), p. 230509.

[87] Aakarshitha Suresh et al. "A quantum circuit obfuscation methodology for security and privacy". In: *arXiv preprint arXiv:2104.05943* (2021).

[88] Robert S Sutor. *Dancing with Qubits: How quantum computing works and how it can change the world*. Packt Publishing Ltd, 2019.

[89] Jerry Tan et al. *Extending and Defending Attacks on Reset Operations in Quantum Computers*. 2023. arXiv: `2309.06281 [cs.AR]`.

[90] Bitdefender Cryptography Research Team. *Private Set Intersection from Homomorphic Encryption: A Python Implementation*. `https://bit-ml.github.io/blog/post/private-set-intersection-an-implementation-in-python/`. Accessed: 2024-02-07.

[91] *Tensorflow*. `https://www.tensorflow.org/quantum`. Accessed: 2024-02-01.

[92] *The LLVM Compiler Infrastructure*. `https://llvm.org/`. Accessed: 2024-03-01.

[93] *The State of Quantum Open Source Software 2023: Survey Results*. `https://unitary.fund/posts/2023_survey_results/`. Accessed: 2024-04-01.

[94] Jules Tilly et al. "The variational quantum eigensolver: a review of methods and best practices". In: *Physics Reports* 986 (2022), pp. 1–128.

[95] *Tket*. `https://www.quantinuum.com/developers/tket`. Accessed: 2024-02-01.

[96] Chen Wang et al. "Measurement and control of quasiparticle dynamics in a superconducting qubit". In: *Nature communications* 5.1 (2014), p. 5836.

[97] Göran Wendin and VS Shumeiko. "Quantum bits with Josephson junctions". In: *Low Temperature Physics* 33.9 (2007), pp. 724–744.

[98] John van de Wetering. "ZX-calculus for the Working Quantum Computer Scientist". In: *arXiv preprint arXiv:2012.13966* (2020).

[99] Karen Wintersperger et al. "Neutral atom quantum computing hardware: performance and end-user perspective". In: *EPJ Quantum Technology* 10.1 (2023), p. 32.

[100] Chuanqi Xu, Ferhat Erata, and Jakub Szefer. "Classification of quantum computer fault injection attacks". In: *arXiv preprint arXiv:2309.05478* (2023).

[101] Chuanqi Xu, Ferhat Erata, and Jakub Szefer. "Exploration of Quantum Computer Power Side-Channels". In: *arXiv preprint arXiv:2304.03315* (2023).

Authors' contacts

Mădălina Bolboceanu: *mbolboceanu@bitdefender.com*
Sorin Boloș: *sorin.bolos@transilvania-quantum.com*
Adrian Coleșa: *acolesa@bitdefender.com*
Andrei Kisari: *akisari@bitdefender.com*
Andrei Luțaș: *vlutas@bitdefender.com*
Dan Luțaș: *dlutas@bitdefender.com*
Radu Mărginean: *radu.marginean@transilvania-quantum.com*
Andrei Muntea: *amuntea@bitdefender.com*
Radu Portase: *rportase@bitdefender.com*
Miruna Roșca: *mrosca@bitdefender.com*

Public resources

`https://github.com/Transilvania-Quantum/quantum-computing-security-investigations`