Implementing Remote-State Preparation on a Noisy-Intermediate Size Quantum Device

Laura Lewis[1], Alexandru Gheorghiu[1], and Thomas Vidick[1]

[1]*Department of Computing and Mathematical Sciences, California Institute of Technology*

September 2020

Abstract

In this research project, we investigate the implementation of a protocol for remotely preparing quantum states, with applications to blind and verifiable delegated quantum computation. The remote-state preparation protocol relies on the use of cryptographic primitives known as *Noisy Trapdoor Claw-Free* (NTCF) functions. These functions are based on conjectured post-quantum secure problems, such as *Learning with Errors* (LWE). The goal of the project is to implement the evaluation of such functions in superposition on a *Noisy-Intermediate Size Quantum* (NISQ) Device, allowing us to assess the potential of NISQ devices for cryptographic applications. We designed and optimized quantum circuits for the implementation of the NTCF functions and analyzed their performance using the quantum programming languages Q#, Qiskit, and Cirq. We also considered different post-quantum secure problems as the basis for these functions (such as *Learning Parity with Noise*, *Learning with Rounding* and *Ring Learning with Errors*) to find circuits that are optimal in terms of depth and required number of qubits.

1 Introduction

The purpose of this project is to delegate and verify the preparation of quantum states on *Noisy Intermediate-Size Quantum* (NISQ) devices. A NISQ device is a quantum computer having a relatively small number of qubits (on the order of 100) and whose gates are subject to noise [1]. This means that every time a quantum operation is performed, there is some probability that an error is introduced into the computation. While it is expected that the detrimental effects of this noise can be mitigated using *quantum error correcting codes* and protocols for *fault-tolerant quantum computation*, such protocols generally require large numbers of qubits [2] and so are not suitable for current and near-term devices. Nevertheless, as the recent demonstration of *quantum computational supremacy* by the Google group showed, it is possible to obtain significant computational advantages from using NISQ devices [3].

But one challenge we face in using these NISQ devices is *verifying* the results they produce. In other words, how can classical users that delegate computations to these NISQ devices efficiently check that the results they obtained are correct?

This question is partially answered by a number of recent works that have proposed protocols for certifiable randomness generation [4], remote-state preparation [5], self-testing of a single quantum device [6] and verification of general quantum computation [7]. All of these protocols make use of a cryptographic primitive defined in [4, 7] known as *Noisy Trapdoor Claw-Free* (NTCF) functions. A trapdoor function is a type of one-way function (a function that is easy to evaluate but hard to invert), that can be efficiently inverted given a secret key known as a trapdoor [4]. A pair of functions f_1, f_2 is claw-free if there is no efficient algorithm to find two inputs x_1, x_2 that satisfy the equality $f_1(x_1) = f_2(x_2)$ [4]. Thus, a trapdoor claw-free function satisfies both of these properties.

Furthermore, the specific family of NTCF functions we are considering are conjectured to be *post-quantum secure*, i.e. even a quantum algorithm should not be able to efficiently find two inputs x_1, x_2 that satisfy the equality $f_1(x_1) = f_2(x_2)$ [4, 8, 7, 5]. This can be accomplished by defining the functions based on a problem called *Learning with Errors* (LWE), first defined in [9]. Informally, the problem is to solve a system of approximate linear equations over a finite field. More precisely, as described in [9, 10], the problem is that given a uniformly random matrix $A \in \mathbb{Z}_q^{m\times n}$ and a vector $y \in \mathbb{Z}_q^m$, we want to find the vector $s \in \mathbb{Z}_q^n$ such that $y = As + e$, for $e \in \mathbb{Z}^m$. Here, the e vector, referred to as an error vector, must be chosen according to a Gaussian distribution, denoted χ [9]. This distribution χ is centered around the 0 vector and so the sampled e vectors have small norm, with high probability. This Learning with Errors problem is conjectured to be post-quantum secure and so we base our NTCF function on LWE. Specifically, the NTCF function is, as defined in [8, 7, 5, 4]

$$g(b, x) = Ax + b \cdot (As + e') + e \pmod q \tag{1}$$

where $b \in \{0, 1\}, A \in \mathbb{Z}_q^{m\times n}, x \in \mathbb{Z}_q^n, s \in \{0, 1\}^n, e' \in \mathbb{Z}_q^m$, and $e \in \mathbb{Z}^m$. Furthermore, $g(b, x)$ can be interpreted as a pair of functions by fixing b:

$$g_0(x) = Ax + e \pmod q$$

$$g_1(x) = Ax + As + e' + e \pmod q$$

With this perspective, we can observe how $g(b, x)$ is approximately two-to-one. Consider the functions g_0, g_1 above, but with inputs related by s:

$$g_0(x) = Ax + e \pmod q$$
$$g_1(x - s) = Ax + e' + e \pmod q$$

Here, $g_0(x)$ and $g_1(x - s)$ only differ by the e' vector. However, since we know that e' is sampled from a probability distribution with high probability on low-weight vectors from the definition of LWE [10], then e' is likely to be low-weight. Thus, $Ax + e$ and $Ax + e' + e$ should be fairly close, allowing the NTCF function to be two-to-one with high probability.

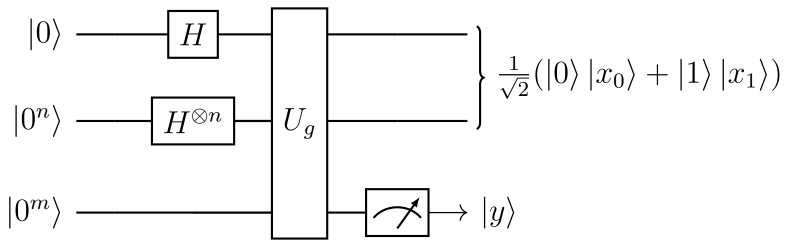
The protocols listed previously regarding certifiable randomness generation [4], remote-state preparation [5], self-testing of a single quantum device [6] and verification of general quantum computation [7] assume that the quantum device can evaluate this NTCF function $g(b,x)$ from Equation 1 *coherently* (i.e. in superposition). However, this cannot be achieved perfectly on a NISQ device because of the errors incurred when performing quantum operations. Nevertheless, we wish to investigate the possibility of implementing claw-free functions on a NISQ device. While it is likely that such an implementation (even a highly optimized one) will incur a significant amount of noise in order to achieve all of the desired the security guarantees, our goal is to see whether we can achieve a balance between accuracy and security. Furthermore, this project mainly focuses on the implementation of the remote-state preparation (RSP) protocol in [5], which makes use of these functions.

The remote-state preparation protocol was developed as a follow-up to Mahadev's protocol for verifiable delegated quantum computation, using only classical communication [7]. Delegated quantum computation is a problem where a (classical) user has a description of a quantum circuit C, and wants to delegate the evaluation of $C|00\dots0\rangle$ to a quantum computer. Importantly, the user wants to be able to efficiently *verify* that the quantum computer performed this computation correctly. In this setting, the user is typically referred to as the *verifier*, whereas the quantum server is the *prover*. While Mahadev's scheme was the first such protocol to utilize only classical communication between the verifier and prover, one disadvantage is that in order to delegate a quantum circuit consisting of G gates, the prover has to prepare a highly entangled quantum state comprising of $poly(G)$-many qubits [7]. In addition, Mahadev's protocol does not achieve a cryptographic property known as *blindness*. Roughly speaking, blindness means that the quantum circuit C is not revealed to the prover throughout the protocol.

To address these limitations, Gheorghiu and Vidick developed the remote-state preparation protocol and showed how it can be used to achieve blind and verifiable delegated quantum computation (using only classical communication) [5]. Their protocol has two stages. First, the verifier delegates to the prover the preparation of certain single-qubit quantum states and checks that these states were prepared correctly. This is the RSP subprotocol, which we are focusing on. Then, the verifier instructs the prover to perform a general quantum computation by suitably entangling and measuring these states according to the protocol of Fitzsimons and Kashefi [11]. The fact that the protocol can be separated into the state preparation phase and the delegated computation phase is in contrast to Mahadev's protocol which has a more monolithic structure. RSP thus provides a way to "dequantize" quantum communication from quantum protocols. We are therefore motivated to see whether this primitive can be implemented on near-term devices.

In particular, at the moment, this project is focused on a portion of the RSP protocol depicted in the diagram below, which implements the evaluation of these NTCF functions as well as a measurement to create a convenient state for the rest of the protocol.

Here the first register is the b input to $g(b,x)$ and the second register is the x input. Furthermore, the unitary operation U_g is the operation to evaluate the function on the first two registers, which is then stored in the third register. After evaluating the function in superposition, we measure the result of the computation, i.e. the third register, to obtain an image of the function y. This measurement collapses the first two registers to the state specified in Figure 1, where we can obtain information about the preimages of $g(b,x)=y$ from this state. This information about the preimages can then lead to tests for verifiability of the protocol. Hence, this circuit that we are implementing is a key part of the RSP



$$g_0(x_0) = g_1(x_1) = y$$

Figure 1: Evaluating NTCF Function Circuit Diagram

protocol, which then has the applications previously mentioned.

Specifically, there are two tests for verifiability: the *preimage test* and *equation test* [4]. Following the preparation of the state $\frac{1}{\sqrt{2}}(|0\rangle|x_1\rangle + |1\rangle|x_2\rangle$ according to the circuit above, the preimage test requests for the prover to measure this state in the computational basis to obtain b, x. Then, the verifier can check that $g(b, x) = y$, and if this is satisfied, the test is passed. On the other hand, the equation test requests a string d, obtained by measuring the collapsed preimage state in the Hadamard basis. In this case, the verifier checks that

$$d \cdot (x_1 \oplus x_2) = b \tag{2}$$

using the trapdoor from g to compute both preimages x_1, x_2. Satisfying this condition is known as the prover generating a *valid equation*. In both applications of certifiable randomness and remote state preparation, each of these tests are performed with equal probability [5, 4]. In this project, we mainly measure success as the probability that the prover passes the equation check.

Thus, in summary, the overall objective for this project is to investigate cryptographic applications of NISQ devices, and specifically the feasibility of evaluating claw-free functions on these devices. A number of companies, such as Google, IBM and Rigetti, are at the forefront of developing NISQ devices and some, such as IBM and Rigetti, have made them available to users on the Internet through their cloud services [12, 13]. It is therefore not only important to have NISQ applications but also to have ways of verifying the results for these applications.

2 Methods

In this section, we describe and analyze the techniques used to implement the remote-state preparation protocol from [5]. We also discuss optimizations made to reduce the number of qubits required as well as the depth of the circuit.

We start by detailing the high level implementation of the evaluation of the NTCF function from Equation 1 in Section 2.1, which we then decompose into elementary operations in Section 2.2. Then, in Section 2.3, we analyze the performance of the implementation for small instances, quantified with the Hellinger overlap. In Section 2.4, we consider a different approach influenced by the problem Learning with Rounding (LWR) as well as Learning with Errors (LWE) and briefly discuss it's performance. Finally, optimizing this LWR approach, we analyze several techniques for reducing qubit usage and depth in Section 2.5.

2.1 Initial Implementation

Recall the function from Equation 1:

$$g(b, x) = Ax + b \cdot (As + e') + e \pmod q \tag{1}$$

where $b \in \{0, 1\}, A \in \mathbb{Z}_q^{m\times n}, x \in \mathbb{Z}_q^n, s \in \{0, 1\}^n, e' \in \mathbb{Z}_q^m$, and $e \in \mathbb{Z}^m$. The evaluation of this function in superposition involves only matrix-vector operations, which can be efficiently parallelized and performed in depth that is logarithmic with respect to the size of the input. In the original implementation, we used Microsoft's Q# quantum programming language, which has built-in operations for modular arithmetic operations.

For example, in the code block below `MultiplyAndAddByModularInteger` and `IncrementByModularInteger` are such operations included in Microsoft's Quantum Development Kit for performing modular arithmetic in quantum registers [14].

```
    // add quantum registers x and y into y, mod modulus
    operation addRegisters (x : Qubit[], y : Qubit[], c : Int, modulus : Int) : Unit is Adj + Ctl {
        MultiplyAndAddByModularInteger(c, modulus, LittleEndian(x), LittleEndian(y));
    }

    // add constant int to register res, mod modulus
    operation addConstantToRegister (c : Int, modulus : Int, res : Qubit[]) : Unit is Adj + Ctl {
        IncrementByModularInteger(c, modulus, LittleEndian(res));
    }
```

Listing 1: Original Implementation

`MultiplyAndAddByModularInteger` and `IncrementByModularInteger` implement the transformations

$$|x\rangle\,|y\rangle \to |x\rangle\,|(cx + y) \bmod N\rangle$$

$$|x\rangle \to |(x + c) \bmod N\rangle$$

respectively, for some constant c and modulus N. We have simply wrapped these functions in our own `addRegisters` and `addConstantToRegister` for convenience. These are very useful for implementing the evaluation of the function in Equation 1. Since we have $As + e'$ and A itself classically, if we have $|b\rangle\,|x\rangle\,|e\rangle$, then $Ax + e$ can be computed by `MultiplyAndAddByModularInteger` to compute the inner product between each row of A and x and add this result to the e register. Moreover, $b\cdot(As+e')$ can be added in by utilizing a controlled version of `addConstantToRegister`[1].

[1]For a unitary operation U, implemented as a function in Q#, one can create the controlled version of U using the keyword `Controlled`.

2.2 Decomposition of High Level Operations

Despite the convenience of utilizing Q#'s built-in functionality as seen in the previous section, these operations act as a black box, and the documentation for them is not explicitly available to analyze what the operations are actually executing in terms of elementary gates. However, because NISQ devices introduce noise for each quantum operation, we want our circuit to have optimal depth. By this we mean having an implementation that requires as few quantum operations as possible, which will result in less noise in the output of the protocol. Hence, we decomposed the high level operations into circuits composed of rotation gates and quantum fourier transforms (QFT), using the ideas and circuits presented in [15, 16, 17]. In [16], `MultiplyAndAddByModularInteger` and `IncrementByModularInteger` correspond to the controlled multiplier gate and modular adder gate, respectively.

An example of one such decomposed operation can be seen in the figure below. Note that the version in the code block implements the transformation

$$|x\rangle \rightarrow |(x+c) \bmod N\rangle$$

only for $N = 2^n$, where n is the size of the register we are adding the constant into. We present this version for simplicity, as the general modulus version is much more involved. In the literature, this operation corresponds to the circuit from Figure 3 in [16], thus decomposing the operation `IncrementByModularInteger` into these single qubit rotations using `R1Frac` as well as the QFT witb `QFTLE`.

```
    // maps |x> to |x + c (mod modulus)>
    // this version only works if modulus = 2^n, where n = Length(x) =
   Length(y)
    operation addConstantToRegister(c : Int, modulus : Int, res : Qubit[])
    : Unit is Adj + Ctl {
        body (...) {
            let n = Length(res);
            QFTLE(LittleEndian(res));
            for (i in 0 .. n - 1) {
                R1Frac(c, (n - 1) - i, (res)[i]);
            }
            Adjoint QFTLE(LittleEndian(res));
        }
        adjoint invert;
        controlled distribute;
        controlled adjoint distribute;
    }
```

Listing 2: Example Operation

Moreover, we also implemented the QFT (called `QFTNormal` in our implementation), as displayed below. This implementation is based off of that given in the Qiskit documentation [12] as well as the circuit known for the QFT [17]. Note that in the Qiskit documentation, the QFT was implemented recursively, but here, we transformed it into an iterative implementation in order to make it easier to transform it into its controlled version.

```
    operation QFTNormal(qs : LittleEndian) : Unit is Adj + Ctl {
        body (...) {
            let n = Length(qs!);
            for (i in 0 .. n - 1) {
                H((qs!)[n - i - 1]);
                if (i != n - 1) {
                    for (j in 0 .. n - i - 2) {
                        Controlled R1Frac([(qs!)[j]], (1, n - 1 - i - j, (
    qs!)[n - i - 1]));
                    }
                }
            }
            for (i in 0 .. (n / 2) - 1) {
                // SWAP((qs!)[i], (qs!)[n - 1 - i]);
                CNOT((qs!)[i], (qs!)[n - 1 - i]);
                CNOT((qs!)[n - 1 - i], (qs!)[i]);
                CNOT((qs!)[i], (qs!)[n - 1 - i]);
            }
        }
```

Listing 3: Quantum Fourier Transform Implementation

With each of these operations decomposed into rotations and controlled rotations, we could estimate the depth of the overall circuit for evaluating the function from Equation 1, which is roughly given by

$$\text{Depth} \approx 18mn + 2 \tag{3}$$

for a modulus $q = 4$ and where A is an $m \times n$ matrix. This is a very rough estimate, in that it assumes none of the rotations or controlled rotations can be parallelized. Moreover, the real depth also depends on the sparsity of the matrix A, so this expression for the depth can be thought of as an extreme upper bound on the true depth. Furthermore, this implementation utilizes

$$\#\ \text{Qubits} = 1 + m \log q + n \log q \tag{4}$$

where, again, q is the modulus and A is $m \times n$. For a concrete example, consider $q = 4, n = 3, m = 6$, which results in a depth of 326 and 19 qubits.

2.3 Analysis of Hellinger Overlap between Distributions

Recall from Section 1 that the function from Equation 1 is only approximately two-to-one. This can be seen by fixing b:

$$g_0(x) = Ax + e \pmod q, \text{ for } b = 0$$
$$g_1(x - s) = Ax + e' + e \pmod q, \text{ for } b = 1$$

Further recall that e' is likely to be a low-weight vector, so $Ax + e$ and $Ax + e' + e$ should be fairly close to equal. This can be seen in the figure below, where we interpret $g_0(x)$ and $g_1(x - s)$ as probability distributions, with variations from their respective centers Ax and $Ax + e'$ due to the error vector e.

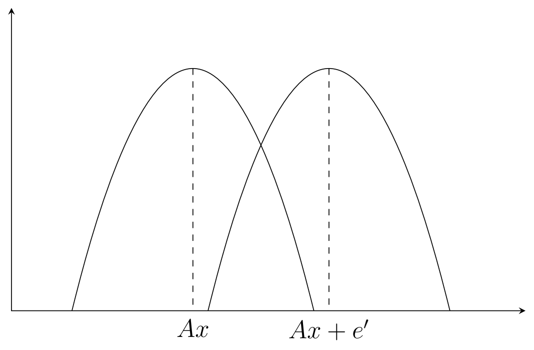


Figure 2: Overlapping Distributions

In order to optimize the performance of the circuit on both simulators and real quantum hardware, certain parameters of the function can be adjusted accordingly to allow for a higher amounts of overlap between the two distributions. Moreover, if there is greater overlap, we will also obtain a valid equation with higher probability, increasing the success rate of our implementation. Of these adaptable factors, the most effective is in calibrating the error distribution for e from Equation 1. We investigated the amount of overlap for several different distributions for e by using the measure known as the Hellinger overlap, which is defined as the following from [4]:

$$\text{Overlap} = 1 - H^2(D, D+e) = \sum_{e \in \mathbb{Z}_q^m} \sqrt{D(e)D(e+e')} \tag{5}$$

where for our case, D is the error distribution for e.

Also, recall when evaluating this function, we want to do so over all possible inputs, so e must be in a superposition with amplitudes that reflect the probability distribution it is chosen from. In [4, 7], they define this distribution as a truncated Gaussian distribution. However, to make a superposition over this distribution would add too much overhead to our circuit. To simplify, we tried choosing distributions that are easier to create "quantumly" but also allow for low-weight errors to be chosen with higher probability. Thus, we considered three different types of distributions: uniform, binomial, and multinomial.

Specifically, the uniform distribution is over all e vectors that have an L1 norm bounded by a certain chosen threshold. All other e vectors occur with probability 0. Then, for the binomial case, we consider each entry in the e vector independently. For each entry, we choose to have either a non-zero entry with probability p or 0 with probability $1-p$. Moreover, when choosing the non-zero entries, each is chosen uniformly at random from the possible $q-1$ options[2]. Finally, the multinomial case is a generalized case of the binomial distribution, where each nonzero entry of the e vector is now chosen from the possible $q-1$ options with different probabilities depending on its absolute value. Furthermore, we wanted a multinomial distribution where small errors were more likely than large errors to more closely reflect the truncated Gaussian.

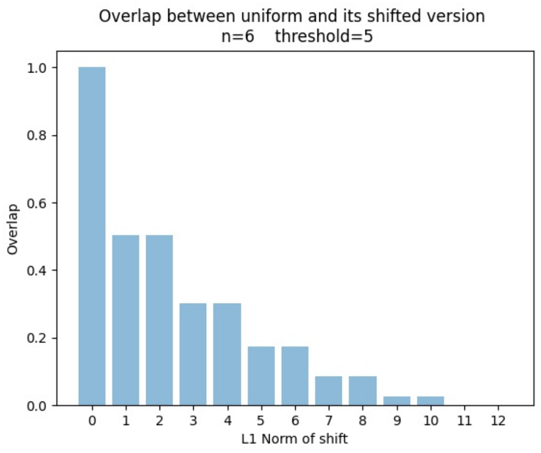
[2]Note that this is not technically a binomial distribution but is similar in the sense that we have 0 with probability $1-p$ and nonzero entries with probability p.

For each of these distributions, we calculated the Hellinger overlap numerically in Python and plotted the L1 norm of the shift vector e' against this overlap while varying the parameters of the distributions. Ideally, for low weight shifts e', we want very high overlap (close to 1), while for high weight shifts, we want overlap close to 0. Also, we performed these calculations for the case of $q = 4$, since this allows us to use larger LWE instances in the construction of the NTCF function.

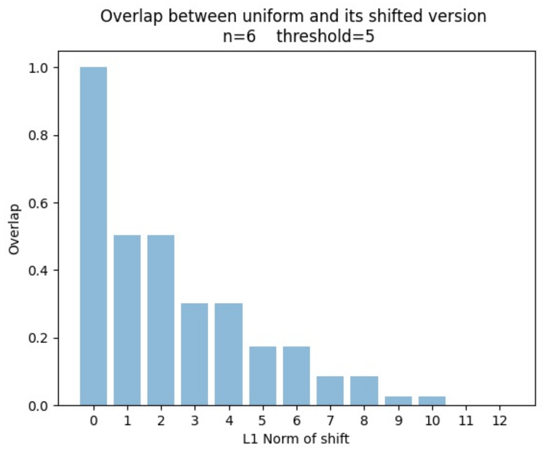
First, we considered the uniform distributions, which can be defined as

$$D_u(e, \text{threshold}) = \begin{cases} 0 & \text{if } ||e|| > \text{threshold} \\ \frac{1}{\text{number of possible errors s.t. } ||e|| \leq \text{threshold}} & \text{if } ||e|| \leq \text{threshold} \end{cases}$$

Out of different thresholds from 0 to 12, the results for thresholds of 5 and 6 seemed the most promising. However, even despite this, as seen in the plots below, the amount of overlap for shifts of norm 1 or 2 were not as high as was necessary. Ideally, these small shifts of 1 or 2 should have overlap significantly higher than 0.5.



(a) Threshold of 5

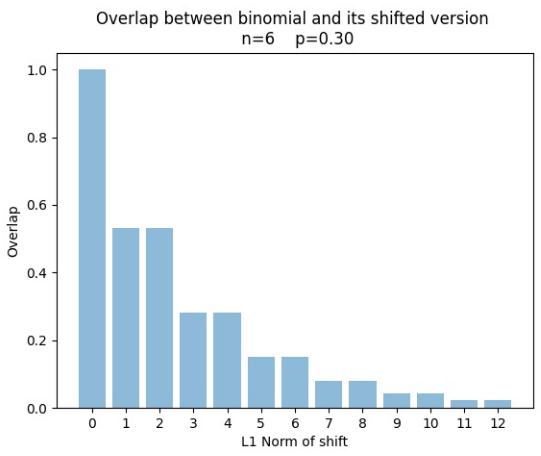


(b) Threshold of 6

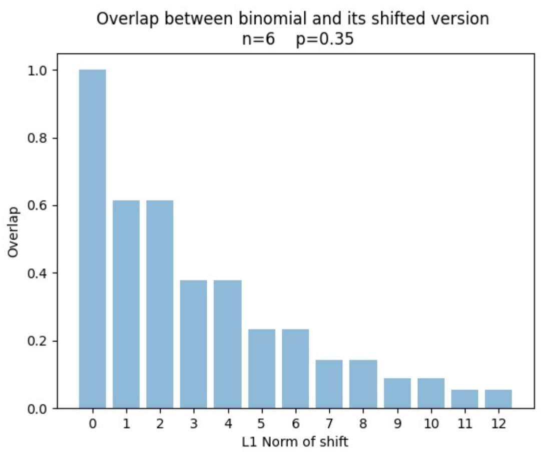
Figure 3: Uniform Distribution Plots

We next considered the binomial distribution[3]. Below are the plots for $p = 0.3$ and $p = 0.35$ for the binomial distribution:

[3]Again, strictly speaking this is not a binomial distribution, however it has similar properties in that each component of the e vector is non-zero with probability p.



(a) Probability $p = 0.3$



(b) Probability $p = 0.35$

Figure 4: Binomial Distribution Plots

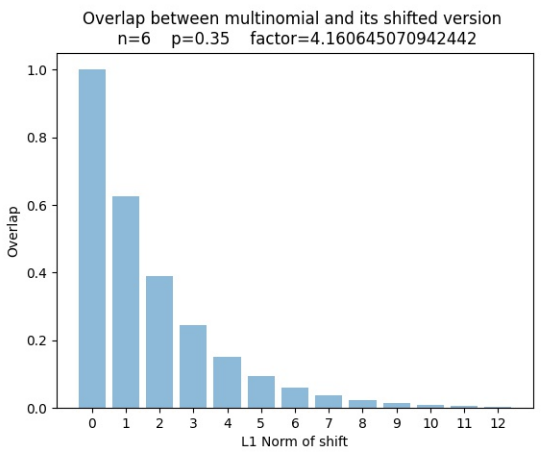
The results are better than the uniform case as the overlap is lower for higher shifts. Also, we can easily create a superposition mimicking this distribution using controlled-Y rotations. Each entry in the error state e can be expressed as the superposition below for the binomial distribution.

$$e_i = \sqrt{1-p}\,|00\rangle + \sqrt{\frac{p}{3}}\,|01\rangle + \sqrt{\frac{p}{3}}\,|01\rangle + \sqrt{\frac{p}{3}}\,|11\rangle$$

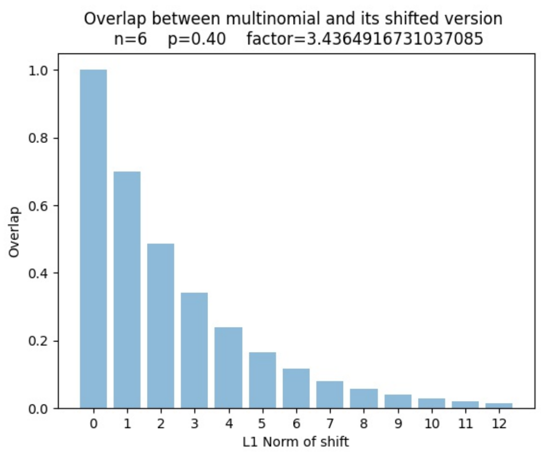
Finally, the multinomial distribution had the greatest potential, as its properties resembled the truncated Gaussian most closely. In particular, we considered multinomial distributions such that $+1$ and $+3$ errors had a factor f times higher probability of occurring than $+2$. Viewing $+3$ as the equivalent of -1 modulo 4, we can see that $+3$ is a low-weight error which we want with higher probability. Also, in order to make shifts of the same norm have equal overlap, we fixed this factor depending on p to be

$$f = \frac{(2-2p) \pm \sqrt{4-4p}}{2p}$$

Using this factor, we calculated the overlap between the original multinomial distribution and one shifted distribution, shifted by e', as seen in the plots below.



(a) Probability $p = 0.35$



(b) Probability $p = 0.4$

Figure 5: Multinomial Distribution Plots

Here, the multinomial distributions were the most promising, with overlap significantly less for higher norm shifts e' and also high overlap for low norm shifts. Furthermore, a superposition resembling this distribution can be created very similarly to the binomial but with an extra parameter of the factor f.

$$e_i = \sqrt{1-p}\,|00\rangle + \sqrt{\frac{fp}{2f+1}}\,|01\rangle + \sqrt{\frac{p}{2f+1}}\,|10\rangle + \sqrt{\frac{fp}{2f+1}}\,|11\rangle$$

We assessed the performance of each of these distributions by evaluating the NTCF function in superposition with each respective distribution for the e register. Following this evaluation, we checked how often we obtained a superposition over preimages in the first register, which is the expected outcome. In particular, we expected that there would be mainly two terms in superposition with amplitudes close to $1/\sqrt{2}$. Unfortunately, even with all three of these approaches, the fidelity of the superposition was quite low. We found this is largely due to the small instances we were considering in order to fit everything in around 15 qubits.

2.4 Construction of Functions Using Learning with Rounding

Given the results from the previous section, we turned to a different approach regarding claw-free functions based on rounding or truncation. Such functions are mentioned briefly in [4], defined as

$$g(b, x) = \lfloor Ax + b \cdot (As + e') \rceil \tag{6}$$

where $\lfloor \cdot \rceil$ is specified to be "a rounding function that truncates 'many' of the least significant bits of the operand" [4]. In other words, we only take the most significant bits of the number being rounded. Also, in Equation 6, all operations are done modulo q. We can view this function as utilizing rounding to create the properties we need, rather than adding in extra noise, as we did before by adding e.

Upon reading the rounding scheme presented in [18], we applied this to the $q = 4$ and $q = 8$ cases. In [18], the rounding scheme is such that the q elements in $\mathbb{Z}_q$, i.e. $\{0, 1, \ldots, q-2, q-1\}$, are separated into k separate "bins." Then, for a vector x, each of its components are then rounded to the index of the bin that the component is in. Also, as stated in the paper, if q, k are powers of two, this is just taking the $\log k$ most significant bits of each entry of x, just as described in the NTCF function from [4].

Furthermore, upon testing our implementation utilizing this rounding scheme for $k = 2$ (i.e. just taking the most significant bit)[4], we successfully obtained two preimages in superposition with 100% probability. Also, as a different test for success, we also obtained a valid equation 100% of the time. We discuss the results of this implementation further in Section 3. Moreover, this implementation utilizes

$$\#\text{ Qubits} = 1 + n \log q + m \log q + m \tag{7}$$

This is an increase by m qubits from the previous approach, where the number of qubits is given by Equation 4. However, there are several ways to decrease this number, as discussed in Section 2.5 below.

2.5 Approaches for Reducing Qubit Usage and Depth

In this section, we describe different methods for reducing qubit usage and depth. Here, we also discuss rough worst-case depth estimates to compare the strategies. However, the accurate depths are presented in Section 3, as computed by Cirq [19] for certain problem instances. Each approach has its own trade-offs in terms of success as well, which we also detail in Section 3.

2.5.1 Workspace Ancilla Register

One way to significantly reduce qubit usage is to utilize a workspace register. However, before detailing this approach, we will first review the unoptimized technique for comparison. Recall that the initial implementation for the rounding idea used $1 + n \log q + m \log q + m$ qubits. In order to evaluate the function from Equation 6, we have four quantum registers. The first two together are known as the *preimage register*, as they store the equal superposition of b, x, i.e. the inputs to the function. The third register then stores the result of evaluating the function prior to rounding, i.e. $Ax + b \cdot (As + e')$. Finally, the fourth register stores the rounded result, which is simply a vector whose entries are the most significant bit of each entry in the vector $Ax + b \cdot (As + e')$. These registers are summarized in the expression below.

$$|b\rangle\,|x\rangle\,|Ax + b \cdot (As + e')\rangle\,|\lfloor Ax + b \cdot (As + e')\rceil\rangle \tag{8}$$

Thus, because b is a single bit and x is an n-dimensional vector modulo q, they use one qubit and $n \log q$ qubits, respectively. Furthermore, $Ax + b \cdot (As + e')$ is an m-dimensional vector modulo q, hence corresponding to $m \log q$ qubits. Finally, the rounded result is a

[4]As a note, this $k = 2$ case is ideal, as the most significant bit can be stored in just one qubit, making our rounded register m qubits even without using the phase encoding.

binary vector since we just take the most significant bits, leading to m additional qubits, in total adding to the number from Equation 7.

In order to reduce qubit usage, consider the following idea. Each entry in the rounded result vector only depends on one entry from the vector $Ax + b \cdot (As + e')$. Hence, we do not need to compute $Ax + b \cdot (As + e')$ all at once, but could compute one entry, copy the most significant bit into the result register, then repeat for each entry. This is what we mean to utilize a workspace register, where the ancillas we use to calculate each entry of $Ax + b \cdot (As + e')$ one at a time act as a workspace for us to do intermediary computations. Furthermore, because we are computing one entry at a time, this utilizes only $\log q$ qubits as opposed to the previous $m \log q$, for a total of

$$\# \text{ Qubits} = 1 + n \log q + \log q + m$$

This is a significant decrease in qubit usage. As a concrete example, consider the case of $n = 2$, $m = 6$, $q = 8$. With the original implementation, we would use 31 qubits, while with this version we would only use 19. Unfortunately, reusing these ancilla qubits requires us to reset them after computing each entry of $Ax + b \cdot (As + e')$, which roughly doubles the depth. However, because we still obtain 100% success rate with this modification, we can afford an increase in error. Furthermore, we have found that the number of qubits is generally more restrictive than the depth, as a larger number of qubits can lead to more noisy operations as well.

Another minor adjustment we made in regards to this was to save one extra qubit from the result register. When calculating the last entry of $Ax + b \cdot (As + e')$, we can take the rounded result directly from the ancilla register, rather than copying it into a new qubit since we no longer need to reset the ancilla. Thus, our final qubit usage for this optimization is

$$\# \text{ Qubits} = n \log q + \log q + m \tag{8}$$

This small adjustment also reduces depth slightly, as we do not reset the ancilla. Furthermore, the success rate is maintained at 100%. A rough, worst-case depth estimate is given by

$$\text{Depth} \approx 3 + 2mn(\log q)^2 + 6m \log q - m \tag{9}$$

2.5.2 Phase Encoding Rounded Result

Another adjustment for reducing qubit usage can be used in addition to the one detailed previously. We refer to this technique as *phase encoding*. The approach is based off of *quantum random access codes* (QRAC), introduced in [20]. Here, the idea is to encode k classical bits into one qubit and by measuring in a certain basis, we can recover any of the k classical bits with high probability.

Similarly, in this phase encoding technique, we encode several of our rounded result bits into the phase of one qubit, which we refer to as a *phase qubit*. However, because these phase qubits are not orthogonal, we sacrifice success rate for using fewer qubits. Furthermore, the more phase qubits we have, the closer to orthogonal, and hence more accurate, the whole state will be. Thus, there is a balance between reducing the number of qubits and the success rate. We present the specific numerics of phase encoding and the success rate in Section 3, while here we discuss the circuits required to encode two or three bits into one phase qubit.

First, in order to encode two bits into one phase qubit, we use the following circuit. Note that $a_0, a_1 \in \{0, 1\}$ are the bits we wish to encode, where $|a_0\rangle, |a_1\rangle$ are the qubits that store the respective bits.

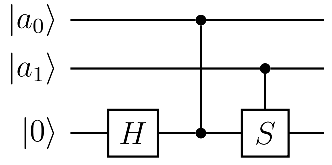


Figure 6: Circuit to Phase Encode 2 Bits

This circuit utilizes $\frac{\pi}{2}$ Z rotations in order to encode these bits into the phase qubit represented by the bottom wire. Furthermore, this can also be described by the following transformation:

$$\begin{aligned}
|00\rangle &\to |+\rangle \\
|01\rangle &\to |+_{\pi/2}\rangle \\
|10\rangle &\to |+_{\pi}\rangle \\
|11\rangle &\to |+_{3\pi/2}\rangle
\end{aligned}$$

where $|+_\theta\rangle = \frac{1}{\sqrt{2}}(|0\rangle + e^{i\theta}|1\rangle)$. Similarly, we can encode three bits into one phase qubit using the circuit

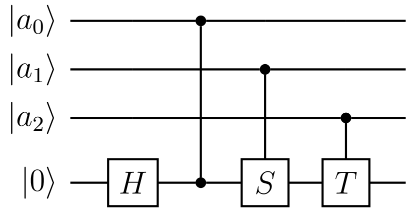


Figure 7: Circuit to Phase Encode 3 Bits

This circuit utlizes $\frac{\pi}{4}$ Z rotations rather than $\frac{\pi}{2}$ in order to encode more bits into a single phase qubit. Furthermore, this can be described by the transformation:

$$\begin{aligned}
|000\rangle &\to |+\rangle \\
|001\rangle &\to |+_{\pi/4}\rangle \\
&\vdots \\
|111\rangle &\to |+_{7\pi/4}\rangle
\end{aligned}$$

This strategy for reducing qubit usage does not affect the depth of the circuit, so this is still given by the expression from Equation 9. However, as discussed previously, it does affect the accuracy of our results. Moreover, the qubit usage varies depending on how many bits are encoded into one phase qubit. Let c be the number of bits encoded. Then, the number of qubits required is given by the following expression

$$\#\text{ Qubits} = 1 + n \log q + \log q + \left\lceil \frac{m}{c} \right\rceil \tag{10}$$

2.5.3 Direct Computation of Result in Phase Using QRAC

Finally, the last significant strategy is similar to the previous one, in that it is also based off of quantum random access codes (QRAC) [20]. Recall that the idea is to encode k classical bits into one qubit and by measuring in a certain basis, we can recover any of the k classical bits with high probability. Here, both of our approaches utilize QRAC to reduce qubit usage, but they differ in what they encode. In contrast to phase encoding, here we are computing result $Ax + b \cdot (As + e')$ directly in phase then performing a special measurement to retrieve the most significant bit. Recall that in phase encoding, we were computing $Ax + b \cdot (As + e')$ in the computational basis and then encoding the rounded results in phase.

To further explain the specifics of the idea, we have m phase qubits, similar to in phase encoding. However, rather than using the workspace register, we can directly compute each entry of $Ax + b \cdot (As + e')$ into the phases of each of the corresponding phase qubits. For instance, say a is the i-th row of A and $y = As + e'$, where y_i is the i-th entry of y. Then, after our computation, the phase qubit will be in the state

$$|+_\theta\rangle \,, \; \theta = \frac{\pi}{4} \cdot (\langle a, x \rangle + b \cdot y_i)$$

which can be done using controlled Z rotations. Moreover, because A, y both have entries modulo q, then there can only be q total possible $|+_\theta\rangle$ states. For this approach, we only considered $q = 8$ to optimize performance, so for this case, we have eight possible $|+_\theta\rangle$ states. We can visualize these states in the XY plane of the Bloch sphere, as seen in the figure below.

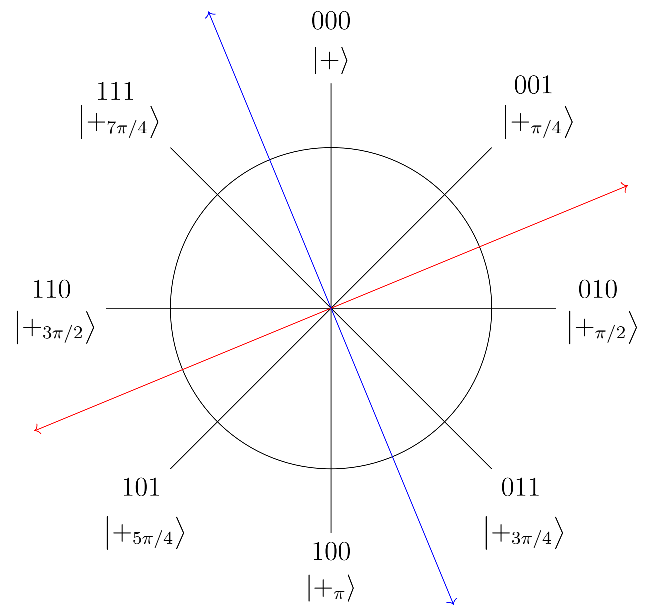


Figure 8: Visualization of States in XY Plane of Bloch Sphere

Here, the four states with most significant bit 0 are on the right side of the blue line, while the other four states with most significant bit 1 are on the left side of the blue line. Thus, if we perform a measurement along the red axis, we should get the correct most significant bit with high probability. This red axis corresponds to measuring in the basis $\{|+_{3\pi/8}\rangle, |-_{3\pi/8}\rangle\}$.

Now, we will analyze the probability with which we obtain the most significant bit. Assuming that we have one of the four states with most significant bit of 0, then we want to find the probability that we measure a 0. Without loss of generality, we can assume that the four states occur with equal probability. Then, notice from Figure 8 that $|+_{\pi/4}\rangle, |+_{\pi/2}\rangle$ form an angle of $\pi/8$ with the 0 outcome (red line). Similarly, $|+\rangle, |+_{3\pi/4}\rangle$ form an angle of $3\pi/8$ with the 0 outcome. Recall that the probability of an outcome is given by the cosine squared of half of the angle between the two states on the Bloch sphere. Thus, in our case, the probability of obtaining 0 if we have one of the states with most significant bit 0 is

$$\frac{1}{2}\left(\cos^2\left(\frac{3\pi}{16}\right)+\cos^2\left(\frac{\pi}{16}\right)\right)\approx 0.82 \tag{11}$$

Furthermore, it can be shown that this is approximately the optimal success probability with which we can encode and recover $k=3$ bits. In order to see this, consider Nayak's bound presented in [20, 21] which states that if a QRAC encodes m classical bits into n qubits such that we can recover each of the m classical bits with probability p, then it must be that

$$n \geq (1-H(p))m$$

where $H(p)$ is the binary entropy function

$$H(p) = -p\log_2(p) - (1-p)\log_2(1-p)$$

In our case, we want to encode 3 classical bits into 1 qubit. As seen in Equation 11, we can do so with probability 0.82. Thus, calculating the binary entropy function, we have

$$H(0.82) \approx 0.68$$

Furthermore, using Nayak's bound, we know that

$$\frac{1}{3} \geq 1 - H(p) \approx 0.32$$

Hence, we can see that the probability with which we recover the most significant bit is close to optimal by Nayak's bound [20, 21].

Overall, this approach is a large deviation from the original implementation, but with a significant reduction in depth as well as a small decrease in qubit usage. Because we are computing $Ax + b \cdot (As + e')$ in phase and measuring to obtain the rounded result, we no longer need the ancilla register. Thus, our qubit usage decreases by $\log q$ for a total of

$$\#\ \text{Qubits} = 1 + n \log q + m \tag{12}$$

Furthermore, the circuit for performing the computation in phase is much simpler than in the previous approaches. As a rough, worst-case depth estimate, we have

$$\text{Depth} = 4 + mn \log q + m \tag{13}$$

However, in exchange for these improvements, the success rate also decreases significantly. Despite the most significant bit being recovered with high probability, even if one is wrong, this will cause the entire equation to be invalid during our tests.

3 Results

In the course of optimizing these circuits, we also discovered an interesting phenomena regarding the hardness of LWE. While testing the LWR based implementation detailed in Section 2.4, we found that for the case of $q = 4$ the circuit for the evaluation of the trapdoor claw-free function is entirely Clifford. Furthermore, it is known by the Gottesman-Knill Theorem [22] that Clifford circuits can be efficiently simulated classically. However, the NTCF functions we are considering are hard to break under the LWE assumption. In other words, it should be hard to efficiently evaluate the functions from Equations 1, 6 in superposition without utilizing a quantum computer, given that LWE is hard[5]. Thus, because this evaluation is easy to simulate classically for the case of $q = 4$, this implies that LWE is easy in this case, which was previously unknown.

Following this discovery, we shifted focus to the case of $q = 8$, which is the next smallest power of 2[6]. For this case, we tested all of the different optimizations presented in Section 2.5

[5]This is an assumption because LWE has not yet been proven to be hard (classically or quantumly). However, we have good reason to believe that this is the case, as the best known (classical and quantum) algorithms for LWE run in exponential time. Furthermore there are reductios from worst-case lattice problems to LWE. This topic is further discussed in [9, 10]

[6]We are only choosing moduli that are powers of 2 because it greatly simplifies the circuits required for modular arithmetic. For more information on general moduli arithmetic in quantum registers, see [16].

on noiseless simulators and have recorded their various success rates and depths in Tables 1, 2. Moreover, we have tested on the IBM online devices [12], but without significant results due to the limited connectivity of their devices[7]. Table 1 also records the number of qubits required when using each optimization. This is omitted in Table 2 because, as previously discussed, the depth for the various phase encoding schemes is the same while differing in qubit usage and success rate, which can be viewed in Table 1.

Optimization	n	m	# Qubits	Success Rate
None	3	12	24	100%
None	3	10	22	100%
None	2	6	15	100%
Phase Encoding $2 \to 1$	3	12	19	92.02%
Phase Encoding $2 \to 1$	3	10	18	85.4%
Phase Encoding $3 \to 1$	3	12	17	82.7%
Phase Encoding $3 \to 1$	3	10	17	81.8%
QRAC	2	6	13	69.8%

Table 1: Success Rates for Various Optimizations

The optimization listed is in reference to Section 2.5. "None" means that we have not adjusted the qubit usage other than the use of the workspace register described in Section 2.5.1. Note that QRAC indicates the optimization from Section 2.5.3. Also, success rate is the probability with which we obtain a valid equation. For phase encoded versions, the success probability is calculated by computing the total variation from the original version. Furthermore, all data are reported for the case of $q = 8$ and utilizing the Qiskit simulators [12].

Optimization	n	m	Depth
None	3	12	274
None	3	10	222
None	2	6	93
QRAC	2	6	14

Table 2: Depth of Circuit for Different Instances

The optimization listed is in reference to Section 2.5. "None" means that we have not adjusted the depth other than the use of the workspace register described in Section 2.5.1. Note that QRAC indicates the optimization from Section 2.5.3. Also, the depth of no optimization is the same as for phase encoded versions, so we have omitted the latter to avoid redundancy. Furthermore, all data are reported for the case of $q = 8$ and utilizing the Qiskit simulators [12].

The results running on real quantum hardware are still to be determined. However, upon running simulations with noise, the circuits in terms of native gates as well as the original both generate valid equations approximately 55% of the time. This serves as a rough estimate for the success of the experiment, which we hope to run soon.

[7]This connectivity prevents controlled operations between any two qubits. Thus, when compiling a circuit to run on their devices, this depth is much higher than that of the original circuit, as we must perform SWAP gates when performing controlled operations between two qubits that are not neighbors.

Acknowledgements. The authors thank Norman Yao, Greg Meyers, and Soonwon Choi for useful discussions. We also thank Chris Monroe and Daiwei Zhu for their help in running the experiment. Laura Lewis is supported by the Michael Mathes Endowed Scholarship Fund, Robert and Amelia Peeler Scholarship Fund, the Alcorn Scholarship, and Caltech Summer Undergraduate Research Fellowship (SURF) Program. Alexandru Gheorghiu and Thomas Vidick are supported by MURI Grant FA9550-18-1-0161 and the IQIM, an NSF Physics Frontiers Center (NSF Grant PHY-1125565) with support of the Gordon and Betty Moore Foundation (GBMF-12500028). Thomas Vidick is also supported by NSF CAREER Grant CCF-1553477, AFOSR YIP award number FA9550-16-1-0495, and a CIFAR Azrieli Global Scholar award.

References

[1] John Preskill. Quantum computing in the nisq era and beyond. *Quantum*, 2:79, 2018.

[2] Austin G Fowler, Matteo Mariantoni, John M Martinis, and Andrew N Cleland. Surface codes: Towards practical large-scale quantum computation. *Physical Review A*, 86(3):032324, 2012.

[3] Frank Arute et al. Quantum supremacy using a programmable superconducting processor. *Nature*, 574(7779):505–510, 2019.

[4] Zvika Brakerski, Paul Christiano, Urmila Mahadev, Umesh Vazirani, and Thomas Vidick. A cryptographic test of quantumness and certifiable randomness from a single quantum device. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 320–331. IEEE, 2018.

[5] Alexandru Gheorghiu and Thomas Vidick. Computationally-secure and composable remote state preparation. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 1024–1033. IEEE, 2019.

[6] Tony Metger and Thomas Vidick. Self-testing of a single quantum device under computational assumptions. *arXiv preprint arXiv:2001.09161*, 2020.

[7] Urmila Mahadev. Classical verification of quantum computations. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 259–267. IEEE, 2018.

[8] Alexandru Gheorghiu and Matty J Hoban. Estimating the entropy of shallow circuit outputs is hard. *arXiv preprint arXiv:2002.12814*, 2020.

[9] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[10] Oded Regev. The learning with errors problem. *Invited survey in CCC*, 7:30, 2010.

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

[12] IBM Contributors. IBM Quantum Experience. https://www.ibm.com/quantum-computing/technology/experience/. Online; accessed 25-July-2020.

[13] Rigetti Computing Revision. Welcome to the Docs for the Forest SDK. http://docs.rigetti.com/en/stable/. Online; accessed 25-July-2020.

[14] Microsoft Contributors. Microsoft Quantum Documentation. https://docs.microsoft.com/en-us/quantum/?view=qsharp-preview. Online; accessed 25-July-2020.

[15] Archimedes Pavlidis and Dimitris Gizopoulos. Fast quantum modular exponentiation architecture for shor's factorization algorithm. *arXiv preprint arXiv:1207.0511*, 2012.

[16] Stephane Beauregard. Circuit for shor's algorithm using 2n+ 3 qubits. *arXiv preprint quant-ph/0205095*, 2002.

[17] Thomas G Draper. Addition on a quantum computer. *arXiv preprint quant-ph/0008033*, 2000.

[18] Joël Alwen, Stephan Krenn, Krzysztof Pietrzak, and Daniel Wichs. Learning with rounding, revisited. In *Annual Cryptology Conference*, pages 57–74. Springer, 2013.

[19] Cirq Developers Revision. Cirq. https://cirq.readthedocs.io/en/stable/#. Online; accesed 25-July-2020.

[20] Andris Ambainis, Ashwin Nayak, Ammon Ta-Shma, and Umesh Vazirani. Dense quantum coding and a lower bound for 1-way quantum automata. In *Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 376–383, 1999.

[21] Ashwin Nayak. Optimal lower bounds for quantum automata and random access codes. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 369–376. IEEE, 1999.

[22] Daniel Gottesman. The heisenberg representation of quantum computers. *arXiv preprint quant-ph/9807006*, 1998.