

Lecture 1: Class Introduction

Instructor: Fermi Ma

September 14, 2026

1 Overview of quantum cryptography

This is a graduate seminar on quantum cryptography. Roughly speaking, we'll define quantum cryptography as encompassing any topic that arises at the intersection of cryptography and quantum computing. But why should you care about quantum cryptography?

If you're a cryptographer, you should care because all of modern cryptography is built on the notion of efficient computation. Cryptographic protocols are all about exploiting a computational asymmetry: an honest party should be able to run the protocol on a polynomial-time computer, but it should be secure against *any* polynomial-time attacker. Quantum computers are an *extension of classical computers*, i.e., they can do everything classical computers can do, but potentially more. And this additional capability gives more power to both **honest parties** and **attackers**. To elaborate:

- Quantum computers enable new attacks on existing cryptosystems. The most famous example of this is Shor's algorithm, which breaks RSA and many other number-theory-based public-key cryptosystems. We'll start covering Shor's algorithm later today.
- Quantum computers also give more power to honest parties, in the sense that honest parties that run cryptographic protocols on quantum computers can achieve capabilities and security guarantees that you can't achieve with classical protocols. The most famous example of this is BB84 Quantum Key Distribution, which is a protocol for two parties with quantum computers to agree on a random shared key with perfect security (meaning even an unbounded time eavesdropper cannot learn the key). This is provably impossible with a classical protocol. BB84 is just one of many examples of interesting new protocols that arise when honest parties use quantum capabilities.

If you're coming from a quantum background, my pitch is this: cryptography has been a gold mine of ideas, tools, and abstractions for understanding some of the most fundamental phenomena in quantum computing. Shor's algorithm and BB84 are two major examples of this, but here are some recent examples that we may cover in this class:

- **Classical verification of quantum computation** How can a classical computer *verify* the results of a quantum computation? This was a longstanding open question, which was finally resolved by Mahadev in 2018 using tools from cryptography.
- **Fully quantum complexity theory.** In general, a quantum computation transforms a quantum input state into a quantum output state. But traditional complexity theory is about the hardness of computing *functions*, which is a map from classical inputs to classical outputs e.g., the halting problem itself is about computing the function that takes as input the description of a Turing machine and outputs whether or not it halts. What, if anything, can classical complexity theory say about general quantum computations? It turns out that our best understanding of this question comes from studying the hardness of breaking quantum

cryptography.

- **Quantum pseudorandomness.** Is there a way to efficiently implement random unitaries? We'll answer this question using cryptography. As it turns out, this question has physical significance: highly chaotic systems such as black holes are often modeled as implementing random unitary transformations, which is at odds with the fact that physical systems must be computationally efficient.

2 Logistics

Grading

- 50% in-class participation. This is broken down into
 - asking (and answering) questions in class
 - e-mailing me a short write-up of what you learned after each class. The writeup should be two parts. Please try to send this within three days of the lecture.

Part 1. Imagine you are explaining what you learned in lecture to a friend who missed the class. Explain the key ideas that were covered, what made sense to you, and what you found confusing. **Please do not use LLMs for Part 1.**

Part 2. If you found anything confusing, please try to clear up your confusion by asking an LLM. If the LLM proves to be helpful, please write up what you asked the LLM and what you got from its explanation. If you're still confused, please say so in the report. **Even though this part involves talking to an LLM, you must still write up everything in your own words.** If everything in the lecture made perfect sense, feel free to skip part 2.

If you any other feedback for me about the lecture, feel free to include it.

- 25% final project presentation
- 25% final project report

For the final project, you have two options:

- A reading project: read 3-6 papers on a topic that was not explicitly covered in class, and write a report (up to 10 pages) motivating the topic and explaining the key ideas in your own words. You may use LLMs to assist your understanding of these papers, but you must write up the report in your own words.
- A research project: (attempt to) conduct original research on any quantum cryptography topic. You may use LLMs to assist your research, but you are responsible for understanding everything that you write in your final report.

A final project proposal will be due on November 2.

The last class meetings (Dec 7 and Dec 14) will be reserved for student presentations.

3 Shor's algorithm and quantum bootcamp

Perhaps the most famous application of quantum computers is Shor's algorithm, which is a quantum polynomial-time algorithm for factoring. For the rest of this lecture, we will sketch Shor's algorithm. Next lecture, we'll go through Shor's lecture again more slowly, and use it as a chance to go through the basics of quantum info/computation.

Shor's algorithm can be broken down into a classical step and a quantum step:

1. Classical step: reduce factoring to period-finding
2. Quantum step: use a quantum computer to solve the period-finding problem.

Before we talk about anything quantum, let's see why factoring reduces to period-finding.

3.1 Reducing factoring to period-finding

Our goal is to factor $N = pq$. We'll do this as follows:

1. Pick a random integer a from $2, 3, \dots, N - 1$. If we get extraordinarily lucky and $\gcd(a, N) > 1$, then we've already found a factor and we're done.
2. Otherwise, we know a is coprime to N . Then we'll use a quantum computer to compute the smallest r such that $a^r \equiv 1 \pmod{N}$. r is known as the *order* of a .
3. Since $a^r \equiv 1 \pmod{N}$, we know that $a^r - 1$ is a multiple of N , i.e., $a^r - 1 = cN$ for some integer c . Now, **assume for a moment that r is even**. Then we can factor the left-hand side as $(a^{r/2} - 1)(a^{r/2} + 1) = cN$. Let us **also assume that $(a^{r/2} + 1)$ is not divisible by N** . Note that the other factor, $(a^{r/2} - 1)$ cannot be divisible by N (if it were, then $a^{r/2} \equiv 1 \pmod{N}$, but this would contradict minimality of r). So, our setup is that $(a^{r/2} - 1)(a^{r/2} + 1)$ is divisible by pq , but neither factor is divisible by pq . The only way this can happen is if one factor is divisible by p (and not q) and the other is divisible by q (and not p). So, taking the gcd of either factor with N gives us a non-trivial factor of N .

As it turns out, for a random a , these assumptions we made hold simultaneously with probability at least $1/2$. This follows from some elementary number theory, but we're just going to take it as given for this lecture.

So now, we've reduced factoring to a period-finding problem. Concretely, I have a sequence $a^0, a^1, a^2, \dots \pmod{N}$ that has period r . I want to find r . This seems hard since r is likely exponential, and so naively brute-force computing $a^0, a^1, a^2, \dots$ until something repeats will take too long.

3.2 Sketch of the quantum period-finding algorithm

We'll first sketch how the quantum period-finding algorithm works at a high-level. I'll highlight the quantum concepts necessary to understand each step, and then next lecture we'll fill these concepts in.

1. We'll pick some Q which is a power of 2, such that Q is roughly N^2 . Then we'll prepare the

following quantum state:

$$\frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} |k\rangle \quad (1)$$

If you haven't seen this notation before, we'll go over it in more detail in the next class.

2. Next, we'll use two things. First, that there is an efficient classical algorithm, namely repeated squaring, to compute $a^k \bmod N$ from k .¹ Here, efficient means that you can do this in time *logarithmic* in k . Second, we'll use the fact **that quantum computing is an extension of classical computing**. In particular, anything you can compute classically, you can also compute quantumly, and so, a quantum computer can efficiently transform the above state into:

$$\frac{1}{\sqrt{Q}} \sum_{k=0}^{Q-1} |k\rangle |a^k \bmod N\rangle. \quad (2)$$

3. Next, we'll measure the second register, i.e., the part of the state containing $|a^k \bmod N\rangle$. Measuring this gives some $y \in \{0, 1, \dots, N-1\}$. The first register collapses to a superposition over all k such that $a^k \bmod N \equiv y \bmod N$. (If this step is unclear, we'll come back to it when we talk about how quantum states behave under **partial measurement**.)

Let s be the smallest k such that $a^s \equiv y \bmod N$. Observe that the remaining state is of the form:

$$\sum_{\substack{k \in \{0, 1, \dots, Q-1\}, \\ a^k \equiv y \bmod N}} |k\rangle = |s\rangle + |s+r\rangle + |s+2r\rangle + \dots \quad (3)$$

4. So, we've gotten the hidden r to show up in the *spacing* of the above "comb" state. How can we extract it? Note that if we simply measure the above state, we're not going to learn r : since we don't know s , it's just going to look like a uniformly random value. Here is the key step where we use quantumness.

We're going to apply the (Q -ary) Quantum Fourier Transform, which is a change of basis for extracting information about *periodic structure*. In this case, the useful property is that if you apply the QFT to the comb state $|0\rangle + |r\rangle + |2r\rangle + \dots$, the result is the comb state $|0\rangle + |Q/r\rangle + |2Q/r\rangle + \dots$ (omitting normalization).

Next class, we'll go through the **Quantum Fourier Transform** in more detail.

5. If I measure this, I'm going to learn sQ/r for some uniformly random s . I know Q , so I can divide it out and get s/r . If we're lucky and s and r are co-prime, this is enough to learn r . Otherwise, we can just take multiple samples and take the LCM of all the denominators.

Of course, we can't just assume that t is 0 and that r divides Q . However, you can show that if you shift everything by t , this only shows up in the *phase* of the Fourier transformed state. Also, if r doesn't divide Q , the Fourier transformed state isn't exactly what I claimed above, but it turns out to be close, and you can show the approximation is pretty good as long as Q is large relative to r . More on this next time.

¹The algorithm is simple: if k is a power of 2, you compute a^k by computing $a, a^2, a^4, a^8, \dots$. General k can be done by breaking k down into its binary decomposition, and then applying the previous idea for each power of 2.

So, to recap:

1. We computed the periodic function $f(k) = a^k \bmod N$ in superposition.
2. We then measured the output to get a superposition of basis states where the period shows up in the spacing between them,
3. Finally, we applied the quantum Fourier transform and measure to extract information about the period.