Simulation of Attacks on Quantum Key Distribution protocols

(Discussion on cryptography in the era of Quantum Computers and simulation of attacks on QKD protocols using Qiskit)

Supreeth BS, Aritra Mukhopadhay, Sagnik Sinha Choudhuri, Atanu Singha, Kirty Ranjan Sahoo, Sagar Gaur, Anirudh Arora, Soniya, Samyak Parashar and Abbena U

Course: Quantum computation: Introduction to algorithms and implementation using Qiskit 2022

Teachers-in-charge: Dr. Chetan Waghela, Dr. Dintomon Joy and Dr. Raghavendra V

July 2022

Contents

Abstract

In this paper, we demonstrate how the intercept and resend attack can happen on the well-known quantum key distribution protocols – BB84 and E91. But before this, we talk about the need for key distribution and establish the importance of reliable key distribution. On that account, we begin with a short introduction to cryptography and the types involved in it. Next, we explain how quantum computers pose a threat to asymmetric cryptography thereby indicating the need to resort to symmetric cryptography. Nonetheless, symmetric cryptography presents its own challenge — the reliable key distribution through a public channel. It turns out that we could use the very principles of Quantum mechanics such as superposition, entanglement, and no-cloning theorem to securely distribute the key. This security mainly originates from the fact that Quantum information unlike classical information is disturbed upon being intercepted by a third-party. How exactly these principles are used in the distribution of the key and detection of an interception forms the basis of these protocols.

The no-cloning theorem is the main principle used in all the protcols. BB84 specially uses superposition, non-orthogonality, and the uncertainty principle while the E91 uses the principle of entanglement and takes advantage of Bell-CHSH inequality to check for interception. The former protocol is fairly easy to understand for any freshman while the latter requires a deeper knowledge of quantum foundations. Therefore, in this paper, we have also broken down the mystery of entanglement and elucidated the rationale behind the Bell-CHSH inequality and its adoption in the E91 protocol.

Employing these protocols will ensure the secure distribution of the key. It is secure in the sense that the eavesdropper will never go undetected. Finally, we simulate the intercept attack by an eavesdropper and demonstrate how it can be detected in both the protocols thereby allowing us to abort the mission.

1 Introduction to Cryptography

Key distribution is the most crucial facet of cryptography. To understand it, we begin with an introduction to cryptography and the processes involved in communication.
The following are the steps involved in the process of communication:

Information → source compression → channel encoding → encryption → decryption → channel decoding → source decompression → information.

Cryptography concentrates on encryption and decryption. The first question we ask is what is the need for encryption and decryption? To answer this, let's take an example. Suppose Alice wants to share a one-time password '010' to Bob, she first encodes this as part of the error correction process using a repetition code 000111000, this is done because the channel may introduce some errors such as bit-flips or erasures. If she sent 010 and the channel introduced a bit-flip to the last bit, the message is changed to 011 which is not what she intended to communicate to Bob. To avoid this, she uses a 3-bit repetition code and encodes the message 010 to 000111000; this is the channel-encoded message. Even if the channel introduced bit-flips, Bob can use a majority vote decoder to decode the message correctly.

However, it does not suffice to just send the channel-encoded message. Not only are we required to protect the message from errors introduced by the channel, but we are also required to protect the message from falling into the hands of an eavesdropper who goes by the name Eve. If Alice tried to send 000111000 to Bob and it fell into the hands of Eve. Eve could, in principle, understand that Alice was using a repetition code to send the message 010 and easily misuse the obtained information and therefore indulge in unethical means.

To protect the message from Eve, we need to convert the message into a form that cannot be understood by Eve even if she accessed it. It is important to note that the access to the message may not be prevented, but surely it can be ensured that the accessed message makes no sense to Eve thus rendering it useless for exploitation.

This process of converting a meaningful message into what seems gibberish to Eve is called Encryption and this happens at Alice's end. Similarly, the process of converting the received gibberish into a meaningful message is called decryption. The practice and study of techniques associated with encryption and decryption are called Cryptography.

All of this can be thought of as a lock and key mechanism. To encrypt the message, Alice uses a key to lock the message. Now that the message is encrypted/locked, it is secure. To decrypt the encrypted/locked message, Bob also uses a key. Depending on whether Alice and Bob use the same key or a different key, Cryptography is classified into two different types, namely symmetric cryptography (private key cryptography) and Asymmetric key cryptography (public key cryptography).

1.1 Symmetric Cryptography

In symmetric cryptography, both Alice and Bob use the same cryptographic keys for both encryption and decryption. This can be thought of as a safe, where the message is locked by Alice with a key. Bob in turn uses a copy of this key to unlock the safe. For example, if Alice wants to send a message 000111000 to Bob, she uses a randomly generated key e.g. k=110010100. A copy of this key has to be distributed to Bob so that he could use this for unlocking. Now, Alice simply adds each bit of the message with the corresponding bit of the randomly generated key to obtain S = 000111000+110010100 = 110101100 (binary addition modulo 2 without carrying). This is the locked message or encrypted message or gibberish. It is then sent to Bob, who decrypts the message by subtracting the key and obtains m = s - k = 110101100 - 110010100 = 000111000, which is the

desired message. Upon successful distribution of the copy of the key, it is ensured that only Alice and Bob know the keys and the whole process is as safe as anything could be. Although perfectly secure, the problem is that Alice and Bob need to possess a common secret key; this cannot happen without the prior distribution of the key. During the cold war, the communication between Russia and America was happening with a prior distribution of the key through the embassies as they could not risk sharing it through an insecure channel. Implementing secure communication for the masses using this technology is economically not viable. Because of this reason, a new type of cryptography called asymmetric cryptography was proposed which is still utilised to this day.

1.2 Asymmetric Cryptography

Here, Alice and Bob use different keys for encryption and decryption. This pair consists of a public key (which may be known to others) and a private key (which may not be known by anyone except the owner). The first implementation was done using the RSA and is still widely used. We will show an example of how the RSA works to better understand this scheme Say Alice wants to send a message to Bob. To do this securely, Bob prepares some public and private keys. The public key allows anyone to send him an encrypted message (i.e ciphertext) and the private key allows only Bob to decrypt the ciphertext into meaningful plaintext. To do this, Bob begins by choosing two distinct prime numbers 'p' and 'q'. For example, say p=17 and q=41. Bob keeps these two numbers secret but reveals their product,$n = p \times q = 17 \times 41 = 697$, and sends it to Alice. Bob then computes the $\phi = (p-1)(q-1) = 16 \times 40 = 640$, which Bob keeps secret. Then he tries to find an integer e such that $1 < e < \phi$ and e are coprime to ϕ, meaning e and ϕ have the greatest common divisor equal to 1 which can be written as gcd(e, ϕ)=1. We can use Euclid's algorithm to find an e such that gcd(e,ϕ) = 1. Bob has to select one of the many values of 'e' which satisfy the above-mentioned conditions. Say Bob selects e = 3 which is less than ϕ and co-prime of ϕ. He publishes the numbers 'e' which can be used as a public key by Alice to send a message to Bob. Finally, Bob computes $d = e^{-1} \ mod \ \phi = 3^{-1} \ mod \ 640 = 427$, where the modulo operation(mod) returns the remainder of a division example q mod w returns the remainder which we get while dividing q by w. Here d is the modular multiplicative inverse of e mod ϕ. From number theory, it is assured that d always exists as e was chosen coprime to ϕ. Hence, We should get 1 as a result of $(d \times e \ mod \ \phi) \ mod \ \phi = ((e^{-1} \ mod \ \phi) \times (e \ mod \ \phi)) \ mod \ \phi = (e^{-1} \times e) \ mod \ \phi \ = 1 \ mod \ \phi = 1$

The number d is Bob's private key, so he keeps it a secret.d will allow him to decrypt messages sent to him. Now, let's see how public keys 'e' and 'n' are used to encrypt the message, and private keys 'd' and 'n' are used to decrypt it. Say Alice has to send A=2 as a message to Bob. She computes the ciphertext $C = A^e \ mod \ n = 2^3 \ mod \ 697 = 8$ and she sends it to Bob. When Bob receives the ciphertext C, he computes $C^d \ mod \ n = A^{ed} \ mod \ n = A^1 \ mod \ n = A$, thus receiving the message A. Continuing our example, Bob computes $8^{427} \ mod \ 697 = 2$, which is the message Alice wants to send to Bob. Decryption is hard for an eavesdropper(Eve) because she does not know the secret key d. She only knows Bob's public keys, n and e. To find the secret key $d = e^{-1} \ mod \ \phi$, Eve needs to know $\phi = (p-1)(q-1)$, but this requires knowing p and q, which involves factoring n. If we choose p and q to be large prime numbers, then there is no known efficient, classical algorithm for factoring large numbers, although a quantum computer can efficiently factor numbers using Shor's algorithm.

2 Quantum Computers: A threat to asymmetric cryptography

If we have a large number (N) which is the product of two prime numbers, factoring such a number to find the original two numbers is a tedious guess even for the best of classical supercomputers.

As the size of the number (N) gets bigger it reaches a point where the time required by any currently existing classical algorithm to find such factors is unreal. This fact is employed in the RSA encryption protocol described above to encrypt data today in such a way that decrypting the data requires prime factors of a large number.

Our current best method to find such factors is essentially to guess a number that might be a factor and then check if it is indeed a factor, and if it isn't, we repeat the process until we find the factor. This is called the Brute-force method. There are clever ways to make better guesses, however, even with those, the process is simply too slow. Hence a sufficiently large number (N) becomes impossible to factor in any reasonable amount of time.

Shor's algorithm (using quantum superposition and entanglement) exponentially decreases the time required to find the factor of large numbers. Roughly speaking, Shor's algorithm starts with a random guess that might share a factor with the target number, and then the algorithm transforms it into a much better guess that has a considerably higher probability of sharing a factor with the target number and the process is repeated with better guesses until the required factor is found.

It can be run on a classical computer as well, however, it is the process of turning a bad guess into a better one that requires a lot of processing power.

We start with a number N which is a product of two unknown numbers and make a guess g which might share a factor with N. Now for any pair of the whole number (here g, N) $= g^p = mN + 1$(for some number p and m) $\rightarrow g^p - 1 = mN \rightarrow (g^{p/2} + 1)(g^{p/2} - 1) = mN$

So, now we have two factors of N. As simple as this process might appear there are a few problems and the most prominent one is how to find p. Trying to simply guess the number p brings us back to our original problem. This is why we need Quantum computers to solve this problem efficiently.

Here since we are dealing with multiples of N rather than N itself the factors on the left-hand side might be multiples of factors of N rather than the factors themselves. But in such cases, we can find shared factors using Euclid's algorithm. But now there are a few problems with these new guesses. Firstly one of the new guesses might itself be a multiple of N then the other would be a factor of m therefore neither shall be useful. Secondly, the power P might come out to be an odd number in which case p/2 is not a whole number and therefore the guess raised to a fraction probably also is not a whole number which is not what we are looking for, we require integers to solve the problem. Lastly, the biggest problem is finding the value of p in the first place. For a classical computer, the job of finding the power p takes a lot of energy and time. In the case of a significantly long number, the process of finding p might require more time and effort than finding the factors by brute force calculations in the first place.

This is where Quantum computers provide an exponentially greater advantage. Quantum computers, unlike their classical counterparts, can simultaneously calculate multiple results for a single input by using Quantum superposition. In Quantum computers the work is based on superpositions that calculate all possible answers at once, while the probabilities of all the wrong answers, represented by states of the qubits interfere destructively and the correct ones interfere constructively, which is exactly what Shor's algorithm does.

The key to fast and reliable results is to set up a quantum superposition to calculate all possible answers simultaneously while the wrong answers destructively interfere with each other. Therefore, when we actually measure the output we are most likely to get the correct answer. Shor's algorithm does this job in a quantum computer for finding the power p.

To begin, the Quantum system is set up in such a way that it takes the number $|x\rangle$ as input and raises the guess g to the power of x. The system keeps track of both $|x, g^x\rangle$. Then the system compares the result to m.N, calculating the difference. We take this difference as the remainder and write $|x, +r\rangle$ for some value of r. A quantum computer takes multiple inputs of x and does the calculation for each case simultaneously. Thus gaining a crucial advantage over classical computers that can only take one input at a time. However to exploit the Quantum mechanical properties we

exploit the following mathematical fact to make the guesses better:

$$g^x = m_1.N + r$$

$$g^{x+p} = m_2.N + r$$

Given that Quantum computers pose a problem for asymmetric cryptography, we go back to symmetric cryptography which is secure. But the problem here as discussed earlier was the key distribution through a public channel. To circumvent this problem, we again use the principles of Quantum Mechanics to securely distribute the key through a public channel. Let's see how this is done.

3 Quantum Key Distribution

The main vulnerability of symmetric key cryptography is that an untrusted channel/medium should be used to share the secret symmetric keys. Since the medium is untrusted, it is highly susceptible to attacks by eavesdroppers. From here, the most important question arises, that is, how to securely distribute symmetric keys between two parties?

The solution to this key distribution problem is the quantum key distribution (QKD). It is a cryptographic technique that offers security that is guided and highly guaranteed by the laws of physics. It provides the means to distribute secret keys that can be used with quantum-safe symmetric algorithms like Advanced encryption standard (AES) or one-time pad encryption. QKD involves the encoding of information in the quantum states of photons which follows some notions from quantum physics known as key principles of QKD.

3.1 Principles used in QKD

3.1.1 Uncertainty Principle

In quantum mechanics, the act of measurement is one of the most essential aspects. This uncertainty principle tells us that if a measurement is done on a quantum observable of the quantum system, then it will automatically change the other properties of the system. This immediately leads us to the result that it is impossible to measure the values of two non-commuting observables for a quantum system simultaneously.

Let's say Alice sends the information/quantum states in encoded messages to Bob via a classical communication channel. If an eavesdropper tries to decode the message by doing the quantum measurement, this will change Bob's output state.

That means, this principle ensures that no eavesdropper can perform a measurement that keeps the quantum state undisturbed.

3.1.2 Non-orthogonality Principle

Consider a two-state quantum system, $|\psi_1\rangle$and $|\psi_2\rangle$. Here

$$|\psi_1\rangle = a|0\rangle + b|1\rangle \;\; where \;\; |a^2| + |b^2| = 1$$

$$|\psi_2\rangle = c|0\rangle + d|1\rangle \;\; where \;\; |c^2| + |d^2| = 1$$

The non-orthogonality principle says that if $|\psi_1\rangle$ and $|\psi_2\rangle$ are non-orthogonal, that is, then it is impossible to distinguish between these two states.

Consider the 1st case where x_0 and x_1 encoded in bits 0 and 1 respectively are orthogonal, hence the measurement can distinguish between them easily. Unlike the 1st case, in the 2nd case, the

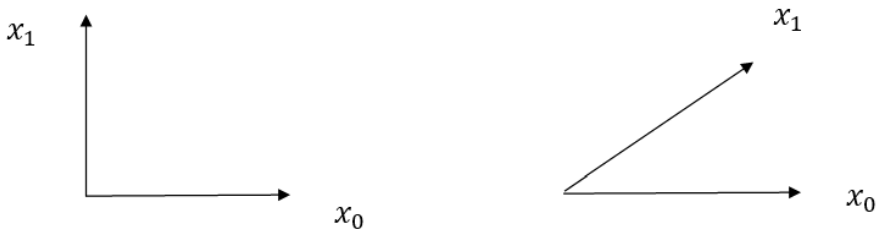

Figure 1: Quantum states

bits 0 and 1 are non-orthogonal. So, there exists no measurement which can distinguish these two states.

This feature is highly useful in Quantum Key Distribution. Because, if the bits 0 and 1 are encoded by non-orthogonal states in the message, then Eve cannot decode the message without making errors.

3.1.3 Authentication Channel

QKD includes the quantum channel to send qubits and a classical channel to figure out the message from the quantum state exchange. But the problem is that the classical channel is a public channel where the message can be heard by the eavesdropper. Hence, the classical channel needs to be authenticated to prevent a man-in-middle attack. In order to solve this issue, an authentication algorithm is used to authenticate Alice and Bob. An authentication algorithm(symmetric in nature) with a pre-shared key is used to authenticate Alice and Bob.

3.1.4 No Cloning Theorem

This no-cloning theorem states that it is impossible to perfectly clone an unknown/arbitrary qubit state. That is if we try to clone the state onto the state to obtain, then quantum mechanically it is impossible.

This is one of the important aspects used in QKD. Because this forbids the eavesdropper to make exact copies of the quantum signal. However, the eavesdropper can make approximate or imperfect copies of the quantum state / gain some information about the quantum channel. Hence, Alice and Bob need to erase all the partial information to ensure that they obtain a perfectly-secure key.

3.1.5 Entanglement

It is a phenomenon that occurs when a system of multiple Quantum particles interact in such a way that the particles cannot be described as independent systems but only as one system as a whole. They are entangled in the sense that when a particular property is measured in one particle, the opposite state will be observed on the other particle instantaneously. This is true regardless of the distance between the particles. It is impossible, however, to predict prior to measurement what state will be observed upon the measurement on either of the particles. We will discuss this phenomenon and its connection to the CHSH inequality in greater detail in section-5 as it forms the basis for the E91 protocol.

4 BB84

4.1 The Protocol

BB84 is a QKD Protocol based on the quantum phenomenon of superposition. Superposition is the ability of a quantum system to be in multiple states at the same time until it is measured. Here, Alice sends qubits to Bob through a quantum channel. If Eve tries to eavesdrop on the channel, Alice and Bob have a process by which they will be able to detect the presence of Eve in the channel. Hence they will discard the transmitted key and try again to evade Eve.

Steps:

1. Alice randomly generates a pair of classical bit strings a and b. According to these realised classical bits, she creates the qubits.

2. The classical bit b determines the basis of encoding. When it is 0, the qubit is encoded in the z basis and when it is 1, the qubit is encoded in the x basis. The classical bit a denotes the encoded state. When it is 0, the qubit is encoded in the state corresponding to the eigenvalue 1 and when it is 1, the qubit is encoded in the state corresponding to the eigenvalue -1. All of this is tabulated below:

a	b	encoded qubit	e.value
0	0	$\|0\rangle$	$+1$
0	1	$\|+\rangle$	$+1$
1	0	$\|1\rangle$	-1
1	1	$\|-\rangle$	-1

3. These two bits are generated together for the preparation of every qubit. Alice then sends these prepared qubits to Bob through a quantum channel.

4. At this point, Bob doesn't know anything about the qubit state or the basis.

5. Bob generates his own classical bit b' randomly. This bit determines the basis of measurement by Bob. When it is 0, he measures the qubit in the z basis and when it is 1, he measures it in the x basis. He generates another bit a' according to the result of these measurements which are eigenvalues +1 or -1, when the eigenvalue is +1, he assigns a' = 0 and when it is -1, he assigns a' = 1.

6. These two bits are generated by Bob before and after every measurement of the qubit.

7. After all the measurements, Bob announces through a public channel that he has completed the measurements. This is when Alice and Bob share their bit string b and b' with each other.

8. They discard the results a and a' whenever b is not equal to b' and retain the results a and a' whenever b = b'. The idea employed here is that when b = b', Bob used the same basis as Alice and his measurement of the qubit didn't alter its state and consequently, the eigenvalue he got corresponds to the eigenstate in which Alice prepared. This ensures that a' = a and that's why we retain the results. This way they have generated the key successfully.

4.2 Detection of Intercept and Resend Attack

At this point, Alice and Bob cannot be completely sure that they have the same key because Eve could have intercepted the qubit and changed the state of the qubit.

9. To be certain that Eve did not intercept along the way, Alice and Bob reveal a fraction of their shared secret key. For eg: If Alice and Bob want 256 bits in their shared key, they can generate 306 bits and reveal 50 of them. What is the probability that they detect Eve's interception, if Eve is measuring every qubit along the way?

10. To answer this, let us start by revealing one bit. Say Alice and Bob reveal a bit where both of them used the Z basis and Alice sent a qubit in the state $|0\rangle$. Then Alice reveals that her bit is 1, while Bob could reveal that his bit is 0 or 1, the probabilities of these outcomes are shown in the following diagram.

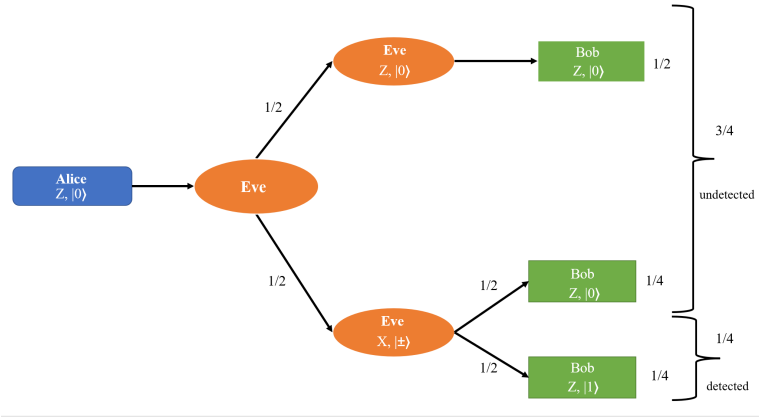

Figure 2: Probability of detecting Eve

11. There's a 75 percent chance that both have bit 1 as their bits and a 25 percent chance that Alice has 1 and Bob has 0. If they reveal n bits of their shared secret key, the probability that eve is undetected for all n is $(\frac{3}{4})^n$ while the probability that Eve is detected is $1-(\frac{3}{4})^n$

12. If they reveal 50 bits of their shared secret key, the probability that they detect eve is $1-(\frac{3}{4})^{50} = 0.99$ which is very close to certainty.

4.3 Simulation of the attack

4.3.1 Flowchart

The serial numbers below correspond to the numbers on the flowchart.

1. Alice asks Quantum Medium for the length of the Key.

2. Alice sends that many numbers of qubits to the quantum medium.

   (a) Eve can ask the quantum medium for the length of the Key and send those many bases to Quantum Medium to be read.

3. Bob asks Quantum Medium to get the length of the Key to be read.

4. Bob sends those many choices of bases to the quantum medium. The quantum medium measures the qubits in the desired basis and returns the results to Bob.

5. Alice sends her bases to the Classical Medium, and in return gets Bob's bases.

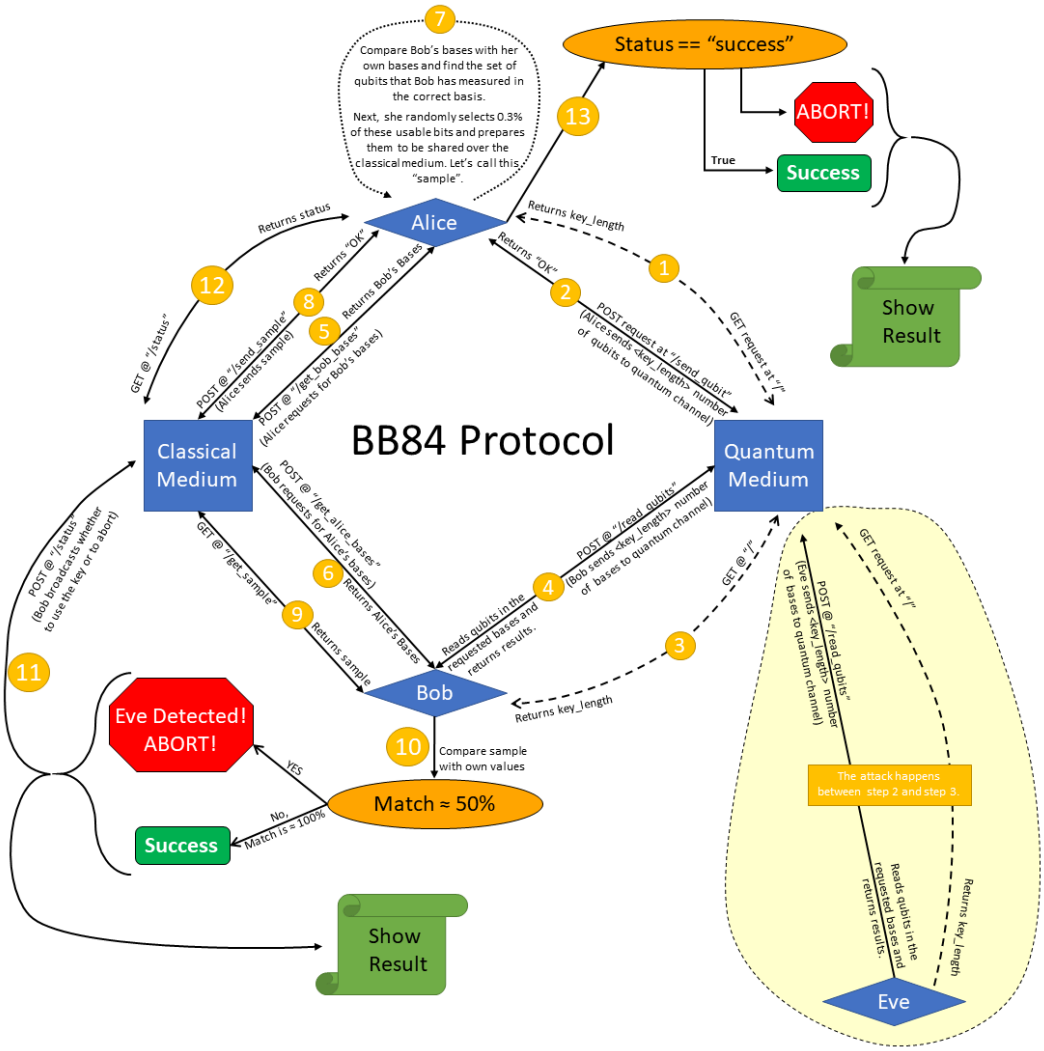

Figure 3: Flow Chart of BB84 protocol with attack

6. Bob also sends his bases and in return gets Alice's bases. (Note: suppose Alice requests the server for Bob's bases by giving her bases, but Bob hasn't given his bases to the classical medium yet, then classical medium makes Alice wait till Bob sends his bases)

7. Alice randomly chooses a part of the bits associated with the qubits which have been read by Bob in the correct basis and makes a Sample of base-value pairs.

8. Alice sends the Sample to Classical Medium.

9. Bob gets the Sample from Classical Medium.

10. Bob matches his own bits with the Sample of Alice's.

11. Bob decides whether to Abort or to Use the Key. He tells his decision to the Classical Medium

12. Alice gets to know about the decision made by Bob.

13. Alice acts accordingly.

4.3.2 Code

Simulation of BB84 Protocol using Flask and Qiskit modules of Python. We used these 5 files for this purpose.

- quantumMedium.py: This is the quantum medium. It is responsible for measuring the qubits in the basis desired by the parties and also retuning the results to the respective parties.

```
from flask import Flask, request
from qiskit import *
from tqdm import tqdm

key_length = 2500
qubits = None
comp = Aer.get_backend("qasm_simulator")

def prepare(inp):
    """Prepares the qubits and returns a list of key_length quantum
   circuits."""
    bits = inp["bits"]
    bases = inp["bases"]
    message = []
    for i in tqdm(range(key_length)):
        qc = QuantumCircuit(1,1)
        if int(bits[i]): qc.x(0)
        if bases[i] == "*": qc.h(0)
        qc.barrier()
        message.append(qc)
    return message

def measure1(qc, basis):
    """Measures the supplied qubit in the desired basis and returns the
   result."""
    if basis == "*":
        qc.h(0)
    elif basis != "+":
        raise ValueError('"basis" must be "+" or "*"!')
    qc.measure(0, 0)
    results = list(execute(qc, comp, shots = 1).result().get_counts().keys
   ())[0]
```

```
    # if random() > 0.85: results = str(1-int(results))  # adding errors
    return results

def measure(qubits, mbasis):
    """takes all the qubits and all the bases and returns a string
    containing measurements."""
    if len(qubits) != len(mbasis):
        raise ValueError(f"length of qubits (= {len(qubits)}) and length of
    mbasis (= {len(mbasis)}) aren't equal")
    ret = ""
    for i in tqdm(range(len(mbasis))):
        qc = qubits[i]
        basis = mbasis[i]
        ret += measure1(qc, basis)
    return ret

app = Flask(__name__)

@app.route("/")
def login():
    """Anyone can know the key-length by raising a request to this endpoint
    ."""
    print(f"someone asks for the key-length, Quantum Medium return {
    key_length}")
    return str(key_length)

@app.route("/send_qubit", methods=["POST"])
def sendqubit():
    """
    Alice will send a dictionary of the form:
    data = {
            "bits": bits,  # key_length long string of 0s and 1s
            "bases": bases,  # key_length long string of + and *
            "name": name  # here name = "Alice"
            }

    this function will generate quantum circuits from then and store them
    in a list
    """
    global qubits
    name = request.form["name"]
    if name == "alice":
        print(f"\nAlice sends all the {key_length} qubit")
        qubits = prepare(request.form)
        return "Qubit Added"
    else:
        return f"{name.title()} is not authorised to send qubits!"

@app.route("/read_qubits", methods=["POST"])
def getqubit():
    """anyone can send a request along with a choice of bases at this
    endpoint to get the qubits read in that base."""
    if not qubits:
        print("\nSomeone sends bases and tries to measure the qubits. But
    Alice hasn't sent the Qubits yet! So he waits!")
        return "Wait"
    print("\nSomeone measures the qubits in his/her choice of bases.")
    bases = request.form["basis"]
    return measure(qubits, bases)

```

```
app.run(
    host = "localhost",  # change this to your local ip
  port = 5050,
  debug = True,
 )
```

- classicalMedium.py: This is the classical medium through which the 3 three parties (Alice, Bob, and eve) will communicate after the qubits are measured by the quantum medium.

```
from flask import Flask, request

app = Flask(__name__)

# Defining different values
bob_bases, alice_bases = None, None
sample = None
status = None

@app.route('/get_bob_bases', methods=['POST'])
def get_bob_bases():
    """Alice sends her bases and requests for Bob's bases.
    If Bob has already sent his bases, it returns the bases,
    otherwise, it returns "wait", so that Alice waits for
    some time and comes back again to ask for Bob's bases."""
    global alice_bases
    alice_bases = request.form['bases']
    print(f"\nAlice sent her bases and is requesting for Bob's bases.", end
    = " ")
    if bob_bases:
        print("The Classical Medium already has Bob's bases, so it gives it
    to her.")
        return bob_bases
    print("But Bob hasn't sent his bases yet, so she waits.")
    return "wait"

@app.route('/get_alice_bases', methods=['POST'])
def get_alice_bases():
    """Similarly Bob sends his bases and requests for Alice's bases.
    If Bob has already sent his bases, it returns the bases, else returns "
   wait"."""
    global bob_bases
    bob_bases = request.form['bases']
    print(f"\nBob sent his bases and is requesting for Alice's bases.", end
    = " ")
    if alice_bases:
        print("The Classical Medium already has Alice's bases, so it gives
   it to him.")
        return alice_bases
    print("But Alice hasn't sent her bases yet, so he waits.")
    return "wait"

@app.route("/send_sample", methods=["POST"])
def send_sample():
    """Alice sends her sample."""
    global sample
    print(f"\nAlice sent her sample.")
    sample = request.form["sample"]
    return "Got it!"

@app.route("/get_sample")
```

```
def get_sample():
    """Bob requests for Alice's sample
    If Alice has sent her sample, it returns the sample, else returns "wait
    "."""
    print(f"\nBob is requesting for Alice's sample.", end = " ")
    if sample:
        print("The Classical Medium already has Alice's sample, so it gives
    it to him.")
        return sample
    print("But Alice hasn't sent her sample yet, so he waits.")
    return "wait"

@app.route("/status", methods=["GET", "POST"])
def get_status():
    """Bob after knowing Alice's sample, calculates the match factor.
    Depending upon the match factor he either decides to keep the key or
    not.
    He declares his decision in the classical Medium.
    Later Alice requests for the decision."""
    global status
    if request.method == "GET":
        print(f"\nAlice is requesting for Bob's decision on whether to
    proceed with the key or start the process of resending another key.")
        if status != None:
            print("The Classical Medium already has Bob's decision, so it
    gives it to her.")
            return status
        print("But Bob hasn't sent his decision yet, so she waits.")
        return "wait"
    else:
        print(f"\nBob is sending his decision on whether to proceed with
    the key or start the process of resending another key.")
        status = request.form["status"]
        return "done"

app.run(
    host = "localhost",  # change this to your local ip
  port = 5000,
  debug = True,
 )
```

- alice.py: This is Alice. She uses this code to carry out the communication with/through the two above-mentioned media.

```
from random import choice as IDK
from random import sample as take_any
import requests
from time import sleep

name = "alice"
print(f"Hi! This is {name.title()}!")

qIP, qPORT = "localhost", "5050"
cIP, cPORT = "localhost", "5000"
quantumMedium   =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel

def prepare(key_length):
    """prepares key_length number of random choices for qubits
    to be prepared in. a denotes the values of the qubits and b
```

```
    denotes the corresponding bases to be used to encode the qubit"""
    a = ""  # X or not
    b = ""  # H or not
    for i in range(key_length):
        a += IDK(["0", "1"])
        b += IDK(["+", "*"])
    return a, b

def sendQubit(key_length):
    """gets the qubit preparation choices and sends them to quantum channel
    """
    bits, bases = prepare(key_length)
    requests.post(f"{quantumMedium}send_qubit",
                  data = {
                      "bits": bits,
                      "bases": bases,
                      "name": name
                      }
                  ).text
    return bits, bases

def get_bob_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_bases", data = {"
    bases": bases}).text
    except:
        sleep(key_length/120)
        return get_bob_bases(bases)
    if  ret == "wait":
        sleep(key_length/120)
        return get_bob_bases(bases)
    return ret

def send_sample(usable_bits, n = 0.3):
    """prepares and sends a sample of the qubits to quantum channel"""
    sample_indices = sorted(
        take_any(usable_bits, int((key_length/2)*n))
    )
    print("sample length =", len(sample_indices))
    sampleD = {x: bits[x] for x in sample_indices}

    # Encoding it before sending
    sample = ""
    for index, bit in sampleD.items():
        sample += f"{index}:{bit}-"
    sample = sample[:-1]
    # sending
    requests.post(f"{classicalMedium}send_sample", data = {"sample": sample
    }).text

    return sampleD

def compile_key(bits, sample, usable_bits):
    """compiles the key from the usable bits and discarding the sample"""
    use = [i for i in range(key_length) if i in usable_bits and i not in
    sample.keys()]
    ret = ""
    for i in use:
        ret += bits[i]
    return ret
```

```

def successful():
    try:
        ret = requests.get(f"{classicalMedium}status").text
    except:
        sleep(5)
        return successful()
    if  ret == "wait":
        sleep(5)
        return successful()
    if ret == "success":
        return True
    return False



# Know the key length
key_length = int(requests.get(quantumMedium).text)

# Send those many qubit preparation choices to quantum channel.
# Quantum channel will prepare them as requested
bits, alice_bases = sendQubit(key_length)

# Get the bases of Bob
bob_bases = get_bob_bases(alice_bases)

# know which qubits have been read by Bob in the correct basis
# to calculate which qubits are usable and which aren't
usable_bits = [i for i in range(key_length) if bob_bases[i] == alice_bases[
    i]]

# Send a sample of the usable qubits to quantum channel
sample = send_sample(usable_bits, n = 0.3)

# If successful, compile the key, else abort the mission
if successful():
    key = compile_key(bits, sample, usable_bits)
    print(f"key = {key}")
else:
    print("Eve Detected, mission abort!")
```

- bob.py : Similarly this is the code for Bob.

```
import requests
from random import choice as IDK
from time import sleep

name = "bob"
print(f"Hi! This is {name.title()}!")

qIP, qPORT = "localhost", "5050"
cIP, cPORT = "localhost", "5000"
quantumMedium  =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel

def readQubit():
    bases = ""
    for i in range(key_length):
        bases += IDK(["+", "*"])
    # print(basis)
```

```
18      bits = requests.post(f"{quantumMedium}read_qubits", data = {"basis":
    bases}).text
19      if bits == "Wait":
20          print("waiting...")
21          sleep(1)
22          return readQubit()
23      return [bits, bases]
24
25  def get_alice_bases(bases):
26      try:
27          ret = requests.post(f"{classicalMedium}get_alice_bases", data = {"
    bases": bases}).text
28      except:
29          sleep(1)
30          return get_alice_bases(bases)
31      if  ret == "wait":
32          sleep(1)
33          return get_alice_bases(bases)
34      return ret
35
36  def get_sample():
37      try:
38          ret = requests.get(f"{classicalMedium}get_sample").text
39      except:
40          sleep(1)
41          return get_sample()
42      if  ret == "wait":
43          sleep(1)
44          return get_sample()
45      return ret
46
47  def process_sample(sample):
48      ss = sample.split("-")
49      ret = {}
50      for s in ss:
51          a, b = s.split(":")
52          ret[int(a)] = int(b)
53      return ret
54
55  def compile_key(bits, sample, usable_bits):
56      use = [i for i in range(key_length) if i in usable_bits and i not in
    sample.keys()]
57      ret = ""
58      for i in use:
59          ret += bits[i]
60      return ret
61
62  key_length = int(requests.get(quantumMedium).text)
63
64  bits, bob_bases = readQubit()
65
66  alice_bases = get_alice_bases(bob_bases)
67
68
69  usable_bits = [i for i in range(key_length) if bob_bases[i] == alice_bases[
    i]]
70
71  sample = process_sample(get_sample())
72
73  check = [sample[i] == int(bits[i]) for i in sample.keys()]
```

```
74
75 print(f"match = {sum(check)/len(check)*100}%")
76
77 if sum(check)/len(check)>=(0.9-0.05):
78     requests.post(f"{classicalMedium}status", data = {"status": "success"})
79     print("Success")
80     key = compile_key(bits, sample, usable_bits)
81     print(f"key length = {len(key)}")
82     print(f"key = {key}")
83 else:
84     requests.post(f"{classicalMedium}status", data = {"status": "abort"})
85     print("Eve Detected, mission abort!")
```

- eve.py: Similarly this is the code for Eve.

```
1  import requests
2  from random import choice as IDK
3  from time import sleep
4
5  name = "eve"
6  print(f"Hi! This is {name.title()}!")
7
8  qIP = "localhost"
9  cIP = "localhost"
10 quantumMedium  =  f"http://{qIP}:5050/"  # quantum channel in port 5050
11 classicalMedium = f"http://{cIP}:5000/"  # classical channel in port 5000
12
13 def readQubit():
14     bases = ""
15     for i in range(key_length):
16         bases += IDK(["+", "*"])
17     # print(basis)
18     bits = requests.post(f"{quantumMedium}read_qubits", data = {"basis":
       bases}).text
19     if bits == "Wait":
20         print("waiting...")
21         sleep(1)
22         return readQubit()
23     return [bits, bases]
24
25 key_length = int(requests.get(quantumMedium).text)
26
27 bits, bob_bases = readQubit()
28
29 print("completed eavesdropping!")
```

4.3.3 Output

The output can be viewed in the README.md file on GitHub.

Without Eve's Interception:

"quantumMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\quantumMedium.py
 * Serving Flask app 'quantumMedium' (lazy loading)
 * Environment: production
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5050/ (Press CTRL+C to quit)
someone asks for the key-length, Quantum Medium return 2500
127.0.0.1 - - [19/Jul/2022 10:07:55] "GET / HTTP/1.1" 200 -
someone asks for the key-length, Quantum Medium return 2500
127.0.0.1 - - [19/Jul/2022 10:07:55] "GET / HTTP/1.1" 200 -

Alice sends all the 2500 qubit
100%|████████████████████████████| 2500/2500 [00:00<00:00, 8338.95it/s]
127.0.0.1 - - [19/Jul/2022 10:07:57] "POST /send_qubit HTTP/1.1" 200 -

Someone measures the qubits in his/her choice of bases.
100%|████████████████████████████| 2500/2500 [00:21<00:00, 116.72it/s]
127.0.0.1 - - [19/Jul/2022 10:08:19] "POST /read_qubits HTTP/1.1" 200 -
```

"classicalMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\classicalMedium.py
 * Serving Flask app 'classicalMedium' (lazy loading)
 * Environment: production
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5000/ (Press CTRL+C to quit)

Alice sent her bases and is requesting for Bob's bases. But Bob hasn't sent his
bases yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:07:59] "POST /get_bob_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. The Classical Medium
already has Alice's bases, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:08:21] "POST /get_alice_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. The Classical Medium
already has Bob's bases, so it gives it to her.
```

```
127.0.0.1 - - [19/Jul/2022 10:08:22] "POST /get_bob_bases HTTP/1.1" 200 -

Bob is requesting for Alice's sample. But Alice hasn't sent her sample yet, so he
waits.
127.0.0.1 - - [19/Jul/2022 10:08:23] "GET /get_sample HTTP/1.1" 200 -

Alice sent her sample.
127.0.0.1 - - [19/Jul/2022 10:08:24] "POST /send_sample HTTP/1.1" 200 -

Bob is requesting for Alice's sample. The Classical Medium already has Alice's
sample, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:08:26] "GET /get_sample HTTP/1.1" 200 -

Alice is requesting for Bob's decision on whether to proceed with the key or start
the process of resending another key.
But Bob hasn't sent his decision yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:08:26] "GET /status HTTP/1.1" 200 -

Bob is sending his decision on whether to proceed with the key or start the
process of resending another key.
127.0.0.1 - - [19/Jul/2022 10:08:28] "POST /status HTTP/1.1" 200 -

Alice is requesting for Bob's decision on whether to proceed with the key or start
the process of resending another key.
The Classical Medium already has Bob's decision, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:08:33] "GET /status HTTP/1.1" 200 -
```

"alice.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\bob.py
Hi! This is Bob!
match = 100.0%
Success
key length = 929
key =
0100011010110111111100101110110100011110001001010100001101011001110111111111101000
0000101110101010011100010101011100110010111110000100110011010010110111100000100111
0001111100100110111110010011101100010110010100100111100100101110111011011101000111
0011010110111110100001001101101001101000100011000010100011011011100011101110011110
1000001101110000000010100111010000100110011111101110100011110010011111110111011011 0
0000011010100101101111010110110100100010000000101101001000000110001101011000101000
0100110010110001100000110101010011010111010101011110010000100110100111000111111111
1110011111101011110110111101010011111111111011101111101001001011011011001100111001
1100111100101000110100100111100001111111111001011011000000100101101011111010001100
1100000011011000000111100110110001011100100101011111110001110111011101101111111001
0000110111001100011100011100100001110110110010110000000100110101011111011100000000
1001110101110011111100001101
```

"bob.py":

```
Hi! This is Alice!
sample length = 375
key =
01000110101101111111100101110110100011110001001010100001101011001110111111111101000
00001011101010100111000101010111001100101111100001001100110100101101111000001001111
00011111001001101111100100111011000101100101001001111001001011101110110111010001111
00110101101111101000010011011010011010001000110000101000110110111000111011100111110
10000011011100000001010011101000010011001111101110100011110010011111110111011011O
00000110101001011011110101101101001000100000001011010010000001100011010110001010000
01001100101100011000001101010100110101110101010111100100001001100101110001111111111
11100111111010111101110111010100111111111111011101111101001001011011011001100111001
11001111001010001101001001111000011111111111001011011000000100101101011111010001100
11000000110110000001111001101100010111001001010111111100011101110111011011111111001
00001101110011000111000111001000011101101100101100000001001101010111110111001000000
10011101011100111110000110 1
```

With Eve's Interception:

"quantumMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\quantumMedium.py
 * Serving Flask app 'quantumMedium' (lazy loading)
 * Environment: production
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5050/ (Press CTRL+C to quit)
someone asks for the key-length, Quantum Medium return 2500
127.0.0.1 - - [19/Jul/2022 10:02:23] "GET / HTTP/1.1" 200 -

Alice sends all the 2500 qubit
100%|██████████████████████████| 2500/2500 [00:00<00:00, 7175.33it/s]
127.0.0.1 - - [19/Jul/2022 10:02:26] "POST /send_qubit HTTP/1.1" 200 -
someone asks for the key-length, Quantum Medium return 2500
127.0.0.1 - - [19/Jul/2022 10:02:31] "GET / HTTP/1.1" 200 -

Someone measures the qubits in his/her choice of bases.
100%|██████████████████████████| 2500/2500 [00:24<00:00, 101.28it/s]
127.0.0.1 - - [19/Jul/2022 10:02:58] "POST /read_qubits HTTP/1.1" 200 -
someone asks for the key-length, Quantum Medium return 2500
127.0.0.1 - - [19/Jul/2022 10:03:07] "GET / HTTP/1.1" 200 -

Someone measures the qubits in his/her choice of bases.
100%|██████████████████████████| 2500/2500 [00:23<00:00, 104.44it/s]
127.0.0.1 - - [19/Jul/2022 10:03:33] "POST /read_qubits HTTP/1.1" 200 -
```

"classicalMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\classicalMedium.py
 * Serving Flask app 'classicalMedium' (lazy loading)
 * Environment: production
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5000/ (Press CTRL+C to quit)

Alice sent her bases and is requesting for Bob's bases. But Bob hasn't sent his
bases yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:02:28] "POST /get_bob_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. But Bob hasn't sent his
bases yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:02:51] "POST /get_bob_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. But Bob hasn't sent his
bases yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:03:13] "POST /get_bob_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. The Classical Medium
already has Alice's bases, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:03:35] "POST /get_alice_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. The Classical Medium
already has Bob's bases, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:03:36] "POST /get_bob_bases HTTP/1.1" 200 -

Bob is requesting for Alice's sample. But Alice hasn't sent her sample yet, so he
waits.
127.0.0.1 - - [19/Jul/2022 10:03:37] "GET /get_sample HTTP/1.1" 200 -

Alice sent her sample.
127.0.0.1 - - [19/Jul/2022 10:03:38] "POST /send_sample HTTP/1.1" 200 -

Bob is requesting for Alice's sample. The Classical Medium already has Alice's
sample, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:03:40] "GET /get_sample HTTP/1.1" 200 -

Alice is requesting for Bob's decision on whether to proceed with the key or start
the process of resending another key.
But Bob hasn't sent his decision yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:03:40] "GET /status HTTP/1.1" 200 -

Bob is sending his decision on whether to proceed with the key or start the
process of resending another key.
127.0.0.1 - - [19/Jul/2022 10:03:42] "POST /status HTTP/1.1" 200 -

Alice is requesting for Bob's decision on whether to proceed with the key or start
the process of resending another key.
The Classical Medium already has Bob's decision, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:03:47] "GET /status HTTP/1.1" 200 -
```

"eve.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\eve.py
Hi! This is Eve!
completed eavesdropping!
```

"alice.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\alice.py
Hi! This is Alice!
sample length = 375
Eve Detected, mission abort!
```

"bob.py":

```
PS C:\Users\amukh\Desktop\QKD\BB84_1> python .\bob.py
Hi! This is Bob!
match = 59.46666666666667%
Eve Detected, mission abort!
```

5 On the EPR paradox and the motivation for Bell's Inequality

5.1 EPR's Argument

Einstein, Podolsky, and Rosen argued that Quantum mechanics could not be a complete theory and that it should be supplemented by additional variables. Their argument with the example advocated by Bohm and Aharonov is as follows:

Consider a neutral pion decaying into an electron and a positron. These two particles move in the opposite direction. On account of the conservation of Angular Momentum, which is sacrosanctum, it requires that the total spin of the combined electron-positron system is zero. This forms what many physicists believed known as a singlet entangled state which EPR was not happy about.

The measurements can be made on selected components of the spin of individual particles. Alice makes the measurement on the electron and Bob makes the measurement on the positron. We denote the $\boldsymbol{\sigma}$ a as the spin measurement operator on the electron in the direction and $\boldsymbol{\tau}$b as the spin measurement operator on the positron in the b direction. Notice that we use the symbols sigma and tau for the electron and positron respectively to keep the notation simple. Let MA be the measurement outcome by Alice and MB be the measurement outcome by Bob, the measurement outcome is always ± 1. If Alice performs the measurement $\boldsymbol{\sigma}$ a and the measurement outcome MA is +1, then it follows from the angular momentum principle that the measurement $\boldsymbol{\tau}$ a by Bob is -1 and vice-versa. The crucial assumption that EPR make here is the locality assumption, by this, they mean that, if the two measurements by Alice and Bob are made remotely, the outcome of Alice's measurement doesn't influence the outcome of Bob's.

Again from the conservation of Angular momentum, we could predict in advance the result of measurement of any chosen component of $\boldsymbol{\tau}$(positron's spin), by formerly measuring the same component of $\boldsymbol{\sigma}$(electron's spin). The word any is the key here. Since we can predict in advance the result of measurement of ANY chosen component of the positron's spin, it should follow that the result of these measurements was predetermined, meaning that both particles had well-defined spins in all directions from the time they were created. This is the element of the reality of each particle.

The idea that the electron and positron had well-defined spins all along from the time they were created is the realist's view which EPR subscribed to. But the orthodox view holds that neither particle had a well-defined spin until the act of measurement forced it to take a particular spin direction. It is in an entangled state.

The EPR's argument is that if the spins were not predetermined as the orthodox view claims and we made the measurements on electron and positron simultaneously when they are 100 lightyears away, it would imply that upon measuring the electron's spin along the direction, the result of the measurement was reaching the positron instantaneously thus producing the positron's spin measurement result in accordance with the conservation of angular momentum leading to the violation of the locality principle.

This instantaneous communication is what Einstein called preposterously. Now, owing to the postulate that no influence can propagate faster than the speed of light, let's just stick to the realist's view and say that both the particles had well-defined spins all along. Now, it follows that If the particles had predetermined spins all along from the time of their creation Quantum mechanics as a theory could not completely describe the state of either particle in the sense that all it talked about was superposition and probabilities and didn't definitely predict the outcomes of any measurement, it implies that there's a possibility of a more complete specification of the state. Their argument, therefore was that the Quantum theory is an incomplete theory, the wavefunction is an incomplete description of the reality, and there's some quantity $\boldsymbol{\lambda}$ in addition to ψ, that is required to specify

the state of the system completely.

$\boldsymbol{\lambda}$ the local hidden variable that represents the elements of reality associated with the spins. So, for each electron-positron system created, there's a particular $\boldsymbol{\lambda}$that is fixed and local to each particle and it contains in itself the information to yield the results of spin measurements in all the directions from the time the particles were created. Bell's genius lies in the fact that he came up with a relationship that proves that no local hidden variable theory can explain the results of the experiments. We will look at what exactly the experiments reveal.

Let's look a little more into why the orthodox physicists believed the description of reality was as complete as an entangled state is.

Charlie performs the Pion decay experiment and her role now is to distribute the electron and positron to Alice and Bob respectively. The wave function governing this composite system, according to the orthodox physicists is called the entangled singlet state and is given by

$$|\psi_{sing}\rangle = 1/\sqrt{2}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle$$

To them, this wave function is as complete a description of the combined system as it is possible to make.

These electron-positron pairs are generated several times, Alice performs the measurement in the z-direction, she obtains +1 in 50 percent of the trials and -1 in the other 50 percent of the trials, and the same goes with bob. We have observed that we can know nothing at all about the outcome of the individual measurement of any component of the spin when a pion decays into electron-positron pair.

How does mathematics express this fact using the entangled wave function? To see this, we calculate the expectation of $\sigma.\hat{z}$

$$\langle\Psi_{sing}|\sigma.\hat{z}\otimes I|\Psi_{sing}\rangle = \frac{1}{2}(\langle\uparrow\downarrow| - \langle\downarrow\uparrow|)|\sigma.\hat{z}\otimes I|(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle) = \frac{1}{2} + 0 + 0 - \frac{1}{2} = 0$$

This can be shown as $\sigma.\hat{z}, \sigma.\hat{y}, \sigma.\hat{x}$ or $\tau.\hat{z}, \tau.\hat{y}, \tau.\hat{x}$ or in general for any direction. The expectation value zero implies that the experimental outcome is equally likely to be +1 or -1, this very well agrees with the experiments that the outcome of an individual measurement is completely uncertain. So, we can know everything about this composite system - everything there is to know– that's the complete description of the wave function and we can still know nothing about its constituent parts, this is the weirdness of this electron-positron pair generated from a pion. This is what they called entanglement.

Knowing the exact state of the system, even if it is entangled, must tell us something. And in fact it does, when we consider the composite observables. Now let's say both Alice and Bob measure in the z direction simultaneously, the observable therefore is $\sigma.\hat{z}\otimes\tau.\hat{z}$, the expectation is now given by:

$$\langle\Psi_{sing}|\sigma.\hat{z}\otimes\tau.\hat{z}|\Psi_{sing}\rangle = \frac{1}{2}(\langle\uparrow\downarrow| - \langle\downarrow\uparrow|)|\sigma.\hat{z}\otimes\tau.\hat{z}|(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle) = \frac{1}{2}(\langle\uparrow\downarrow| - \langle\downarrow\uparrow|)|(-|\uparrow\downarrow\rangle + |\downarrow\uparrow\rangle) = -1$$

The expectation of the product of the measurement outcomes of Alice and Bob is called the correlation. This implies that when both of them measure the spin in the same direction z, they come back, compare their recorded outcomes in all the trials, and see that they have measured opposite values and therefore the product is always -1. Sometimes Alice measures +1 and Bob measures -1, other times vice-versa. Taking the expectation over several trials indeed gives -1 experimentally as well. This result is also correctly predicted by the entangled wave function as shown in the above calculation.

In fact, this is nothing surprising, Einstein would argue that this could be a result of a completely classical setup where the particles have a physical reality. It's as if Charlie prepared two particles,

one in spin up and the other spin down and distributed them to Alice and Bob in each trail without knowing which is which. Here as well, sometimes Alice gets +1 and Bob gets -1, other times vice-versa and therefore leads to the same correlation.

But then we come to something that has no classical analogy, instead of measuring, $\sigma.\hat{z} \otimes \tau.\hat{z}$, Alice and bob measure in the x-basis, $\sigma.\hat{x} \otimes \tau.\hat{x}$,

$$\langle \Psi_{sing} | \sigma.\hat{x} \otimes \tau.\hat{x} | \Psi_{sing} \rangle = \frac{1}{2}(\langle\uparrow\downarrow| - \langle\downarrow\uparrow|)|\sigma.\hat{x} \otimes \tau.\hat{x}|(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle) = \frac{1}{2}(\langle\uparrow\downarrow| - \langle\downarrow\uparrow|)|(|\downarrow\uparrow\rangle - |\uparrow\downarrow\rangle) = -1$$

This is such a startling result, nonetheless agrees with the experiments. Startling because in a classical setup, if a spin is measured in the z direction and it gives you +1 or -1, it means that the spin of the particle is aligned in the z direction and if we measured again in the x basis, a classical setup would straight out throw at us a zero for both Alice and Bob and the correlation would be zero. But what we measure experimentally is -1. This is the absolute weirdness of these particles, the orthodox opinion is not just another opinion, it is an informed opinion. At Least as long as we have not found any hidden variable that represents the element of physical reality which reproduces the quantum mechanical correlation and still satisfies the principle of locality.

In general, it can easily be derived from the entangled state using the mathematics of quantum mechanics that when Alice and Bob choose the directions arbitrarily, the correlation , $\langle M_a.M_b \rangle = -\hat{a}.\hat{b} = -cos(\theta)$ where θ is the angle between $\hat{a}$ and $\hat{b}$ direction which very well agrees with the experimental results. We set out to show the derivation.
When both of them measure in the same basis, they get opposite results, therefore

$$I \otimes \vec{\tau}.\hat{b} \, |\psi\rangle = -\vec{\sigma}.\hat{b} \otimes I \, |\psi\rangle$$

If Alice and Bob choose to measure in arbitrary directions $\hat{a}$ and $\hat{b}$ respectively, we have

$$\vec{\sigma}.\hat{a} \otimes \vec{\tau}.\hat{b} \, |\psi\rangle = \vec{\sigma}.\hat{a}.I \otimes \vec{\tau}.\hat{b} \, |\psi\rangle = -(\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \otimes I \, |\psi\rangle$$

Hence

$$\langle\psi| \, \vec{\sigma}.\hat{a} \otimes \vec{\sigma}.\hat{b} \, |\psi\rangle = - \langle\psi| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \otimes I \, |\psi\rangle$$

RHS

$$\begin{aligned}
&= -[(\frac{1}{\sqrt{2}} \langle\uparrow\downarrow| - \frac{1}{\sqrt{2}} \langle\downarrow\uparrow|)(\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \otimes I_B(\frac{1}{\sqrt{2}} |\uparrow\downarrow\rangle - \frac{1}{\sqrt{2}} |\downarrow\uparrow\rangle)] \\
&= -[\frac{1}{\sqrt{2}} \langle\uparrow\downarrow| - \frac{1}{\sqrt{2}} \langle\downarrow\uparrow| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b})(\frac{1}{\sqrt{2}} |\uparrow\downarrow\rangle - \frac{1}{\sqrt{2}} |\downarrow\uparrow\rangle)] \\
&= [-\frac{1}{2} \langle\uparrow\downarrow| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\uparrow\downarrow\rangle + \frac{1}{2} \langle\downarrow\uparrow| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\downarrow\uparrow\rangle] \\
&= [-\frac{1}{2} \langle\uparrow| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\uparrow\rangle \langle\downarrow \mid \downarrow\rangle + \frac{1}{2} \langle\downarrow| \, (\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\downarrow\rangle \langle\uparrow \mid \uparrow\rangle] \\
&= [-(\frac{1}{2}((\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b})\delta^A_{|\uparrow\rangle}) + (\frac{1}{2}((\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b})\delta^B_{|\downarrow\rangle})] \\
&= [-\frac{1}{2}Tr((\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\uparrow\rangle \langle\uparrow|) + \frac{1}{2}Tr((\vec{\sigma}.\hat{a})(\vec{\sigma}.\hat{b}) \, |\downarrow\rangle \langle\downarrow|)]
\end{aligned}$$

$$
\begin{aligned}
&= \sum_i \sum_j -\frac{1}{2} a_i b_j tr(\sigma_i \sigma_j) \\
&= \sum_i -\frac{1}{2} a_i b_1 tr(\sigma_i \sigma_1) - \frac{1}{2} a_i b_2 tr(\sigma_i \sigma_2) - \frac{1}{2} a_i b_3 tr(\sigma_i \sigma_3) \\
&= -a_1 b_1 - a_2 b_2 - a_3 b_3 \\
&= -\hat{a}.\hat{b} = -\cos\theta
\end{aligned}
$$

This correlation has been experimentally verified. To arrive at this correlation theoretically, we indeed used the entangled state. The entangled state, as argued by the EPR violates locality. So the question John Bell like everyone else asked is, would it be possible to arrive at this correlation with a local hidden variable model? Bell eventually answered the question; he came up with a theorem that proves any local hidden variable theory cannot reproduce this correlation. Therefore the entangled state is the complete description of reality.

5.2 CHSH Inequality

Consider the relation:

$$C = (M_A + M_{A'})M_B + (M_A - M_{A'})M_{B'}$$

Where M_A and $M_{A'}$ are the results of measurement by Alice in the A and A' direction respectively, M_B and $M_{B'}$ are the results of measurement by Bob in the B and B' direction respectively. These results can take values +1 or -1.

C can be rewritten as

$$C = M_A.M_B + M_{A'}.M_B + M_A.M_{B'} - M_{A'}.M_{B'}$$

Taking the average:

$$\langle C \rangle = \langle M_A.M_B \rangle + \langle M'_A.M_B \rangle + \langle M_A.M_{B'} \rangle - \langle M_{A'}.M_{B'} \rangle$$

The way we take the average is, for eg: Charlie conducts a million experiments thereby generating a million electron-positron pairs. These are distributed to Alice and Bob one by one for measurement. They randomly pick the measurement direction in each trial. After each trial, they note down their measurement direction and the result in their respective notebooks. After a million trials, they both come back, look at each other's notebooks to calculate the above 4 quantities. For eg: To calculate $< M_A.M_B >$, they count the number of trials where Alice picked A and Bob picked B, and then calculate the value $\langle M_A.M_B \rangle$ of all these trials, remember that M_A and M_B can take values from +1,-1, and therefore $M_A.M_B$ can take +1 or -1. Finally, they add the MA.MB value of all these trials and divide by the number of trials. We will try to upper bound $\langle C \rangle$ in a local hidden variable, according to it, the results of measurements in all these directions are predetermined for every electron-positron pair. Therefore these are the possible combinations of measurement results:-

M_A	$M_{A'}$	M_B	$M_{B'}$
-1	-1	-1	-1
-1	-1	-1	1
-1	-1	1	-1
-1	-1	1	1
-1	1	-1	-1
-1	1	-1	1
-1	1	1	-1
-1	1	1	1
1	-1	-1	-1
1	-1	-1	1
1	-1	1	-1
1	-1	1	1
1	1	-1	-1
1	1	-1	1
1	1	1	-1
1	1	1	1

Every electron pair released has one of the above configurations. When $M_A = M_{A'}$, $M_A + M_{A'} = 2$, this implies $|C| = |(M_A + M_{A'})M_B + (M_A - M_{A'})M_{B'}| = 2$. Similarly when $M_A = -M_{A'}, M_A + M_{A'} = 0$, this also implies $|C| = |(M_A + M_{A'})M_B + (M_A - M_{A'})M'_B| = 2$.

Therefore, in any local hidden variable theory, we can bound $|\langle C\rangle|$ as follows:

$$|\langle C\rangle| \leq \langle |C|\rangle = 2$$

$$|\langle C\rangle| = |\langle M_A.M_B\rangle + \langle M_{A'}.M_B\rangle + \langle M_A.M_{B'}\rangle - \langle M_{A'}.M_{B'}\rangle| \leq 2$$

The Above relation is called the CHSH inequality. Now consider the directions as seen in the diagram below.

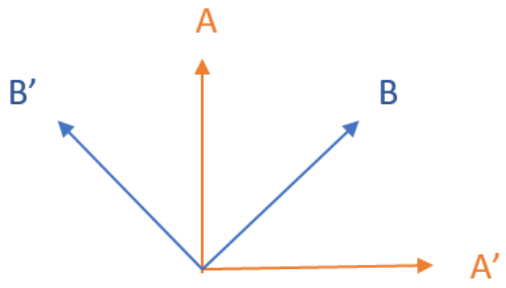

Figure 4: Alice's measurement directions in orange and Bob's in blue

When these directions are chosen by Alice and Bob in the experiments, they see that $\langle M_A.M_B\rangle = \frac{-1}{\sqrt{2}} = -A.B$ which is exactly predicted by the mathematics of Quantum mechanics involving the entangled wave function. Similarly, $\langle M_{A'}.M_B\rangle = \frac{-1}{\sqrt{2}}, \langle M_A.M_{B'}\rangle = \frac{-1}{\sqrt{2}}, \langle M_{A'}.M_{B'}\rangle = \frac{1}{\sqrt{2}}$. Calculation of $|\langle C\rangle|$ yields a value of $2\sqrt{2}$ Which is a clear violation of the bounds predicted by any local hidden variable theory. Therefore, Bell concluded that no local hidden variable theory can reproduce the correlations predicted by quantum mechanics which is in conjunction with the experiments in all the angles.

Therefore, we are compelled to agree that the wave function governing the electron-positron pair is indeed an entangled wave function, i.e; neither particle has a well defined spin. It is indeed as though the result of the measurement of one particle is instantaneously communicated to the

other. This was a shock to the physics community — to discover that nature itself is fundamentally non-local.

We use the entangled particles for key distribution, that is the basis of E91 protocol. To verify whether the particles are entangled or in other words non-local, we use the CHSH inequality.

6 E91

6.1 The Protocol

1. Say, Alice wants to communicate with Bob. They assign the job of generating the entangled particles in a singlet state to Charlie

$$|\psi_{sing}\rangle = 1/\sqrt{2}(|\uparrow\downarrow\rangle - |\downarrow\uparrow\rangle$$

2. Charlie generates and distributes them to Alice and Bob. When they both measure in the same basis, the probability that Alice measures +1 and Bob measures -1 is $\frac{1}{2}$, similarly the probability that Alice measures -1 and Bob measures +1 is also $\frac{1}{2}$. The probability that both of them measure the same is always zero.

3. They assign bit 0 to +1 result and bit 1 to -1 result. As long as they are measuring in the same basis, their results are exactly the opposite. They use this fact to establish the key.

Now that we have understood this, the protocol is as follows.

1. After every entangled pair generated by Charlie, Alice randomly chooses from the directions on the left circle and Bob from the right circle for their respective measurements.

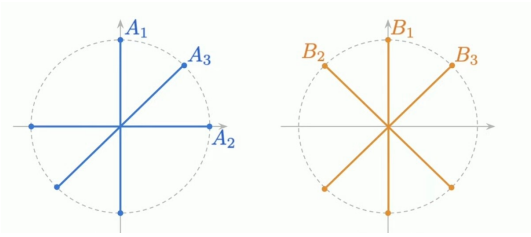

Figure 5: Alice's measurement choices in blue and Bob's measurement choices in yellow

2. Upon each measurement, both of them note down the basis they have used and tabulate them as shown below as an example for 10 measurements.

Alice's basis	A1	A2	A3	A2	A1	A3	A2	A1	A2	A3
Bob's basis	B1	B3	B3	B1	B2	B2	B3	B2	B1	B3
	✓		✓							✓

3. After all the measurements are made, they exchange information about the basis they used through the public channel.

4. Now that they have each other's bases, they divide the measurements into two different groups. Group 1 for which they used the same bases and group 2 for which they used different bases. Both of them retain the results of group 1. These are used for the key generation, indicated by the check marks above. In group 1 measurements, it is assured that they have the opposite results. So Bob simply inverts his result to match exactly with Alice's result. He then assigns 0s and 1s, this way both of them have the same bits and they have thus generated the key.

5. But how do they know they have the same key? Recall that the same key is generated only when the particles are entangled. That is only when the particles are entangled, is it assured that they get opposite results and Bob can invert his results to match Alice's. It becomes, therefore, quintessential to establish that the particles they have been receiving are indeed entangled.

6.2 Detection of Intercept and Resend Attack

6. The particles are ideally assumed to be entangled as long as there was no eavesdropping. Once there is eavesdropping or interception by Eve on either of the particles of the entangled pair, the particles are no more entangled, and the wave function governing the entanglement has collapsed, which means the particles now have a definite spin. In other words, both particles have elements of physical reality.

7. Simply put, Eve's interception has introduced elements of physical reality into the system.

8. In order to finally establish the secret key, Alice and Bob should test whether the particles they have been receiving are indeed entangled as mentioned before, this helps them detect the interception by Eve.

9. Now is when the CHSH inequality comes in handy. Recall that only the correlations arising from the entangled particles violate the inequality while the correlation arising from the particles with elements of physical reality satisfies the inequality.

10. The group 2 measurement bases (A1, B3), (A1, B2), (A2, B2) and (A2, B3) are used to check the CHSH inequality, it is as follows:

$$|\langle C\rangle| = |\langle M_{A1}.M_{B2}\rangle + \langle M_{A1}.M_{B3}\rangle + \langle M_{A2}.M_{B2}\rangle - \langle M_{A2}.M_{B3}\rangle| \leq 2$$

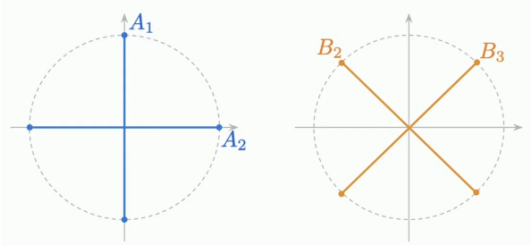

Figure 6: Group 2 measurement bases

11. The left-hand side is called the test statistic, to calculate this value, the 4 correlations have to be first calculated. This means Alice and Bob have to publicly share the measurement results of the group 2 basis with each other and it is fine to do so, as these results are not being used to establish the key.

12. Once both of them obtain these results from each other, they calculate the correlations. For eg: To calculate $\langle M_{A1}.M_{B2}\rangle$, they count the number of measurements where Alice picked A1 and Bob picked B2, and then calculate the value $M_{A1}.M_{B2}$ of all these measurements. Finally, they add the $M_{A1}.M_{B2}$ value of all these measurements and divide by the number of measurements.

13. If the test statistic is lower than or equal to 2, they abort. If it is greater than 2, they proceed to use the established key in the conventional symmetric cryptography.

6.3 Simulation of the attack

6.3.1 Flowchart

The serial numbers below correspond to the numbers on the flowchart.

0. (Optional) Eve gets to know the number of singlets from Quantum Medium. She then sends her choice of bases to Quantum Medium.

1. Alice and Bob get to know the number of singlets from Quantum Medium.

2. Alice and Bob send their choice of bases to the Quantum Medium one by one. As soon as Quantum Medium has the choice of bases from Alice and Bob (whether or not Eve has sent her bases), Quantum Medium measures the singlets. If Eve has sent her choice of bases before this time, her measurements are made before that of Alice and Bob.

3. Quantum Medium has measured the qubits according to everyone's choices of bases. Now Bob and Alice request it to return the measurement outcomes.

4. Alice and Bob share their choices of bases through the classical medium.

5. Alice and Bob share their unused bits through the classical medium for the calculation of CHSH test statistic.

6. Finally they individually calculate the CHSH test statistic. Depending on the obtained value, they decide whether to use the established key or to start the process all over again.

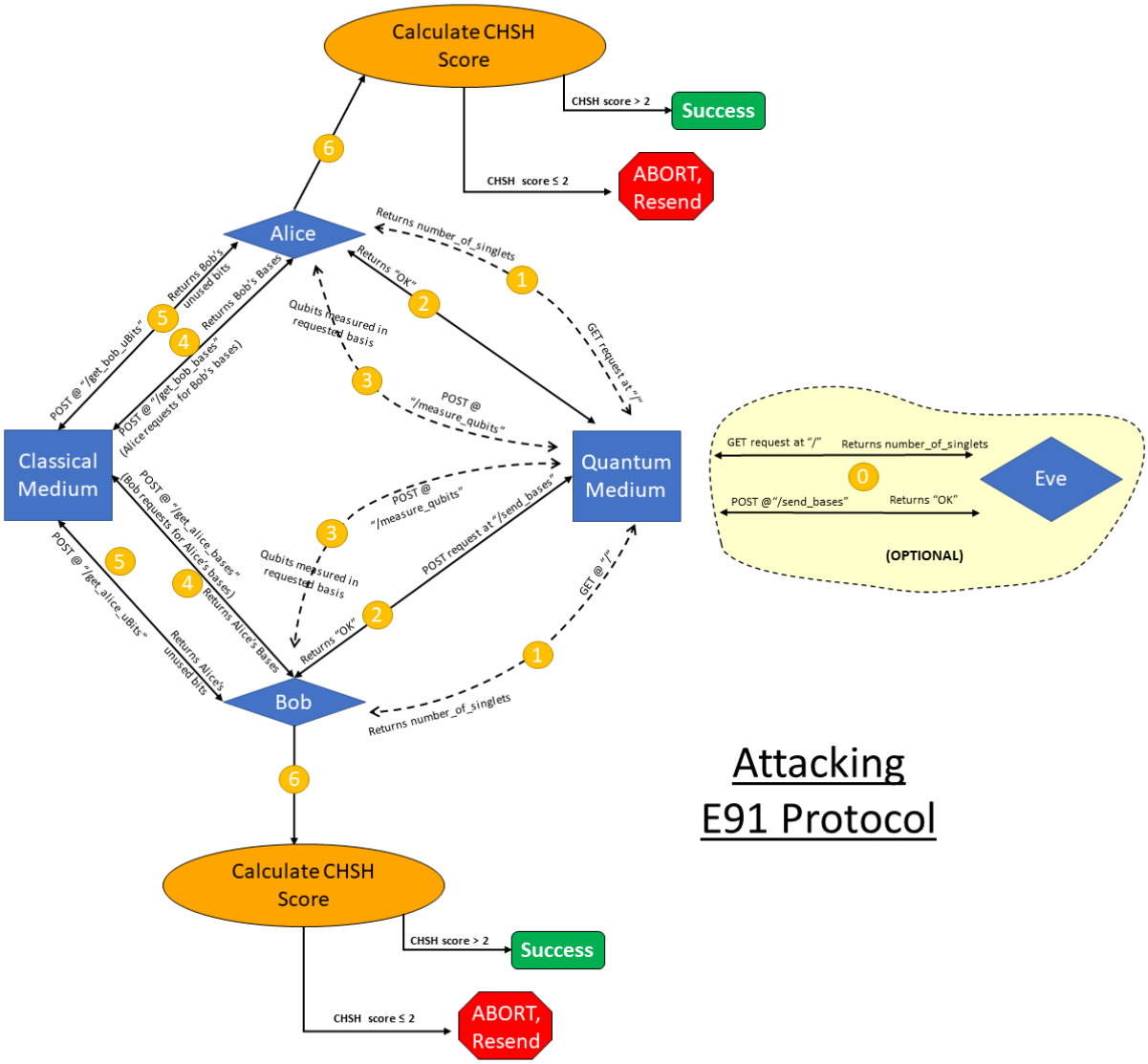

Figure 7: Flow chart of the E91 protocol with attack

6.3.2 Code

We have written the following 6 files for simulating E91. We have used Flask, Requests and Qiskit python modules to complete the task.

- quantumMedium.py: This is the quantum medium. It is responsible for generating the entangled particles and measuring them on anyone's request.

```
from flask import Flask, request
from qiskit import *

number_of_singlets = 500
qcomp = Aer.get_backend("qasm_simulator")
bases = {"alice": None, "bob": None, "eve": None}
results = None
singlets = []
chsh_res = None
chsh_corr_val = None

def alice_measure(qc, basis):  # basis = "X", "W" or "Z"
    """Alice's measurement circuits"""
    if basis == "X":  # X basis
        qc.h(0)
    elif basis == "W":  # W basis
        qc.s(0)
        qc.h(0)
        qc.t(0)
        qc.h(0)
    elif basis == "Z":  # Z basis
        pass
    else:
        raise ValueError("Value of parameter 'basis' can be one of 'X', 'W'
    or 'Z'")

def bob_measure(qc, basis):  # basis = "W", "Z" or "V"
    """Bob's measurement circuits"""
    if basis == "W":  # W basis
        qc.s(1)
        qc.h(1)
        qc.t(1)
        qc.h(1)
    elif basis == "Z":  # Z basis
        pass
    elif basis == "V":  # V basis
        qc.s(1)
        qc.h(1)
        qc.tdg(1)
        qc.h(1)
    else:
        raise ValueError("Value of parameter 'basis' can be one of 'W', 'Z'
    or 'V'")

def calc():
    global results, singlets, chsh_res
    alice_bases = bases["alice"]
    bob_bases = bases["bob"]
    eve_bases = bases["eve"]
    for i in range(number_of_singlets):
        singlet = QuantumCircuit(2, 4)
        singlet.x(0)
```

```
        singlet.x(1)
        singlet.h(0)
        singlet.cx(0, 1)
        if eve_bases:
            alice_measure(singlet, eve_bases[i])
            # bob_measure(singlet, eve_bases[i])
        alice_measure(singlet, alice_bases[i])
        bob_measure(singlet, bob_bases[i])
        singlet.measure(0, 0)
        singlet.measure(1, 1)
        singlets.append(singlet)
    result = execute(singlets, qcomp, shots=1).result().get_counts()
    chsh_res = result
    result = [list(x.keys())[0] for x in result]
    alice_result = "".join([x[-1] for x in result])
    bob_result = "".join([x[-2] for x in result])
    results = {"alice": alice_result, "bob": bob_result}

app = Flask(__name__)

@app.route("/")
def login():
    print(f"\nsomeone asked for the number of singlets... returned {
   number_of_singlets}")
    return str(number_of_singlets)

@app.route("/send_bases", methods=["POST"])
def send_bases():
    global bases
    name = request.form["name"]
    bases[name] = request.form["bases"]
    print(f"\n{name.title()} sent choice of bases")
    if bases["bob"] and bases["alice"]:
        print("\nAs the Quantum Medium has both the bases of Alice and Bob,
    it can start making measurements...")
        calc()
        print("\nmeasurements completed!")
    return "Ok"

@app.route("/measure_qubits", methods=["POST"])
def measure_qubits():
    name = request.form["name"]
    print(f"\n{name.title()} is requesting for the measurements.", end = "
   ")
    if results:
        print("Measurements are now ready, so Quantum Medium is  returning
   the results.")
        return results[name]
    else:
        print("But measurements are not ready yet, so Quantum Medium
   requests him/her to wait a bit before asking again!")
        return "wait"

app.run(
    host="localhost",
    port=5001,
    debug=True,
 )
```

- classicalMedium.py: This is the classical medium through which the 3 three parties (Alice, Bob and Eve) communicate after they have sent their qubits.

```
from flask import Flask, request

app = Flask(__name__)

bob_bases, alice_bases = None, None
bob_uBits, alice_uBits = None, None
status = None

@app.route('/get_bob_bases', methods=['POST'])
def get_bob_bases():
    global alice_bases
    alice_bases = request.form['bases']
    print(f"\nAlice sent her bases and is requesting for Bob's bases.", end = " ")
    if bob_bases:
        print("The Classical Medium already has Bob's bases, so it gives it to her.")
        return bob_bases
    print("But Bob hasn't sent his bases yet, so she waits.")
    return "wait"

@app.route('/get_alice_bases', methods=['POST'])
def get_alice_bases():
    global bob_bases
    bob_bases = request.form['bases']
    print(f"\nBob sent his bases and is requesting for Alice's bases.", end = " ")
    if alice_bases:
        print("The Classical Medium already has Alice's bases, so it gives it to him.")
        return alice_bases
    print("But Alice hasn't sent her bases yet, so he waits.")
    return "wait"

@app.route('/get_bob_uBits', methods=['POST'])
def get_bob_uBits():
    global alice_uBits
    alice_uBits = request.form['uBits']
    print(f"\nAlice sent her unused Bits and is requesting for Bob's unused Bits.", end = " ")
    if bob_uBits:
        print("The Classical Medium already has Bob's unused Bits, so it gives it to her.")
        return bob_uBits
    print("But Bob hasn't sent his unused Bits yet, so she waits.")
    return "wait"

@app.route('/get_alice_uBits', methods=['POST'])
def get_alice_uBits():
    global bob_uBits
    bob_uBits = request.form['uBits']
    print(f"\nBob sent his unused Bits and is requesting for Alice's unused Bits.", end = " ")
    if alice_uBits:
        print("The Classical Medium already has Alice's unused Bits, so it gives it to him.")
        return alice_uBits
```

```
    print("But Alice hasn't sent her unused Bits yet, so he waits.")
    return "wait"

app.run(
    host = "localhost",
    port = 5000,
    debug = True,
 )
```

- module.py: We can clearly see that most parts of the codes for Alice, Bob and Eve are similar. So we decided to write all the important functions and put them in this file. Thus the codes for Alice and Bob will be smaller and more readable.

```
import random
import requests
from time import sleep

qIP, qPORT = "localhost", "5001"
cIP, cPORT = "localhost", "5000"
quantuMedium  =  f"http://{qIP}:{qPORT}/"  # URL of quantum channel
classicalMedium = f"http://{cIP}:{cPORT}/"  # URL of classical channel


def send_bases(baseOpt, number_of_singlets, name):
    choice = "".join([baseOpt[random.randint(0, len(baseOpt)-1)] for i in
   range(number_of_singlets)])
    print("Sending bases...")
    requests.post(f"{quantuMedium}send_bases",
                  data = {
                      "name": name,
                      "bases": choice,
                      }
                  ).text
    return choice

def measure_qubits(name):
    try:
        print("Reading qubits...")
        ret = requests.post(f"{quantuMedium}measure_qubits", data={"name":
   name}).text
    except:
        sleep(10)
        return measure_qubits(name)
    if  ret == "wait":
        sleep(10)
        return measure_qubits(name)
    return ret

def get_bob_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_bases", data = {"
   bases": bases}).text
    except:
        sleep(1)
        return get_bob_bases(bases)
    if  ret == "wait":
        sleep(1)
        return get_bob_bases(bases)
    return ret

```

```
def get_alice_bases(bases):
    try:
        ret = requests.post(f"{classicalMedium}get_alice_bases", data = {"bases": bases}).text
    except:
        sleep(1)
        return get_alice_bases(bases)
    if  ret == "wait":
        sleep(1)
        return get_alice_bases(bases)
    return ret

def get_bob_uBits(uBits):
    try:
        ret = requests.post(f"{classicalMedium}get_bob_uBits", data = {"uBits": uBits}).text
    except:
        sleep(1)
        return get_bob_uBits(uBits)
    if  ret == "wait":
        sleep(1)
        return get_bob_uBits(uBits)
    return ret

def get_alice_uBits(uBits):
    try:
        ret = requests.post(f"{classicalMedium}get_alice_uBits", data = {"uBits": uBits}).text
    except:
        sleep(1)
        return get_alice_uBits(uBits)
    if  ret == "wait":
        sleep(1)
        return get_alice_uBits(uBits)
    return ret

def chsh(uBitsA, uBitsB, alice_bases, bob_bases):
    search = ["00", "01", "10", "11"]  # f"{bob}{alice}"
    XW = [0, 0, 0, 0]
    XV = [0, 0, 0, 0]
    ZW = [0, 0, 0, 0]
    ZV = [0, 0, 0, 0]

    for i in range(len(alice_bases)):
        try:
            if uBitsA[i] == " ":
                continue
        except IndexError:
            raise IndexError(f"i = {i}")

        if alice_bases[i] == "X" and bob_bases[i] == "W":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    XW[j] += 1
                    break
        if alice_bases[i] == "X" and bob_bases[i] == "V":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    XV[j] += 1
                    break
```

```
        if alice_bases[i] == "Z" and bob_bases[i] == "W":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    ZW[j] += 1
                    break
        if alice_bases[i] == "Z" and bob_bases[i] == "V":
            for j in range(4):
                if f"{uBitsA[i]}{uBitsB[i]}" == search[j]:
                    ZV[j] += 1
                    break

    # expectation values of XW, XV, ZW and ZV observables (2)
    chsh_corr_val = [
                (XW[0] - XW[1] - XW[2] + XW[3]) / sum(XW),  # -1/sqrt(2)
                -(XV[0] - XV[1] - XV[2] + XV[3]) / sum(XV),  # 1/sqrt(2)
                (ZW[0] - ZW[1] - ZW[2] + ZW[3]) / sum(ZW),  # -1/sqrt(2)
                (ZV[0] - ZV[1] - ZV[2] + ZV[3]) / sum(ZV)  # -1/sqrt(2)
        ]

    chsh_corr_val = sum(chsh_corr_val)
    return chsh_corr_val
```

- alice.py: Code to be used by Alice.

```
from module import *

name = "alice"
print(f"Hi! This is {name.title()}!")
baseOpt = ['X', 'W', 'Z']

number_of_singlets = int(requests.get(quantuMedium).text)
alice_bases = send_bases(baseOpt, number_of_singlets, name)
bits = measure_qubits(name)
bob_bases = get_bob_bases("".join(alice_bases))

key = ""
uBitsA = ""
for i in range(number_of_singlets):
    if alice_bases[i] == bob_bases[i]:
        bit = int(bits[i])
        if bit: key += "1"
        else: key += "0"
        uBitsA += " "
    else:
        uBitsA += bits[i]

key_length = len(key)
mismatch = number_of_singlets - key_length

uBitsB = get_bob_uBits(uBitsA)

chsh_score = float(chsh(uBitsA, uBitsB, alice_bases, bob_bases))
print(f"chsh score: {chsh_score}")
success = chsh_score < -2

if success:
    print(key)
else:
    print("Eve is here! Abort!")
```

- bob.py : Code to be used by Bob.

```
from module import *

name = "bob"
print(f"Hi! This is {name.title()}!")
baseOpt = ['W', 'Z', 'V']

number_of_singlets = int(requests.get(quantuMedium).text)
bob_bases = send_bases(baseOpt, number_of_singlets, name)
bits = measure_qubits(name)
alice_bases = get_alice_bases("".join(bob_bases))

key = ""
uBitsB = ""
for i in range(number_of_singlets):
    if bob_bases[i] == alice_bases[i]:
        bit = int(bits[i])
        if bit: key += "0"
        else: key += "1"
        uBitsB += " "
    else:
        uBitsB += bits[i]

key_length = len(key)
mismatch = number_of_singlets - key_length

uBitsA = get_alice_uBits(uBitsB)

chsh_score = chsh(uBitsA, uBitsB, alice_bases, bob_bases)
print(f"chsh score: {chsh_score}")
success = chsh_score < -2

if success:
    print(key)
else:
    print("Eve is here! Abort!")
```

- eve.py: Code to be used by Eve.

```
from module import *

name = "eve"
print(f"Hi! This is {name.title()}!")
baseOpt = ['W', 'Z']

singlets_sent = int(requests.get(quantuMedium).text)
send_bases(baseOpt, singlets_sent, name)
print("completed eavesdropping!")
```

6.3.3 Output

The output can be viewed on the README.md file on GitHub.

Without Eve's Interception:

"quantumMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\quantumMedium.py
 * Serving Flask app 'quantumMedium' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5001/ (Press CTRL+C to quit)

someone asked for the number of singlets... returned 2000
127.0.0.1 - - [19/Jul/2022 10:30:08] "GET / HTTP/1.1" 200 -

Alice sent choice of bases
127.0.0.1 - - [19/Jul/2022 10:30:10] "POST /send_bases HTTP/1.1" 200 -

someone asked for the number of singlets... returned 2000
127.0.0.1 - - [19/Jul/2022 10:30:10] "GET / HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:30:12] "POST /measure_qubits HTTP/1.1" 200 -

Bob sent choice of bases

As the Quantum Medium has both the bases of Alice and Bob, it can start making
measurements...

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:30:24] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:30:36] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:31:03] "POST /measure_qubits HTTP/1.1" 200 -

measurements completed!
127.0.0.1 - - [19/Jul/2022 10:31:03] "POST /send_bases HTTP/1.1" 200 -
```

```
Bob is requesting for the measurements. Measurements are now ready, so Quantum
Medium is  returning the results.
127.0.0.1 - - [19/Jul/2022 10:31:05] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. Measurements are now ready, so Quantum
Medium is  returning the results.
127.0.0.1 - - [19/Jul/2022 10:31:15] "POST /measure_qubits HTTP/1.1" 200 -
```

"classicalMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\classicalMedium.py
 * Serving Flask app 'classicalMedium' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5000/ (Press CTRL+C to quit)

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:31:07] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:31:10] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:31:13] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:31:16] "POST /get_alice_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. The Classical Medium
already has Bob's bases, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:31:17] "POST /get_bob_bases HTTP/1.1" 200 -

Alice sent her unused Bits and is requesting for Bob's unused Bits. But Bob hasn't
sent his unused Bits yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:31:19] "POST /get_bob_uBits HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. The Classical Medium
already has Alice's bases, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:31:19] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his unused Bits and is requesting for Alice's unused Bits. The Classical
```

```
Medium already has Alice's unused Bits, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:31:21] "POST /get_alice_uBits HTTP/1.1" 200 -

Alice sent her unused Bits and is requesting for Bob's unused Bits. The Classical
Medium already has Bob's unused Bits, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:31:22] "POST /get_bob_uBits HTTP/1.1" 200 -
```

"alice.py":

```
Hi! This is Alice!
Sending bases...
Reading qubits...
Reading qubits...
Reading qubits...
Reading qubits...
Reading qubits...
chsh score: -2.831877117951897
00100000110101011111000010011010001001010000011000010000000000010100010110100010111
10100001101000010010011000110010001010001001101110110010100010001101000100100010111
00100111110011100011111010010110101001011101110001011111101000111100010110010100111
01010111011110000101100110010001010100000010001110100100011101110001001000110001011
01000110100011011010100000101100001001101011001101100010010100100110011111100101110
10110101101101101010101101
```

"bob.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\bob.py
Hi! This is Bob!
Sending bases...
Reading qubits...
chsh score: -2.831877117951897
00100000110101011111000010011010001001010000011000010000000000010100010110100010111
10100001101000010010011000110010001010001001101110110010100010001101000100100010111
00100111110011100011111010010110101001011101110001011111101000111100010110010100111
01010111011110000101100110010001010100000010001110100100011101110001001000110001011
01000110100011011010100000101100001001101011001101100010010100100110011111100101110
10110101101101101010101101
```

With Eve's Interception:

"quantumMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\quantumMedium.py
 * Serving Flask app 'quantumMedium' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
```

```
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5001/ (Press CTRL+C to quit)

someone asked for the number of singlets... returned 2000
127.0.0.1 - - [19/Jul/2022 10:35:08] "GET / HTTP/1.1" 200 -

Eve sent choice of bases
127.0.0.1 - - [19/Jul/2022 10:35:10] "POST /send_bases HTTP/1.1" 200 -

someone asked for the number of singlets... returned 2000
127.0.0.1 - - [19/Jul/2022 10:35:34] "GET / HTTP/1.1" 200 -

someone asked for the number of singlets... returned 2000
127.0.0.1 - - [19/Jul/2022 10:35:34] "GET / HTTP/1.1" 200 -

Alice sent choice of bases
127.0.0.1 - - [19/Jul/2022 10:35:36] "POST /send_bases HTTP/1.1" 200 -

Bob sent choice of bases

As the Quantum Medium has both the bases of Alice and Bob, it can start making
measurements...

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:35:38] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:35:50] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:36:02] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. But measurements are not ready yet, so
Quantum Medium requests him/her to wait a bit before asking again!
127.0.0.1 - - [19/Jul/2022 10:36:28] "POST /measure_qubits HTTP/1.1" 200 -

measurements completed!
127.0.0.1 - - [19/Jul/2022 10:36:28] "POST /send_bases HTTP/1.1" 200 -

Bob is requesting for the measurements. Measurements are now ready, so Quantum
Medium is  returning the results.
127.0.0.1 - - [19/Jul/2022 10:36:30] "POST /measure_qubits HTTP/1.1" 200 -

Alice is requesting for the measurements. Measurements are now ready, so Quantum
Medium is  returning the results.
127.0.0.1 - - [19/Jul/2022 10:36:40] "POST /measure_qubits HTTP/1.1" 200 -
```

"classicalMedium.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\classicalMedium.py
 * Serving Flask app 'classicalMedium' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production
deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Restarting with stat
 * Debugger is active!
 * Debugger PIN: 108-787-847
 * Running on http://localhost:5000/ (Press CTRL+C to quit)

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:36:32] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:36:35] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:36:39] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. But Alice hasn't sent her
bases yet, so he waits.
127.0.0.1 - - [19/Jul/2022 10:36:42] "POST /get_alice_bases HTTP/1.1" 200 -

Alice sent her bases and is requesting for Bob's bases. The Classical Medium
already has Bob's bases, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:36:42] "POST /get_bob_bases HTTP/1.1" 200 -

Alice sent her unused Bits and is requesting for Bob's unused Bits. But Bob hasn't
sent his unused Bits yet, so she waits.
127.0.0.1 - - [19/Jul/2022 10:36:44] "POST /get_bob_uBits HTTP/1.1" 200 -

Bob sent his bases and is requesting for Alice's bases. The Classical Medium
already has Alice's bases, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:36:45] "POST /get_alice_bases HTTP/1.1" 200 -

Bob sent his unused Bits and is requesting for Alice's unused Bits. The Classical
Medium already has Alice's unused Bits, so it gives it to him.
127.0.0.1 - - [19/Jul/2022 10:36:47] "POST /get_alice_uBits HTTP/1.1" 200 -

Alice sent her unused Bits and is requesting for Bob's unused Bits. The Classical
Medium already has Bob's unused Bits, so it gives it to her.
127.0.0.1 - - [19/Jul/2022 10:36:48] "POST /get_bob_uBits HTTP/1.1" 200 -
```

"eve.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\eve.py
Hi! This is Eve!
Sending bases...
completed eavesdropping!
```

"alice.py":

```
PS C:\Users\amukh\Desktop\QKD\E91> python .\alice.py
Hi! This is Alice!
Sending bases...
Reading qubits...
Reading qubits...
Reading qubits...
Reading qubits...
Reading qubits...
chsh score: -1.9656904472046612
Eve is here! Abort!
```

"bob.py":

```
Hi! This is Bob!
Sending bases...
Reading qubits...
chsh score: -1.9656904472046612
Eve is here! Abort!
```

7 Conclusion

In the near future, quantum computers will be able to solve problems that have been impossible to solve before. They use the power of quantum mechanics to calculate things much faster and will be easily capable of threatening contemporary encryption mechanisms. These mechanisms employ the fact that factoring large numbers is a tedious job even for the best of supercomputers. Therefore, in the era of quantum computers, it is best to resort to private key encryption. This is where the principles of quantum mechanics are utilised to distribute the private keys securely to the parties involved in the communication. Using quantum mechanical principles is secure because the security is guaranteed by the laws of Physics. They let us detect the presence of an eavesdropper thereby allowing us to abort the mission. One conventional eavesdropping strategy is the "intercept and resend" attack which has been simulated for both the protocols — BB84 and E91 in our work. As can be seen in the output, Alice and Bob — the parties involved in the communication detect the presence of the eavesdropper — Eve, and abort the mission.

8 Appendix

1. All the Codes have been kept in the GitHub repository: `https://github.com/PeithonKing/Attacking_QKD_Protocols`

9 References

[1] Nicolas Gisin, Gregorie Ribordy, Wolfgang Title and Hugo Zbinden, Quantum cryptography `https://journals.aps.org/rmp/abstract/10.1103/RevModPhys.74.145`

[2] Eddie Woo, The RSA Encryption Algorithm (1 of 2: Computing an Example) `https://www.youtube.com/watch?v=4zahvcJ9glg&t=17s`

[3] Eddie Woo, The RSA Encryption Algorithm (2 of 2: Computing an Example) `https://www.youtube.com/watch?v=oOcTVTpUsPQ&t=1s`

[4] minutephysics, How Quantum Computers Break Encryption — Shor's Algorithm Explained,`https://www.youtube.com/watch?v=lvTqbM5Dq4Q&t=1s`

[5] Vimal Gaur, Devika Mehra, Anchit Aggarwal, Raveena Kumari, Srishti Rawat, Quantum Key Distribution: Attacks and Solutions `https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3563118`

[6] Matthew Campagna, et al., Quantum Safe Cryptography and Security, An introduction, benefits, enablers and challenges. `https://www.etsi.org/images/files/ETSIWhitePapers/QuantumSafeWhitepaper.pdf`

[7] Jochen Rau, Quantum key distribution I: BB84 protocol `https://youtu.be/u_K9jPBrOwA`

[8] Thomas G. wong, Introduction to Classical and Quantum Computing `https://www.thomaswong.net/introduction-to-classical-and-quantum-computing-1e3p.pdf`

[9] J. S. Bell, On the Einstein Rosen Prodolsky Paradox `https://cds.cern.ch/record/111654/files/vol1p195-200_001.pdf`

[10] D. Bohm and Y. Aharonov, Discussion of Experimental Proof for the Paradox of Einstein, Rosen, and Podolsky `https://journals.aps.org/pr/abstract/10.1103/PhysRev.108.1070`

[11] Leonard Susskind and Art Friedman, Quantum Mechanics: The Theoretical Minimum https://www.goodreads.com/book/show/18210750-quantum-mechanics

[12] John Preskill, Quantum Information and Computation Chapter 4, http://theory.caltech.edu/~preskill/ph229/notes/chap4_01.pdf

[13] Artur K. Ekert, Quantum cryptography based on Bell's theorem https://cqi.inf.usi.ch/qic/91_Ekert.pdf

[14] Michal Hajdušek, Overview of Quantum Communication, https://www.youtube.com/watch?v=JZFK4jr3c7M&list=PLCTGenrx1-SOC-b98RCC1uEGI-Sc-N3C-

[15] Github: Qiskit Community Implementation of E91

[16] Quantum Computing group, IIT Roorkee, Fundamentals of Quantum Key Distribution — BB84, B92 E91 protocols https://bit.ly/3okOfpz