Towards an Antivirus for Quantum Computers

Sanjay Deshpande
Yale University
New Haven, CT, USA
sanjay.deshpande@yale.edu

Chuanqi Xu
Yale University
New Haven, CT, USA
chuanqi.xu@yale.edu

Theodoros Trochatos
Yale University
New Haven, CT, USA
theodoros.trochatos@yale.edu

Yongshan Ding
Yale University
New Haven, CT, USA
yongshan.ding@yale.edu

Jakub Szefer
Yale University
New Haven, CT, USA
jakub.szefer@yale.edu

***Abstract*—Researchers are today exploring models for cloud-based usage of quantum computers where multi-tenancy can be used to share quantum computer hardware among multiple users. Multi-tenancy has a promise of allowing better utilization of the quantum computer hardware, but also opens up the quantum computer to new types of security attacks. As this and other recent research shows, it is possible to perform a fault injection attack using crosstalk on quantum computers when a victim and attacker circuits are instantiated as co-tenants on the same quantum computer. To ensure such attacks do not happen, this paper proposes that new techniques should be developed to help catch malicious circuits before they are loaded onto quantum computer hardware. Following ideas from classical computers, a compile-time technique can be designed to scan quantum computer programs for malicious or suspicious code patterns before they are compiled into quantum circuits that run on a quantum computer. This paper presents ongoing work which demonstrates how crosstalk can affect Grover's algorithm, and then presents suggestions of how quantum programs could be analyzed to catch circuits that generate large amounts of crosstalk with malicious intent.**

I. Introduction

Researchers across the world are today working towards building large-scale quantum computers, e.g., IBM most recently announced their 127-Qubit Quantum processor called eagle in 2021 [1]. In parallel research, several schemes are being proposed for sharing quantum computer hardware following ideas of multi-tenancy [2], [3], [4]. These schemes envision throughput optimization by allowing multiple users to run their circuits on the same quantum computer, but using different qubits. Although this idea has a good potential to improve utilization of quantum computer resources, it opens up quantum computers to security threats due to crosstalk errors [5].

Crosstalk errors can occur when one or more qubits are activated, which adversely affects the state of other nearby qubits [6]. If the former qubits are controlled by an attacker circuit, and the latter belong to a victim circuit, then the attacker can affect computational results of the victim [7].

As this work-in-progress proposes, it may be possible to achieve both multi-tenancy and security by introducing techniques to catch malicious quantum computer programs during compilation, before they are able run on the quantum computer. In particular, quantum computer users today develop their designs in high-level programming languages or in quantum assembly language, which are then transpiled using tools such as Qiskit[1] into circuits that are scheduled and eventually loaded onto quantum computer hardware for execution. During this transpilation process, we propose that antivirus-like software can scan the user's programs and locate any malicious or suspicious code or circuit patterns. Based on existing work [7] and our exploration, quantum `CNOT` gates can be used to generate crosstalk errors. Following this, antivirus-like software can be used to scan if quantum programs contain suspicious patterns of `CNOT` gates.

II. Background and Related Work

Existing quantum computers are prone to different errors such as single-qubit and two-qubit gate errors, decoherence errors, crosstalk errors, and readout errors [8].

This work was supported in part by NSF grant 1901901.

[1]Qiskit is an open-source software development kit for working with quantum computer programs, available at: https://qiskit.org/.

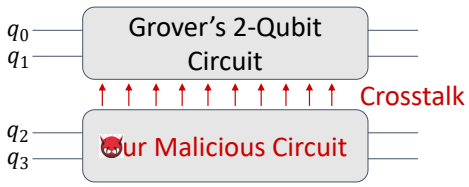

Fig. 1. Block diagram of the experimental setup. A 5-qubit quantum computer from IBM, namely the `ibmq_lima`, is used to evaluate crosstalk effects of different attacker circuits on 2-qubit Grover's algorithm. Note that the fifth qubit is unused in this example and only two qubits are needed for victim and two more for the attacker.

Out of these errors, crosstalk errors are of special interest as they lead to interesting security vulnerabilities [5]. The work from Ghosh et al. [7] shows that the crosstalk errors could be used in a fault injection attack. It also showed how an adversary can launch a denial of service attack on a victim circuit using crosstalk errors.

To counter such threats threats, existing work proposes, for example, execution of the target circuit and performing measurements to determine if the target circuits was affected by crosstalk errors from a specific attacker circuit [9]. In contrast, our work-in-progress proposes that malicious circuits could be caught at compile time using an antivirus-like approach, before any circuit is executed on a quantum computer.

III. Defining and Testing Potential Malicious Quantum Computer Circuits

To design an antivirus program, we first need to define what a virus is to quantum computers. We define a virus to be a malicious circuit that causes the crosstalk errors and affects the output fidelity of another target victim circuit. However, as with classical computer antivirus, the definitions of the virus can be updated and broadened over time as new understanding of potentially malicious circuits is developed. As such, we propose to keep a dynamic library of virus quantum circuits. When a potential attacker circuit is evaluated, we label it as a virus circuit if it indeed causes crosstalk errors in an adjacent victim co-tenant. Patterns of such virus circuits can then be added to a database that the antivirus software uses when scanning for malicious attackers.

A. Finding Malicious Circuit Types

As a preliminary demonstration, we use the 5-qubit IBM quantum computer `ibmq_lima` [10] for conducting our experiments. The attack example is shown in Figure 1. We use Grover's 2-qubit algorithm circuit [11] as the target victim circuit which is fixed on qubits q_0 and q_1. For the malicious circuit we use qubits q_2 and q_3, and we define different series of `CNOT` gates as potential malicious circuits, following preliminary work by Ghosh et al.'s work [7] which used a series of `CNOT` as an attack circuit to perform fault injection attack.

We observe that without active attack, the output probability of the Grover's circuit is $P_{original} = 0.87$. Interestingly, when the attacker circuit is a pure sequence of `CNOT` gates, we do not notice significant reduction in the $P_{original}$. The reason for that is, the transpile function when ran in optimization mode decomposes all `CNOT` gates and converts them in to a delay value equivalent to all the `CNOT` gates that are connected in series. If the number of `CNOT` gates connected in series are even, the series of gates is converted into delay and scheduled at the beginning of the operation and if it is odd then it is converted in to delay and one `CNOT` gate connected in series and scheduled at the beginning of the operation. Interestingly, this result is in contrast to [7], which did not seem to consider such existing transipler optimizations and only tested a pure sequence of `CNOT` gates.

In order to bypass the optimizations, we add delay gates [12] with minimum delay in between each `CNOT` gate in the malicious circuit as shown in the Figure 2. And we noted that with this approach we were able to bypass the transpile optimizations and preserve the `CNOT` gates in the malicious circuit. By running this circuit on the quantum device, the output probability value of target circuit is lower than $P_{original}$.

We define two parameters to perform further analysis and find more malicious circuits. The first parameter is the delay value from the delay gate [12]. The delay unit we use is dt which translates to $2/5$ nanoseconds on the IBM quantum computers. The second parameter is K that defines the number of gates in the malicious circuit. These malicious circuits are ran alongside Grover's 2-qubit algorithm in all cases. In the plots, the `CNOT` gates are labeled as CX, while the other gates are the Pauli gates X, Y, and Z, and the I gate. The delay gates are labeled as $Delay(i)$, where i is the delay of the delay gate given in units of dt.

Below we present the experiments performed for three cases of possible malicious circuits: series of `CNOT` and delay gates in an interleaved fashion, as discussed before, series of only delay gates, and series of interleaved Pauli and delay gates. In all the cases we ran experiments by varying K from 0 to 5000, but for brevity we only plot results for K from 0 to 300.

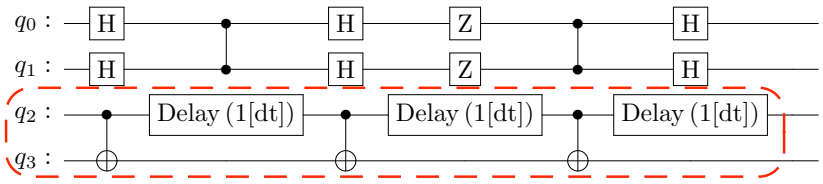

Fig. 2. Circuit diagram for one possible attacker circuit with series of interleaved CNOT and delay gates. The attacker circuit is outlined with a red dotted line, while the other circuit on qubits q_0 and q_1 is the victim Grover's circuit.

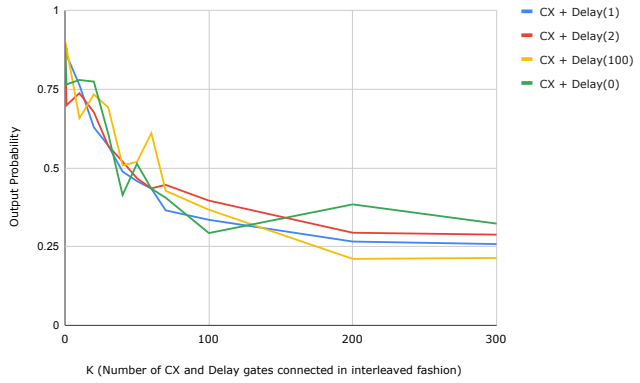

Fig. 3. K vs. Output Probability plot where malicious circuit is CNOT (labled CX in the figure) and delay gates connected in an interleaved fashion.

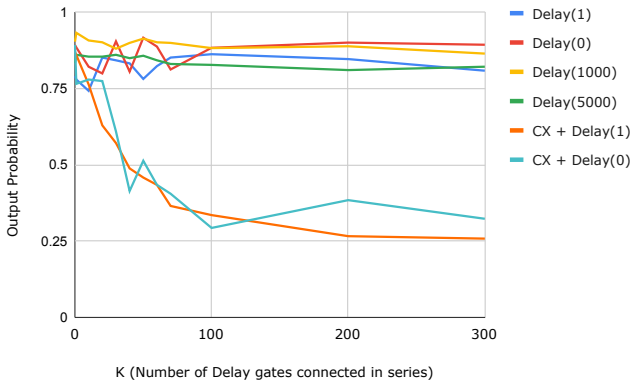

Fig. 4. K vs. Output Probability plot where malicious circuit is series of delay gates, delay length in units of dt is given in the parenthesis in the legend; data for CNOT gate experiments is shown for comparison.

Considering CNOT gates interleaved with delay gates, we ran experiments by fixing delay value to 1 and varying K and observe that as K increase the output probability of target circuit decreases and it saturates around 0.2 as shown in Figure 3. We ran similar experiments by fixing delay values to 2 and 100 and notice similar results. We also tried the delay value to be 0 (i.e. no delay) and we notice that the transpiler optimization can be tricked even with the zero delay value and this has the same effect on the output probability as in cases of delay equal 1, 2, 100, as shown in Figure 3.

Considering only delay gates, we used the delay gates as malicious circuits to observe their effect on the target circuit. We ran experiments by varying K and observed that there is not any significant change in the output probability of the target circuit as shown in Figure 4. The reason for this is that the transpile function identifies all the unwanted delays and schedules them at the beginning of the circuit and performs the operation of target circuit at the end. We also plot the results where the malicious circuit is series of CNOT gates and delay gates in the same plot to provide a comparison.

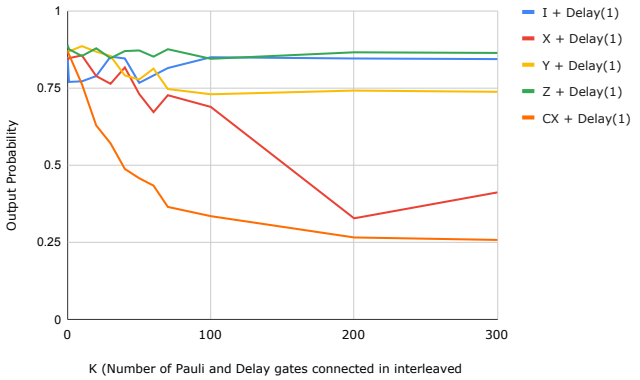

Fig. 5. K vs. Output Probability plot where malicious circuit is Pauli gates and delay gates connected in interleaved fashion; data for CNOT gate experiments is shown for comparison.

Considering Pauli and delay gates, we used the series of Pauli and delay gates connected in interleaved fashion as malicious circuits. We ran experiments by varying K and observed that there is some change in the output probability of the target circuit in case of X and Y gates

(not as much as in case where malicious circuit is series CNOT and delay gates) and we also observe there is not any significant change in output probability in case of I and Z gates as shown in Figure 5. We plot the results where the malicious circuit is series of CNOT gates and delay gates with delay value 1 in the same plot to provide a comparison.

IV. Towards an Antivirus for Quantum Computers

The preliminary results show that the malicious circuits can be series of CNOT gates interleaved with delay gates. They can also be circuits with interleaved gates such as Pauli X and Y gates, and thus can form a large or even an infinite malicious circuit set. On the other hand, pure delay gates, or I and Z gates do not produce noticeable crosstalk errors. Further, pure sequence of CNOT gates is optimized away, so it does not induce crosstalk errors in the victim circuit. The former patterns that can be malicious could be used as the initial set of virus patterns that should be detected by the antivirus during the transpilation process.

A. Quantum Computer Antivirus as a Qiskit Extension

In order to detect and eliminate harmful circuits, the currently used Qiskit framework for developing quantum computer programs can be extended with the antivirus and pattern matching features.

As one possibility, an algorithm can be developed to count the number of appearances of each pattern in the quantum circuit. With a database of malicious patterns, the antivirus software can scan the user's code to find and count occurrences of the patterns. For example, as we have identified, when the value K is low, there is low impact on the output probability of the victim Grover's circuit. As result if in a circuit being scanned there is only one CNOT gate followed by a delay, i.e. $K = 1$, then this is likely not a cirucit that could be malicious. But if the count of CNOT gate followed by a delay is greater than 5 or 10 this could be judged as possibly malicious circuit.

To realize the algorithm, both the quantum circuit and the patterns can be described by a set of Qasm instructions [13]. Therefore, the problem of quantum circuit matching can be reduced to find the instruction pattern in an instruction list. Compared with conventional string matching problem, the complexity of quantum circuit matching problem lies in the additional qubits (and possibly the classical bits and other parameters) for each instruction. Instructions that are not adjacent in the source code may still be gates in a series as well. There also may be multiple patterns dispersed among different qubits and in different order. Different patterns may cross with each other as well. All these challenges need to be addressed when creating a functional antivirus software for quantum computers.

V. Conclusion

This work proposes that multi-tenant quantum computers can be protected by developing mechanisms for scanning user's circuits for malicious patterns. By using an antivirus for preventing attackers from instantiating circuits which could be causing harmful crosstalk errors, secure multi-tenant cloud-based quantum computers can be realized.

References

[1] "IBM Unveils Breakthrough 127-Qubit Quantum Processor," https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor, accessed Jan. 2022.

[2] X. Dou and L. Liu, "A new qubits mapping mechanism for multi-programming quantum computing," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques*, 2020, p. 349–350.

[3] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," in *Proceedings of the International Symposium on Microarchitecture*, 2019, p. 291–303.

[4] L. Liu and X. Dou, "Qucloud: A new qubit mapping mechanism for multi-programming quantum computing in cloud environment," in *Proceedings of the International Symposium on High-Performance Computer Architecture*, 2021, pp. 167–178.

[5] A. A. Saki, M. Alam, and S. Ghosh, "Experimental characterization, modeling, and analysis of crosstalk in a quantum computer," *IEEE Transactions on Quantum Engineering*, vol. 1, pp. 1–6, 01 2020.

[6] K. M. Rudinger, M. Sarovar, D. Langharst, T. J. Proctor, K. Young, E. Nielsen, and R. J. Blume-Kohout, "Classifying and diagnosing crosstalk in quantum information processors." https://www.osti.gov/biblio/1513744, accessed Jan. 2022.

[7] A. Ash-Saki, M. Alam, and S. Ghosh, "Analysis of crosstalk in NISQ devices and security implications in multi-programming regime," in *Proceedings of the International Symposium on Low Power Electronics and Design*, 2020, p. 25–30.

[8] Y. Ding and F. T. Chong, "Quantum computer systems: Research for noisy intermediate-scale quantum computers," *Synthesis Lectures on Computer Architecture*, 2020.

[9] M. Sarovar, T. Proctor, K. Rudinger, K. Young, E. Nielsen, and R. Blume-Kohout, "Detecting crosstalk errors in quantum information processors," *Quantum*, vol. 4, p. 321, 9 2020.

[10] "ibmq_lima," https://quantum-computing.ibm.com/services?services=systems&system=ibmq_lima, accessed Jan. 2022.

[11] "Qiskit, Grover's algorithm," https://qiskit.org/textbook/ch-algorithms/grover.html, accessed Jan. 2022.

[12] "Qiskit, Delay," https://qiskit.org/documentation/stubs/qiskit.circuit.Delay.html, accessed Jan. 2022.

[13] "Qiskit, Qasm," https://qiskit.org/documentation/apidoc/qasm.html, accessed Jan. 2022.