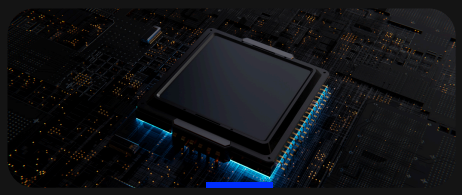
# Quantum Cryptography with Qiskit

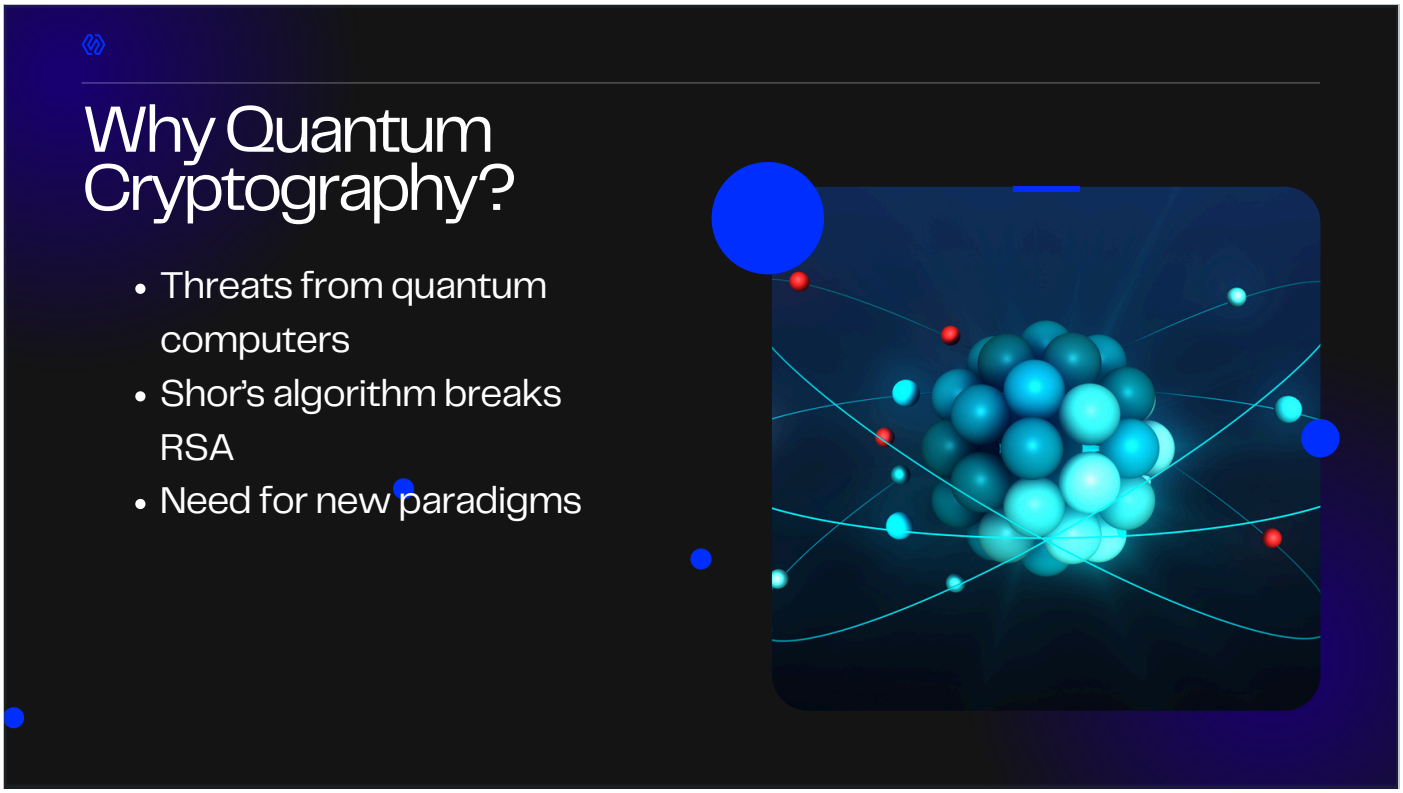
Name : Harshita Deewan
Presented Under The Guidance Of : Dr. Trivedi Harsh Chandrakant
College : LNMIIT, Jaipur



Hello, everyone! My name is Harshita Deewan, and today I am thrilled to present a topic that sits at the exciting frontier of physics and cybersecurity — Quantum Cryptography with Qiskit. This session is more than just theoretical—it's about understanding real-world security implications and actually seeing quantum code in action.

So what exactly are we going to explore today? First, we'll understand the fundamental problem quantum cryptography is trying to solve, particularly in light of the threats posed by powerful quantum computers. Then, we'll dive into how quantum physics, specifically its core properties like superposition and no-cloning, can be used to secure communication. Our journey will include a deep dive into the BB84 protocol — a classic in quantum key distribution — and we'll implement it step by step using IBM's Qiskit SDK.

By the end of this talk, you'll not only understand how quantum cryptography works theoretically but also have the tools and know-how to simulate it yourself. So let's begin this fascinating journey.




To appreciate the need for quantum cryptography, let's first look at the risks posed by quantum computing. Most modern cryptographic systems like RSA and ECC are based on mathematical hardness—problems like factoring large integers or computing discrete logarithms. These are tough for classical computers to solve, and that's why they're secure.

But in 1994, Peter Shor proposed an algorithm that changes the game. Shor's algorithm, when executed on a sufficiently powerful quantum computer, can efficiently factor large numbers — completely undermining the security of RSA. It's not a matter of if this becomes a threat, but when. Research in quantum hardware is progressing rapidly, and it's only a matter of time before today's cryptography becomes obsolete.

Quantum cryptography doesn't rely on hard math. Instead, it leverages the laws of quantum physics — laws that are fundamentally unbreakable. This shift from mathematical assumptions to physical principles offers security even in a future dominated by quantum computers.

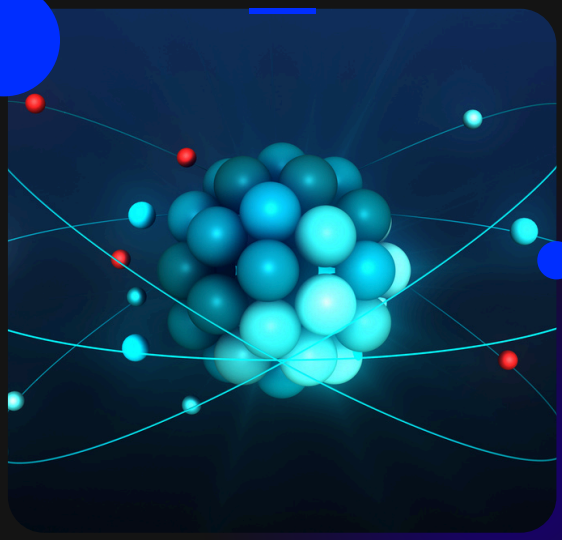

Quantum cryptography leverages two key properties of quantum mechanics. First, the no-cloning theorem: you can't make an exact copy of an unknown quantum state. Second, measurement disturbance: when you observe a quantum state, you inevitably alter it.

These phenomena form the backbone of protocols like Quantum Key Distribution or QKD. In QKD, two parties can share a secret key over a public channel and detect any third-party interference. The key difference is: in classical communication, an eavesdropper can copy messages undetected. In quantum communication, copying or measuring a qubit disrupts it, alerting the users.

A major point: quantum cryptography doesn't encrypt the message directly. Instead, it secures the key used for encryption. So, you could still use AES for data encryption — but the key is shared using QKD, ensuring it wasn't intercepted.

# Quantum Key Distribution (QKD)

- Purpose: share a secret key securely
- Detects eavesdropping
- Most famous: BB84 protocol



Quantum Key Distribution addresses a fundamental problem in secure communication: how to share a key safely when someone could be listening. Unlike classical protocols like Diffie-Hellman, QKD doesn't rely on math puzzles. It uses qubits.

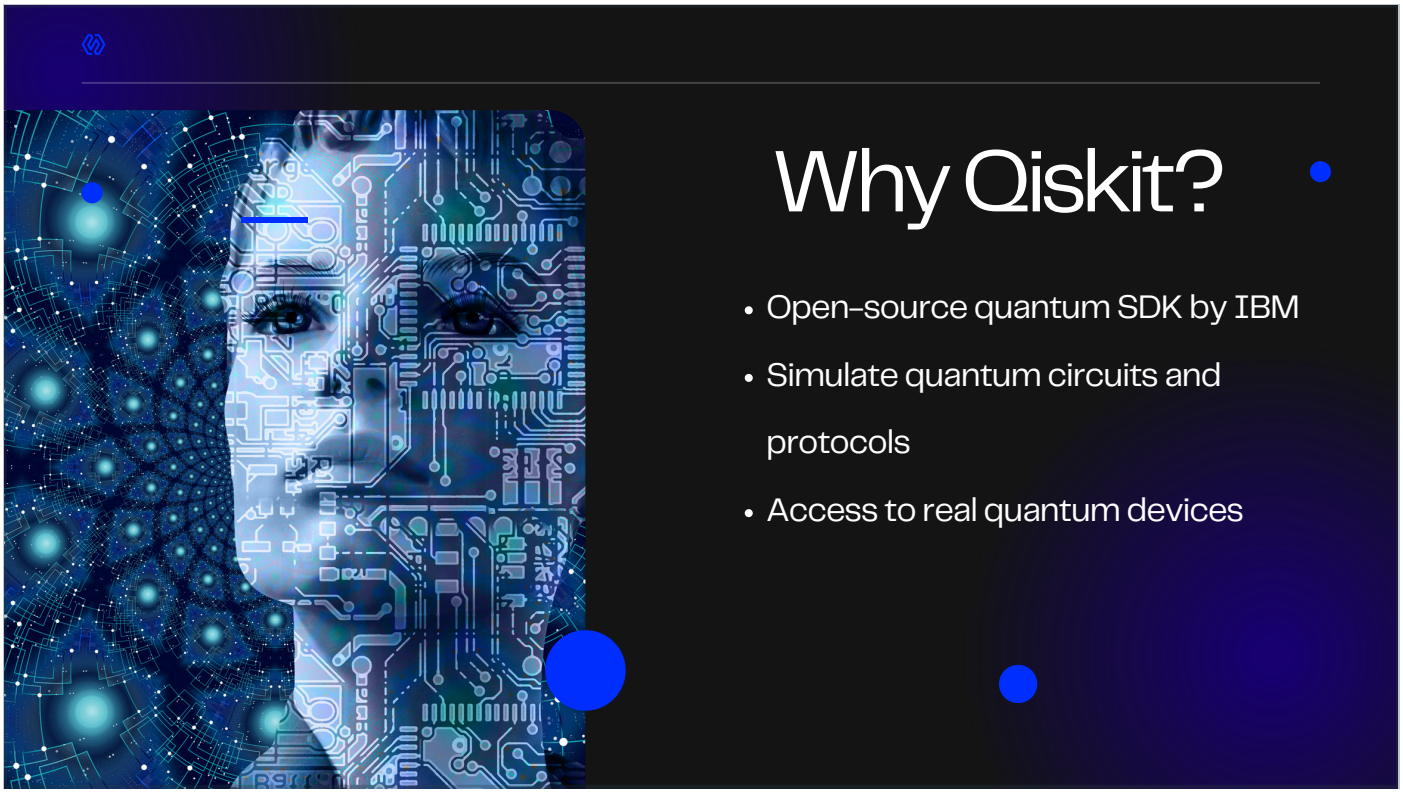
Take BB84, for instance — developed by Bennett and Brassard in 1984. It's simple but powerful. Alice sends qubits encoded in randomly chosen bases. Bob receives them and measures using his own random bases. Later, they publicly discuss which bases they used — not the actual bits — and discard mismatches.

If an eavesdropper, say Eve, tries to intercept the qubits, she has to guess the basis. If she guesses wrong, she disturbs the qubits, introducing detectable errors. That's the magic of quantum mechanics — any unauthorized observation is inherently detectable.


The BB84 protocol is a brilliant use of quantum mechanics for secure key exchange. Let's break it down. There are two bases: rectilinear (which we denote as '+') and diagonal (denoted as '×'). Each basis allows encoding of 0 or 1 differently.

Alice randomly picks a bit value and a basis for each qubit. Bob, without knowing Alice's choices, also picks a random basis to measure. After the transmission, they compare their bases over a public channel. Wherever the bases match, Bob is likely to get the correct bit. Those bits are kept; the rest are discarded.

They then test a sample for error rate. If it's low, they proceed. If not, it suggests Eve may have tampered. This entire scheme works because quantum states collapse when measured incorrectly. No sophisticated math — just pure physics.




Qiskit is IBM's open-source SDK for quantum programming in Python, and it's the perfect tool for our purpose today. It simplifies quantum computing by offering a high-level way to create circuits, run simulations, and even access real quantum hardware through the IBM Quantum Experience.

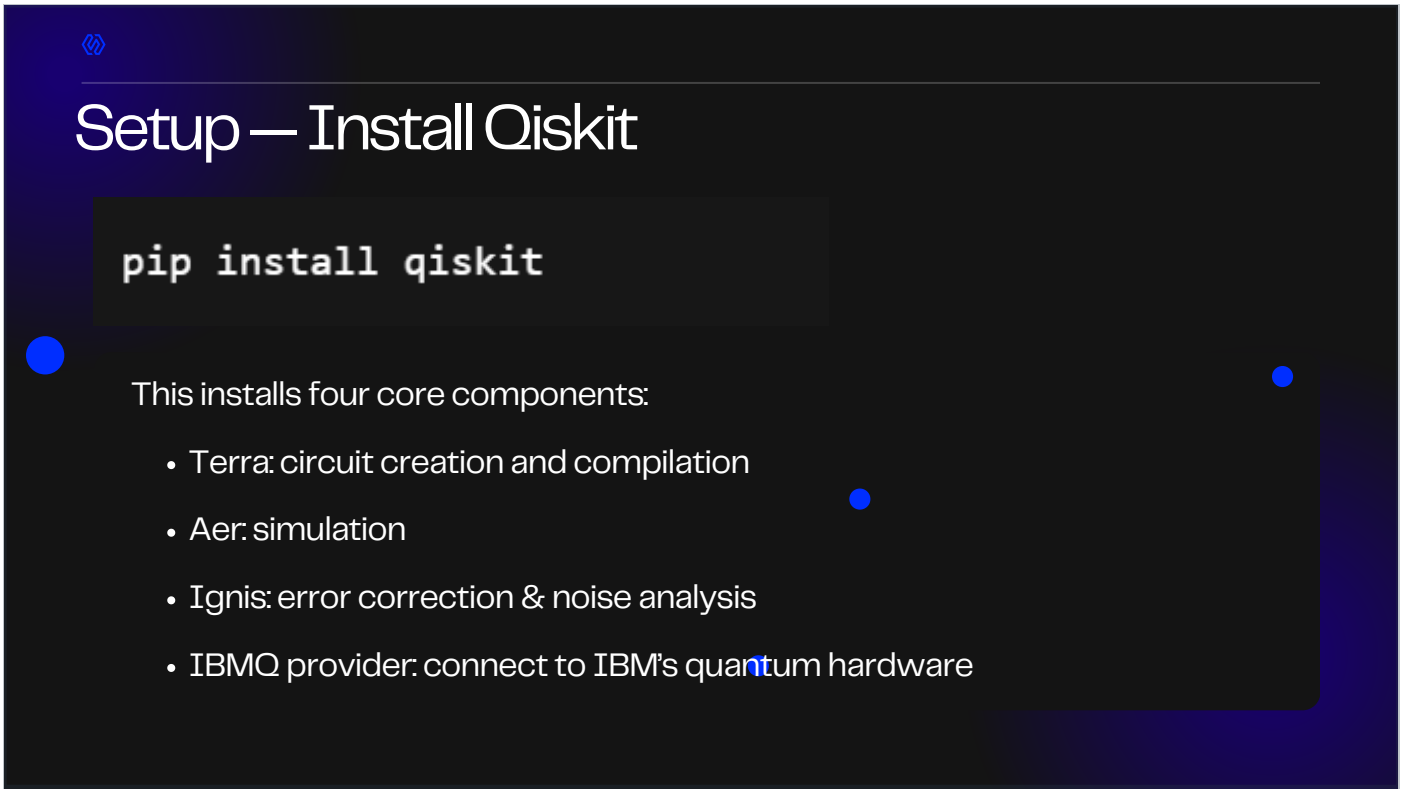
For our implementation of BB84, Qiskit gives us:

Terra, to build circuits.

Aer, to simulate them.

IBMQ, to run them on real devices.

This makes quantum cryptography accessible even if you're not a physicist. You don't need to understand every nuance of quantum gates — Qiskit provides abstractions, visuals, and an environment to learn by doing.




Before we dive into coding the BB84 protocol, we need to set up our environment. Qiskit is easy to install using Python's pip package manager. Just type:
pip install qiskit

This command installs four important components of the Qiskit framework:

Terra – This is the foundation. It allows us to create and manipulate quantum circuits. You can think of it as the syntax layer for building quantum programs.

Aer – This is the simulator. Since we don't all have quantum computers on our desks, Aer lets us test our quantum circuits using high-performance classical simulation. This is essential for debugging and educational purposes.

Ignis – Although not used heavily in our BB84 example, Ignis is crucial for analyzing and correcting noise in quantum systems. This is useful if you're working on quantum error correction or benchmarking real quantum devices.

IBMQ Provider – This lets us connect to real quantum computers hosted by IBM via the cloud. All you need is a free IBM Quantum Experience account and an API token.

Once you install Qiskit, you're equipped with a full quantum computing toolkit. And thanks to its open-source nature, there's a vibrant global community and plenty of tutorials to explore.

# Step 1 – Generate Random Bits and Bases

- Alice's random bits & bases
- Bob's random bases

Now, let’s start building our BB84 protocol from scratch.

The very first step involves generating randomness — a cornerstone of cryptographic security.  Alice needs to prepare two things:

A random sequence of bits (0s and 1s). These will be the actual values she wants to communicate .

A random sequence of bases — either '+' (rectilinear) or '×' (diagonal). Each bit will be encoded in  one of these bases.

Why is this randomness essential? Because if the bits or bases were predictable, an eavesdropper could potentially guess or replicate the communication. Randomization ensures  unpredictability, which is vital for security.

Meanwhile, Bob, who doesn’t know Alice’s choices, also randomly selects a basis for each qubit  he receives. He’ll use this basis to measure the qubits.

This mismatch in basis selection is what makes BB84 powerful. It introduces a way to detect if  someone has interfered with the communication, since an incorrect basis collapses the qubit unpr edictably.

In our implementation, we usually work with a small number like 10 or 20 qubits to demonstrate

the concept, but in real-world QKD systems, we'd be dealing with thousands or millions of bits to  generate keys suitable for strong encryption.

# Python Code — Alice's Preparation

```
import random
n = 10
alice_bits = [random.randint(0,1) for _ in range(n)]
alice_bases = [random.choice(['+', 'x']) for _ in range(n)]
```

Now, let's look at the code for generating Alice's random bits and bases.

In Python, we can generate random bits using:

alice_bits = [random.randint(0, 1) for _ in range(n)]

This gives us a list of 0s and 1s. For the bases, we typically use:

alice_bases = [random.choice(['+', 'x']) for _ in range(n)]

This randomly chooses between the rectilinear and diagonal bases.

So at this point, Alice has two arrays:

One that holds the bit values she wants to send.

Another that tells her how she'll encode each of those bits — in '+' or '×'.

This setup mirrors what happens in a real quantum key distribution system: the transmitter prepares qubits based on these choices, and sends them over a quantum channel — typically an optical fiber or even free space using photons.

And even though this looks simple in code, this is where quantum communication begins —

randomness, bases, and preparation form the heart of the protocol.

# Step 2 – Build Quantum Circuits

- Apply X gate for bit=1
- Apply H gate for diagonal basis

With Alice's random bits and bases ready, the next step is to encode them into quantum states.

We start with the basic quantum state: $|0\rangle$. Every qubit in Qiskit starts in this ground state.

Here's how Alice encodes her bit:

If the bit is 0 and the basis is rectilinear ('+'), she does nothing — the qubit remains $|0\rangle$.

If the bit is 1 and the basis is rectilinear ('+'), she applies an X gate, flipping the qubit to $|1\rangle$.

If the basis is diagonal ('×'), she first prepares $|0\rangle$ or $|1\rangle$ as usual, then applies a Hadamard gate (H) to rotate the qubit into the diagonal basis.

What's amazing is that with just these two gates — X and H — we can represent all four possible combinations:
$|0\rangle$ (bit 0, + basis)
$|1\rangle$ (bit 1, + basis)
$|+\rangle$ (bit 0, × basis → $H|0\rangle$)
$|-\rangle$ (bit 1, × basis → $HX|0\rangle$)

This process is implemented in Qiskit by creating a circuit for each qubit. Each circuit will have 1 or 2 gates, depending on the bit and basis. It's a clean and elegant way to simulate how a real quantum transmitter, like a photon emitter, would operate in a QKD system.

## Python Code — Alice's Circuits

```
from qiskit import QuantumCircuit
circuits = []
for bit, base in zip(alice_bits, alice_bases):
    qc = QuantumCircuit(1,1)
    if bit == 1:
        qc.x(0)
    if base == 'x':
        qc.h(0)
    circuits.append(qc)
```

In code, this step becomes quite visual.

We loop through each index of the bit and basis arrays. For every bit:

We start with a new quantum circuit.

If the bit is 1, we apply an X gate.

If the basis is diagonal ('×'), we apply a Hadamard gate.

Here's what it looks like:

```
if bit == 1:
   circuit.x(0)
if basis == 'x':
   circuit.h(0)
```

Each of these circuits is then stored in a list. Later, this list of circuits will be passed on to simulate  transmission and measurement by Bob.

Even though we're using simulation, this mimics how real hardware would prepare the quantum  states — especially in photonic systems, where different polarizations are used to represent bases .

One great thing about Qiskit is that it allows us to visually inspect these circuits. We can draw them and see exactly which gates are applied — making it a fantastic educational tool for both quantum computing and quantum cryptography.

# Step 3 – Bob's Measurement

- Bob applies H if measuring in diagonal basis
- Measures qubit

Now the qubits prepared by Alice are transmitted to Bob — remember, in a real-world implementation, this would be done using photons traveling through fiber optic cables or even free-space optics.

But here's the catch: Bob doesn't know which basis Alice used for each qubit. So, he must guess. For each incoming qubit, he randomly chooses either the rectilinear ('+') or diagonal ('×') basis to measure.

The measurement strategy is as follows:

If Bob chooses rectilinear, he measures the qubit directly. This basis corresponds to the computational basis, meaning no gate needs to be applied — just a straightforward measurement.

If Bob chooses the diagonal basis, he needs to apply a Hadamard (H) gate before measurement. This is because a Hadamard gate effectively rotates the qubit from diagonal to rectilinear space, enabling a meaningful measurement.

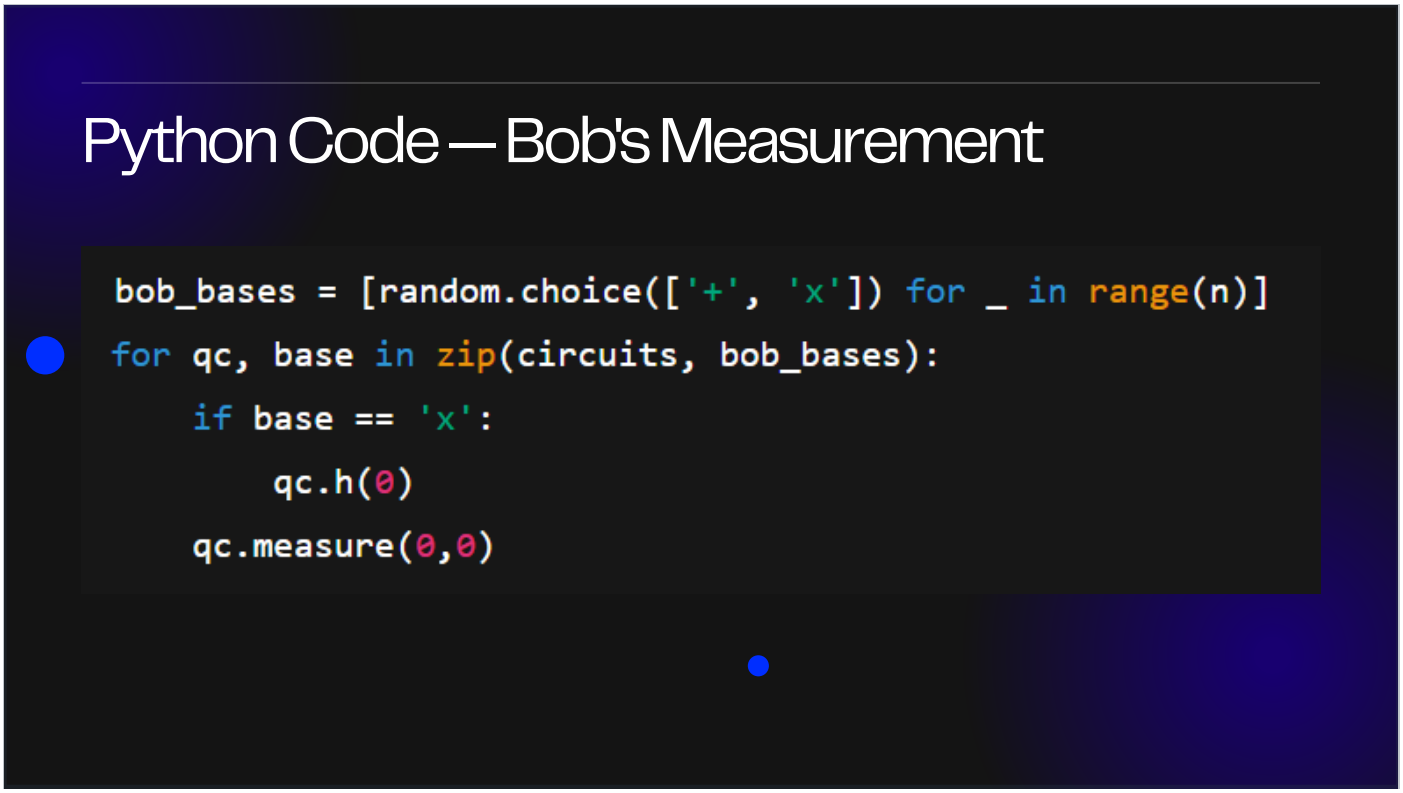
The brilliance of BB84 lies in this uncertainty — Bob's random choice of basis, coupled with quantum measurement disturbance, creates a situation where incorrect guesses cause errors, and these errors become a telltale sign of interference or eavesdropping.

In code, we'll loop through Bob's list of random bases, and for each qubit:

If the basis is diagonal, apply H.

Then measure.

This step converts the quantum information into classical bits, which we can later compare with  Alice’s to extract the final key.




Let's look at the code that simulates Bob's measurement.

First, Bob generates his own list of random bases:

bob_bases = [random.choice(['+', 'x']) for _ in range(n)]

Then, for each qubit circuit received from Alice:

If Bob's basis is diagonal, we add a Hadamard gate before measurement.

Finally, we measure the qubit and store the result.

Here's a simplified version:

```
if bob_basis == 'x':
   circuit.h(0)
circuit.measure(0, 0)
```

Each measurement collapses the qubit into either 0 or 1, depending on both the qubit's original state and Bob's chosen basis.

The measurement outcome is probabilistic. If Bob's basis matches Alice's, he likely gets the correct bit. If not, there's a 50% chance he's wrong. This uncertainty adds a layer of security — if

an eavesdropper tries to measure the qubit in the wrong basis, it introduces detectable noise.

This measurement phase is critical because it finalizes the 'quantum' part of BB84 — the information has now been observed and transformed into classical data. From here on, we move  into classical post-processing.

# Step 4 – Simulate in Qiskit

- Use Aer simulator
- Get measurement results

Since we are running this experiment on a classical machine, we need to simulate the behavior of our quantum system.

Qiskit's Aer module provides a high-fidelity simulator called qasm_simulator, which simulates quantum circuits using probabilistic models. This lets us 'observe' qubit behavior as if they were executed on a real quantum device.

Why simulate instead of directly running on real quantum hardware? Well, for one, quantum computers are still scarce, have limited qubit counts, and are prone to noise. Simulation gives us:

Deterministic and reproducible results, helpful for learning.

Faster execution, especially for basic circuits.

Noise-free outputs, unless we explicitly add noise models.

To simulate, we bundle all the circuits into a list and submit them to the simulator using:

execute(circuits, backend, shots=1)

Notice we set shots=1, because in BB84, each qubit is only transmitted and measured once. This aligns with the protocol's use of single-photon states in real-world implementations.

Once executed, we receive measurement outcomes, which we'll use in the next phase to construct the shared key.

## Python Code — Run Simulation

```
from qiskit import Aer, execute
backend = Aer.get_backend('qasm_simulator')

results = []
for qc in circuits:
    job = execute(qc, backend=backend, shots=1)
    counts = job.result().get_counts()
    bit = int(list(counts.keys())[0])
    results.append(bit)
```

Let’s walk through the code to simulate the measurement process.

After Bob adds measurement operations to each circuit, we collect all those circuits and run:

job = execute(circuits, backend=qasm_simulator, shots=1)

This launches a simulation of the circuits. When complete, we retrieve the measurement results:

result = job.result()
counts = result.get_counts()

Each counts[i] contains the measurement result (a string like '0' or '1') for the i-th qubit. Since we  only use one shot per qubit, each result is unique.

These results are stored in Bob’s measured bits array. We now have all the classical data we nee d:

Alice’s original bits.

Alice’s encoding bases.

Bob’s measurement bases.

Bob’s measured bits.

The quantum part is over. The next steps involve reconciling this classical data to extract a shared  key and detect potential eavesdropping.

# Step 5 – Basis Reconciliation

- Compare bases
- Keep bits where bases matched

After transmission and measurement, the next crucial step is basis reconciliation.

Here's how it works:

Alice and Bob publicly reveal the sequence of bases they used — but not the actual bit values.

They compare the sequences and identify which positions had matching bases.

The bits at those positions are considered valid and kept.

The bits where bases differ are discarded.

This step is done over a classical public channel, which anyone can hear — but that's okay because we're not revealing the actual bits, only the bases. Without knowing the original states or having intercepted the qubits, an eavesdropper gains no useful information.

The result is a raw key — a string of bits where both Alice and Bob used the same basis, and thus Bob likely measured the correct value.

In practice, this raw key is not used as-is. It undergoes further processing: error correction (to fix any discrepancies) and privacy amplification (to reduce any partial knowledge Eve may have).

But for the purpose of learning BB84, this reconciliation gives us a solid understanding of how

quantum physics and classical communication interact to ensure security.

# Python Code — Extract Final Key

```
key = []
for a_base, b_base, a_bit, b_bit in zip(alice_bases, bob_bases, alice_bits, results):
    if a_base == b_base:
        key.append(a_bit)
```

This is the moment where Alice and Bob attempt to extract their shared secret key from the data collected so far.

Let's break it down. We iterate through all the bits and:

Check if Alice's and Bob's bases match at each index.

If they do, we assume Bob's measurement is likely accurate.

We add Alice's original bit at that index to the shared key.

In code, it looks something like:

```
if alice_bases[i] == bob_bases[i]:
  final_key.append(alice_bits[i])
```

Why not use Bob's bit here? Because we're using a simulation and assuming ideal conditions, Alice's bit is guaranteed. In real-world implementations, we'd compare both to detect discrepancies caused by noise or potential eavesdropping.

This filtered list of bits is known as the raw key. It's not yet the final secure key — further processing like error correction and privacy amplification would still be needed to remove any errors and limit potential knowledge gained by an attacker.

It’s important to understand that quantum key distribution (QKD) is not about transmitting messages — it’s about securely distributing keys. Once a shared key is established, Alice and Bob can use it in classical symmetric encryption schemes like AES or One-Time Pads, which offer confidentiality for actual messages.

# Sample Output

| Index | Alice Bit | Alice Basis | Bob Basis | Bob Measured Bit | Kept in Key? |
|---|---|---|---|---|---|
| 0 | 1 | + | + | 1 | ✓ |
| 1 | 0 | x | + | 1 | ✗ |
| 2 | 1 | + | + | 1 | ✓ |
| 3 | 0 | + | x | 0 | ✗ |
| 4 | 1 | x | x | 1 | ✓ |
| 5 | 0 | x | + | 0 | ✗ |
| 6 | 1 | + | + | 1 | ✓ |
| 7 | 0 | x | x | 0 | ✓ |
| 8 | 1 | x | x | 1 | ✓ |
| 9 | 0 | + | x | 1 | ✗ |

Final Shared Key (Alice's bits where bases matched):

```
[1, 1, 1, 0, 1]
```

Let's visualize what all this effort has achieved with a sample output from our simulation.

Here we can see:

The random bits Alice generated.

The bases Alice used to encode those bits.

Bob's randomly chosen measurement bases.

Bob's measured bits.

Then we do basis comparison:

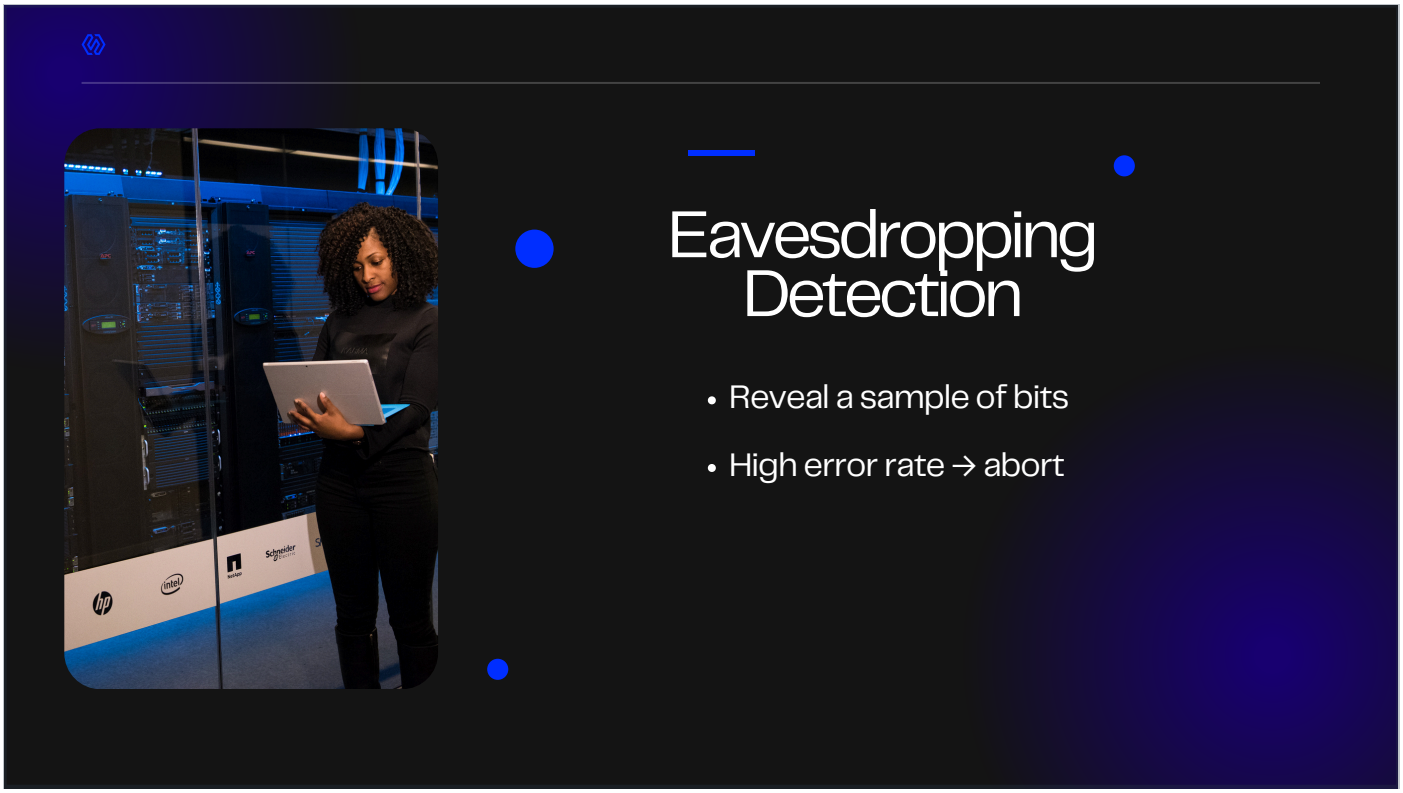
We check each index — if the bases match, we mark it with a ✓.

If they don't match, we mark it with an ✗ and discard that bit.

The bits that survive this filtering become the shared key.

This table is incredibly useful for debugging and understanding the protocol visually. In real implementations, this reconciliation would be done programmatically, and the sample used for eavesdropping detection would be chosen at random.

The goal is to confirm that the shared key bits — those where bases matched — are sufficiently accurate and haven't been tampered with. This visibility helps us understand why the BB84 protocol is so effective and secure, even in the presence of potential attackers.




One of the most fascinating aspects of quantum cryptography is its ability to detect eavesdropping.

Here's how: After key generation, Alice and Bob agree to publicly compare a small random sample of the shared bits from their raw key. If these samples show a high rate of mismatches, it means the quantum channel was disturbed.

Why does this work?

Because any measurement on a qubit — whether by Bob or an attacker like Eve — collapses the state. If Eve tries to measure qubits during transmission:

She doesn't know the basis Alice used.

If she guesses wrong and measures in the wrong basis, she disturbs the qubit.

When Bob receives that qubit and uses the correct basis, the disturbance may cause a wrong bit to be registered.

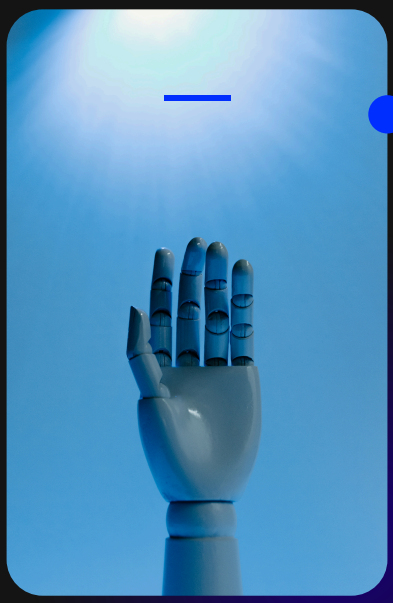
This causes an increase in error rate, which can be detected in the sample comparison step.

If the error rate exceeds a certain threshold, the session is aborted, and the key is discarded. If it's within tolerance, Alice and Bob proceed with error correction and privacy amplification.

So, this isn’t just theoretical. BB84 provides unconditional security, assuming the laws of quantum  mechanics hold — a massive leap forward from classical cryptography.

# Visualization in Qiskit

```
circuits[0].draw('mpl')
```



One of the most user-friendly and educational features of Qiskit is its ability to visualize quantum ci rcuits.

As we've built our BB84 simulation step-by-step, each qubit has been represented by a circuit.  Qiskit allows us to generate diagrams that show exactly what operations were applied to each qubi t.

For example, we can see:

If an X gate was applied (for bit=1).

If an H gate was applied (for diagonal basis).

When and how a measurement was performed.

These diagrams make the abstract world of quantum computing much more tangible. They're  useful for:

Debugging your logic.

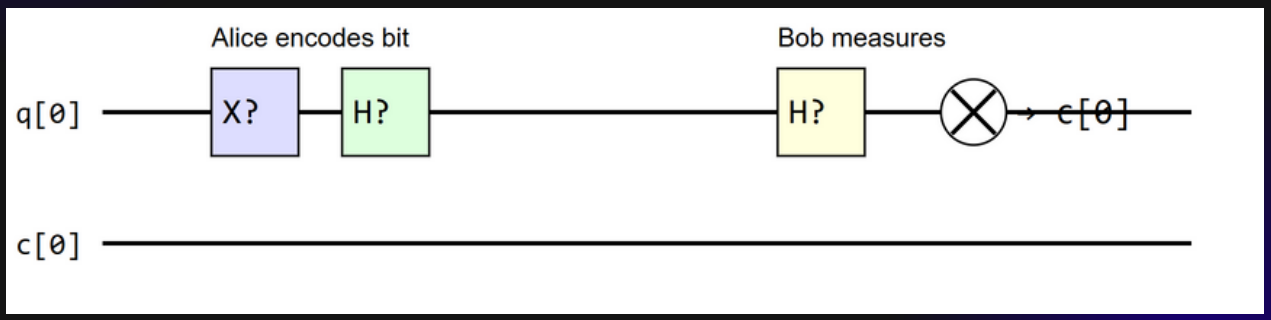
Understanding quantum algorithms.

Teaching others how quantum gates interact.

In BB84, seeing the visual difference between a qubit in $|0\rangle$, $|1\rangle$, $|+\rangle$, and $|-\rangle$ states reinforces the idea that encoding and measuring are basis-dependent processes.

Whether you're a student or researcher, circuit visualization is an indispensable tool in your quantum computing journey.

# Circuit Image




Here, we have a concrete example — a circuit diagram for how Alice prepares a single qubit.

Let's decode it:

We start with a default $|0\rangle$ state.

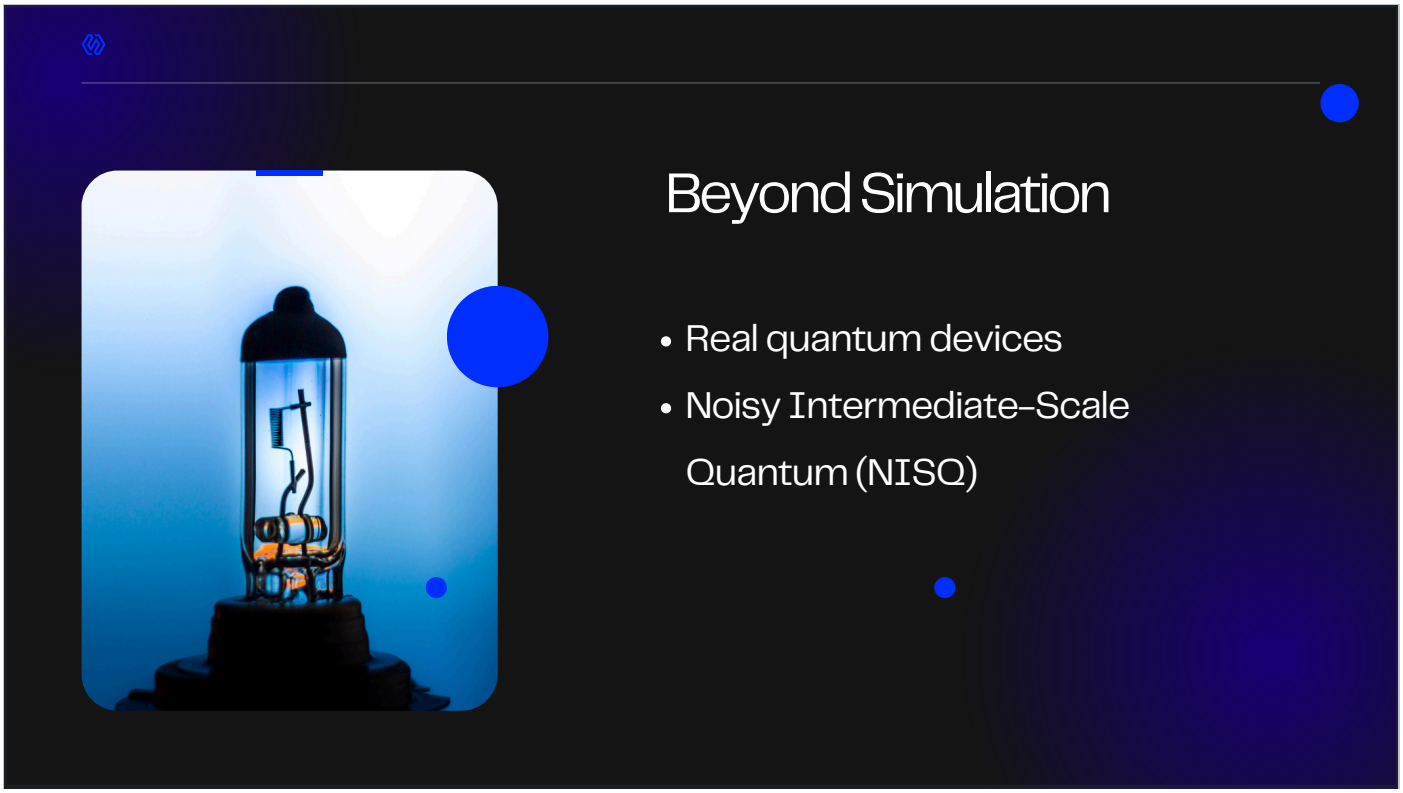
If the bit to be encoded is 1, we apply an X gate.

If the basis is diagonal, we apply an H gate after the X or on $|0\rangle$.

Finally, if we were visualizing Bob's side, we'd see an optional H gate followed by a measurement operation.

These circuit diagrams are the 'blueprints' of our quantum processes. They let us trace exactly how a quantum state was manipulated, and in what sequence.

In a lab setting, or even on real IBM Quantum devices, having a clear visualization of your quantum circuit is crucial for verifying correctness — especially in sensitive cryptographic protocols where precision matters.




Until now, we've simulated our quantum circuits — but what if we want to go further?

Qiskit supports real hardware execution via IBM Quantum's cloud-based access to superconducting quantum processors. These machines are noisy, limited in qubit number, and have gate errors — but they represent real quantum computing power.

We call this era NISQ — Noisy Intermediate-Scale Quantum. It's a transitional period where we don't have fault-tolerant, large-scale quantum machines yet, but we have enough capability to run small experiments and test theoretical ideas in practice.

BB84 on a real device introduces realistic challenges:

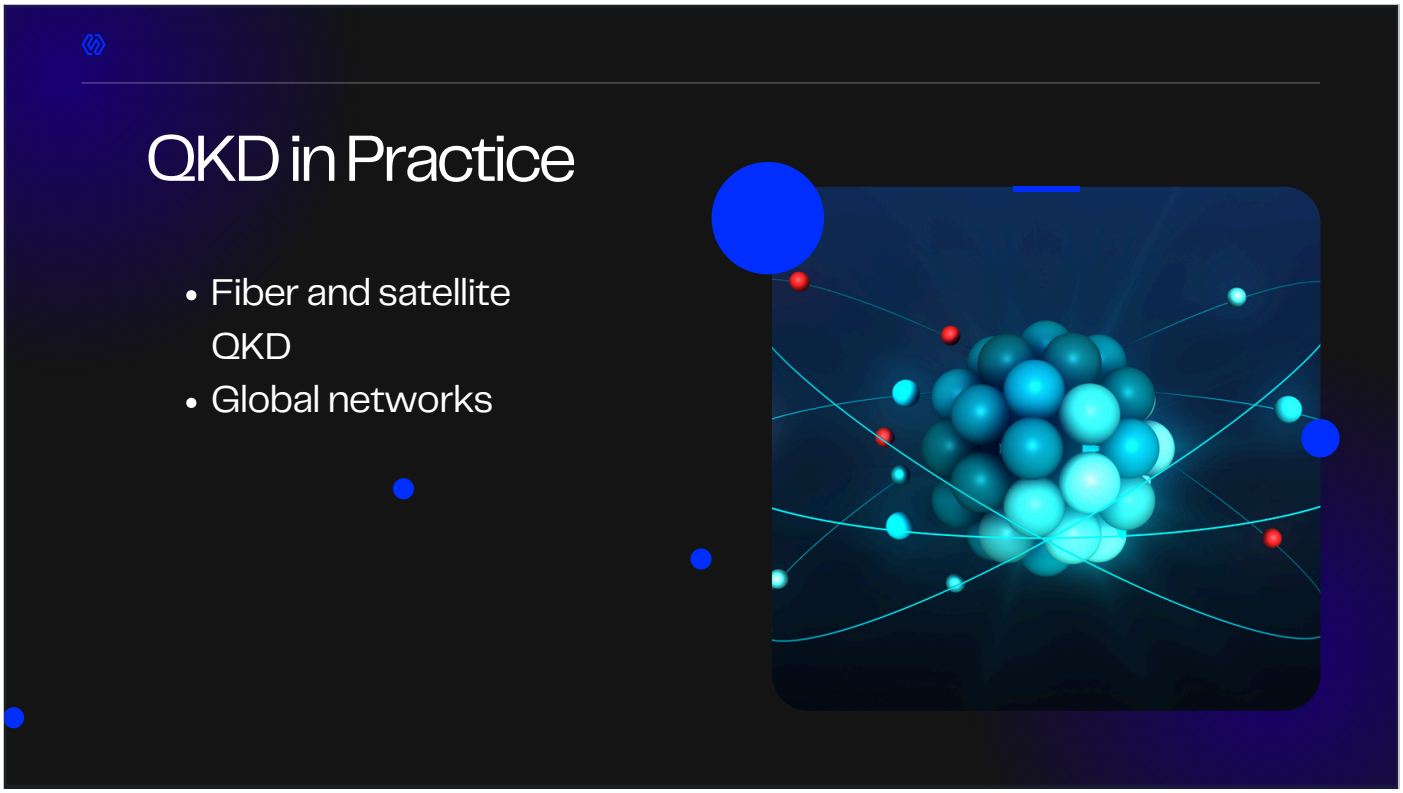
Gate errors may distort states.

Decoherence may affect fidelity.

Measurement noise may create false mismatches.

Testing BB84 on a NISQ device helps us understand how error correction and noise modeling must be integrated into future secure systems.

So even though today's hardware has limitations, the hands-on experience and insight you gain from running circuits on real quantum processors is invaluable.




Quantum Key Distribution is no longer just a theoretical experiment — it's happening in the real world.

Let me give you a few compelling examples:

China has deployed a 2000-kilometer quantum communication network connecting Beijing and Shanghai using fiber optics. It's called the Beijing-Shanghai QKD Backbone.

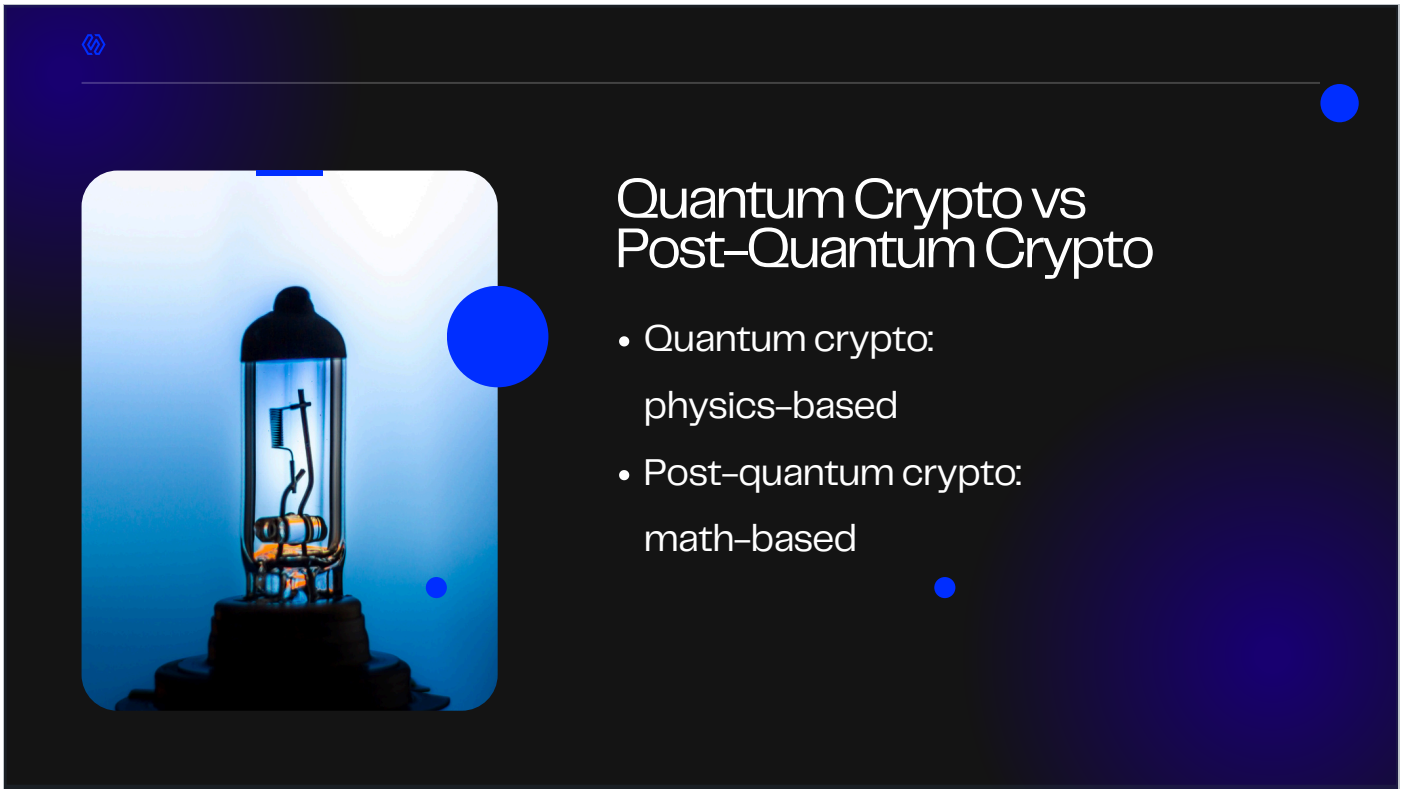
The Micius Satellite, launched in 2016, successfully performed space-to-ground QKD — transmitting entangled photons over 1,200 km between ground stations. This marked the first intercontinental quantum-encrypted video call.

European countries, Japan, and the U.S. have launched pilot projects and urban QKD testbeds.

These networks rely on specially engineered hardware like single-photon emitters, detectors, and synchronization systems.

We're now entering an era of quantum-secured global communication, where encryption keys are distributed via light — literally — and the laws of physics, not assumptions about computational difficulty, secure our data.

This is not just about privacy — it's about sovereignty, trust in critical infrastructure, and building future-proof communication systems.




It’s important to distinguish between Quantum Cryptography and Post-Quantum Cryptography (PQC) — both are responses to the same problem: the rise of quantum computers.

Quantum Cryptography uses the laws of physics — such as the no-cloning theorem — to secure communication. It's hardware-based, requires quantum channels, and includes protocols like BB84.

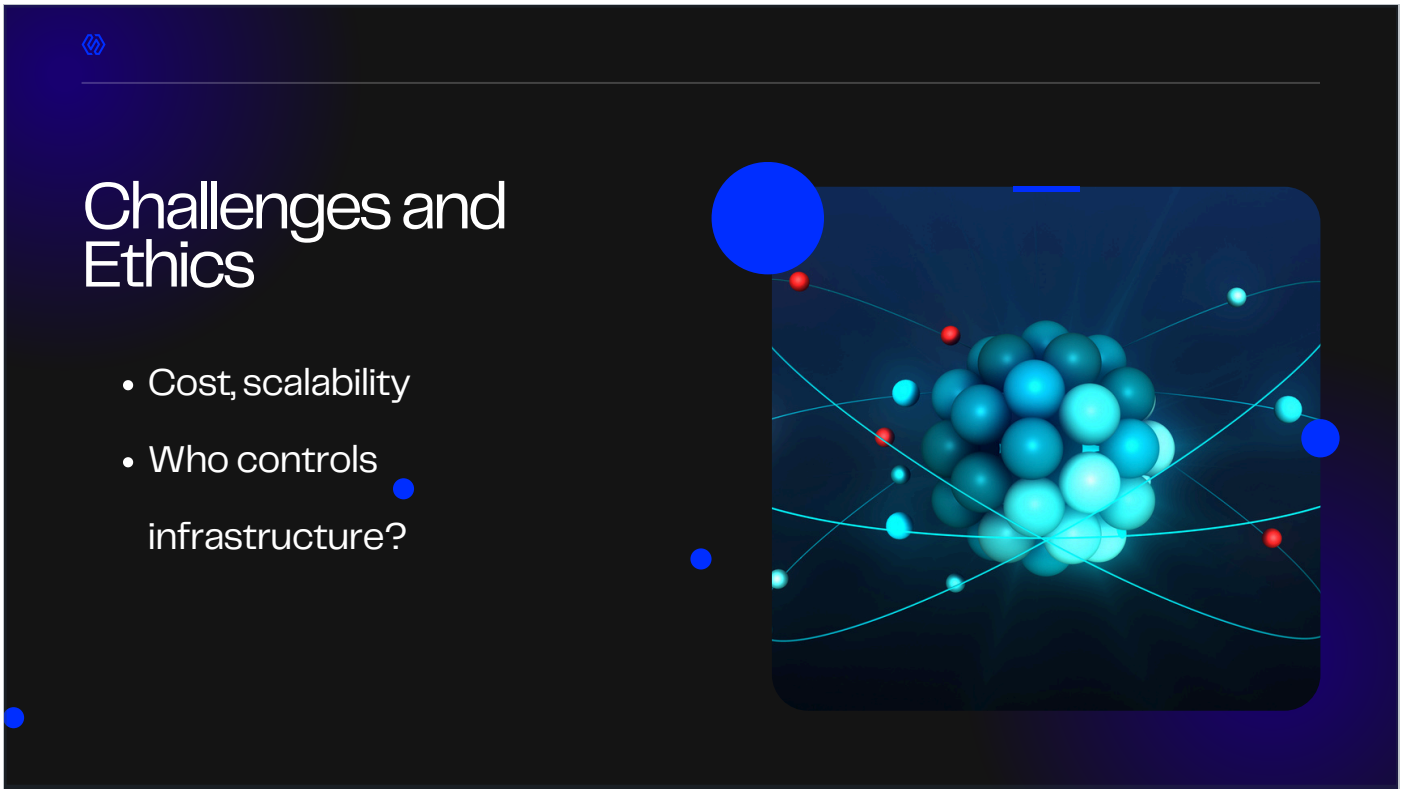
Post-Quantum Cryptography, on the other hand, sticks to classical hardware and software but uses new mathematical problems that are thought to be resistant to quantum algorithms — such as lattice-based cryptography, code-based cryptography, and multivariate polynomial problems.

Each has its pros and cons:

QKD offers provable security but is hard to scale.

PQC is easier to deploy but may still be broken in the future.

The future of cryptography likely lies in a hybrid approach — deploying PQC algorithms while also building out quantum-secure infrastructures where possible. The two aren’t enemies — they’re allies tackling the same quantum threat from different angles.




As promising as quantum cryptography sounds, it comes with real-world challenges and ethical concerns.

Cost: Quantum hardware like single-photon sources, detectors, and phase modulators is expensive. This raises questions about accessibility.

Scalability: Distributing quantum keys across long distances or through standard internet infrastructure isn't trivial. Repeaters, entanglement swapping, and satellites are complex and costly solutions.

Centralization and Control: Who builds and controls these networks? Will a few powerful nations or corporations monopolize quantum-secure communication, or will it be a decentralized, global effort?

Digital Sovereignty: Should governments have the right to monitor or control quantum communication networks for national security?

As technologists and scientists, we must not only innovate but also ensure these technologies are inclusive, transparent, and fair. Ethics must guide quantum innovation, especially when it intersects with privacy and security.


To wrap up, here are the key takeaways from today's session:

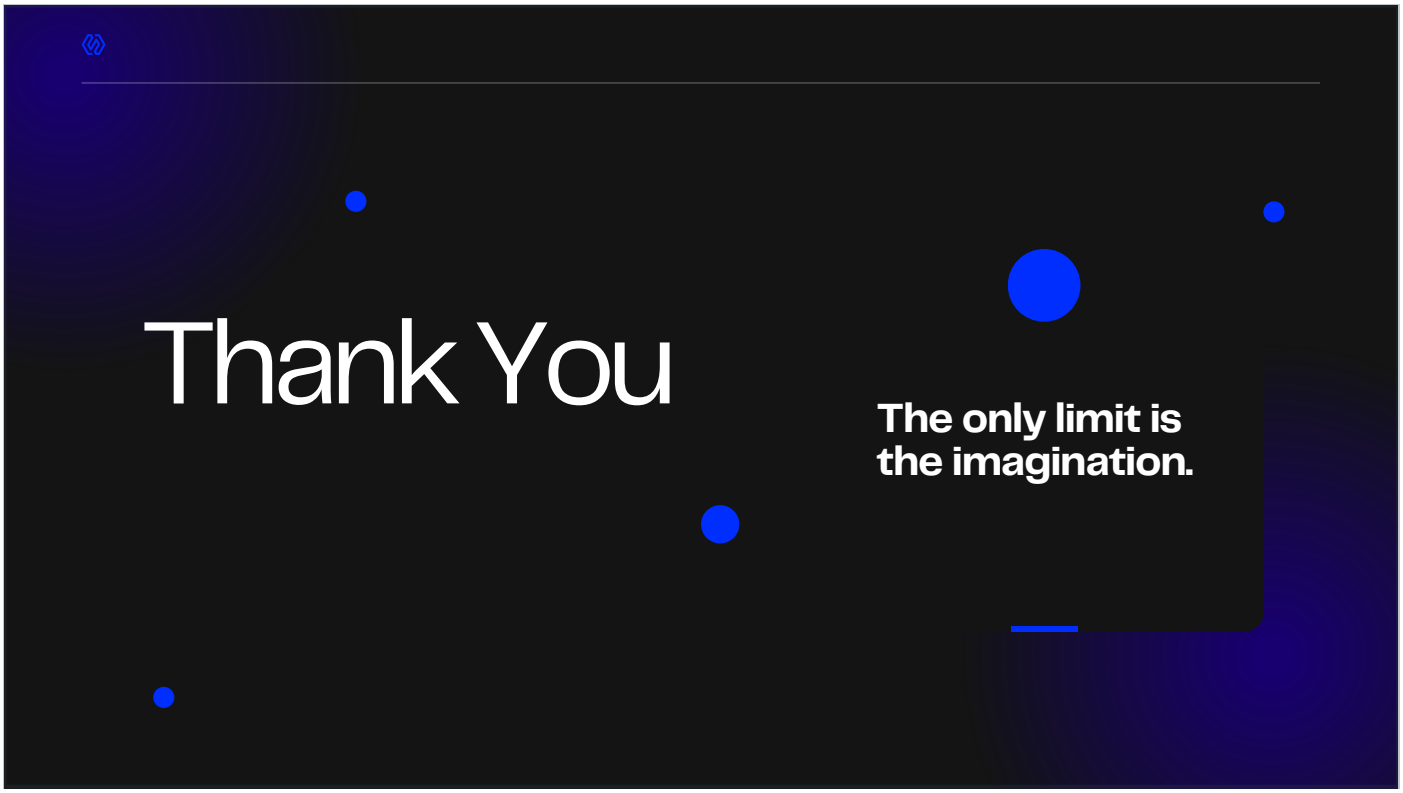
The BB84 protocol is a foundational technique in quantum cryptography. It enables two parties to generate a shared, secure key — with guaranteed detection of any eavesdropping.

Qiskit is a powerful, accessible tool that lets us simulate and visualize quantum protocols like BB84. It bridges the gap between theory and implementation.

Quantum cryptography is real and evolving. From lab experiments to global QKD networks and satellite demonstrations, it's shaping the future of secure communication.

But with promise comes responsibility — we must tackle scalability, cost, and ethical deployment as the field grows.

The beauty of quantum cryptography is that it combines hard science with deep philosophical implications — security not from secrecy, but from nature itself. It's not just a technological advance; it's a paradigm shift in how we protect information.




Thank you all for your time and attention.

I hope this session not only introduced you to the technical mechanics of quantum cryptography but also sparked a sense of wonder. We are standing at the edge of a new era — one where imagination, physics, and technology merge to secure our digital future.

Remember, quantum computing is still young. You don't have to be an expert today. What matters is curiosity, persistence, and a willingness to learn.

I'm grateful to my mentor, Dr. Trivedi Harsh Chandrakant, and to LNMIIT Jaipur for the opportunity to explore and present this topic. If you have any questions or would like to collaborate, please feel free to reach out.

As the slide says — 'The only limit is the imagination.' Thank you.