A Quantum Computer Trusted Execution Environment

Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, Jakub Szefer
Yale University

Abstract—We present the first architecture for a trusted execution environment for quantum computers. In the architecture, to protect the user's circuits, they are obfuscated with decoy control pulses added during circuit transpilation by the user. The decoy pulses are removed, i.e. attenuated, by the trusted hardware inside the superconducting quantum computer's fridge before they reach the qubits. This preliminary work demonstrates that protection from possibly malicious cloud providers is feasible with minimal hardware cost.

Index Terms—quantum computing, cloud computing, control pulses, RF switches, attenuation, obfuscation, dilution refrigerator

1 Introduction

A number of cloud vendors today already provide access to Noisy Intermediate-Scale Quantum (NISQ) quantum computers for remote access by users. IBM Quantum [1], for example, is both a vendor of quantum computers and cloud provider making these machines accessible. Meanwhile, Amazon Braket [2] and Azure Quantum [3] are cloud provider services, which give access to quantum computers from vendors who are separate, such vendors as Rigetti or Quantinuum, for example.

In the cloud setting, the cloud provider may have access to the circuits (and the resulting control pulses) that execute on the quantum computer. Given knowledge of the circuits (or equivalently of the control pulses, which can be reverse-engineered into the circuits), the cloud provider has full access to what the users are executing. The cloud-based model of quantum computing benefits users as it allows for on-demand access to quantum computation resources, but it endangers the intellectual properties and secrecy of users' algorithms and circuits executing on cloud-based quantum computers.

Despite the promising progress in the design of blind quantum computation schemes proposed to protect users from the untrusted quantum computer cloud providers [4], [5], [6], there are no practical realizations of blind quantum computation protocols today. As an alternative approach to blind quantum computation protocols, we propose a new architecture, the Quantum Computer Trusted Execution Environment (QC-TEE), that could be realized with today's technology to aid in the protection of the quantum circuits from honest-but-curious cloud providers.

1.1 Motivation for QC-TEE

Classical hardware providers provide hardware security features in their processors today, such as Intel SGX [7]. The cloud providers today also already are providing these classical hardware security features to users [8]. Security-conscious users are willing to pay extra for access to more secure hardware. We use this as motivation to develop hardware security features for quantum computers, which users could use to protect their circuits; and which cloud providers and hardware vendors will likely be motivated to provide – just as they provide such features for classical hardware today.

1.2 Main Idea of QC-TEE

The main idea behind our design is to provide protection from an honest-but-curious quantum cloud provider with minimal hardware modifications and without the need for quantum users or quantum networking resources. One way to achieve this is by confusing the quantum computer provider about which quantum gates are actually executed on the quantum computer. The honest-but-curious provider can observe the operation of the RF pulses and could learn the quantum gates used, and thus the algorithm, executed by the user. If the control pulses can be obfuscated and randomized from the view of the cloud provider, he or she will no longer be able to learn what the quantum circuits are. Obfuscation of control pulses on the quantum server requires two parts: first, the addition by the user of control pulses, i.e. quantum gates, which are not actually executed on the quantum computer, but which the provider cannot distinguish from regular control pulses sent to the quantum computer. We call these the decoy control pulses. Second, the removal of these decoy control pulses within the quantum computer, so that they do not actually affect the qubits. Our insight is that this can be easily achieved by the addition of RF switches inside the quantum computer's refrigerator. These switches can attenuate the decoy pulses after they have entered the dilution refrigerator.

This work was supported in part by NSF grants 2312754 and 2245344.

2 Background

This section gives a brief introduction to fundamental parts of quantum programs or circuits, such as qubits, quantum gates, and control pulses, which we aim to protect from being stolen.

Qubits and Quantum Gates: A qubit is the fundamental unit of information on a quantum computer and may be represented using a vector $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$, which is a superposition of the basis states $|0\rangle$ and $|1\rangle$ such that $|\alpha|^2 + |\beta|^2 = 1$. When measured, a qubit collapses to a binary 0 or 1 with probabilities $|\alpha|^2$ and $|\beta|^2$, respectively. Quantum gates such as single-qubit (e.g., Pauli-X gate) or multiple-qubit (e.g., 2-qubit CNOT gate) gates modulate states of qubits and thus perform computations.

Control Pulses and Quantum Circuits: Microwave RF pulses are typically used to control superconducting qubits. To operate each native gate on a quantum computer, the necessary control pulses for each gate must be created and supplied to the quantum computer. Each quantum circuit is decomposed into native gates, which then are executed by sending the corresponding control pulses to the quantum computer.

3 Threat Model

Our work aims to protect users from honest-but-curious cloud providers, who do not modify the hardware nor perform active

attacks, but who may want to covertly and passively learn what computations the user is performing. In the cloud setting, we assume the quantum server's dilution refrigerator forms a natural trust boundary, where the cloud provider cannot extract or access any information or tamper the hardware inside the refrigerator.[1]

We also assume an honest-but-curious cloud provider who does not manipulate the control pulses or classical information sent to the quantum computer for execution, but he or she can observe this information to try to learn what the user is executing. The classical information sent to the quantum computer, such as our (encrypted) bitmap, discussed later, is assumed to be protected with quantum-safe cryptography.

Note, that since we assume the cloud provider can see all the control pulses, which define the gates and the circuit, as a result the cloud provider does have access to the gates and the circuit as a whole. However, they control pulses include our decoy gates, discussed later. Since the honest-but-curious cloud provider cannot distinguish the real and decoy pulses, and cannot decrypt the encrypted bitmap, they do not actually know which are the real pulses (and thus gates) and which are decoy pulses (and thus gates).

On the hardware end, we assume the backend can be modified with minimal hardware changes (details of the modifications are discussed in the Section 4). We assume the modifications to the backend's hardware are within the cooling power of today's fridges. We assume the added classical hardware logic has been validated or verified, and side-channel attacks on this classical hardware logic are out of the scope of this work. In classical computing, security architectures such as Intel SGX are implemented by Intel. We make the same assumption that, in our case, quantum computer developers should implement the security modifications and the implementation is trusted.

On the software end, following Figure 1, we assume the user is able to transpile his or her circuits locally, as is possible today with Qiskit or similar frameworks. The transpiled circuits can contain any valid gates from the basis library, including any decoy gates added by the user as explained in Section 5. In addition to the transpiled circuits, we assume the user is able to send additional classical data, which is the encrypted *input bitmap*. The *input bitmap* is used to specify which pulses belong to the original circuit and which pulses are decoy. This information is used inside the trusted hardware to attenuate the decoy pulses.

4 Architecture of the QC-TEE

An illustration of the obfuscation is shown in Figure 2a and an illustration of the input bitmap is shown in Figure 2b. An illustration of the attenuation of decoy pulses is also shown in Figure 2c. To realize the QC-TEE, a number of hardware components need to be added to the internals of the dilution refrigerator. Following our threat model and assumptions, components in the refrigerator are trusted and cannot be tampered by the honest-but-curious quantum computer cloud provider.

- **Decryption Engine** is used to decrypt the encrypted input bitmap and then store the decrypted input bitmap in the Input Bitmap Memory. The decryption engine works by first using public-key cryptography to establish a shared secret key (symmetric key) and then using symmetric-key cryptography to decrypt the input bitmap itself. Both the public and private-key algorithms need to be quantum secure. The memory is used to store the decrypted input bitmap, which is later used to control the attenuation of the pulses.
- **Switch Control Module** is used to send the bits of the decrypted input bitmap to the attenuation switches. Each drive and control channel is associated with one attenuation switch. In each 160dt time period, each of the drive and control channels for all the qubits is provided with one bit: 0 for no attenuation and 1 for attenuation.
- **Attenuation Switches** are used to attenuate the decoy control pulses which were added to confuse the potential attackers but are not actually used for computation. Each of the RF switches requires 1 bit of input to set if the switch should or should not attenuate the input RF signal at any provided time period.
- **TRNG** is used to generate truly random numbers used in controlling flipping of the qubits state, i.e. applying `X` gate, at the end of the user's circuit. If qubit flipping is enabled, the TRNG generates the flipping bits. In parallel, for each qubit, the information of whether the qubit state was flipped (represented by classical bit 0) or not (represented by classical bit 1) is saved into the output bitmap.
- **Encryption Engine** is used if qubit flipping is enabled. The engine encrypts the output bitmap, so that the user can correctly interpret the qubit measurements. The engine uses symmetric-key encryption to encrypt the output bits with a randomly-generated key and public key encryption to encrypt the symmetric key with the user's public key.

5 The Obfuscation Algorithm

The first step in our algorithm is to split the input circuit into slots. Given a specific backend, it determines the `CNOT` coupling with the longest duration, as well as the duration of single-qubit gates. The algorithm then takes the input circuit already transpiled for the target backend, and adds barriers before and after `CNOT` gates to separate portions of the circuit with single qubit gates from ones with `CNOT` gates. Each barrier thus determines a start of a *slot*:

- $Slot_{CX}$: is a slot that consists of at least one `CNOT` gate and optionally any single qubit gates on the other qubits where the `CNOT` gate is not connected. The duration of $Slot_{CX}$ slots is determined by choosing the longest `CNOT` gate duration out of all possible couplings on the backend, and then rounding it up to be an even integer multiple of the single-gate duration.
- $Slot_{SQ}$: is a slot that consists of only single qubit gates and optionally any delays if required. The duration of these slots is determined at compile time and chosen based on the target security level. Currently, two duration periods are suggested: half-delay and max-delay. Half-delay means the duration of $Slot_{SQ}$ is half of the duration of $Slot_{CX}$, recall $Slot_{CX}$ slots are set to be of duration this is even multiple of single-qubit gate duration. Max-delay means the duration of $Slot_{SQ}$is equal to duration of $Slot_{CX}$.

The slotted input circuit is next updated by inserting decoy gates. Each slot is divided into equal-length sub-slots. The length of the sub-slot is equal to the duration of a single-qubit gate, e.g., 160dt on IBM quantum computers today. Figure 2a

1. The dilution refrigerator keeps extremely low temperature and target pressure and any access or opening of the fridge will disturb temperature and pressure, effectively destroying qubit state. The changes can also trigger self-erase or self-destruct features in QC-TEE to erase any decrypted bitmap or cryptographic keys. We leave defenses of the dilution refrigerator as future work and simply assume the dilution refrigerator is the trust boundary.

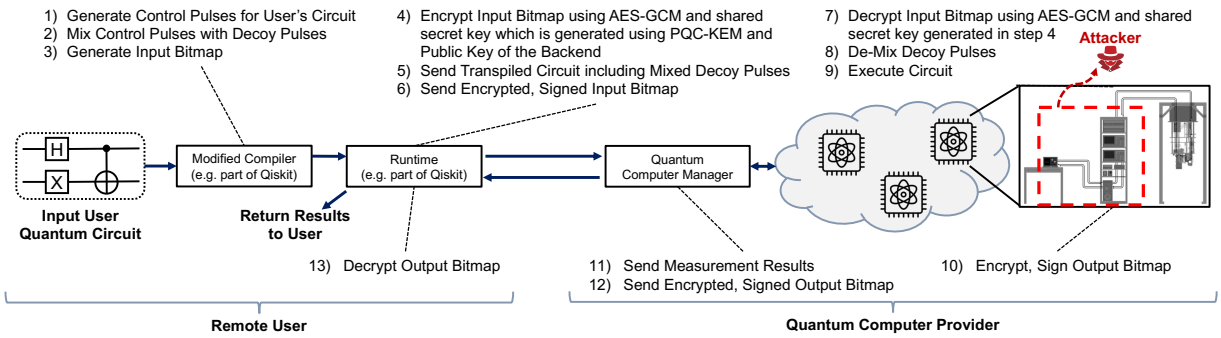

Fig. 1: Workflow of QC-TEE overlaid on typical cloud-based quantum computation workflow.

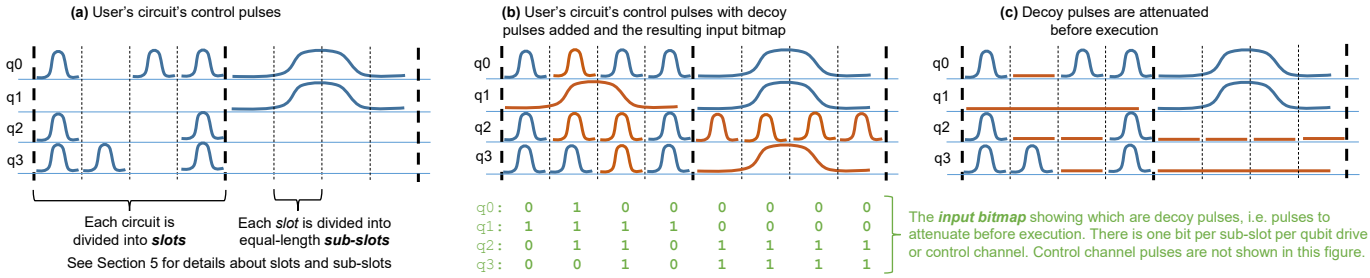

Fig. 2: Example illustration of how control pulses are obfuscated, the pulses shown are a simplified graphical representation. (a) Control pulses correspond to the input circuit of the user. (b) Control pulses with decoy pulses added, along with the input bitmap, specifying which are real control pulses and which are decoy pulses to be attenuated. (c) Original control pulses are executed on the quantum computer and after the attenuation of decoy pulses, attenuation is done based on the information from the input bitmap. The time is quantized into *sub-slots* of 160dt.

shows the input circuit with the slots and sub-slots. The sub-slots which are not occupied by a gate are filled with the decoy gates, i.e. control pulses.

For each $Slot_{CX}$ in the circuit, on the qubits that do not consist of a `CNOT` gate in that slot, we either add a `CNOT` gate if `CNOT` coupling of the backend allows, or we add a random mix of `X` and `SX` gates. Figure 2b earlier in the paper shows (in red color) the addition of a decoy `CNOT` gate next to user's `CNOT` gate in a $Slot_{CX}$.

For each $Slot_{SQ}$, in addition to the existing single-qubit gates from the original circuit, a random mix of `X` and `SX` gates are added in each empty sub-slot. Figure 2b shows (in red color) the addition of decoy single-qubit gates in a $Slot_{SQ}$. The `Rz` gates from the original circuit are added to the slots but are not counted towards a sub-slot because the `Rz` gates are virtual.

In addition to the decoy gates, an additional layer of `X` gates can be added at the end of the circuit to randomize the output. The randomizing of the output happens inside the dilution refrigerator and uses the TRNG module. With use of the `X` gates at the end of the circuits, we provide option for protecting not just the circuit but also the output. The circuit will be appended with a layer of `X` gates on all qubits. But in each shot of execution, each `X` gate will be randomly chosen to be attenuated or not based on the generated random number. The provider will see the layer of `X` gates but cannot know which `X` gate is attenuated in each shot. This information about which `X` gates actually executed is encrypted and sent back to the user to recover the correct results. Previously use of `X` gates has been applied in context of mitigating measurement errors [9], but not from security perspective.

6 Experimental Setup

To evaluate the overhead of QC-TEE hardware additions, we measured fidelity differences of various quantum algorithms executed with and without the proposed TEE. We use a real 7-qubit IBM machine (backend), `ibm_perth`, currently available to researchers and the public through the IBM Quantum cloud-based service [1].

The QASMBench Benchmark Suite is used in our work to analyze the impact of the QC-TEE design on different circuits. We transpile each benchmark for the target backend. The transpiled code is then used as input to the obfuscation algorithm, which is described in Section 5. QASMBench includes three categories of benchmarks of different sizes, and we use the first category, which refers to the small-scale quantum circuits that use 2 to 10 qubits. Because our access is limited to a real quantum computer with 7 qubits, we only select the benchmarks that can be run on this machine. We keep the number of shots constant at 8192.

We evaluated four different configurations: *baseline*, *half-delay*, *max-delay* and *randomize-output*. The *baseline* is simply the benchmark transpiled for the target backend without any of our modifications. The *half-delay* means the duration of $Slot_{SQ}$ is half of the duration of $Slot_{CX}$, recall $Slot_{CX}$ slots are set to be of duration this is even multiple of single-qubit gate duration. The *max-delay* means the duration of $Slot_{SQ}$ is equal to the duration of $Slot_{CX}$. The *randomize-output* includes an additional layer of `X` gates before measurement.

We measure the variational distance between each of the three configurations *half-delay*, *max-delay*, *randomize-output*, and the *baseline* in order to define it as a metric to examine how our design affected the output probability of the benchmark circuits. Informally, the variational distance of two output probability distributions is the measure of how one probability

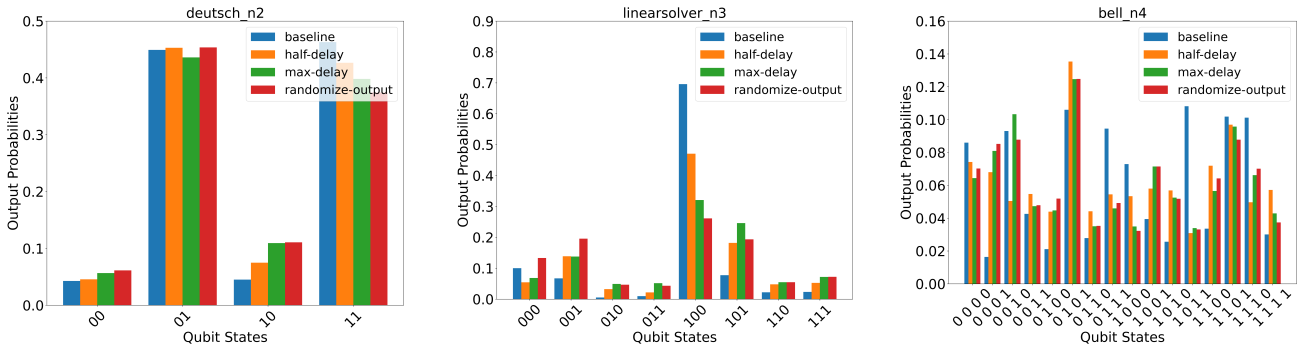

Fig. 3: Example output probability of *deutsch*, *linearsolver* and *bell* benchmarks. Tested on *ibm_perth* with custom SX, X, CX gates of **0.01%** amplitude, representing non-ideal RF switches.

TABLE 1: Average depth increase factor for the QASMBench benchmarks for IBM Perth. The average depth increase by factor is reported in each of the three Obfuscation Levels: *half-delay*, *max-delay*, *randomize-output*; the depth increase is computed relative to the unmodified benchmark, i.e. the *baseline* configuration.

Obfuscation Level:	**half-delay**	**max-delay**	**randomize-output**
IBM Perth	9.45x	11.51x	11.55x

TABLE 2: Average Variational Distance (VD) for the QASMBench benchmarks when emulating imperfect RF switches, which attenuate the decoy pulses to 0.01% and 0.1% amplitude. Data is from experiments on IBM Perth.

Obfuscation Level:	**half-delay**	**max-delay**	**randomize-output**
QASMBench Average VD (0% amplitude, perfect RF switches)	0.1647	0.2099	0.2453
QASMBench Average VD (0.01% amplitude)	0.2322	0.2652	0.2691
QASMBench Average VD (0.1% amplitude)	0.3356	0.3647	0.3768

distribution is different from the other. In general, the total variation distance between P and Q is defined as:

$$\delta(P,Q) = \frac{1}{2}\sum |P - Q|$$

7 Evaluation

Our obfuscation technique works by adding decoy pulses and then attenuating them before actual execution. Once the decoy pulses are attenuated, they effectively become delays. Table 1 shows that the depth[2] of the QASMBench benchmarks when using our defense, compared to baseline, is increased on average by a factor of 10. Table 2 shows the average variational distance for the different obfuscation levels over the set of QASMBench benchmarks. First row of the table shows perfect RF switches. Second row of the table shows imperfect switches that attenuate the decoy pulses to 0.01% of amplitude. Last row of the table shows worst switches that attenuate decoy pulses to 0.1% of amplitude. To simulate and test imperfect switches, we developed custom gates that have the same pulse shape as `X`, `SX`, and `CX` gates, but they have only 0.01% or 0.1% amplitude. Figure 3 shows the effect of imperfect switches with attenuation to 0.01% of the original amplitude of decoy gates on the output probabilities in *deutsch*, *linearsolver*, and *bell* benchmarks. Higher amount of obfuscation results in larger variational distance, however, benchmarks such as *deutsch* give correct solution even with the larger variational distances.

8 Conclusion

This work proposed the first architecture for a trusted execution environment for quantum computers, to protect from honest-but-curious cloud providers who may try to steal or learn what quantum program the user is executing. The preliminary proposed QC-TEE architecture imposes minimal hardware changes, mostly the addition of RF switches and related control logic, to the fridge of a superconducting qubit quantum computer. To protect the circuits, they are obfuscated with decoy control pulses added during circuit transpilation by the user. The decoy pulses can later be removed, i.e. attenuated, by the QC-TEE hardware before they reach the qubits.

2. The definition of "depth" of the circuit refers to the `QuantumCircuit.depth()` in Qiskit code.

References

[1] "Ibm quantum," https://quantum-computing.ibm.com/.
[2] "Amazon braket," https://aws.amazon.com/braket/.
[3] "Azure quantum," https://azure.microsoft.com/en-us/products/quantum.
[4] V. Dunjko, J. F. Fitzsimons, C. Portmann, and R. Renner, "Composable security of delegated quantum computation," in *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014, pp. 406–425, https://arxiv.org/pdf/1301.3662.pdf. [Online]. Available: https://doi.org/10.1007%2F978-3-662-45608-8_22
[5] J. F. Fitzsimons and E. Kashefi, "Unconditionally verifiable blind quantum computation," *Physical Review A*, vol. 96, no. 1, jul 2017, https://arxiv.org/pdf/1203.5217.pdf. [Online]. Available: https://doi.org/10.1103%2Fphysreva.96.012303
[6] A. Broadbent, J. Fitzsimons, and E. Kashefi, "Universal blind quantum computation," in *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, 2009, pp. 517–526, https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5438603.
[7] J. Szefer, "Principles of secure processor architecture design," in *Principles of Secure Processor Architecture Design*. Springer, 2018, pp. 113–124.
[8] "AWS Nitro System — aws.amazon.com," https://aws.amazon.com/ec2/nitro/, [Accessed 20-Jul-2023].
[9] S. S. Tannu and M. K. Qureshi, "Mitigating measurement errors in quantum computers by exploiting state-dependent bias," in *Proceedings of the 52nd annual IEEE/ACM international symposium on microarchitecture*, 2019, pp. 279–290.