Dynamic Pulse Switching for Protection of Quantum Computation on Untrusted Clouds

Theodoros Trochatos
Yale University
theodoros.trochatos@yale.edu

Sanjay Deshpande
Yale University
sanjay.deshpande@yale.edu

Chuanqi Xu
Yale University
chuanqi.xu@yale.edu

Yao Lu
Yale University
physics.lu@yale.edu

Yongshan Ding
Yale University
yongshan.ding@yale.edu

Jakub Szefer
Yale University
jakub.szefer@yale.edu

***Abstract*—Quantum computing resources are now becoming easily accessible from various cloud providers. Although still in the Noisy Intermediate Scale Quantum regime, quantum computers hold promise to be able to execute novel algorithms and create invaluable data. However, just as with any other type of computing resource, they may be vulnerable to security attacks and should have defenses built into their design. This paper explores a particular threat of untrusted cloud providers, and how to protect user's quantum programs and data from the untrusted cloud provider. By leveraging trusted hardware in the quantum computer, a new obfuscation-based protection is developed based on switching of control pulses between different drive and control channels of the quantum computer. This work demonstrates that simple hardware modifications can enable dynamic, run-time pulse switching, which makes it extremely difficult for the cloud provider to decode what actual circuit is executed on the quantum computer. This work presents a basic architecture that employs pulse switching, and an extended architecture that includes use of dummy qubits for increased protection. The overhead of the proposed changes, as well as attack complexity for different types of user circuits and obfuscation levels is evaluated in this work.**

I. Introduction

Quantum computing is one of the emerging technologies which holds the promise to solve complex scientific, optimizations, and machine learning tasks [5]. With the progress in quantum computing technology, multiple cloud providers have opened up access to small and medium-scale quantum computers to the customers as Infrastructure as a Service (IaaS). For example, cloud-based services such as IBM Quantum, Amazon Braket, and Azure Quantum already provide access to the NISQ (Noisy Intermediate Scale Quantum) quantum computers remotely for users.

Since today the quantum computer cloud provider can see users' circuits and all the control pulses, which define the gates and operations of the circuit and as a result, the cloud provider has full knowledge of what the user is executing. This can endanger the privacy or intellectual property of the user's circuits and the data they contain. Therefore, there is a need to protect users' circuits from untrusted or malicious cloud providers. To address this need, we propose CASQUE, which is a new hardware design to secure superconducting quantum computers by leveraging trsuted hardware.

In the CASQUE design, we propose that trusted hardware can be incorporated into the quantum computer to help protect the user's circuits and data. Similar to how the CPU package in classical trusted execution environments forms the trust boundary, we select the dilution refrigerator as the natural trust boundary for the superconducting quantum computer. Opening or manipulating the refrigerator will cause temperature and pressure changes that destroy qubit states, and can be easily sensed to trigger protection of our trusted classical hardware in the refrigerator. Our trusted hardware consists of minimal hardware changes to the quantum computer hardware and leverages off-the-shelf components.

The main modification to a quantum computer hardware that CASQUE introduces is a Beneš network of RF switches to allow for switching of control pulses between different qubit and coupling drive and control channels. The input circuits are modified by our CASQUE software to randomly switch control pulses between different channels, and only inside the refrigerator, they are switched back to the correct channels based on encrypted information received from the user. The untrusted cloud provider cannot access the switching information and does not know which pulses actually execute on which qubits. In the extended CASQUE+ design, we also introduce means of adding dummy qubits and dummy gates for further increasing attack complexity. To support the switching of control pulses, our modifications also account for adjusting the frequency and amplitudes of the control pulses.

The hardware modifications are relatively minor in the context of the complexity of the rest of the quantum computer hardware. We are inspired to propose addition of the security features and hardware based on classical security hardware, such as Intel SGX [9], [43]. Cloud customers are today willing to pay extra for additional protections in hardware, and hardware providers have the motivation to add security features in hardware to help protect the code. We assume quantum computer cloud providers and hardware vendors are motivated to provide such new hardware security features – just as they provide such features for classical hardware today.

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

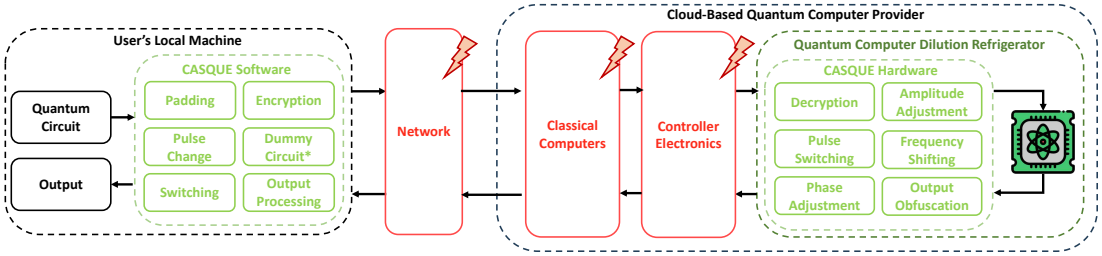


Fig. 1: System overview of today's cloud-based quantum computers, overlaid with the components of CASQUE, distinguishing trusted (green) and untrusted (red) elements. Trust is assigned to CASQUE operations and the quantum computer (refrigerator), while transmission links between users, cloud providers, and classical computers/controllers are considered untrusted.

A. Contributions

The contributions of this work are as follows:

- Design of CASQUE hardware architecture for superconducting qubit quantum computers which protects user's circuits and data with trusted hardware
- Development of novel qubit-switching as an obfuscation mechanism for hiding control pulses from untrusted cloud providers and malicious attackers
- Evaluation of hardware (FPGA) implementation of the CASQUE logic
- Evaluation of security attack complexity
- Design of CASQUE+ with novel use of dummy qubits and dummy gates for further increase of attack complexity with low additional cost

II. Background and Motivation

A. Control Pulses

Microwave 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. The pulses for all native gates on IBM Quantum are published as part of the specification of the quantum computers and their parameters are routinely updated through calibrations to maintain fidelity over time. A pulse is usually defined by the envelope, frequency, and phase. As an instance of the superconducting qubit control, the envelope specifies the shape of the signal which is generated by the arbitrary waveform generator (AWG), a common lab instrument, and the frequency and phase specify a period signal that will be used to modulate the envelope signal.

B. Pulse-Level Circuit Description

To fully describe a quantum program, all pulses for all the channels need to be defined, including when the pulses should start relative to the starting point of the circuit, to what qubits the pulses will be applied, and other physical operations like frequency or phase change, need to be specified. This information, referred to as pulse information, along with other useful information forms a so-called pulse-level circuit description. Pulse-level circuits and pulse information are important and valuable to be provided to users, because they enable users to verify quantum circuits and check execution details.

C. From Gate-Level to Pulse-Level Circuits

In order to actually generate pulse-level circuits, a number of steps are needed. The first step in developing a quantum circuit or pro-gram is to build a logic-level circuit with a quantum development kit, such as Qiskit [10], Amazon Braket SDK [20], Q# [38] and Cirq [36]. Analogous to classical computing, logic-level quantum circuits usually contain high-level descriptions. A series of operations need to be done to transform them into low-level and hardware-specific instructions, which is similar to the preprocessing, compilation and assembly process for classical computing programs. The second step is then to *transpile* the circuits, which is the term used by Qiskit to represent the operations and transformations that are like preprocessing and compilation. The process of transpiling involves many steps, including decomposing non-native quantum gates into groups of native gates, grouping and removing quantum gates to reduce the number of gates, mapping the logic qubits in the original circuits to the physical qubits on the specified quantum computers, routing the circuit under limited topologies, potentially optimizing circuits to lower error, and so on. The third step is termed *schedule* in Qiskit, which transforms gate-level circuits into pulse-level circuits. Scheduling further maps quantum circuits to microwave pulses, which are the ultimate physical operations used to regulate and control qubits. Based on previously calibrated data for each basis gate on each qubit or qubit pair, scheduling creates microwave pulse sequences that are ready to be carried out for quantum programs. The end result is a circuit composed only of control pulses representing basis gates that can be executed on the target quantum computer.

D. Prior Work

1) Blind Quantum Computing (BQC): BQC is a method designed to enable a client to conduct quantum computations on a distant server while maintaining the confidentiality of the computation's content. The primary goal of BQC is to uphold the privacy of quantum computations, ensuring that, despite the server executing the computation, it remains unaware of its nature or purpose. Nevertheless, the majority of BQC protocols currently in use assume that the client possesses a basic quantum device and relies on the existence of a quantum network for communication between the client and the server [6], [8], [12], [13], [24], [32]. Recent research has

put forth BQC protocols that alleviate the requirement for clients to host a quantum device on their premises [12], [17], [18], [21], [27], [40]. However, these protocols introduce a novel approach by depending on the existence of multiple quantum servers. Notably, these protocols operate on the assumption that these servers do not engage in communication with each other, necessitating a level of trust in servers that would otherwise be considered untrusted.

2) Quantum Homomorphic Encryption (QHE): Homomorphic encryption is a cryptographic technique enabling computations to occur on encrypted data, resulting in an encrypted output that, upon decryption, corresponds to the outcome of operations executed on the plaintext [4], [15], [25], [41], [44]. In contrast to the interactive computation inherent in BQC, homomorphic encryption provides a distinct approach. However, the realization of fully-secure Quantum Homomorphic Encryption (QHE), as indicated by the "no-go theorem," introduces exponential computational overhead. This impracticality arises, especially in the near term, as noisy quantum devices struggle to manage the substantial noise accumulation associated with such computations. Furthermore, the prevailing challenge lies in the fact that most homomorphic encryption schemes are specifically designed for classical binary data. This poses a significant hurdle when attempting to adapt these schemes to the intricate landscape of quantum information, which is characterized by principles such as superposition and entanglement. The intricacies of quantum information make the incorporation of homomorphic encryption into quantum contexts demanding.

E. Goal of this Paper

Previous techniques aim to protect arbitrary quantum programs, necessitating a classical client to be augmented with quantum capabilities and quantum networking (BQC), or incurring exponential computational overhead (QHE). For the aforementioned reasons, these techniques are rendered impractical. In this paper, our goal is to secure the quantum circuits while they are running on an untrusted cloud provider's end. This work, aims to timely address an important problem due to the emerging of quantum computers by introducing a number of software and hardware steps for switching of control pulses between different drive and control channels with minimal hardware modifications. In this way, the cloud provider is unable to retrieve information of what actual circuit is executed on the quantum computer.

III. Threat Model

A. Entities in the Threat Model

In our threat model, we consider three entities: users, cloud provider and quantum computer manufacturer. Users may have sensitive data and computation that they want to run on a quantum computer. Cloud provider manages the quantum computers. Quantum computer manufacturer is the entity who makes the quantum computers. Despite sharing names, such as IBM, we treat cloud providers and quantum computer manufacturers as distinct entities, separating runtime security threats (from the cloud provider) from supply chain and manufacturing security concerns (from the quantum computer manufacturer). Numerous examples, like Amazon Braket or Microsoft Azure Quantum, illustrate this separation. Importantly, any hardware modifications by the cloud provider are viewed as analogous to an untrusted manufacturer, a threat model currently outside our scope but a potential focus for future research.

B. Honest-but-Curious Cloud Provider

Our work considers the threat model of an honest-but-curious cloud provider. The honest-but-curious cloud provider assumption encompasses different attacks, ranging from the cloud provider spying on users for intellectual property to potential malicious insider actions. Our threat model treats these scenarios as isomorphic, grouping them under the honest-but-curious cloud provider category. In this model, cloud providers are untrusted and capable of spying on quantum computer operations without modification. The assumption of untrusted cloud providers extends to the possibility of collusion and our work operates independently of relying on non-colluding cloud providers.

C. Threat Model Details

Our research is dedicated to safeguarding users' quantum circuits from untrusted cloud-based quantum computing providers, focusing on defense against passive attacks like eavesdropping and side channels. Active attack protection, involving digital or analog data and signal modification, is considered separately. In our system architecture, shown in Figure 1, trusted and untrusted components overlay the cloud-based quantum computer.

Assumptions include a trusted user and compiler generating transpiled circuits implementing our obfuscation technique for execution by the cloud provider. Correct information about the quantum computers is assumed, allowing proper circuit transpilation. The user verifies the public encryption key for the new CASQUE hardware from a trusted authority. Unencrypted, transpiled circuits are accessible to the cloud provider for scheduling and execution, except for the encrypted CASQUE pulse protection map (PPM), decrypted only by our trusted hardware in the quantum computer.

The untrusted cloud provider is restricted from manipulating circuits or control pulses, but can observe information. Classical data, including the encrypted pulse protection map, is safeguarded with quantum-safe cryptography. Trusted hardware within the quantum computer cannot be manipulated without detection. For superconducting qubit machines, the dilution refrigerator serves as the trust boundary, with any intrusion causing detectable disturbances, ensuring the cloud provider cannot access it without destroying the quantum computation. All hardware within the dilution refrigerator is assumed correct, verified and bug-free.

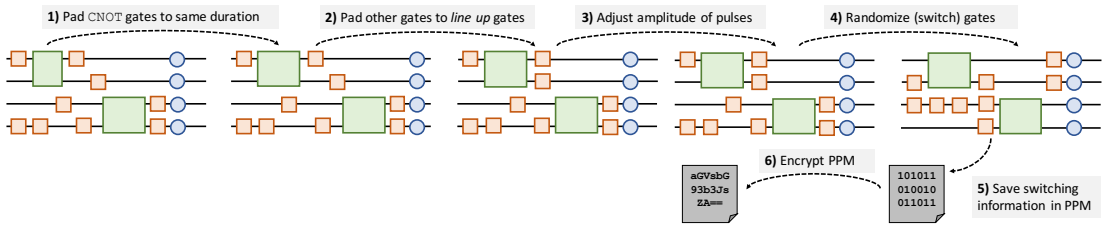


Fig. 2: Overview of the software steps for the update of user's circuits for CASQUE protection.

IV. CASQUE Main Features

Our proposed architecture addresses protection against an untrusted quantum computer cloud provider conducting passive attacks. By obfuscating and randomizing control pulses, we aim to confuse the provider about executed quantum gates and qubit assignments. This entails user-side software modifications involving random switching of transpiled program control pulses and the generation of control information for reassignment. Hardware alterations include incorporating RF switches inside the quantum computer's dilution refrigerator, creating a network to switch incoming signals among qubits or couplings. Notably, these modifications within the dilution refrigerator require no additional signal generators, minimizing power consumption within cooling constraints.

A. Pulse Switching

The key idea is pulse switching. Today, a cloud provider can directly see which pulses execute on which qubit. However, if new hardware is added to allow to switch any control pulse to any channel, then for each time period there are $\binom{n}{k}$, i.e. n *choose* k, possibilities that k pulses can execute on n qubits; a more detailed evaluation of the complexity is presented in Section VI-B. Within limitations discussed later, after a circuit is transpiled, the control pulses can be re-arranged in $\binom{n}{k}$ ways in each time period and the re-arrangement information can be saved so that the pulses can be switched back during execution. The re-arrangement in software is a simple modification of digital data representing the circuit. In the hardware, the re-arrangement can be achieved by use of a Beneš Networks.

1) Single-Qubit Gates: Single qubit gates such as `X` and `SX` gates can be switched between any qubit channel. The amplitude of the pulses is different for different qubits, thus if there is any switching, the amplitude has to be adjusted. To simplify the CASQUE design, we assume all single qubit pulses on all channels will be initially sent at maximum amplitude, and then the trusted hardware attenuates them according to the target qubit. Since each incoming pulse regardless of the channel will have the same initial amplitude, the attenuation hardware does not need to know the initial amplitude or the channel, only the target channel. More details are presented in the Section IV-F.

2) Two-Qubit Gates: Two qubit gates such as `CNOT` gate cannot be switched between channels, at least in current IBM Quantum computer designs. The reason is that for different qubits and couplings, the exact pulses, not just their amplitudes, are different for different couplings. In base CASQUE, we assume that `CNOT` gates will not be switched. However, as an extension, we present the CASQUE+ design in Section VII. In CASQUE+ we discuss how to increase attack complexity by adding dummy qubits to the user's programs (as long as the quantum computer backend has sufficient physical qubits to accommodate the original qubits and the dummy qubits). Dummy qubits increase the number of possible locations for switching pulses, increasing n in the $\binom{n}{k}$ number of possible combinations. We further leverage the dummy qubits to enable the addition (and elimination) of dummy single-qubit gates, as well as two-qubit gates, such as `CNOT` gates. The dummy gates can be switched to the dummy qubits, effectively removing them – while the cloud provider does not know this is happening since they can only see the input circuit which includes the actual and dummy qubits and actual and dummy gates in the transpiled circuit specification.

B. Pulse Protection Map (PPM)

We propose to save the re-arrangement information in a new data structure called Pulse Protection Map (PPM). The PPM can be viewed as a list containing switching information for each time period. The transpiled circuit can be easily modified to quantize the time into fixed time periods. For example, single-qubit gates on IBM Quantum today execute in 160dt time, while two-qubit gates have variable timing, but can be easily padded so that each two-qubit gate takes a duration that is a fixed multiple of 160dt. As a result, the PPM can be viewed as a list of control bits specifying the switching of pulses at each time period. The PPM needs to be encrypted so that the cloud provider does not access it. Inside the trusted hardware, the PPM can be decrypted. The decrypted control bits can be used directly to control the Beneš Network, and no real-time computation is needed – simply the control bits from PPM can be sent to the RF switches.

C. Attenuation and Phase Map (APM)

While control pulses can be easily switched between different channels, the same control pulse on different channels is slightly different. For example, the control pulse specifying `X` gate executing on qubit 0 may have a different amplitude than the control pulse specifying `X` gate executing on qubit 1. The phase of the pulses may also have to be adjusted. Further, since we want to enable switching any pulse to any channel, the input pulse should have the maximum amplitude of all the channels. Thus, in the transpiled circuit, all the control pulses are at maximum amplitude. This means, regardless of the

circuit, the input pulses going to the quantum computer will be at maximum amplitude, and the attenuation amount does not depend on the circuit. The attenuation amount and any phase change information can be stored in the Attenuation and Phase Map (APM). This is public information, since the properties of the quantum computer, such as shapes and amplitudes of control pulses, are known. The APM can be stored on trusted hardware without any protection. The APM is used by our trusted CASQUE logic for amplitude and phase controller (APC) which we introduce into the dilution refrigerator.

D. Measurement Obfuscation

After all the pulses are switched and the circuit finishes execution, there is a need to protect the measurements. To obfuscate the measurement results, the best approach is to randomly flip qubits before measurement by using `X` gates at the end of the circuit. However, gates cannot be simply "added" to the circuit in the dilution refrigerator. As an alternative, we assume the user transpiles his or her circuit with a layer of `X` gates at the end of the circuit, on half of the qubits used by the circuit qubits. During execution of the circuit, each `X` gate will be randomly switched to a different channel. Half of the qubits will not have `X` gate applied and their output is not flipped, while the other half will have the `X` gate applied and the output will be flipped. Assuming there are n qubits used, then there are $\binom{n}{(n/2)}$ possibilities for which qubits are flipped and which are not.

The switching of the final gates can be further determined at run-time, so for each shot different qubits have their output flipped. To support this, TRNG can be used to generate randomness used to determine which qubits to flip. This information about which `X` gates actually, i.e. which qubits were flipped, needs to be encrypted and sent back to the user so he or she can recover the correct outputs. We note that, as a side benefit, the addition of `X` before measurements may also help to reduce measurement errors [45].

E. Software Overview

To realize the CASQUE architecture, both software and hardware modifications are needed. On the user's software end, there are a number of modifications that are required by CASQUE to be made to the user's circuits. First, the user needs to transpile their circuit for the target quantum computer. Then, subsequent steps required by CASQUE are listed below and shown schematically in Figure 2.

1) Pad each `CNOT` gate with delays such that all `CNOT` gates with their associated padding take the same amount of time, the duration including padding should be a fixed multiple of single-qubit gate duration
2) Add padding between other gates if necessary to "line up" all the gates so that each gate starts at a time that is a multiple of single-qubit gate duration
3) Increase the amplitude of each control pulse to the maximum amplitude needed by any channel
4) For each time period, randomly switch the pulses between channels and add a layer of gates for flipping the output
5) Save the switching information in the pulse protection map (PPM)
6) Encrypt PPM with the cryptographic key associated with the trusted CASQUE hardware inside the target

The transpiled circuit (with all the modifications and with pulses switched between channels) is sent to the cloud provider, along with the encrypted PPM.

F. Hardware Overview

On the hardware end, the pulses need to be switched to the correct channels, and possibly have their amplitude or phase adjusted. We assume the control pulses are correctly generated by the cloud provider, based on the user's transpiled circuit that they received. The untrusted cloud provider who we assume can passively try to spy on the information should not learn details of the circuit since they do not know how the pulses are switched and on which qubits they actually execute. The encrypted PPM cannot be read by the cloud provider who does not have the decryption keys. However, the encrypted PPM is sent to the hardware before the circuit executes so that the trusted CASQUE Logic hardware can decrypt the PPM and send the control bits to the switches and amplitude and phase control. Compared to an unmodified quantum computer, a number of operations are performed on the control pulses which are input to the dilution refrigerator.

1) For each time period, based on the control bits from the PPM, the incoming pulses need to be switched to the correct channels; at each time period, the control bits are loaded into the Beneš Network to re-configure the routing of the incoming signals
2) After the pulses pass the Beneš Network, they pass through amplitude and phase control; the APM specifies the attenuation and any phase shifting needed
3) Finally, the control pulses are now sent to the mixers so that the incoming control signals can be mixed with the carrier signal at the frequency matching the channel
4) For the output protection, at runtime, TRNG generates random bits used to determine how the final `X` gates will be switched, this information is also encrypted and sent back to the user

The details of the hardware design, as well as the reasons for including the mixers inside the dilution refrigerator (while keeping the waveform generators outside), are discussed next.

V. Design of CASQUE Hardware

Switching control pulses requires more than just redirecting the control signals to different qubits. We need to consider other features of the control signals, such as the frequency or amplitude of the signals.

A. Pulse Switching with Beneš Network

Swapping of the control pulses between qubits and couplings can be realized by a Beneš network. A Beneš network can be used to switch signals between N inputs and N outputs, where any rearrangement of the inputs can be achieved without blocking. Beneš network of N inputs has $2 \times log_2(N) - 1$

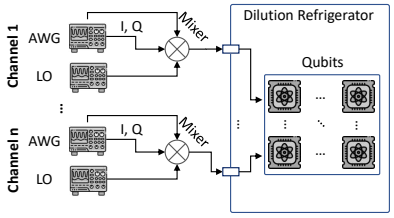


Fig. 3: Schematic of a typical superconducting quantum computer showing arbitrary waveform generators (AWGs) with local oscillators (LOs) and mixers used to mix the I,Q pulses onto the target qubit's or coupling's frequency.

stages, each containing $N/2$ two-by-two crossbar switches, and use a total of $N \times log_2(N) - N/2$ two-by-two crossbar switches. For specifying switching, rather than specify for each qubit or coupling channel its target channel, we can instead directly provide control bits for each switch in the Beneš network at each time period. Recall that all gates in transpiled circuit are padded to that they each start at a time that is multiple of single-qubit duration. In effect, the circuits are quantized into fixed time periods. In each time period, $N \times log_2(N) - N/2$ bits are needed (i.e. one bit per switch to determine if the switch should exchange its inputs, or let them pass unchanged).

The Beneš network's switches can be realized with standard RF switches. For example, the CMD272P3 [39] is a low loss broadband positive control double-pole, double-throw (DPDT) transfer switch. The CMD272P3 covers DC to 10GHz and offers a low insertion loss of 1.6dB and high isolation of 43dB at 5GHz, which is the target frequency for many superconducting qubit quantum computers. The CMD272P3 operates using complementary control voltage logic lines of 0/+5V that can be easily generated by the control logic by converting typical single-ended outputs to differential pair outputs.

B. Frequency Adjustment

In a superconducting qubit quantum computer, each qubit has a target frequency. Pulses generated for one qubit will not work on another if the frequency is incorrect. Figure 3 shows a schematic of a typical superconducting quantum computer showing arbitrary waveform generators (AWGs) with local oscillators (LOs) and mixers used to mix the I,Q pulses onto the target qubit's or coupling's frequency. The AWGs, LOs, and mixers are located outside the dilution refrigerator.

In CASQUE, the Beneš network can switch pulses, but cannot adjust the frequency. Further, we do not want to introduce any high-power equipment, such as signal generators, into the dilution refrigerator. A solution we propose for CASQUE is to move the mixers into the dilution refrigerator. The I and Q pulses are at the same frequency for all qubits, it is only the frequency of the LO that changes for each qubit. Thus, the I and Q pulses can be sent through the switching network to switch the gates. Once the pulses are switched, they can be mixed with the LO signal.

C. Amplitude and Phase Adjustment

One of the main differences between the control pulses, besides the frequency, is the amplitude of the pulse. As discussed before, in CASQUE all pulses are sent at the highest amplitude, and then attenuated. The amplitude adjustment can be achieved with a voltage-controlled attenuator, for example, F2258NLGK8 from Renesas [35]. The example attenuator works in range up to 6GHz, with 1.4dB insertion loss and the control voltage can be from 0V to 3.6V. To generate the attenuation control voltage from digital information, a digital-to-analog converter is needed. Assuming 10 bits of resolution for the digital-to-analog, we can control the amplitude with resolution of over 0.001%. Detailed analysis of the type of the needed attenuation levels, and any residual impact of the attenuation on the control signals, is left for future work.

The phase of the control pulses also affects their operation. The phase information is stored in I and Q signals. If needed, the phase may have to be modified before the I and Q signals reach the mixers. Phase adjustment can be achieved with a phase shifter, for example, HMC649A from Analog Devices [3]. The example phase shifter works in 3GHz to 6GHz frequency range, with 8dB insertion loss, and has a resolution of 6 bits, corresponding to phase adjust resolution of 5.625 degrees.

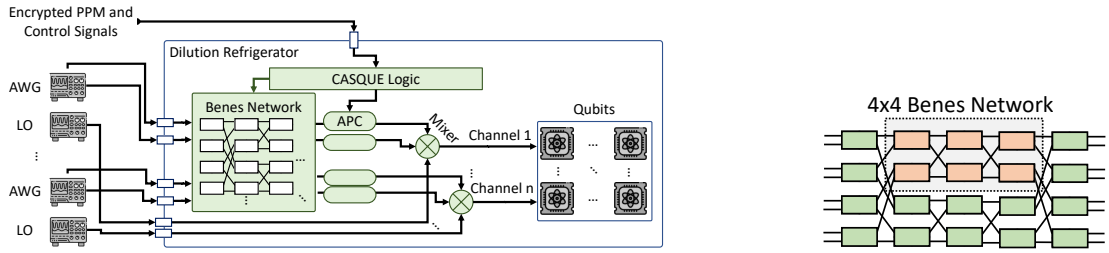
D. Combined Hardware Modifications

Figure 4a shows the combined modifications to the quantum computer hardware that CASQUE introduces. The additions inside the dilution refrigerator include the CASQUE logic, switching network, APC and mixers. These are the trusted components. Further, the CASQUE logic is made of one or more decryption tiles and engines discussed later to support various sizes of switching networks (and the number of control bits they require). The only modification to the untrusted hardware outside the dilution refrigerator is the removal of the mixers, which are now moved inside the fridge.

E. PPM and APM Data Size and Bandwidth

The Beneš network requires $N \times log_2(N) - N/2$ control bits for each time period to allow for arbitrary switching of N channels. Further, $10 \times N$ control bits are needed for the amplitude control on N channels, and $6 \times N$ control bits may be needed for phase adjustment.

The control bits for Beneš network are stored in PPM, and at each time period The Beneš network requires $N \times log_2(N) - N/2$ bits need to be provided from the PPM to switch the switches. Meanwhile, control data related to amplitude and phase is stored in the APM. These control bits from APM only need to be provided before the circuit starts to execute. At each time period, for each channel, it is only needed to specify the type of gate, so the appropriate APM data can be used to adjust the amplitudes, for example. In IBM Quantum there are `X`, `SX`, and `CX` basis gates that use real control pulses, thus $2 \times N = log_2(3) \times N$ control bits are needed for each time period, in addition to the switching bits. The time period corresponding to the duration of single-qubit gates is



(a) (b)

Fig. 4: (a) Combined hardware modifications for support of CASQUE. The additions inside the dilution refrigerator include the CASQUE logic, Beneš Network, Amplitude and Phase Control (APC), and the Mixers. (b) An Example of 8×8 network built from 4×4 networks.

today 160dt in IBM quantum computers, which is equivalent to 35.5ns. This corresponds to a frequency of 28.5MHz and which the switching network control bits need to be provided.

1) Kookaburra Example: As an example, we can consider 1,386-qubit Kookaburra quantum processing unit, which according to IBM roadmap [16] should be available in a few years. For 1,386-qubit Kookaburra, we have 1,386 qubit drive channels, and a similar number of control channels. Because Beneš network requires a power-of-two number of inputs, we can use the next closest power of two, which is $N = 2,048$ for a number of inputs for qubit channel switching.

With $N = 2,048$, the number of control bits needed at each time period is 21,504 for determining swapping qubits. The size of PPM for Kookaburra would then be on the order of $21,504 \times D$, where D is the depth of the transpiled circuit. Regardless of the depth of the transpiled circuit, at each time period, Kookaburra would need 21,504 control bits sent to the switching network plus 2,772 bits for specifying gate type (for amplitude and phase adjustment). At 28.5MHz, this is approximately 75GB/s bandwidth that is needed for communication between the CASQUE logic and the switching network.

F. Security-Performance Tradeoff with Tiled Switching Network Design

According to recent IBM roadmaps, large quantum computer systems will be built from smaller quantum computers or quantum computer chips on the order of 1000 qubits, such as the 1,386-qubit Kookaburra quantum processing unit that can be replicated multiple times to build a larger computer. In a basic approach, CASQUE hardware can be instantiated for each quantum processing unit. Multiple CASQUE hardware can work in parallel, each with its associated quantum processing unit. While approximately 75GB/s bandwidth between each CASQUE logic and its associated switching network is high, it is within reach of today's electronics (e.g. DDR5 can provide 51 GB/s data bandwidth per module, or HBM can provide 256 GB/s data bandwidth). However, so many switches are not needed in practice. From a security perspective, if we consider even circuits with an order of 6 qubits and 100s of gates, our evaluation in Section VI, shows the complexity can be above 2^{256}. Thus, if we only switch a few channels, the attack complexity is huge and there is no need to switch all the channels each time and the 75GB/s bandwidth is unlikely to be actually needed, order or two smaller bandwidth could be sufficient when only smaller number of qubits are switches – while still having high security level.

TABLE I: Number of control bits, number of possible permutations, and corresponding bandwidth for different sizes of Beneš network. Bandwidth computation is based on the number of control bits that need to be provided in each 35.5ns time period.

Size	Perm.	Ctrl. Bits	Bandwidth	Num. Dec. Tiles
2×2	2^1	1	3.5MB/s	1
4×4	$\sim 2^{4.5}$	6	21.1MB/s	1
8×8	$\sim 2^{15.3}$	20	70.4MB/s	1
16×16	$\sim 2^{44.3}$	56	197.2MB/s	1
32×32	$\sim 2^{117.6}$	144	507.0MB/s	2
64×64	$\sim 2^{296.0}$	352	1,873.0MB/s	3

1) Tiling of the Switching Network: $N \times N$ Beneš network is actually built from two $N/2 \times N/2$ networks with added layers. An example of 8×8 a network built from 4×4 networks is shown in Figure 4b. As a result, it is simple to trade off the number of possible permutations vs. the number of control bits (and thus bandwidth) needed. Table I shows different sizes of networks and the associated number of permutations, needed control bits, and bandwidth. Based on the different sizes of the network, a different number of control bits is needed. We note that in our design of the decryption engine which decrypts the PPM, discussed next, we use the AES algorithm, which has a block size of 128 bits. Each decryption tile can provide 128 bits in a time period. Depending on the size of the switching network, multiple tiles can be used in parallel.

G. CASQUE Logic and Decryption Engine Design

In this work, we also implement a hardware design for the decryption engine, which is a key part of the CASQUE logic. The decryption engine helps decrypt the encrypted PPM inside the trusted hardware. For our evaluation, we consider AES-GCM algorithm to encrypt the PPM on the user's end. We use an existing AES-GCM module [22] which can perform both encryption and decryption and implement a wrapper consisting of a controller which helps in loading the encrypted PPM from the BRAM or DRAM, decrypt it, and write it back to the same memory location block by block. We name this module as `decrypt_tile` (shown in Figure 5). The `decrypt_tile`

has the capability of generating parameterized output width for these decrypted PPM bits. Users can choose the width arbitrarily based on the bandwidth requirement. Furthermore, the `decrypt_tile` also supports a high-performance mode where multiple `decrypt_tile` could be stacked together to decrypt large PPMs efficiently. The key required by the AES-GCM module is established using a quantum-safe public key algorithm, e.g., Classic McEliece [2]. In our hardware design, for the public key algorithm, we use the `mceliece38864` decapsulation module described in Section 5.4 of [7]. Other quantum-safe algorithms could be used as well.

In addition to the decryption engine, to protect the output generated from the quantum computer (discussed in Section IV-D), we also implement a True Random Number Generator (TRNG). For the `TRNG` module, we use an existing `SHAKE256` module [7] and implement a wrapper around it to feed an arbitrary size seed (we assume that there is a random number generator (RNG) inside the trusted hardware boundary that provides uniformly distributed random bits to our module as an initial seed) and squeeze out the required number (N) of random bits (*RB*), where N is the number of qubits for the given quantum computer. As discussed in Section IV-D, based on these bits, an `X` gate is either applied or not on the output of the quantum circuit. The wrapper also facilitates the encryption of *RB* using the `AES-GCM` module from the `decrypt_tile`, which is sent to the user.

The `TRNG` module uses the `SHAKE256` with the smallest performance parameter configuration, i.e, `parallel_slices = 1` (described in [7]). The `top_module` combines all other modules as shown in Figure 5. The hardware utilization of our CASQUE hardware module is as follows: 3,340 LUTs, 1,158 FFs, 10 BRAMs, and it operates at a frequency of 103 MHz when targeted to Xilinx Artix 7 `xc7a100t`. To handle a 1 MB PPM, our hardware module takes 7.7 ns. These results do not include the resources used for the public-key algorithm. we do not report the BRAM utilization required for the PPM storage. This is because the size of the PPM changes as per the target quantum computer and quantum circuit. For large quantum computers and large circuits, the size of the on-chip BRAM may not be enough for the PPM storage, in which case we could use off-chip storage units such as DRAM. Our CASQUE hardware design supports the usage of either of them. We also note that we chose to target a lightweight hardware implementation for this evaluation. However, the parameterizable capability of our design allows switching to a high-speed parallel implementation easily.

1) Serializer-Deserializer for Outputs: Although the operating frequency of the CASQUE hardware module is higher (103 MHz), the frequency at which the pulse switching happens is much slower (i.e., 28.5 MHz, as described in Section V-E). One possible solution to tackle this is to run the complete design at the (slower) switching frequency, but the bandwidth requirement of the Beneš network may not be met. Consequently, we run our design at the fast frequency and use an asynchronous FIFO (`ASYNC_FIFO`) to handle the crossing clock domains. Figure 5 shows usage of two sets of `ASYNC_FIFO`s, one for switching network and the other to control the application of `X` gate on the output. From the (`ASYNC_FIFO`), the decrypted PPM output is then loaded into the Serializer-Deserializer to arrange bits as per required bandwidth. We note that we successfully conducted practical experiments by running our hardware design on Xilinx Artix 7 `xc7a100t` FPGA and interfacing them with the RF switches (described in Section V-A). We simulate the quantum computer control pulses, going into the RF switches, using a lab signal generator; and an oscilloscope is used to validate that the switches attenuate the pulses when needed.

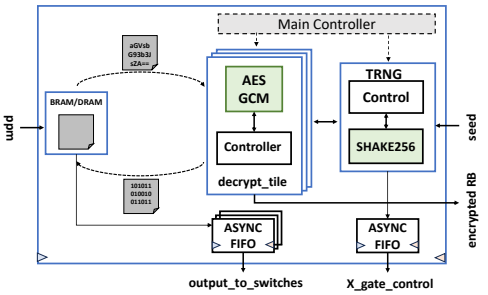


Fig. 5: Top-level design of CASQUE hardware module, which implements key parts of the CASQUE logic. Quantum-safe public-cryptographic modules (`mceliece38864` decapsulation) for establishing the shared secret for use in the AES would be extra hardware.

VI. Evaluation

Our evaluation focuses on fidelity evaluation using Variational Distance (VD), as well as computation of complexity of how many circuit the attacker would have to try based on the obfuscation provided by the switching of the pulses. We use selected QASMBench benchmarks [23] for the evaluation. Fidelity evaluation is done on the 7-qubits real IBM Perth quantum machine.

A. Fidelity Evaluation

Assuming ideal operation of the switches and other added components, we focus on the impact of added delays due to "lining up" of gates done as part of CASQUE software steps. When `CNOT` gates are padded with delays to have fixed duration, we observe that with more delays, the variational distance increases, i.e. the fidelity decreases. For this reason, we measure the Variational Distance (VD) between each of the eight selected benchmarks and the unmodified benchmark from the QASMBench benchmarks suite. Informally, the variational distance of two output probability distributions is the measure of how one probability distribution is different from the other. In general, the total variation distance between P and Q is defined as: $\delta(P,Q) = \frac{1}{2}\sum|P-Q|$.

The impact of the extended duration of the selected benchmarks is presented in Table II.

B. Security Analysis

In the computation of the security level and the security analysis, we take into account the following aspects: number of

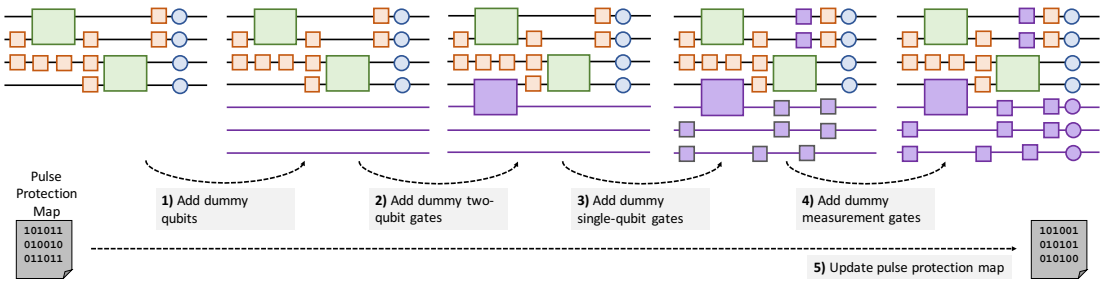


Fig. 6: Software steps in the extended CASQUE+ architecture.

TABLE II: Variational Distance (VD) metric for selected QASM-Bench benchmarks [23]. The Qubits, Gates (single-qubit gates) and CNOT numbers are pre-transpilation.

Benchmark	Qubits	Gates	CNOT	VD
wstate	3	30	9	0.126
basis_change	3	53	10	0.116
variational	4	54	16	0.080
vqe	4	89	9	0.195
qec_en	5	25	10	0.642
error_correctiond3	5	114	49	0.184
simon	6	44	14	0.195
qaoa	6	270	54	0.203

qubits (n_{qubits}), total number of single-qubit slots (p), number of single-qubit gate pulses in each slot (ms_i), total number of CNOT slots (q), number of single-qubit slots within duration of a CNOT slot (r), number of CNOT gate pulses we have in each CNOT slot (c_j), number of single-qubit gate pulses in each slot within a CNOT slot ($mc_{j,k}$):

$$C = \prod_{i=0}^{p-1} \binom{n_{qubits}}{ms_i} \times \prod_{j=0}^{q-1} \left(\prod_{k=0}^{r-1} \binom{(n_{qubits} - (2 \times c_j))}{mc_{j,k}} \right) \quad (1)$$

In the above equation, we use $\binom{n}{k}$ notation to represent n choose k computation. In our implementation, each CNOT gate slot is padded with delays such that all CNOT gates within the circuit take the same duration of time (equal to the duration of the longest CNOT gate on any of the coupling in the target quantum computer). The CNOT gate duration of time including padding is set to be a multiple of the single-qubit gate slot duration, currently 160dt. Since all CNOT gate slots are of the same duration, regardless which CNOT slot j is considered, the number of single-qubit gate slots that fit within the CNOT slot is the same and equal to r. As result r depends on the quantum computer backend which determines the duration of CNOT gates. Meanwhile, p, q, ms_i, $mc_{j,k}$, and c_j depend on the user's circuit, its structure and the gates used. n_{qubits} depends on the number of qubits used by the user's circuit, but clearly cannot be larger than the number of qubits available on the target quantum computer. The approximate attack complexity on selected bencharks is shown in Table III. The table also shows the complexity for CASQUE+ extended architecture, discussed next.

VII. CASQUE+ Architecture

Our proposed CASQUE provides very good protection at a very high obfuscation level. However, we observed that a novel application of the switches can be realized if we assume that additional dummy qubits can be added to the user's circuit. Specifically, if dummy qubits (qubits not otherwise used by the original circuit) are added to the design, then it is possible to switch control pulses from the other qubits to the dummy qubits. Pulses switched to the dummy qubits do not affect nor perform any useful computation, and thus they are effectively eliminated from the circuit. Further, pulses can be added to the dummy qubits, and pulses can be switched among the dummy qubits, which also does not affect the actual computation.

A. Increasing Obfuscation with Added Dummy Qubits and Dummy Gates

Adding dummy qubits and dummy gates increases the number of possibilities for switching the gates – increasing the complexity for the attacker. Considering our prior Equation 1, adding dummy qubits increases n_{qubits}. Adding dummy single-qubit gates increases ms_i and $mc_{j,k}$. Adding two-qubit gates increases c_j. However, the complexity increases further as in Equation 1 and baseline CASQUE design we do not alter the two-qubit CNOT gates. Now, with dummy qubits, we can add dummy two-qubit gates on original qubits and then switch them to the dummy qubits to eliminate them. As a result, the attacker (cloud provider) no longer is certain that a CNOT gate will execute (as it did in CASQUE). Rather, each CNOT could be a real gate that executes or could be a gate that is eliminated (by switching it to a dummy qubit) before execution. The updated complexity for the number of possible circuits that the untrusted cloud provider would have to guess from is:

$$C' = \prod_{i=0}^{p-1} \binom{n_{qubits}}{ms_i} \times \prod_{j=0}^{q-1} \left(\left(\prod_{k=0}^{r-1} \binom{(n_{qubits} - (2 \times c_j))}{mc_{j,k}} \right) \times 2^{c_j} \right) \quad (2)$$

B. Preventing Dummy Qubit Detection

Although the attacker does not know which are the dummy qubits from the input circuit, they could use the circuit structure to guess the dummy qubits. First, if the circuit can be partitioned into two disjoint circuits not connected by a two-qubit gate, then the attacker could easily say that one of the two circuits is made up of the dummy qubits. Thus, we

TABLE III: Approximate attack complexity on selected QASMBench benchmarks [23]. Complexity calculated for zero, two, four and eight number of added dummy qubits. The Qubits, Gates (single-qubit gates) and CNOT gates numbers are pre-transpilation.

Benchmark	Qubits	Gates	CNOT	Complexity			
				0 dummy qubits (CASQUE)	w/ 2 dummy qubits (CASQUE+)	w/ 4 dummy qubits (CASQUE+)	w/ 8 dummy qubits (CASQUE+)
wstate	3	30	9	2^{8}	2^{15}	2^{19}	2^{25}
basis_change	3	53	10	2^{43}	2^{66}	2^{80}	2^{100}
variational	4	54	16	2^{20}	2^{26}	2^{30}	2^{36}
vqe	4	89	9	2^{45}	2^{64}	2^{77}	2^{95}
qec_en	5	25	10	2^{17}	2^{22}	2^{26}	2^{32}
error_correctiond3	5	114	49	2^{103}	2^{132}	2^{155}	2^{191}
simon	6	44	14	2^{20}	2^{25}	2^{28}	2^{33}
qaoa	6	270	54	2^{268}	2^{324}	2^{365}	2^{428}

must ensure that there is at least one dummy two-qubit gate that connects one of the original qubits with one of the dummy qubits. Second, if there is no measurement on a qubit, it can be identified as an ancillary qubit (actually used by the circuit, but not measured) or as the dummy qubit. Thus, we must ensure there is a measurement gate on all qubits, so that all qubits look like they are part of the circuit, even if the measurement will be discarded by the user.

C. Switching of Multi-qubit Gates

The current design of CASQUE does not switch multi-qubit gates due to the complexity of the waveform of the multi-qubit gates. However, this can also be done with enough knowledge of the waveform of the multi-qubit gates. For example, if the `CNOT` gates on different qubit pairs have the same waveform, then this can be easily done with the same scheme as in CASQUE. Nevertheless, the current design of the `CNOT` gate on the IBM cloud introduces different duration and pulse patterns, which may require additional hardware, such as the hardware to change the duration of pulses. The study of switching multi-qubit gates is left for future work, but we acknowledge that switching `CNOT` gates would further increase attack complexity.

D. Operation of CASQUE+

Figure 6 shows schematically the steps of extending the user's circuit with CASQUE+ protections. We assume that the input is the CASQUE protected circuit. The steps are done on top of, or in addition to, the protection applied by CASQUE. The steps to add dummy qubits and dummy gates are:

1) Add dummy qubits to the circuit
2) Randomly add two-qubit gates in `CNOT` gate slots, ensure that at least in one slot one of the added gates uses both real and dummy qubits
3) Randomly add single qubit gates
4) Add measurement gates on all qubits
5) Update pulse protection map

E. Analysis of Increased Complexity

Based on Table III, we can achieve higher complexity by adding additional dummy qubits to the circuits. We tested for two, four, and eight additional dummy qubits for every application. We observe the highest complexity for the 6-qubit benchmark *qaoa*, as this benchmark has the highest number of gates and `CNOT` gates among all the tested benchmarks. We acknowledge that in CASQUE+ architecture, some extra qubits for the machine are occupied and prevented from being used by other applications to run in parallel, however, we consider this to be a minor issue in larger quantum machines.

VIII. Related Work

Relevant sets of research include work on blind quantum computation, quantum homomorphic encryption and obfuscation of quantum computation. Considering protection of programs from untrusted cloud computing providers, BQC has been explored by various researchers [1], [6], [11], [14], [19], [26], [28]–[31], [33], [34], [42]. Unlike our work, blind quantum computation remains mostly theoretical since it assumes existence of quantum networking to connect to the cloud-based quantum computer, as well as, a trusted quantum computer owned by the user. On the other hand, QHE facilitates the inclusion of classical clients on a single server, yet it encounters significant challenges, including exponential computational overheads and the necessity for advanced quantum error correction [4], [15], [25], [41], [44]. Our work removes need for these assumptions by leveraging trusted hardware inside the quantum computer.

Our research aligns with recent works on arXiv, particularly [37], which lacks trusted hardware and adopts a compile-time strategy, allowing users to insert RX gate pairs for confusion. In contrast, our dynamic approach forces the untrusted provider to test an exponential number of gate-switching combinations. In [46], [47], trusted hardware is used to attenuate decoy control pulses, akin to our work, but we employ additional trusted hardware without extending user circuits with idle decoy pulses.

IX. Conclusion

This work introduced a novel hardware architecture, CASQUE, for securing computation on superconducting qubit quantum computers by enabling dynamic runtime pulse switching. CASQUE+ enhanced the protections further by allowing users to add dummy qubits and dummy gates for heightened security. The proposed protections can be especially useful in cloud-based quantum computing systems to help protect from untrusted cloud providers. The architectures allow users to select different obfuscation levels to balance security and resource use.

REFERENCES

[1] Dorit Aharonov, Michael Ben-Or, and Elad Eban. Interactive proofs for quantum computations, 2008. https://arxiv.org/pdf/0810.5375.pdf.

[2] Martin R. Albrecht, Daniel J. Bernstein, Tung Chou, Carlos Cid, Jan Gilcher, Tanja Lange, Varun Maram, Ingo von Maurich, Rafael Misoczki, Ruben Niederhagen, Kenneth G. Paterson, Edoardo Persichetti, Christiane Peters, Peter Schwabe, Nicolas Sendrier, Jakub Szefer, Cen Jung Tjhai, Martin Tomlinson, and Wen Wang. Classic McEliece. Technical report, National Institute of Standards and Technology, 2020. available at https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions.

[3] Analog. Hmc649a documentation.

[4] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. A guide to fully homomorphic encryption. *Cryptology ePrint Archive*, 2015.

[5] Francesco Bova, Avi Goldfarb, and Roger G Melko. Commercial Applications of Quantum Computing. *EPJ quantum technology*, 8(1):2, 2021.

[6] Anne Broadbent, Joseph Fitzsimons, and Elham Kashefi. Universal blind quantum computation. In *2009 50th Annual IEEE Symposium on Foundations of Computer Science*, pages 517–526, 2009. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5438603.

[7] Po-Jen Chen, Tung Chou, Sanjay Deshpande, Norman Lahr, Ruben Niederhagen, Jakub Szefer, and Wen Wang. Complete and improved fpga implementation of classic mceliece. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(3):71–113, Jun. 2022.

[8] Andrew M Childs. Secure assisted quantum computation. *arXiv preprint quant-ph/0111046*, 2001.

[9] Victor Costan and Srinivas Devadas. Intel sgx explained. *Cryptology ePrint Archive*, 2016.

[10] Andrew Cross. The ibm q experience and qiskit open-source quantum computing software. In *APS March meeting abstracts*, volume 2018, pages L58–003, 2018.

[11] Vedran Dunjko, Elham Kashefi, and Anthony Leverrier. Blind quantum computing with weak coherent pulses. *Physical Review Letters*, 108(20), may 2012. https://arxiv.org/pdf/1108.5571.pdf.

[12] Joseph F Fitzsimons. Private quantum computation: an introduction to blind quantum computing and related protocols. *npj Quantum Information*, 3(1):23, 2017.

[13] Joseph F Fitzsimons and Elham Kashefi. Unconditionally verifiable blind quantum computation. *Physical Review A*, 96(1):012303, 2017.

[14] Joseph F. Fitzsimons and Elham Kashefi. Unconditionally verifiable blind quantum computation. *Physical Review A*, 96(1), jul 2017. https://arxiv.org/pdf/1203.5217.pdf.

[15] Caroline Fontaine and Fabien Galand. A survey of homomorphic encryption for nonspecialists. *EURASIP Journal on Information Security*, 2007:1–10, 2007.

[16] Jay Gambetta. Ibm's roadmap for scaling quantum technology. *IBM Research Blog (September 2020)*, 2020.

[17] Alexandru Gheorghiu, Elham Kashefi, and Petros Wallden. Robustness and device independence of verifiable blind quantum computing. *New Journal of Physics*, 17(8):083040, 2015.

[18] Alexandru Gheorghiu, Petros Wallden, and Elham Kashefi. Rigidity of quantum steering and one-sided device-independent verifiable quantum computation. *New Journal of Physics*, 19(2):023043, 2017.

[19] Vittorio Giovannetti, Lorenzo Maccone, Tomoyuki Morimae, and Terry G. Rudolph. Efficient universal blind quantum computation. *Physical Review Letters*, 111(23), dec 2013. https://arxiv.org/pdf/1306.2724.pdf.

[20] Constantin Gonzalez. Cloud based qc with amazon braket. *Digitale Welt*, 5:14–17, 2021.

[21] He-Liang Huang, Qi Zhao, Xiongfeng Ma, Chang Liu, Zu-En Su, Xi-Lin Wang, Li Li, Nai-Le Liu, Barry C Sanders, Chao-Yang Lu, et al. Experimental blind quantum computing for a classical client. *Physical review letters*, 119(5):050503, 2017.

[22] Emilia Käsper and Peter Schwabe. Faster and timing-attack resistant aes-gcm. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 1–17. Springer, 2009.

[23] Ang Li, Samuel Stein, Sriram Krishnamoorthy, and James Ang. Qasmbench: A low-level quantum benchmark suite for nisq evaluation and simulation. *ACM Transactions on Quantum Computing*, 2022.

[24] Qin Li, Chengdong Liu, Yu Peng, Fang Yu, and Cai Zhang. Blind quantum computation where a user only performs single-qubit gates. *Optics & Laser Technology*, 142:107190, 2021.

[25] Urmila Mahadev. Classical homomorphic encryption for quantum circuits. *SIAM Journal on Computing*, (0):FOCS18–189, 2020.

[26] Atul Mantri, Carlos A. Pé rez-Delgado, and Joseph F. Fitzsimons. Optimal blind quantum computation. *Physical Review Letters*, 111(23), dec 2013. https://arxiv.org/pdf/1306.3677.pdf.

[27] Matthew McKague. Interactive proofs for bqp via self-tested graph states. *arXiv preprint arXiv:1309.5675*, 2013.

[28] Tomoyuki Morimae. Continuous-variable blind quantum computation. *Physical Review Letters*, 109(23), dec 2012. https://arxiv.org/pdf/1208.0442.pdf.

[29] Tomoyuki Morimae. Verification for measurement-only blind quantum computing, 2014. https://arxiv.org/pdf/1208.1495.pdf.

[30] Tomoyuki Morimae, Vedran Dunjko, and Elham Kashefi. Ground state blind quantum computation on aklt state, 2011. https://arxiv.org/pdf/1009.3486.pdf.

[31] Tomoyuki Morimae and Keisuke Fujii. Blind topological measurement-based quantum computation. *Nature Communications*, 3(1), sep 2012. https://arxiv.org/pdf/1110.5460.pdf.

[32] Tomoyuki Morimae and Keisuke Fujii. Blind quantum computation protocol in which alice only makes measurements. *Physical Review A*, 87(5):050301, 2013.

[33] Tomoyuki Morimae and Keisuke Fujii. Blind quantum computation protocol in which alice only makes measurements. *Physical Review A*, 87(5), may 2013. https://arxiv.org/pdf/1201.3966.pdf.

[34] Tomoyuki Morimae and Takeshi Koshiba. Composable security of measuring-alice blind quantum computation, 2013. https://arxiv.org/pdf/1306.2113.pdf.

[35] Newark. f2258nlgk8 rf attenuator documentation.

[36] Victory Omole, Akhilesh Tyagi, Calista Carey, AJ Hanus, Andrew Hancock, Austin Garcia, and Jake Shedenhelm. Cirq: A python framework for creating, editing, and invoking quantum circuits, 2020.

[37] Tirthak Patel, Daniel Silver, Aditya Ranjan, Harshitta Gandhi, William Cutler, and Devesh Tiwari. Toward privacy in quantum program execution on untrusted quantum cloud computing machines for business-sensitive quantum needs, 2023.

[38] Kumar Prateek and Soumyadev Maity. Quantum programming on azure quantum—an open source tool for quantum developers. In *Quantum Computing: A Shift from Bits to Qubits*, pages 283–309. Springer, 2023.

[39] Qorvo. Cmd272p3 dpdt transfer switch documentation.

[40] Ben W Reichardt, Falk Unger, and Umesh Vazirani. Classical command of quantum systems. *Nature*, 496(7446):456–460, 2013.

[41] Ronald L Rivest, Len Adleman, Michael L Dertouzos, et al. On data banks and privacy homomorphisms. *Foundations of secure computation*, 4(11):169–180, 1978.

[42] Takahiro Sueki, Takeshi Koshiba, and Tomoyuki Morimae. Ancilla-driven universal blind quantum computation. *Phys. Rev. A*, 87:060301, Jun 2013.

[43] Jakub Szefer. Principles of secure processor architecture design. In *Principles of Secure Processor Architecture Design*, pages 113–124. Springer, 2018.

[44] Si-Hui Tan, Joshua A Kettlewell, Yingkai Ouyang, Lin Chen, and Joseph F Fitzsimons. A quantum approach to homomorphic encryption. *Scientific reports*, 6(1):33467, 2016.

[45] Swamit S Tannu and Moinuddin 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*, pages 279–290, 2019.

[46] Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, and Jakub Szefer. Hardware architecture for a quantum computer trusted execution environment, 2023.

[47] Theodoros Trochatos, Chuanqi Xu, Sanjay Deshpande, Yao Lu, Yongshan Ding, and Jakub Szefer. A quantum computer trusted execution environment. *IEEE Computer Architecture Letters*, 22(2):177–180, 2023.