Design of Quantum Computer Antivirus

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

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

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

Hanrui Wang
EECS
MIT
Cambridge, MA, USA
hanrui@mit.edu

Ferhat Erata
Electrical Engineering
Yale University
New Haven, CT, USA
ferhat.erata@yale.edu

Song Han
EECS
MIT
Cambridge, MA, USA
songhan@mit.edu

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

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

2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST) | 979-8-3503-0062-8/23/$31.00 ©2023 IEEE | DOI: 10.1109/HOST55118.2023.10133711

***Abstract*—The development of quantum computers has been advancing rapidly in recent years. In addition to researchers and companies building bigger and bigger machines, these computers are already being actively connected to the internet and offered as cloud-based quantum computer services. As quantum computers become more widely accessible, potentially malicious users could try to execute their code on the machines to leak information from other users, to interfere with or manipulate results of other users, or to reverse engineer the underlying quantum computer architecture and its intellectual property, for example. To analyze such new security threats to cloud-based quantum computers, this work first proposes and explores different types of quantum computer viruses. This work shows that quantum viruses can impact outcomes of Grover's search algorithm or machine learning classification algorithms running on quantum computers, for example. The work then proposes a first of its kind quantum computer antivirus as a new means of protecting the expensive and fragile quantum computer hardware from quantum computer viruses. The antivirus can analyze quantum computer programs, also called circuits, and detect possibly malicious ones before they execute on quantum computer hardware. As a compile-time technique, it does not introduce any new overhead at run-time of the quantum computer.**

***Index Terms*—quantum computers, viruses, quantum computer viruses, antivirus**

I. Introduction

Quantum computing is an exciting new paradigm of computation that offers the potential to solve problems which are intractable on classical digital computers, e.g., integer factorization using Shor's Algorithm [1]. To realize this potential, researchers worldwide are racing to build bigger and bigger quantum computers. Already, 127-qubit quantum computers are available [2], and IBM, one of the quantum computing vendors, has projected to build 1000-qubit computers by 2023 [3]. Similar projections about upcoming quantum computer sizes have been made by others [4] as well. In addition to increasing sizes, these machines are now easily accessible to anybody through IBM Quantum or other similar cloud-based services such as Amazon Braket or Microsoft Azure.

While much research is focusing on the functional aspects of the quantum computers, there is little research thus far on their security. Different from classical computers, *quantum computers are particularly susceptible to low-level attacks.* The reason is two-fold – firstly, current quantum systems exposes intimate hardware instructions access to their users. Most quantum computers follow the quantum circuit model, where programmers have direct access to the hardware elements (qubits) and their control mechanism (e.g., quantum gates or pulses). An adversarial user could take advantage of this to exploit or harm the hardware. Secondly, a fundamental feature of quantum systems is its sensitivity to environmental disturbance or control noises.

Consequently, in this work, we explore potential vulnerabilities of quantum computers to malicious programs, and then how to protect them. On the attack side, we show through our evaluation that attackers can deliberately lower the success probability of victim's circuits, for instance, resulting in wrong outcomes of the Grover's search algorithm [5], misclassifications in quantum machine learning algorithms [6], or wrong results in molecular energy calculations for drug discovery [7]. On the defense side, to help mitigate the threat of virus circuits, we propose the first *quantum computer antivirus* which protects the quantum computers by analyzing whether an input quantum computer program, also called a circuit, may be a virus circuit. We also develop the first database of possible quantum computer viruses, and the antivirus can be extended in future with new or improved databases as new types of viruses are discovered.

Although the antivirus is sufficient to identify the virus circuits, an attacker who has access to the system on which the antivirus program is running may bypass it, e.g., by modifying the compilation software. Consequently, we also propose and demonstrate adding a protection by running the antivirus inside the Trusted Execution Environment (TEE) of Intel SGX [8]. A TEE, such as realized by Intel SGX enclaves, helps in increasing the application security by using isolation technology [9] to protect programs from other software and even the operating system or the hypervisor. This way, a malicious user cannot attack the enclave and bypass the antivirus. The program

compiled and checked by the antivirus is digitally signed by the SGX hardware, so that the cloud-based quantum computer provider can authenticate that the program was generated and checked correctly. At the same time, because users can run the antivirus locally inside SGX, they do not have to expose the source code of their quantum computer circuits to the quantum computer provider, protecting user's intellectual property.

A. Contributions

The contributions of this work are:

- Demonstrate sample crosstalk-based quantum computer viruses, using 10 types of attacker circuits which could affect operation of victim circuits.
- Quantify the impact of the attacks on different variants of 5 quantum computer victim programs, ranging from Grover's search to Machine Learning.
- Propose a method to detect such virus circuits in quantum programs by expressing both input circuits and malicious virus circuits as graphs, and formulating the virus checking as a sub-graph isomorphism finding problem which can be quickly solved with existing algorithms.
- Evaluate the performance of the antivirus with and without use of Intel SGX enclaves.
- Demonstrate the antivirus to have no false positives on our data set, and zero-overhead at run-time of quantum computers.

II. Principles of Quantum Computers

The bit is the most basic unit in modern computing, and quantum computing uses an analogous concept, the quantum bit, or qubit for short. The qubit has two basis states $|0\rangle$ and $|1\rangle$ but the qubit can be in a state other than $|0\rangle$ or $|1\rangle$. Every qubit $|\psi\rangle$ can be represented as a linear combination of these two states:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle,$$

where α and β are complex coefficients satisfying $|\alpha|^2 + |\beta|^2 = 1$.

Two-dimensional vectors can be used to represent one-qubit quantum states. For example, $|0\rangle = [1\ 0]^T$ and $|1\rangle = [0\ 1]^T$, and then a qubit can be represented as $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = [\alpha\ \beta]^T$. Similarly, there are four basis states for two-qubit quantum states, $|00\rangle$, $|01\rangle$, $|10\rangle$ and $|11\rangle$, and thus two-qubits quantum states can be represented by four-dimensional vectors. More generally, n-qubit quantum states can be represented by 2^n-dimensional vectors.

Quantum computation is realized by applying a series of quantum gates on the input qubits. Quantum gates are unitary operations which take one quantum state to another quantum state, i.e. $|\psi\rangle \rightarrow U|\psi\rangle$, where U is a unitary matrix. One-qubit and two-qubit gates are the most common quantum gates, because any n-qubit quantum gates can be decomposed into them. One-qubit gates are represented by 2×2 matrices and two-qubit gates are represented by 4×4 matrices. For example, the Hadamard gate is a one-qubit gate, taking the $|0\rangle$ state to the $|+\rangle = (|0\rangle + |1\rangle)/\sqrt{2}$ state and taking the $|1\rangle$ state to the $|-\rangle = (|0\rangle - |1\rangle)/\sqrt{2}$ state. `CX` gate is a two-qubit gate. If the most significant qubit is the control qubit and the least significant qubit is the target qubit, then `CX` flips the target qubit when the control qubit is $|1\rangle$ and leaves the target qubit unchanged otherwise, i.e., it takes $|0x\rangle$ to $|0x\rangle$ and $|1x\rangle$ to $|1\bar{x}\rangle$, where $x \in 0, 1$ and $\bar{1} = 0, \bar{0} = 1$. In their matrix representation, Hadamard (`H`) gate and controlled-NOT `CX` gate can be written as:

$$\texttt{H} = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix},\ \texttt{CX} = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{bmatrix}$$

For publicly accessible quantum computers, typically only a few quantum gates are supported natively. For example, in current IBM quantum computers there are: four one-qubit gates (`I`, $\texttt{R}_\texttt{z}$, $\sqrt{\texttt{X}}$, and `X`) and one two-qubit gate (`CX`). Any other gate needed by a program, e.g., Hadamard gate, must be decomposed to these native gates.

A. Device Topologies

Qubit connectivity is an important characteristic of a quantum device. In the case of superconducting quantum architectures, a pair of qubits are linked by circuit wires (via a coupler such as a capacitor) to allow interactions between the pair. We can only apply two-qubit gates on qubits that are directly coupled; interactions between any remote pair of qubits requires moving them to be next to each other via (logical) swapping of qubits. We use a device topology to describe the layout of the physical qubits. Typically, all-to-all topology is challenging to implement and manufacture in superconducting architectures. In fact, sparse topology such as 2D mesh or lattice graph offers fewer connections between qubits but higher control precision and milder crosstalk noise. An example of topologies of some of the IBM quantum computers are shown in Figure 1.

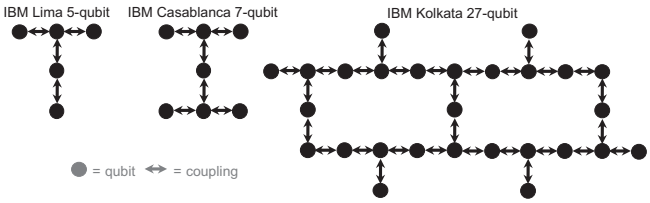


Fig. 1: Sample topologies of IBM quantum computers. Observe that only certain qubits are coupled together. Lima and Casablanca are examples of computers freely available to researchers, while Kolkata is a bigger chip only available with special access. Note that Kolkata and all the larger IBM quantum computers today follow a similar qubit connectivity pattern by repeating the basic pattern of arranging the qubits in the rectangle shape, which can be tiled to form even bigger machines.

B. Noise and Crosstalk in Quantum Computers

Noise in quantum computers can be attributed to two classes of errors: *idle errors* and *operational errors*. Idle errors, such as qubit decoherence and idle crosstalk, are spontaneous errors

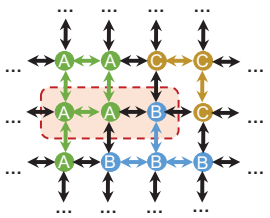


Fig. 2: Example diagram of three users, A, B, and C, being assigned a disjoint set of qubits within a quantum computer. Each user is assigned qubits and couplings between the qubits. Couplings between different users are inactive but physically present, as they cannot be dynamically reconfigured in current superconducting quantum devices. Red dashed lines highlight a potential scenario where 2 qubits from user A can cause crosstalk errors in a qubit of user B.

that happen when qubits interact with each other or with the surrounding environment. Specifically, *qubit decoherence errors* are due to the interaction of the qubits with the environment, resulting in a lost of information stored in the qubits. *Idle crosstalks* are always-on unwanted ZZ-type interactions between qubits due to residual resonance in hardware. In contrast, operational errors are not spontaneous; they happen when we apply quantum control, gates, or measurements with imprecision. For instance, *gate errors* are due to imprecision in the control pulses for some quantum gates, such as the Hadamard gate and `CX` gate. *Readout errors* are errors that occur in measurement operations that affect the probability distribution of the outcome. *Crosstalk errors* can also happen as a result of gate operations – when gates are performed on one or two target qubits, the effect of the gates can propagate to unintended nearby qubits. This can be detrimental to the fidelity of the affected qubits and that of the gates on them.

C. Multi-tenant Quantum Computers

An important trend in the quantum computing is the scaling of the number of qubits. Although existing computers offer up to 127 qubits, hardware with more than 200 qubits is predicted to become a reality in less than 5 years [10], and IBM is projecting 1000-qubit quantum computers soon as well [3]. Such resourceful computers can be used to solve large scale problems or to solve multiple smaller problems to maximize profit in a cloud-based quantum service model. Initial proposals on multi-tenant quantum computing already exist in literature [11]. The ideas for the multi-tenant environment in quantum computers are similar to cloud servers in classical computing, where programs from various users may run on the same server simultaneously [12]. Each program may use different copies of computational blocks or the same copy in a time-multiplexed manner. Similarly, IBM already allows for cloud-based access to time-multiplexed quantum computers, but no multi-tenancy is implemented. When multi-tenancy is enabled, not only time-sharing, but spatial sharing will be possible, with two or more quantum programs running simultaneously on different sets of qubits within a quantum computer; an example is shown in Figure 2. Crosstalk can present a challenge to such a computing model, where different programs (running on adjacent qubits) may generate excessive crosstalk to affect other programs in the spatially shared quantum computer.

III. Threat Model

This work assumes that users with gate-level or pulse-level access to a quantum computer attempts are able to submit quantum programs or circuits to be executed on a target machine. It is assumed the programs are compiled by a trusted compiler before they are allowed to execute on the quantum computer. Today, for example, all programs submitted to IBM are compiled by their compiler using Qiskit framework. In the future, if users do not fully trust the quantum computer provider, they could compile the circuits locally inside a trusted execution environment, such as SGX, and send the compiled circuits plus attestation that it was generated using an approved (by the quantum computer providers) compiler.

We assume that potentially malicious users could try to execute their code on the machines to leak information from other users, to interfere with or manipulate results of other users, or to reverse engineer the underlying quantum computer architecture and its intellectual property, for example. Or that virus circuit could be part of a library that a benign user unintentionally uses. We assume the circuits are expressed at the gate-level or pulse-level, as is common today.

This work considers both single-tenant and multi-tenant or multi-programming settings. In the single-tenant setting, currently available from commercial quantum computers, the malicious quantum computer circuit could be hidden as part of library or third-party code downloaded by users, for example. Or the user could directly try to execute virus circuits. In the multi-tenant setting, the malicious quantum computer circuit could be executed by one user, while the victim circuit is executed concurrently by another user. While multi-tenant quantum computers are not available today, initial proposals on multi-tenant quantum computing already exist in literature [11]. In either setting, we assume the objective of the virus circuit is to leverage crosstalk to affect computation of the victim circuit, such as to get the victim circuit to generate incorrect results, or to leak information from the victim circuit by observing how the victim's operation affects the malicious user. Or the malicious goal can be to perform frequency sweep or other operations to find out details about the underlying hardware.

IV. Attack Evaluation

One of the main weaknesses of quantum computers is the crosstalk. Thus, our work focuses on crosstalk as one of the main attack vectors. To perform the crosstalk evaluation, we use the experimental setup described in Figure 3. We adapt our setup from [12]. We evaluate five benchmark circuits described in Section IV-A against ten variants of virus circuits described in Section IV-B. We use two different IBM quantum computers to perform our evaluation based on the size of our virus circuits. For virus circuits with one and two qubits, we use the 5-qubit IBM quantum computer `ibmq_lima` and for

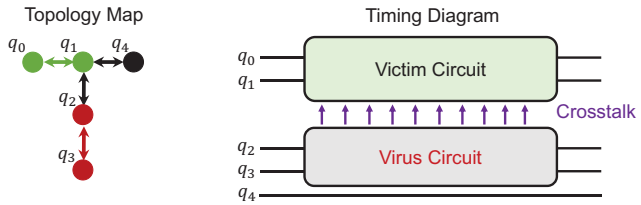


Fig. 3: Topology map of IBM Lima machine and timing diagram of a crosstalk attack example. The example shows a 2-qubit victim circuit on the green qubits and a 2-qubit attacker circuit on the red qubits. Although the couplings between victim and attacker qubits are physically present, they are not used to emulate a multi-tenant setup and isolation of the two users. Note the fifth qubit is idle in this example as only four qubits total are needed by the victim and attacker.

three and four qubit virus circuits, we use the 7-qubit IBM quantum computer `ibmq_casablanca`. Topologies of these computers were previously shown in Figure 1.

A. Benchmark Victim Circuits

This section introduces a number of important quantum algorithms, shown in Figure 4, which we use as benchmark victim circuits to analyze the impact of the attacker circuits on the outputs generated by these circuits.

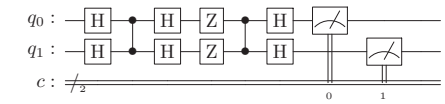
a) Grover's Algorithm:: Grover's algorithm demonstrates the capability of providing quadratic speed-up in an unstructured search. An unstructured search can be defined as follows, given a large list of items ($\mathtt{N} = 2^n$, where n is a number of bits) arranged randomly, among these items is one item `X` that we want to locate. To locate `X` using a classical computation, one would have to check $\mathtt{N}/2$ items on average or `N` in worst case scenario. Using Grover's algorithm, the search complexity reduces to $\sqrt{\mathtt{N}}$. More details on the algorithm can be found at [5].

b) Deutsch-Jozsa Algorithm:: In the Deutsch-Jozsa algorithm [13], we are given an unknown Boolean function f that takes n-bit input and generates 1-bit output. We are promised that f is either 'balanced' or 'constant' – A constant f always returns 0 or always returns 1 for any inputs, whereas a balanced f returns exactly half 0's and half 1's. The task here is to determine whether the function is balanced or constant.

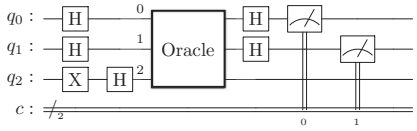
c) Bernstein-Vazirani Algorithm:: The Bernstein-Vazirani algorithm [14] is an extension of the Deutsch-Jozsa algorithm. Rather than finding the type of the function, this algorithm returns a bitwise product of the n-bit input with some n-bit binary string 's' (where 's' is unknown). On a classical computer this would take n function calls to retrieve the 's' completely, whereas on a quantum computer, this could be solved with one call to the function.

d) VQE:: The Variational Quantum Eigensolver (VQE) is a kind of variational algorithm that computes the ground state energy of a molecule. A variational circuit is a trainable quantum circuit where its quantum gates are parameterized (e.g., by angles in quantum rotation gates). The parameterized quantum circuit $\Phi(x, \theta)$ is used to prepare a variational quantum state: $|\psi(x, \theta)\rangle = \Phi(x, \theta)\,|0 \ldots 0\rangle$, where x is the input data related to the computation and θ is a set of free variables for adaptive optimizations. During parameter training, the objective is to minimize the expectation value of the ground state energy. For demonstration purpose, we implement the VQE for H_2 molecule. The circuit uses two qubits with a series of U3 and CU3 gates as the trainable layers for quantum state preparation.

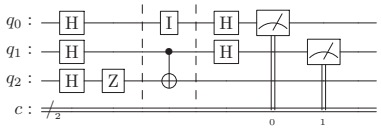
e) QML:: Quantum machine learning (QML) explores performing ML tasks on quantum devices. For demonstration purpose, we implement the Quantum Neural Networks (QNN) for MNIST handwriting digits image classification tasks. The encoder uses rotation gates to embed the pixel values in qubits. Then we use U3 and CU3 gates as the trainable gates. The measurements are performed on the Z-basis and the expectation values are sent to a softmax function to obtain the probability of each digit class. The MNIST 2-classification circuit requires 2 qubit and the 4-classification



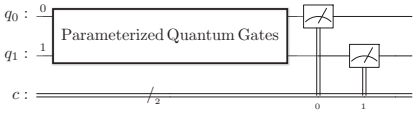
(a) Circuit for 2-qubit Grover's algorithm.



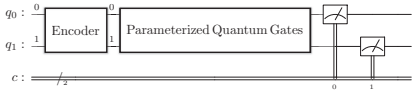
(b) Circuit for 2-qubit Deutsch-Jozsa algorithm, details of the Oracle are not shown due to limited space.



(c) Circuit for 2-qubit Bernstein-Vazirani algorithm.



(d) Circuit for 2-qubit VQE algorithm, details of the Parametrized Quantum Gates are not shown due to limited space.



(e) Circuit for 2-qubit QNN algorithm, details of the Encoder and Parametrized Quantum Gates are not shown due to limited space.

Fig. 4: Timing diagrams of the benchmark victim circuits, later used to demonstrate the effectiveness of different virus circuits' ability to affect output of these victim circuits through crosstalk. The horizontal lines labeled q represent qubits, and the lines labeled c represent the output classical bits read out from the qubits

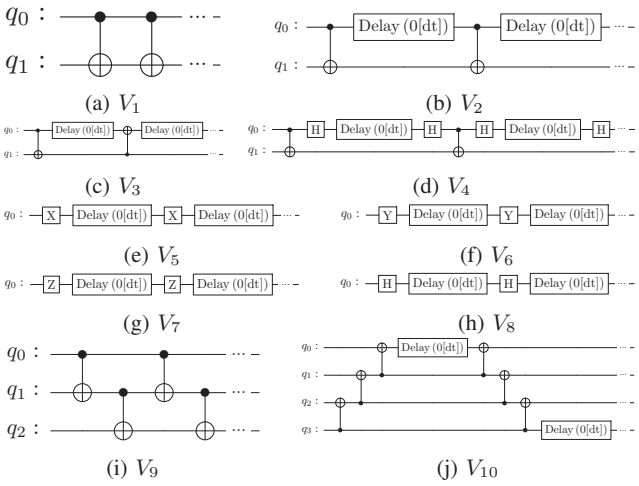


Fig. 5: Virus circuits, each subfigure is shown with 2 copies of the corresponding virus circuit type.

requires 4 qubits.

B. Quantum Computer Virus Circuits

We define a *quantum computer virus circuit* as a circuit that does not necessarily perform any meaningful operation itself, yet its presence would affect (negatively) the output fidelity or success rate of another (victim) circuit that shares the quantum computer with it. Alternatively, the virus circuit could be one which measures its own fidelity to act as receiver of information leak due to the victim's operation. Lastly, it could also measure behavior of the underling hardware to leak information about the hardware's design. Due to limited space, we focus on the first kind of viruses, which aims to disrupt operation of the victim. As the output of a quantum program is the most valuable information, disrupting the output can have negative consequences, such as giving the user the wrong solution to a routing optimization problem, which may then have real-world impact of the user wasting energy or time using incorrect routing in package delivery, for example.

For our experiments, we define the 10 types of preliminary virus circuits which we refer as $V_1 - V_{10}$, due to space limitation they are summarized only as Figure 5. We do want to highlight the use of delay gates with 0 delay as means of tricking the current Qiskit compiler into not optimizing away the repeated gates used in the virus circuits.

C. Virus Attack Evaluation

For the evaluation, we only consider 5-qubit and 7-qubit quantum computers that are freely available to researchers. Consequently, we vary our selection of virus circuits between 1-qubit ($V_5 - V_8$), 2-qubits ($V_1 - V_4$), 3-qubits (V_9), and 4-qubits (V_{10}). We limit the maximum number of qubits used by the virus circuit to be four to be able to fit the victim circuit along with the virus circuit. Figure 6 shows an example of one of our experiments where we run our Grover's 2-qubit algorithm circuit shown in Figure 4a along with our virus circuit

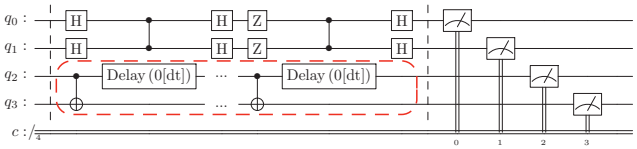


Fig. 6: Circuit diagram for 2-qubit Grover's Circuit alongside with our virus circuit V_2 (highlighted with dotted red line).

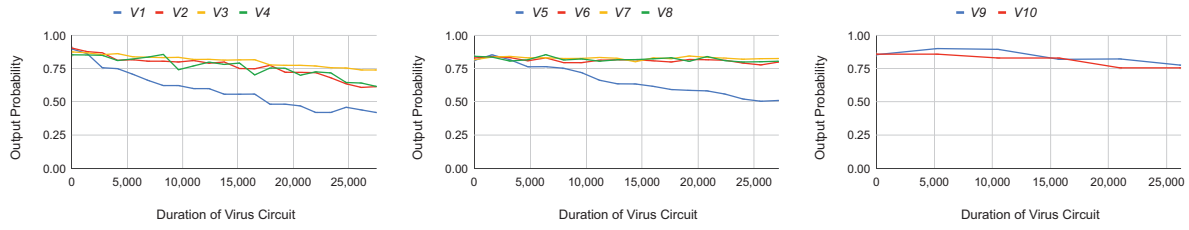
V_2 shown in Figure 5b. We use similar setup across all the benchmarks circuits alongside with all our virus circuits. We run all non-parameterized circuit experiments with $4,000$ shots and parameterized circuits (QNN and VQE) with 8,192 shots. QNN and VQE training leverages the TorchQuantum library and follows the setup of QuantumNAS [15].

We choose two different evaluation metrics and procedures to perform our evaluations:

1) For the benchmarks: Grover's, Deutsch-Jozsa, and Bernstein-Vazirani, we quantify the amount of crosstalk using the output probability. For these experiments, we use the experimental setup shown in Figure 6 and vary duration for each virus pattern (D_M) by varying the number of patterns in virus circuits. We limit the maximum value of D_M based on duration of benchmark circuit (D_B).
2) For the benchmarks: QML and VQE circuits, we quantify the amount of crosstalk based on the accuracy of the circuit. For these experiments, we first run benchmark circuits without any virus circuit (as shown in Figure 4e and Figure 4d) and note the accuracy of these circuits. Following that, we run these circuits alongside with virus circuit. We select the D_M value to be the maximum possible value (based on the value D_B) and note if the virus circuit has any effect on the accuracy of these benchmark circuits.

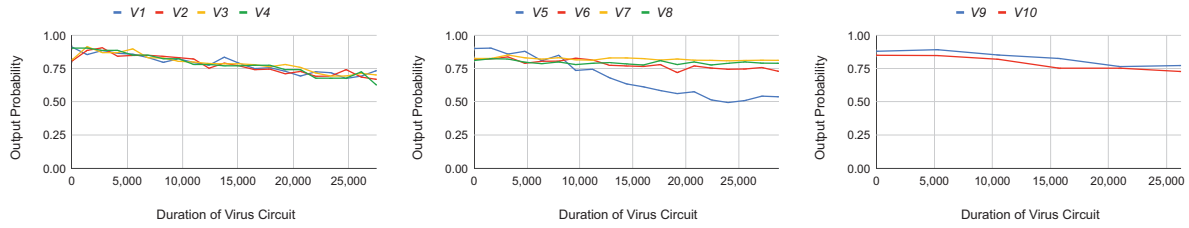
1) Attack Evaluation Results: For the evaluation of Grover's, Deutsch-Jozsa, and Bernstein-Vazirani, we group the results from 1-qubit, 2-qubit, and 3 & 4-qubit virus circuits together. Figure 7a, Figure 8a, and Figure 9a demonstrates the effectiveness of our 2-qubit virus circuits across different benchmark circuits. We make an observation that, as the duration of the virus circuit increases (i.e., as the number of gates in the virus circuit increase) the output probability of all the benchmark circuits is lowered. Figure 7b, Figure 8b, and Figure 9b show the results for our 1-qubit virus circuits. We observe that out of our four 1-qubit virus circuits only V_5 demonstrates the reduction in the output probability of the benchmark circuit.

Figure 7c, Figure 8c, and Figure 9c show the results for our 3-qubit and 4-qubit virus circuits. We observe that although there is some decrease in the output probability of the benchmark circuits it is not as much as we note in the case of the 2-qubit virus circuits. The reason for this is when the `CX` gates are connected step-wise as shown in Figure 5i and Figure 5j the duration of these circuits is longer than



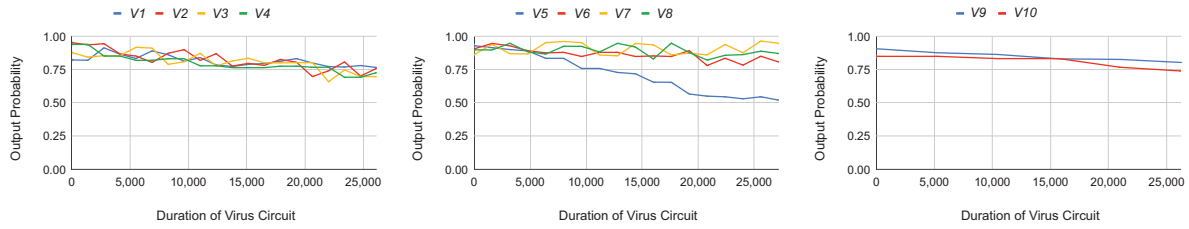
(a) (b) (c)

Fig. 7: Output probability for 2-qubit Grover's circuit vs. different duration of virus circuits. (a) Results for 2-qubit virus circuits (V_1, V_2, V_3 and, V_4), ran on `ibmq_lima` machine. (b) Results for 1-qubit virus circuits (V_5, V_6, V_7 and, V_8), ran on `ibmq_lima` machine. (c) Results for V_9 and V_{10}, ran on `ibmq_casablanca` machine.



(a) (b) (c)

Fig. 8: Output probability for 2-qubit Deutsch-Jozsa circuit vs. different duration of virus circuits. (a) Results for 2-qubit virus circuits (V_1, V_2, V_3 and, V_4), ran on `ibmq_lima` machine. (b) Results for 1-qubit virus circuits (V_5, V_6, V_7 and, V_8), ran on `ibmq_lima` machine. (c) Results for V_9 and V_{10}, ran on `ibmq_casablanca` machine.



(a) (b) (c)

Fig. 9: Output probability for 2-qubit Bernstein-Vazirani circuit vs. different duration of virus circuits. (a) Results for 2-qubit virus circuits (V_1, V_2, V_3 and, V_4), ran on `ibmq_lima` machine. (b) Results for 1-qubit virus circuits (V_5, V_6, V_7 and, V_8), ran on `ibmq_lima` machine. (c) Results for V_9 and V_{10}, ran on `ibmq_casablanca` machine.

when they are connected in series as in case of Figure 5a and Figure 5b. This limits the number of gates in the selected malicious pattern, since we reach the D_B with fewer gates. We note that the more `CX` are in the virus circuit pattern, the crosstalk effect is higher. Out of our ten virus circuits our most effective virus circuits are V_1 to V_5.

Figure 10 and Figure 11 show the results for VQE circuits with 2 qubits on the `ibmq_lima` (V_1-V_8) and `ibmq_casablanca` (V_9-V_{10}) machines with victim circuits of around 15,000 dt (short) and 45,000 dt (long) duration, respectively. The ground truth expectation value is -1.84. For the short circuit, 9 out of 10 virus circuits degrades the performance as the measured value is farther from the ground truth. For long circuit, 8 out of 10 kinds of virus circuits worsen the performance. We observe that the virus circuits with only single qubit gates (V_5-V_8) have less significant negative impact to the performance of the victim.

Figure 12 and Figure 13 show the results for QNN circuit with 4 qubits on the `ibmq_casablanca` machine with victim circuits of around 30,000 dt (medium) and 45,000 dt (long) duration, respectively. We did not experiment with V_{10} because that requires 8 qubits, exceeding the maximum number we can access (7 qubits). For the medium circuit, we observe that 8 out of 9 kinds of virus circuits degrade the accuracy of the QNN classification. The average accurate drop is 6.4%. For the long circuit, we observe that the accuracy is in general lower than the medium one because the noise-free simulation accuracy of the two circuits are similar, while more

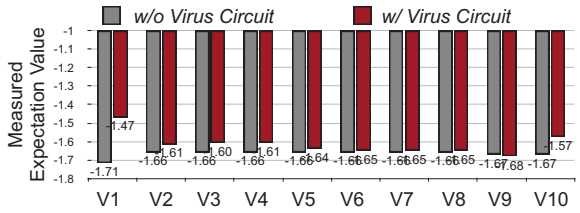


Fig. 10: VQE results for circuit of 15,000 dt duration, with different virus circuits. Lower value is more accurate.

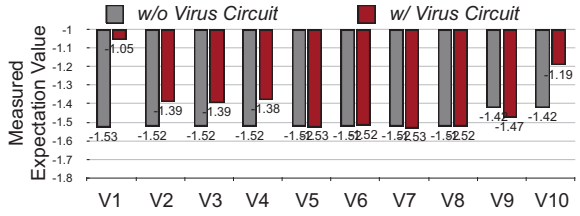


Fig. 11: VQE results for circuit of 45,000 dt duration, with different virus circuits. Lower value is more accurate.

gates in the longer circuits introduce additional noise sources. For the long victim circuit, all virus circuits can negatively impact the measured accuracy, by 3.3% on average. In addition to 4-classification task, we also perform experiments on 2-classification. However, the accuracy drop is not as significant because 2-classification accuracy is typically higher than 90%, which means a larger boundary between classes and better error resilience.

2) Attack Sensitivity to Qubit Mapping: The defense technique recommended in [12] against the crosstalk attacks is to leave an idle qubit between victim circuit and virus circuit. [12] suggests that this technique reduces the nearest neighbor crosstalk. To validate this recommendation, we performed experiments using one Grover's 2-qubit circuit (described in Section IV-A) as a victim circuit and virus circuit V_2 (described in Section IV-B) with duration D_B. We use 7-qubit IBM quantum machine `ibmq_casablanca` to perform this experiment with number of shots set to $8,192$. We first run only the victim circuit on the quantum computer (as shown in Figure 14 (a)) and observe the output probability to be 0.891. We then fix the coupling map to use qubits without any idle qubits between the victim circuit and virus circuit and measure the output probability as shown in Figure 14 (b) and we note that the output probability is 0.669 (output probability lowered due to the crosstalk). Following that, we add an idle qubit between victim and virus circuits and measure the output probability as shown in Figure 14 (c) and we observe the output probability to be 0.692. We note that even with the idle qubit in between the victim circuit and the virus circuit, reduction in output probability can still be observed. Furthermore, keeping one idle bit between adds overhead in the overall hardware required. Consequently, in Section V we introduce our antivirus, which rather than trying to isolate the circuits, helps find and prevent virus circuits from being

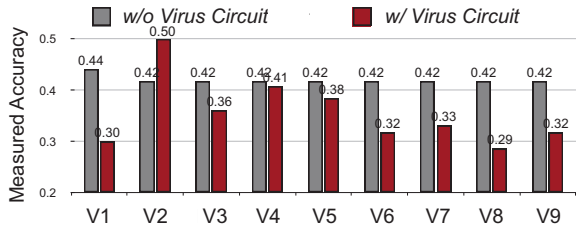


Fig. 12: QML results using circuit of around 30,000 dt duration, measured on `ibmq_casablanca`.

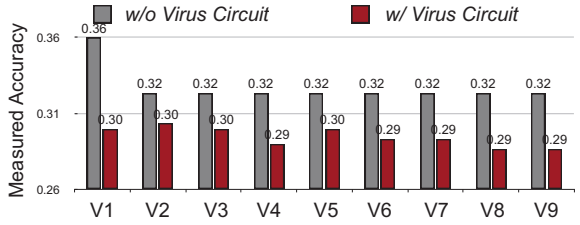


Fig. 13: QML results using circuit of around 45,000 dt duration, measured on `ibmq_casablanca`.

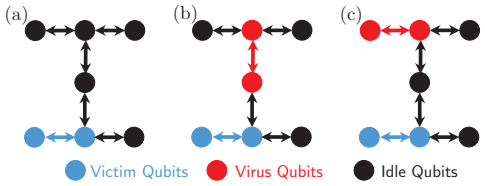


Fig. 14: (a) Coupling map for the experiment running *only* the benchmark victim circuit. (b) Coupling map for the experiment with virus circuit and victim circuit running alongside *without* an idle qubit between. (c) Coupling map for the experiment with benchmark circuit and victim circuits running alongside *with* an idle qubit between. Given topology is for `ibmq_casablanca`.

executed on the quantum computer in the first place.

For NISQ quantum computers, decoherence time is short and thus will also make a difference when circuits are long enough. Decoherence and crosstalk will both interfere to our results, especially when the jobs execute in different calibration periods, since the decoherence time, gate errors, and other physical properties may vary a lot. Our future direction is to obtain the results, considering and getting rid of the decoherence effect.

V. Quantum Computer Antivirus

As we have demonstrated in the prior sections, a virus circuit co-located with a victim circuit can abuse crosstalk to degrade the output fidelity of the victim circuit. We have evaluated a number of potential malicious patterns. As they are able to severely impact the victim circuits, we consider these malicious patterns as viruses – which should be detected and prevented.

The virus circuits can be detected at compile time using the antivirus, which we define and describe in this section. The core idea of the antivirus is to scan input quantum computer

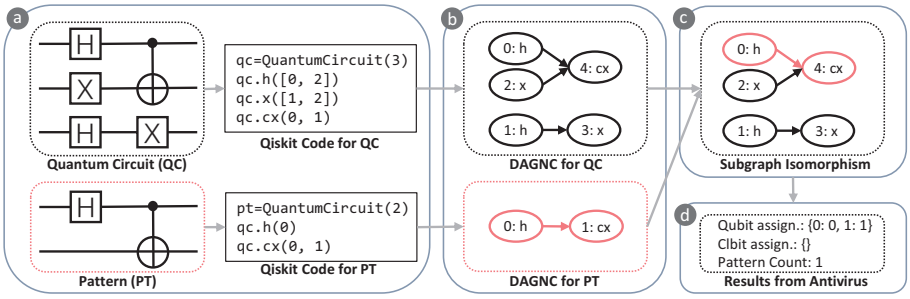


Fig. 15: Overview and workflow of the antivirus program. (a) Specify the quantum circuit (QC) and the pattern (PT) in Qiskit. (b) Convert the *QuantumCircuit* object to the directed acyclic graph with non-commutativity (DAGNC) representation. Vertices in DAGNC are gates in quantum circuits, and edges specify orders between gates. (c) Find patterns using subgraph isomorphism algorithm and check whether the found mappings are exact. (d) Analysis of the results. Qubit and clbit assignment is represented as a Python *Dict* object, whose keys (values) are qubit and clbit indices in QC (PT). Pattern count is the number of occurrences of found PTs in QC.

programs for occurrences of malicious patterns. If there is a considerable number of such patterns in the circuit, it will be classified as malicious. The number of occurrences of these patterns is an important indication of potential maliciousness of a quantum circuit.

A. Architecture of the Quantum Computer Antivirus

In this section, we introduce our approach for quantum computer antivirus. The idea is to firstly specify patterns that are considered to be malicious, such as those in Section IV-B, and then use a pattern matching algorithm to scan for malicious patterns in quantum circuits. In our antivirus program, a directed acyclic graph (DAG) is used to represent a quantum circuit, and thus the pattern matching problem can be reduced to the *subgraph isomorphism problem*, i.e., to find the subgraphs in one graph that are isomorphic to another graph. Our program can find the locations of all given patterns and the associated information, such as their qubits and classical bits.

In quantum computing, DAG has been previously used for gate optimization [16], [17], [18]. For instance, subsets of gates are substituted based on the commutation relations between gates to reduce the depth or number of gates. However, such optimizations (e.g., gate substitution and reordering) can significantly change the circuits' crosstalk effect and thus their maliciousness. To address this problem, the antivirus program should only consider independent gates, i.e., gates operating on disjoint qubits and classical bits, to be commutable, while commutation relations between other gates are neglected. As such, we leverage and modify the existing DAG representation for gate optimization in [16] and the canonical form for quantum circuits [19] to satisfy our needs.

Specifically, in our DAG with non-commutativity (DAGNC) representation, a quantum circuit can be represented as a multigraph. Vertices in the multigraph correspond to gates in the quantum circuit, and edges correspond to orders between gates. The edge from node i to j means that the gates corresponding to node i and j have at least one qubit or classical bit in common, and the gate corresponding to node i executes before the gate corresponding to node j. As shown in Figure 15 (b), all five gates in the quantum circuit are represented by five vertices associated with indices in the multigraph of DAGNC. The edges from vertex 0 and 2 to vertex 4 are created due to that the Hadamard gate and the Pauli `X` gate above will execute prior to the `CX` gate, and so does the edge from vertex 1 to vertex 3. There is no edge between the vertices above and the vertices below, because there is no specified order between them. Compared with DAGNC, there is no edge from vertex 2 to vertex 4 in the DAG of the canonical form, because when the Pauli `X` gate operates on the target qubit of `CX` gate, the Pauli `X` gate and the `CX` gate are commutable.

After converting quantum circuits to the DAGNC representation, finding patterns in the quantum circuit is equivalent to finding subgraphs in the DAGNC of the quantum circuit that are isomorphic to the DAGNC of the pattern – an instance of the subgraph isomorphism problem. A mapping is found when a subgraph matches a target pattern. As efficient methods for subgraph isomorphism already exist, we leverage the Python *networkx* [20] package to solve this problem, which implements the VF2 algorithm [21] for graph isomorphism testing.

In our antivirus program, the subgraph isomorphism is followed by checking whether the found mappings are *exact mappings*. A mapping is exact if and only if there is a qubit and classical bit (clbit) assignment for the found mapping that can fully reconstruct the pattern circuit. Not all found mappings are exact due to the asymmetry of multi-qubit gates. For example, if the pattern circuit consists of "`H`(0); `CX`(0,1)", that is, an `H` gate on qubit 0 and a `CX` gate whose control qubit is 0 and target qubit is 1. The mapping is not exact if we found "`H`(1); `CX`(0,1)", because there is no qubit assignment/reordering that reconstructs this mapping to the pattern circuit. The resulting mappings from VF2 algorithm in *networkx* are not always exact; only exact mappings should be selected. As such, we test if a mapping is exact by building the qubit and clbit assignment in the process of transversing all vertices in the found mapping, and checking if the assignment of the new vertex is contradictory to the current assignment.

The final output of the antivirus is a count for each of the malicious patterns, i.e., the number of occurrences of the

Malicious Circuits	# Copies	# Detected	Detection Accuracy
V_1	8	8	100%
V_2	8	8	100%
V_3	4	4	100%
V_4	6	6	100%
V_5	65	65	100%
V_6	65	65	100%
V_7	65	65	100%
V_8	65	65	100%
V_9	2	2	100%
V_{10}	1	1	100%

TABLE I: Malicious pattern counts found by antivirus program in the virus circuits. The duration of the virus circuits was set to about $10,000$dt, the same as used for attack evaluation in prior sections, and the # Copies lists how many repetitions of each malicious pattern fit in that time.

Victim Circuits	# Detected Copies	Detected Pattern	Labeled as Virus?
Grover	0	–	No
Deutsch-Jozsa	0	–	No
Bernstein-Vazirani	1	V_1	No
VQE	0	–	No
QML	0	–	No

TABLE II: Malicious pattern counts found by antivirus program in the victim circuits.

patterns in a quantum circuit. Based on our evaluation in previous sections, we see that a higher malicious pattern count (i.e., longer duration of virus circuit) results in higher crosstalk errors and poses a higher risk to neighboring victim circuits.

B. Antivirus Evaluation Results

To evaluate the antivirus, we tested it on the malicious patterns which we have evaluated to cause crosstalk in the earlier sections. We further tested it on the benign victim circuits to observe if any will be labeled as a virus.

Table I shows the results of scanning the 10 virus circuits using our antivirus. In all cases the antivirus correctly identified the malicious pattern and number of copies of the pattern that occur in the circuit.

Table II shows the results of scanning the victim circuits. It can be seen that the malicious patterns do not generally appear in the victim circuits. For Bernstein-Vazirani the antivirus detected 1 copy of V_1 within this circuit because V_1 consists only one `CX` gate and there is one `CX` in Bernstein-Vazirani algorithm, but as the evaluation shows for V_1 with just one copy is not enough to cause crosstalk. Please note that V_{10} is malicious with only one copy since V_{10} has duration of $10,000$dt even with one copy, while one copy of V_1 has duration of only about $1,250$dt. Consequently, we did not see any false positives on our tested circuits.

C. Securing Antivirus from Attacks

Although the antivirus is able to detect all the malicious patterns, an attacker may attempt to bypass the antivirus to get his or her virus circuits to run on the quantum computers. A solution is to either run the compilation and antivirus at the quantum computer provider site or run on the client side with use of a Trusted Execution Environment (TEE) to protect the antivirus and attest to the quantum computer provider that the circuits were compiled and checked for viruses using TEE attestation mechanisms. It should be noted that today, users typically run the compiler, i.e. Qiskit on their own computer and send compiled circuits to IBM for execution on the quantum computer backends – there is no protection from malicious or modified Qiskit being used.

In Figure 16 we show the three workflows: (a) the workflow today with no antivirus protection and all the code being compiled on the user's end, (b) code being compiled on the cloud provider's end, (c) code compiled on user's end with SGX protection.

a) Existing Qiskit Workflow: The existing Qiskit workflow does not provide any antivirus-like protections. Further, as the compilation is done on the user's end, the quantum computer provider, i.e. IBM, does not have any assurance about the compiled circuit, other than the user who submitted it. This workflow is shown in Figure 16 (a).

b) Executing Antivirus by Quantum Computer Provider: To provide the antivirus protection, we propose one solution to run the antivirus on the cloud provider's end. Already, the tools, i.e. Qiskit, is vendor specific, so it could be run by the provider. This ensures the provider can enforce the antivirus checking, may be lower cost to users (since compilation and related computation is done by the provider), but users lose control of their code (since the provider accesses raw code to compile and check it). This workflow is shown in Figure 16 (b).

c) Executing Antivirus Locally with SGX: To allow users more control of their code and to run the compilation and the antivirus locally, the antivirus could also be run inside a Trusted Execution Environment (TEE), which is an isolated execution environment for executing code, in which those executing the code can have high levels of trust for the applications running on the device. One such TEE is from Intel Software Guard Extension (SGX) [22] technology available in current classical hardware. Intel SGX offers hardware-based memory encryption that isolates specific application code and data in memory. Intel SGX allows user-level code to allocate private regions of memory, called enclaves, which are designed to be protected from processes running at higher privilege levels. In our work, we port Qiskit to run inside SGX. By running the antivirus inside the SGX enclaves, we avert potential attackers from maliciously modifying or bypassing the antivirus. SGX is also able to provide attestation, so that the quantum provider can verify that the circuit and the antivirus report were generated by genuine antivirus. Further, since antivirus and compiler are run locally, the user exposes less potentially proprietary information to the quantum computer provider. In our work, we use Gramine [23] to run the Qiskit (which is written in Python) inside SGX. This workflow is shown in Figure 16 (c).

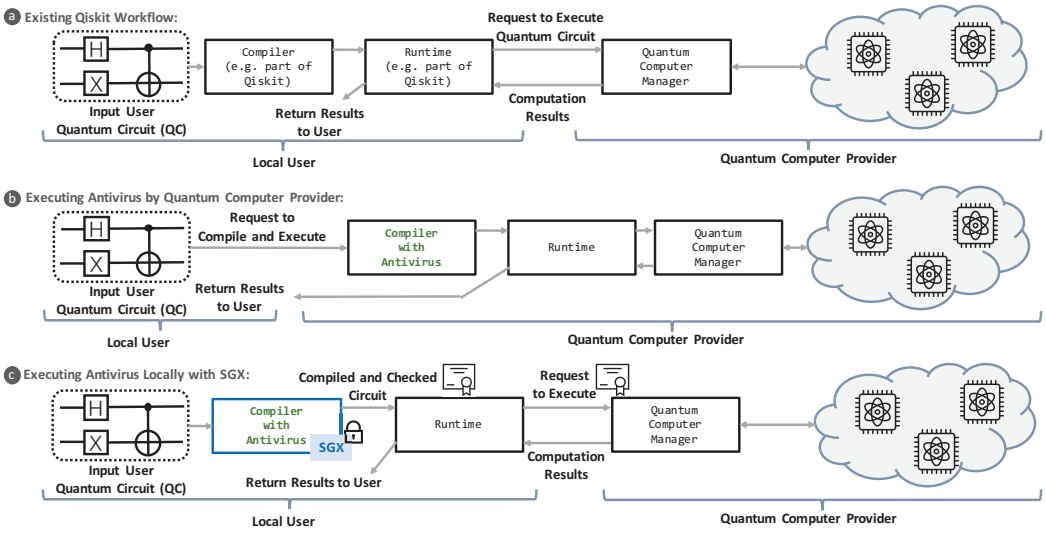


Fig. 16: Three quantum compiler workflows: (a) exiting workflow in Qiskit today, (b) proposed workflow with antivirus running on the quantum provider end, and (c) proposed workflow with antivirus running on the user end with SGX protections.

D. Antivirus Overhead Evaluation

We evaluated the overhead of running the antivirus on an Intel(R) Xeon E-2286G CPU at 4GHz. The timing was measured using the *perf* Linux profiler and the time elapsed was extracted as metric.

The actual pattern scanning operation is very quick, on the order of 1s and the main timing overhead without or with SGX is that of loading Qiskit and the antivirus code. We observed that SGX has a very large, but almost constant overhead factor of about 250s regardless of the tested circuit duration time. This large overhead comes from setting up the SGX enclave and loading all the necessary Qiskit and Python code into the enclave. The time for generating the SGX attestation report as well as sending the circuit to the quantum computer provider for execution is an additional time overhead on top of the values reported in the table. Since IBM does not support receiving SGX attestation reports, this was not evaluated.

We also observed that considering a fixed duration, it takes longer to scan virus circuits V_5 to V_8. For these virus circuits, where each copy uses fewer or simpler gates, more circuit copies fit in the fixed duration of time. The scanning algorithm's complexity depends on the depth (counted in number of gates) of the circuit, and for a fixed time duration, the number of gates is different for different circuits. As a result, some circuits of the same duration take different amount of time to scan.

VI. Related Work

There is currently limited, but growing, literature on security of quantum computers, including surveys [10], [24].

For superconducting quantum computers, recent work [12] shows that the crosstalk errors could be used in fault injection attacks. It also showed how an adversary can launch a denial of service attack on the victim circuit using crosstalk errors, similar to our evaluation. In addition, due to the difference of eigenstates, qubit-sensing employs malicious circuits to sense qubits of victim circuits based on already known statistical information [25]. Researchers have also proposed different methods to fingerprint quantum computer hardware by characterizing error patterns unique to each device or qubit [26], [27]. In trapped-ion quantum computers, repeated shuttle operations can elevate the ion-chain's energy, which can damage the fidelity of victim circuits [28], [29].

Motivated by the attacks, recent work has explored some preliminary defenses. After detecting crosstalk errors by analyzing execution of a circuit [30], they can be mitigated using connectivity reduction, qubit frequency tuning, and coupler tuning are proposed [31]. Other defenses include instruction scheduling modifications [32]. One common trend observed across all the methods is that the crosstalk errors can be controlled up to a certain point, but not eliminated completely. We also demonstrated in this work that the defenses such as using idle qubits, in Section IV-C2, do not prevent crosstalk. As an alternative, the antivirus proposed in this work aims to detect possibly virus circuits at compile time, before the circuits are even allowed to execute on the quantum computers and trigger crosstalk errors. The antivirus especially does not require executing the circuit to do the analysis whether the circuit is malicious.

VII. Conclusion and Future Work

This paper presented the first antivirus for quantum computers. The work first evaluated a number of virus circuits that use different patterns of gates to trigger crosstalk errors on adjacent qubits. The work demonstrated that these virus circuits can impact adjacent victim circuits running on the same quantum computer and cause crosstalk errors that affect the output probabilities of the victim circuits. The work also showed that defenses such as keeping idle qubits do not fully

work, and a solution such as the antivirus is needed. The attack experiments were evaluated on real, publicly accessible cloud-based IBM quantum computers showing the threats are not just theoretical but can be measured on real hardware. As a result, the antivirus was developed as means to proactively check for virus circuits in the source code of the quantum programs. It was shown to have 100% detection with no false positives on the tested virus and victim circuits. The antivirus was also shown to work effectively as part of Qiskit with two use cases: without and with use of SGX enclaves.

Moving forward, we point out further research need to investigate more virus circuits or to decouple possible decoherence issues from crosstalk when evaluating the attacks, for example. Quantum computer security is a nascent research area and more virus definitions can be added in the future, so that the antivirus can check for new viruses as they are discovered. Further, pulse-level viruses can be also analyzed and new antivirus extension developed for them.

ACKNOWLEDGMENTS

This work was supported in part by NSF grants 1901901 and MIT–IBM Watson AI Lab. We would like to thank Dmitrii Kuvaiskii from Intel for help with Intel SGX and Gramine tools.

REFERENCES

[1] F. de Lima Marquezino, R. Portugal, and C. Lavor, *Shor's Algorithm for Integer Factorization*. Cham: Springer International Publishing, 2019, pp. 57–77. [Online]. Available: https://doi.org/10.1007/978-3-030-19066-8_4

[2] "Ibm unveils breakthrough 127-qubit quantum processor," Nov 2021. [Online]. Available: https://newsroom.ibm.com/2021-11-16-IBM-Unveils-Breakthrough-127-Qubit-Quantum-Processor

[3] A. Cho, "Ibm promises 1000-qubit quantum computer—a milestone—by 2023," *Science, September*, vol. 10, 2020. [Online]. Available: doi:10.1126/science.abe8122

[4] S. Castellanos, "Google aims for commercial-grade quantum computer by 2029," *The Wall Street Journal*, 2021. [Online]. Available: https://www.wsj.com/articles/google-aims-for-commercial-grade-quantum-computer-by-2029-11621359136

[5] L. K. Grover, "A fast quantum mechanical algorithm for database search," in *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, ser. STOC '96. New York, NY, USA: Association for Computing Machinery, 1996, p. 212–219. [Online]. Available: https://doi.org/10.1145/237814.237866

[6] E. Farhi and H. Neven, "Classification with quantum neural networks on near term processors," *arXiv preprint arXiv:1802.06002*, 2018.

[7] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta, "Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets," *Nature*, vol. 549, no. 7671, pp. 242–246, 2017.

[8] "Intel sgx," https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html.

[9] J. Greene, "Trusted execution environment." [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/trusted-execution-technology-security-paper.pdf

[10] A. A. Saki, M. Alam, K. Phalak, A. Suresh, R. O. Topaloglu, and S. Ghosh, "A survey and tutorial on security and resilience of quantum computing," 2021.

[12] 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 ACM/IEEE International Symposium on Low Power Electronics and Design*, ser. ISLPED '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 25–30. [Online]. Available: https://doi.org/10.1145/3370748.3406570

[11] P. Das, S. S. Tannu, P. J. Nair, and M. Qureshi, "A case for multi-programming quantum computers," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '52. New York, NY, USA: Association for Computing Machinery, 2019, p. 291–303. [Online]. Available: https://doi.org/10.1145/3352460.3358287

[13] D. Deutsch and R. Jozsa, "Rapid solution of problems by quantum computation," *Proceedings of the Royal Society of London. Series A: Mathematical and Physical Sciences*, vol. 439, pp. 553 – 558, 1992.

[14] E. Bernstein and U. Vazirani, "Quantum complexity theory," *SIAM J. Comput.*, vol. 26, no. 5, p. 1411–1473, oct 1997. [Online]. Available: https://doi.org/10.1137/S0097539796300921

[15] H. Wang, Y. Ding, J. Gu, Y. Lin, D. Z. Pan, F. T. Chong, and S. Han, "Quantumnas: Noise-adaptive search for robust quantum circuits," *arXiv preprint arXiv:2107.10845*, 2021.

[16] R. Iten, R. Moyard, T. Metger, D. Sutter, and S. Woerner, "Exact and practical pattern matching for quantum circuit optimization," *ACM Transactions on Quantum Computing*, vol. 3, no. 1, jan 2022. [Online]. Available: https://doi.org/10.1145/3498325

[17] M. Chen, Y. Zhang, and Y. Li, "A quantum circuit optimization framework based on pattern matching," *SPIN*, vol. 11, no. 03, p. 2140008, 2021. [Online]. Available: https://doi.org/10.1142/S2010324721400087

[18] N. Abdessaied, M. Soeken, R. Wille, and R. Drechsler, "Exact template matching using boolean satisfiability," in *2013 IEEE 43rd International Symposium on Multiple-Valued Logic*, 2013, pp. 328–333.

[19] M. M. Rahman, G. W. Dueck, and J. D. Horton, "An algorithm for quantum template matching," *J. Emerg. Technol. Comput. Syst.*, vol. 11, no. 3, dec 2015. [Online]. Available: https://doi.org/10.1145/2629537

[20] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring network structure, dynamics, and function using networkx," in *Proceedings of the 7th Python in Science Conference*, G. Varoquaux, T. Vaught, and J. Millman, Eds., Pasadena, CA USA, 2008, pp. 11 – 15.

[21] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "An improved algorithm for matching large graphs," in *In: 3rd IAPR-TC15 Workshop on Graph-based Representations in Pattern Recognition, Cuen*, 2001, pp. 149–159.

[22] "Intel software guard extensions." [Online]. Available: https://www.intel.com/content/www/us/en/developer/tools/software-guard-extensions/overview.html

[23] W. Porzyck and M. kowalzyck, "Gramineproject/gramine: A library os for linux multi-process applications, with intel sgx support," Oct 2021. [Online]. Available: https://github.com/gramineproject/gramine

[24] A. A. Saki, M. Alam, and S. Ghosh, "Impact of noise on the resilience and the security of quantum computing," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 186–191.

[25] A. A. Saki and S. Ghosh, "Qubit sensing: A new attack model for multi-programming quantum computing," 2021.

[26] K. Phalak, A. A. Saki, M. Alam, R. O. Topaloglu, and S. Ghosh, "Quantum puf for security and trust in quantum computing," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 11, no. 2, pp. 333–342, 2021.

[27] A. Mi, S. Deng, and J. Szefer, "Short paper: Device- and locality-specific fingerprinting of shared nisq quantum computers," in *Proceedings of the Workshop on Hardware and Architectural Support for Security and Privacy*, ser. HASP, October 2021.

[28] A. A. Saki, R. O. Topaloglu, and S. Ghosh, "Shuttle-exploiting attacks and their defenses in trapped-ion quantum computers," *IEEE Access*, vol. 10, pp. 2686–2699, 2022.

[29] ——, "Muzzle the shuttle: Efficient compilation for multi-trap trapped-ion quantum computers," 2021.

[30] 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, Sep. 2020. [Online]. Available: https://doi.org/10.22331/q-2020-09-11-321

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

[32] P. Murali, D. C. Mckay, M. Martonosi, and A. Javadi-Abhari, "Software mitigation of crosstalk on noisy intermediate-scale quantum computers," *Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems*, Mar 2020. [Online]. Available: http://dx.doi.org/10.1145/3373376.3378477