Trieste Hackathon: Quantum Cryptography

Introduction

Quantum theory is fundamentally probabilistic. That is, we use it to give probabilities for measurement outcomes in experiments. This also means that in each measurement of an experiment we cannot always perfectly predict what will happen. This unpredictability does not seem to come from a lack of knowledge, such as in statistical physics, but from the properties of the laws of physics themselves.

Unpredictability, also called randomness, is a necessary resource in cryptography. In particular, randomness is needed for cryptographic key generation, encryption and authentication. If the key is private then it must be unpredictable to adversaries, so that to decrypt a message you need to try all possible values of the key, which could take forever. Because of the inherent unpredictability of quantum devices, they are an excellent platform for providing cryptographic-quality randomness, especially current and near-term devices.

One difficulty with randomness is that it is notoriously hard to measure; this is made more difficult when your device deviates from the theory due to errors and noise in your system. In this way, we might not be able to distinguish between true quantum randomness and classical signals, which could be due to an unknown yet predictable process. We could use error correction but that becomes costly and lessens the appeal of using current devices. So we need methods to estimate genuine quantum randomness.

In this use case we will consider the randomness that comes from quantum circuits. In this setting there is a toy error model that affects the measurements and the resulting measurement statistics. In this setting you will first come up with circuits to test and use methods to measure the randomness from circuits. In the second part, we will look at techniques to turn the raw randomness from quantum circuits into something cryptographically useful, using **randomness extractors**. In particular, you will use cryptomite, a package developed by the Quantum Cryptographic team at Quantinuum.

This use case is more of a very short research project to get you to think about the issues associated with cryptographic randomness from quantum systems.

Quantifying Randomness

For cryptographic applications, the accepted way to quantify randomness is through the **min-entropy**. We can imagine having a box that produces a fixed number of bits when we request randomness from it. This is what happens when we call `getrandbits()` in the built-in Random python module, where the argument of the function determines the length of the bit-string. In general, for a probabilistic process generating bit-strings $x \in \{0,1\}^n$ of length n. Since it is a probabilistic process, there is a probability distribution denoted $p(x)$ where $\sum_x p(x) = 1$. You might have encountered the Shannon entropy, which is defined as

$$H(p) = -\sum_x p(x)\log_2 p(x).$$

This gives a single value that is n if the probability distribution is the most random, i.e. $p(x) = 2^{-n}$ for all x, and 0 when $p(x)$ is deterministic, i.e. $p(x) = 1$ for a particular x. So this might seem a good quantity to use for randomness generation.

However, the Shannon entropy can overestimate the amount of randomness in a process. Let's look at an example, which we will call **Example 1**. Imagine your random process is the following: with probability $(1-\epsilon)$ the process outputs the all-zeroes string, and with probability ϵ it outputs a random string. So the distribution is $p(x) = (1-\epsilon) + \epsilon 2^{-n}$ if x is the all zeroes string and $\epsilon 2^{-n}$ otherwise. You can use the fact that Shannon entropy is *concave* to prove that $H(p)$ of this process is at least ϵn (verify this to yourself). Therefore, for small values of ϵ the Shannon entropy is telling you the amount of randomness grows with the output. However, this process is not suitable for cryptographic applications. Can you work out why?

The min-entropy has a simple definition, which is

$$H_{min}(p) = -\log_2 \max p(x),$$

where the minimisation is taken over all bit-strings x. In words, we first need to find the most probable bit-string x, then take the logarithm of its probability. Intuitively, the larger the probability the smaller the min-entropy. We also see that it gives the same values as the Shannon entropy for maximally random processes where $p(x) = 2^{-n}$ for all x and deterministic processes. The reason the min-entropy is useful cryptographically is that it captures how predictable an output bit-string is. It is asking how predictable the most predictable output is.

We can demonstrate this by looking at the process in Example 1. Recall that the Shannon entropy was lower-bounded by ϵn. Now convince yourself that the min-entropy of this process is *at most* $-\log_2(1-\epsilon)$, which is independent of n and tends to 0 when ϵ gets small. Therefore the min-entropy tells us that the string of all-zeroes is really predictable and so it would not be wise to use this in a cryptographic application.

Quantum systems are really good for calculating the min-entropy since if we know the quantum states and measurements we can compute it exactly. That is, the basic physics lets us calculate the min-entropy. For classical systems, things become more difficult: we need to know how much we don't know! That is, we need to calculate our ignorance of what a process is. For something like an ideal coin, that seems easy. But computers don't use coins and things become messy. All this being said, things can get messy with quantum systems: our description of the quantum state and measurements can become inaccurate due to noise. Amazingly, even with imperfect systems you can estimate the randomness, in something called the *device-independent* approach; this is the basis of the Quantinuum product, Quantum Origin. In the following exercises you will be considering a toy example of how to estimate randomness from quantum systems.

Exercises

Part 1

Task 1: a min-entropy warm-up

Here we will first consider an idealised set-up where you have quantum circuits, and your task is to calculate the probability distribution over outputs and the min-entropy. Write functions that take a representation of a quantum circuit and then finally give estimates of min-entropy.

Think about the simplest circuits that give you the most min-entropy and use these to test your functions.

Task 2: noisy circuits

Previously you were implementing something like this and calculating the randomness.

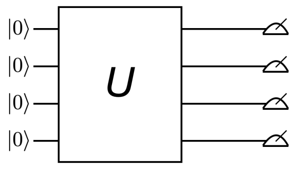

Figure 1: A general quantum circuit

This assumes that you perfectly create the initial quantum states, the unitaries and the measurements. However, this does not happen in practice. All elements of the circuit could deviate from the theory, and we might not know. In this use case we will simplify things and assume that our measurements might have some noise on them. This is motivated by the fact that measurements involve interacting a quantum system with a large environment, so there is ample possibility for things to go wrong. Furthermore, for near-term uses of quantum computers, you will submit your circuits to distant labs and so might not have direct access to what's going on in the labs.

To summarise, when your circuit is actually implemented, the following will happen:

There is some parameter $\lambda \in \{0, 1\}$, which determines if there is a unitary error $\mathcal{E}_i$ applied to each qubit before it's measured in the computational basis. The unitary could be an arbitrary one, about which we do

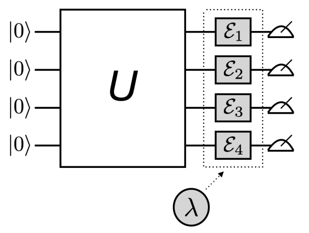

Figure 2: A quantum circuit with errors happening on the measurement

not know anything.

We have provided some code which will simulate these noisy implementations of quantum circuits. You will provide your quantum circuit in a particular format as an argument for a function. The function will then return a sample, or shot, from the circuit.

Your first part of this task is to write a function that will return N shots of the circuit. The second part is then to write functions that will estimate the distribution over n bits (where n is the number of qubits in your circuit) from these shots and min-entropy from this process.

Discussion: think about the limitations of this approach. In particular, what if someone knows the value of λ, could they have more predictive power? This could over-estimate the entropy.

Task 3: conditional predictive power

Have a look at the function simulating the noisy quantum circuit, and use this to estimate the *worst-case conditional min-entropy*. This is the min-entropy when you condition on knowing the value of λ, where you now maximise over all values of lambda. So you will have the conditional probability distribution $p(x|\lambda)$, then the worst-case conditional min-entropy is

$$-\log_2 \max p(x|\lambda),$$

where now we maximise over λ as well as x. In words, we want to find the worst-case scenario λ where the outcome x is the most predictable. Note that we assume that all values of λ are possible.

Using your knowledge of the simulation function, write a new function that will calculate the conditional worst-case min-entropy for a quantum circuit. Test this on different circuits to see how entropy can change due to the error model.

Discussion: at this point because the errors happen locally before the measurement this should give you an idea of what circuits are useful for randomness generation. In Quantum Origin, we allow for more general errors and deviations from target quantum circuits and estimate the conditional min-entropy. It is unfortunately beyond the scope here.

Part 2

There are many examples where you will have a process where it will generate n bits, but the min-entropy is less than n. Loosely speaking, the min-entropy will be diluted. For cryptographic purposes you want to have a bit-string that is as close to maximally random as possible. So we need to take the diluted randomness and "compress" it in a short bit-string that is maximally random. For example, imagine we have a string of length n that has min-entropy $n/2$, we want all the randomness to be in a bit-string of length $n/2$.

The process that takes a string of diluted randomness and makes a shorter string of maximal randomness is called a **randomness extractor**. It is also part of the Quantum Origin product. We have developed an open source version of the randomness extractors into a packaged called cryptomite. In this part of the use case you will become familiar with this package and use it to turn randomness from quantum circuits into maximal randomness, provided there is enough min-entropy.