Quantum Secret Sharing

Franklin Jia, Marc Robert Wong, Ruchir Rastogi

June 8, 2019

1 Introduction

Consider a scenario in which Alice would like to share a secret with Bob and Charlie, such that neither individually could reconstruct the secret nor gain any information about the secret but together they could. This problem, referred to as the secret sharing problem, has been well studied in the classical setting where the secret s is a string in $\{0,1\}^n$. If the secret must only be shared between two individuals, there is a very simple algorithm that can be employed. Alice sends a random string $r \in \{0,1\}^n$ to Bob and $s \oplus r$ to Charlie. Neither Bob nor Charlie can gain any information about s individually, but if they XOR their strings together, they can recover the secret:

$$r \oplus (s \oplus r) = s.$$

More complex schemes can be used to split a secret into n shares where at least k shares are required to recover the secret. Shamir's Secret Sharing algorithm[1], for instance, encodes a one bit secret as a degree $k-1$ polynomial where each of the n shares is an evaluation of the polynomial. If at least k shares are known, polynomial interpolation can then be used to recover the secret.

In this paper, we focus on an analogous problem in the quantum realm. In particular, we examine a quantum secret sharing protocol described by Hillery et al.[2], which we refer to as the HBB protocol, that splits one secret qubit into two. The protocol has the additional advantage that if an eavesdropper tries to interfere (for example by entangling an ancilla qubit with the qubits used in the protocol), the eavesdropper will either gain no information or the tampering will be detected by the honest parties.

In the remainder of this paper, we will describe the HBB protocol and its implementation. Afterwards, we will detail results of running the protocol on both a QVM with noise and the Rigetti QPU. Finally, we will go over some possible next steps.

2 HBB protocol

Let's consider the case where Alice wants to share a quantum secret A with Bob and Charlie.

(1) Alice generates a GHZ triple a, b, c such that only Alice has access to a, only Bob has access to b, and only Charlie has access to c.

[1] Boneh, Shoup. "A Graduate Course in Applied Cryptography" (Chapter 11.6.1)

[2] Hillery et al. "Quantum Secret Sharing". `https://arxiv.org/pdf/quant-ph/9806063.pdf`

(2) Alice measures both A and a in the Bell basis. Let the resulting values be v_1 and v_2.

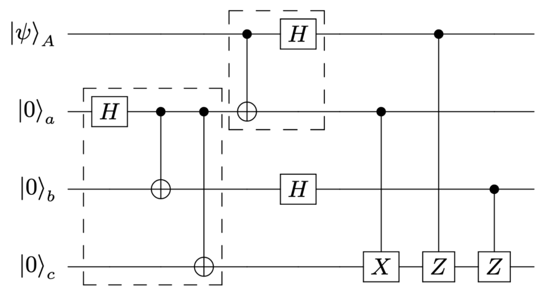
(3) Bob or Charlie are randomly chosen to measure their GHZ qubit (i.e. either a or b, respectively) in the X basis. Let's assume for the sake of clarity that Bob is chosen, and let the resulting value be v_3.

(4) Alice sends v_1 and v_2 to Charlie, and Bob sends v_3 to Charlie.

(5) Finally, Charlie can transform his GHZ qubit c into A up to an overall sign by applying a set of up to two Pauli gates, which depend on the values of v_1, v_2, and v_3, to c. The mapping from (v_1, v_2, v_3) to the corresponding Pauli gates required to transform c into A up to an overall sign are described in equation (19) of the HBB paper.

In practice, we implemented the HBB protocol using PyQuil. Our first implementation behaved exactly as described above by using the `MEASURE` instruction and classical control flow to correctly transform Charlie's GHZ qubit c into Alice's secret A. This implementation can be found in `hbb.py`. While this implementation performed correctly on the QVM simulation, we were unable to run it on the QPU due to some errors which we believe arose because we had PyQuil instructions in our program after we used the `MEASURE` instruction. In order to resolve this situation, we decided to mirror the approach of Joy et al.[3], who used controlled gates with control qubits a, b, c to emulate the measurement and classical control flow section of our original implementation. This quantum circuit is visualized in Figure 2 of the Joy et al. paper, and we have included it below for convenience.

Figure 1: Quantum circuit (source: Joy et al. Figure 2)



Our PyQuil implementation can be found in `hbb_qcs.py`. We also implemented a version of the protocol that works with N-qubit secrets in `hbb_n.py`.

3 Results and discussion

We first implemented the above protocol and ran it on the Aspen-4-4Q-A lattice of the Rigetti QPU. The following are some of the relevant noise characteristics for that lattice: T1 = 27.07 μs, T2 = 21.43 μs, fRO (readout fidelity) = 94.02%.

Table 1 depicts the ability of Charlie to reconstruct the secret sent by Alice for a variety of secrets:

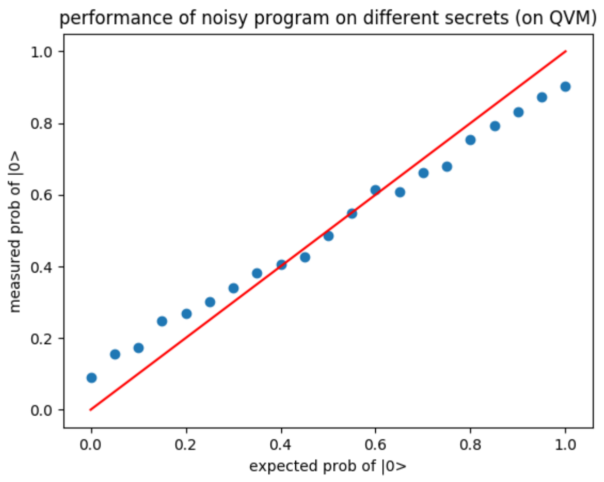
[3] Joy et al. "In principle demonstration of quantum secret sharing in the IBM quantum computer." `https://arxiv.org/pdf/1807.03219.pdf`

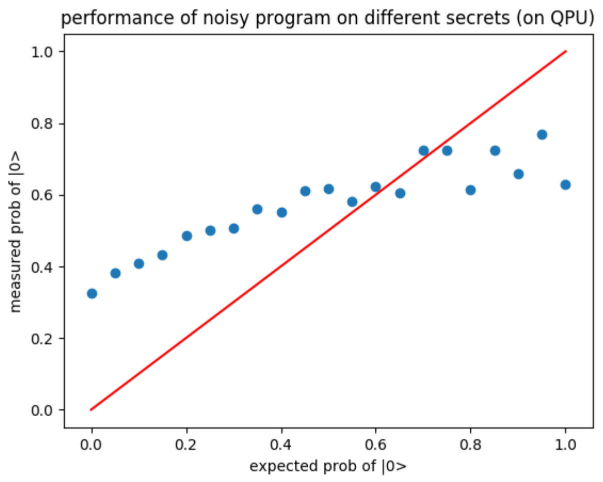
secret construction	expected $P(\|0\rangle)$	$P(\|0\rangle)$ of constructed secret	$P(\|0\rangle)$ of reconstructed secret
X	0.00	0.08	0.15
H	0.50	0.51	0.56
HTH	0.85	0.83	0.70

Table 1: Performance of HBB protocol on QPU (1000 trials)

Measuring the probability of landing in a $|0\rangle$ state helps us characterize how close the reconstructed secret is to the intended secret. The above results show that the the QPU introduces a significant amount of noise into the reconstructed secret, thereby preventing this protocol from being practical in the short term for sharing *quantum* secrets. The protocol could be used to share *classical* secrets (with the added benefit of detection of eavesdropping), but cryptographic schemes that encrypt classical shares would work much better.

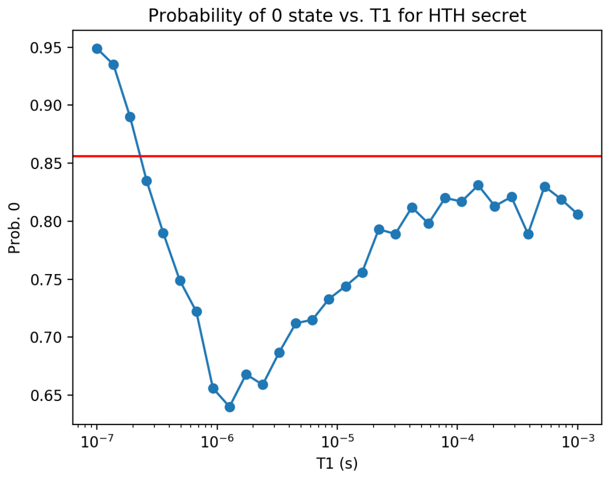
Since the protocol did not successfully run on the QPU, we analyzed its performance on the QVM. First, we compared the performance of the protocol on the QPU with its performance on a QVM simulated to have the same T1, T2, and fRO values. The following graphs compare the noisy QVM with the QPU:

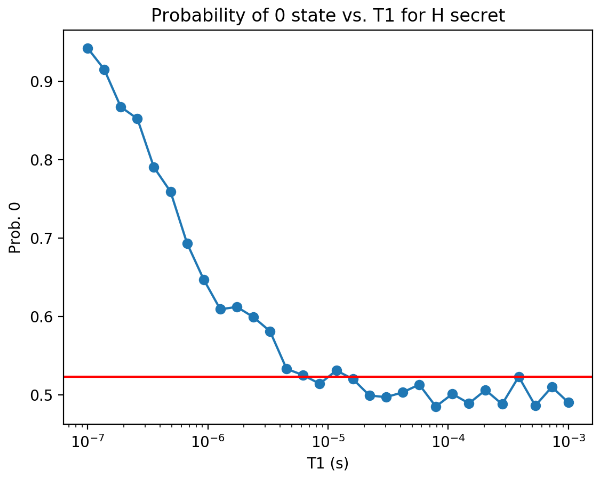




The above graphs illustrate that the errors introduced by the QPU are not just due to decoherence noise and readout errors, as the QVM is simulated with the exact same error values as those of the QPU. This happens because decoherence noise and readout errors are not enough to model the reality of noise on the QPU which can be much more complex.

We also measured the performance of the HBB protocol across a range of noisy QVM simulations by varying the T1/T2 values. This would allow us to estimate what the performance of the protocol would look like if we were able to run the HBB protocol on a less noisy QPU.





On the graphs below, note that the red line is the expected probability of measuring the 0 state. We also made the assumption that $T2 = T1/1.5$ which is roughly consistent with the T1 and T2 values for the QPUs available on Rigetti's QCS. We also used a global readout fidelity of 0.95 which is once again similar to the readout fidelity of Rigetti QPUs.

As expected, the below graphs clearly show that a QVM with less decoherence noise has a more accurate secret reconstruction as the measured probability approaches the theoretical probability. However, it seems that in order to reach the theoretical probability we would need QPUs that have much higher T1/T2 values than are currently available. Note that in the first graph, although the algorithm seems initially accurate because it has probability of 0 close to the expected 0.85, the second graph shows that this is simply due to all measurements trending towards the 0 state with very low T1 values. This seems reasonable based on the definition of T1 coherence. We can also see that even with a noisy QVM—with T1/T2 values that are orders of magnitude higher than the current QPUs—the performance is still not exactly what we expect. This gap is due to the readout fidelity which remained 0.95 for all the noisy simulations; with a readout fidelity of 1.0 the noisy simulations were able to reach the theoretical probabilities.

4 Next steps

4.1 State tomography

In order to obtain a more precise measurement of the performance of the HBB protocol on a QPU and on noisy simulations, we tried to use state tomography. State tomography would allow us to determine the fidelity, a metric which approximates the distance between the theoretical density matrix we expect and the experimental density matrix we measured. In particular, we could calculate the fidelity between Alice's secret A and Charlie's recovered value after transforming c. This would give us a more precise measurement than simply comparing the probability of A and c being 0.

Unfortunately, we ran into several issues when trying to implement state tomography. The state tomography library in Grove looked very promising but unfortunately to our knowledge has not been updated to work with PyQuil 2. In order to resolve this issue, we attempted to update the Grove library manually as well as looking into changing our implementation to work with PyQuil

1.9, which seemed non-trivial as there have been significant changes. Next, we tried to use the Forest benchmarking library. This was also unsuccessful due to lack of documentation of the Forest benchmarking library as well as the fact that the Forest library is much more low-level than the Grove tomography library. Finally, we also attempted to write our own custom tomography function; this also proved to be unsuccessful.

4.2 Multiparty HBB

An extension of the HBB protocol that we considered implementing was to allow for the secret to be shared between more than 2 parties. The HBB paper describes a version of the secret sharing scheme for sharing between 3 parties and another paper by Xiao et al.[4] describes a more general extension of HBB to share quantum information between N parties.

4.3 Threshold QSS schemes

The Cleve et al. paper[5] describes another type of quantum secret sharing that is known as a threshold secret sharing scheme. A (k, n) quantum secret sharing schemes splits quantum information into n shares such that at least k of the n shares are required to recover the secret. Much like the classical Shamir Secret Sharing scheme, the Cleve et al. paper describes a quantum secret sharing scheme that relies on quantum polynomial codes to implement a (k, n) threshold scheme for $n < 2k$.

4.4 Communication efficient quantum secret sharing schemes

Typically in quantum secret sharing, we're interested in minimal authorized sets for reconstruction of the secret (where every proper subset of an authorized set is unable to recover the secret). However, if we allow for non-minimal authorized sets, then we can trade off the size of the authorized sets with the amount of communication required for reconstruction, reducing communication overheads by a factor of $O(k)$.[6]

5 Miscellaneous

Code repository
Our code is available on GitHub here: `https://github.com/rastogiruchir/CS269Q-Final-Project`. We are comfortable with the code and paper being posted on the course website.

Late days
We pooled late days from Franklin (1 late day) and Marc (3 late days) in order to obtain a 1 day extension. In addition, we requested and received an addition 1 day extension from Will because we could not reserve QCS lattice instances which we needed to make some final measurements.

[4] Xiao et al. "Efficient multiparty quantum-secret-sharing schemes". `https://journals.aps.org/pra/pdf/10.1103/PhysRevA.69.052307`

[5] Cleve et al. "How to share a quantum secret." `https://arxiv.org/pdf/quant-ph/9901025.pdf`

[6] Senthoor et al. "Communication Efficient Quantum Secret Sharing" `https://arxiv.org/pdf/1801.09500.pdf`