Mitigating Against Attacks on BB84 Quantum Key Distribution

Charles Buckley

ORCID: https://orcid.org/0000-0003-1647-1248

Submitted: April 2021

Abstract

Quantum key distribution has been one of the promising technologies since quantum computing began. It has the potential to replace the classical secure key generation method that can be used to secure the communication across the network. The protocol has been proven to be secure however, the implementation needs to be regularly checked to ensure there are no ways to get the secure key from the side channel. This research aims to get real-world data by experimentally testing the attacks with and without a mitigation plan. Then each attack will be categorized according to few criteria. This can help to standardize attacks and help ETSI with their standardization work. The categorization also shows attack pattern that can occur in the future. This research has identified that real-world data may not be like the prediction of their respective equation which is a topic of further research. This research concludes that by using a mitigation plan, most of the automated attacks can be mitigated. However, without a mitigation plan, devices will be highly vulnerable to automated attacks even with the quantum key distribution.

Acknowledgements

First and foremost, I would like to express my deepest gratitude to Dr Domenico Vicinanza at Anglia Ruskin University for introducing me to Piotr Rydlichowski. Without this opportunity, I would never have been able to learn and push myself as much. I thank your patience for I did not update you until five days before the deadline.

I especially thank Piotr Rydlichowski for answering many questions about the project as well as many questions that were not directly related to it. I also thank your patience for I requested help a week before the deadline without any update before it and you still helped me.

I must thank all my professors at Anglia Ruskin University for supporting me and teaching me many things that I did not know.

Lastly, I thank my family for supporting me throughout this entire journey.

Table of Contents

List of Figures

List of Tables

1 Introduction

This section focuses on the background, problem statement, the aim of the study and the report structure.

1.1 Background

Quantum computing has the potential to simulate the movement of atoms (O'Malley, et al., 2016), create new medicine (Hernandez, et al., 2019), discover vaccines, and solve certain problems that classical computers cannot (Anon., 1992). One of the promising technologies is the quantum network (Kimble, 2008). Quantum network can allow things such as true peer to peer mesh network (K, 2018), quantum blind computing (Chien, et al., 2015) and quantum clock synchronisation (Chuang, 2000) and (Krco & Paul, 2001). However, the best example would be the quantum key distribution (QKD). QKD is designed to securely share random bits which form the base of encryption keys (Bennet & Brassard, 1984). It also allows the detection of eavesdropper by statistical analysis such as Bell's inequality (Bell, 1964). QKD covers only the creation and distribution of keys but not the encryption of data or transmission of encrypted data.

1.2 Problem Background

QKD is theoretically secure (Shannon, 1949). However, in practice, there are usually ways to break the security. This is not because the protocol is not secure (Tomamichel & Leverrier, 2017) and (Portmann & Renner, 2014), but because the

implementation in the real world is different from the implementation in theory. For example, in classical computing, AES and RSA have been broken by researchers even if they still are theoretically secure. See (Ashokkumar, et al., 2016) for AES and (Genkin, et al., 2013) for RSA.

Therefore, there is a need to constantly monitor and test QKD algorithms to make sure there are no known attacks. If there are known attacks, then it needs to be mitigated as much as possible.

1.3 Aim of the Study

The aim of this study is to design and simulate attacks against the QKD algorithm and to create a mitigation plan against such attacks. This study will help the adoption of QKD in the future by finding attacks and mitigation plan or by confirming that QKD protocol has no attack vectors as of this research.

1.4 Scope of this Research

This research is limited to BB84 protocol and any attacks that may be able to get the keys generated by BB84 protocol. And mitigation plans against such attacks. Experiments will be restricted to attacks that can be simulated. Any attacks that involve hardware or physical access is not possible due to the unavailability of a real quantum computer.

1.5 Report Structure

The remaining of the report is structured as follows. In chapter 2, the theory of quantum information and attacks on QKD is introduced. In chapter 3, the research methodology is described. In chapter 4, the implementation and result are presented. In chapter 5, the results, evaluation, and implication of this research are discussed. In chapter 6, this research is concluded.

2 Literature Review

This section introduces the theory of quantum information and the attacks on BB84 QKD.

2.1 Theory of Quantum Computing

In this section, the author will give a basic introduction to the theory of quantum computing that is necessary to understand attacks against QKD.

2.1.1 History of Quantum Computing

Paul Benloff proposed a Turing machine model in quantum mechanics in 1980 (Benioff, 1980). In 1982, Richard Feynman noted that classical computers cannot solve problems that quantum computers can. He also noted that it is generally not feasible to represent the output of a quantum computer using a classical device (Feynman, 1981). The term "Quantum Computer" was officially used for the first time by David Deutsch in 1985 who suggested that universal quantum computer could be developed by creating quantum gates that function similarly to binary logic gates. David Deutsch also generalised computing methods for quantum computers (Deutch, 1984).

During this period, a new subfield of quantum computing called quantum networking emerged. The first QKD protocol called BB84 was developed by Charles Bennet and Gilles Brassard in 1984 (Bennet & Brassard, 1984). In 1989, the first successful quantum exchange was performed. BB84 got wide recognition because of its security (Bennett & Brassard, 1989).

In 1994, Shor's algorithm was invented by Peter Williston Shor at Bell Laboratories. It is a quantum algorithm that allows for large integer factorization in polynomial time. This means that when quantum computer become powerful, they can break encryptions based on prime number factorization such as RSA (Shor, 1994).

2.1.2 Qubit

Any information can be represented by binary digit. In general, expressed as 0 or 1. One binary digit is known as a bit. A bit can be represented by anything physical that has two values. Such as low or high voltage in a classical computer, heads or tails in a coin or left or right direction that pen points to when it is spun around. Similarly, any information can be represented by a quantum bit or a qubit (Schumacher, 1995). However, qubits have additional properties that can be used to solve interesting problems. For example, a qubit can be in a superposition state which represents both 0 and 1 simultaneously. Qubits can be represented by the polarization of a photon, spin of electron or nucleus and many others[1].

Dirac Notation

Dirac notation also called bra-ket notation is used to express the state of a single qubit as a vector on a complex Hilbert space. This notation was established by Paul Dirac in 1939 (Dirac, 1939).

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$$

[1] https://en.wikipedia.org/wiki/Qubit - Accessed 15th April 2021

$$|\alpha|^2 + |\beta|^2 = 1 \qquad (2.1)^2$$

Coefficients α and β are probability amplitudes and can be complex numbers. A bra looks like $\langle 0|$ and a ket looks like $|0\rangle$. The coefficients in equation (2.1) describes the probability of possible states $|0\rangle$ and $|1\rangle$. For example, the superposition state in Dirac notation is:

$$|\psi\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle \qquad (2.2)^2$$

Equation (2.2) shows that both $|0\rangle$ and $|1\rangle$ have a probability of $\frac{1}{\sqrt{2}}$. This indicates that $|\psi\rangle$ is in a superposition state with $|0\rangle$ and $|1\rangle$ having equally weighted probabilities.

A vector in a Hilbert space can also be used to express a quantum state:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle = \alpha\begin{bmatrix}1\\0\end{bmatrix} + \beta\begin{bmatrix}0\\1\end{bmatrix} = \begin{bmatrix}\alpha\\\beta\end{bmatrix} \qquad (2.3)^3$$

Bloch Sphere

Quantum state can be geometrically represented on a Bloch sphere as a vector pointing towards the surface. Bloch sphere is named after the physicist Felix Bloch in 1946 (Bloch, 1946).

[2] https://en.wikipedia.org/wiki/Quantum_computing - Accessed 15th April 2021

[3] https://en.wikipedia.org/wiki/Quantum_logic_gate - Accessed 15th April 2021

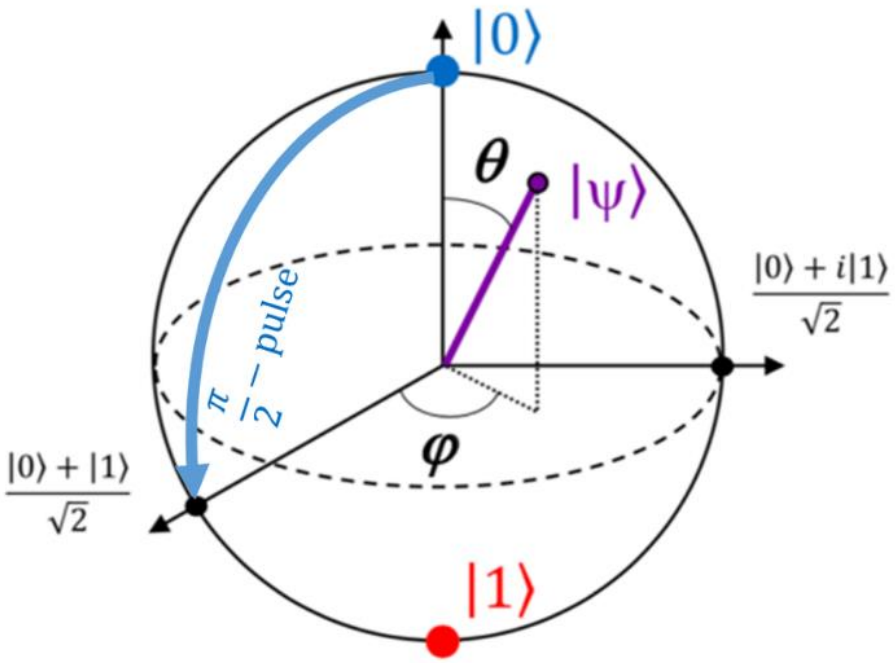

Figure 2.1 Quantum state represented on Bloch sphere[4]

Equation (2.1) written in Dirac notation can be converted to Bloch sphere representation by substituting α for $\cos\frac{\theta}{2}$ and β for $e^{i\varphi}\sin\frac{\theta}{2}$.

$$|\psi\rangle = \cos\frac{\theta}{2}|0\rangle + e^{i\varphi}\sin\frac{\theta}{2}|1\rangle \qquad (2.4)^{1}$$

2.1.3 Single Qubit Quantum Gates

Classical computers use gates to manipulate bits. Similarly, quantum computers have quantum gates to manipulate qubits. Quantum gates that are like Boolean logic gates are generally called unitary gates as they give unitary transformation of qubits states.

[4] https://www.researchgate.net/figure/The-Bloch-sphere-provides-a-useful-means-of-visualizing-the-state-of-a-single-qubit-and_fig1_335028508 - Accessed 15th April 2021

Pauli-X Gate

X gate is like classical NOT gate. It is the equivalent of rotating Bloch sphere by the X-axis.

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = |0\rangle\langle 1| + |1\rangle\langle 0| \qquad (2.5)^3$$

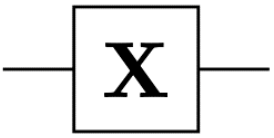

Figure 2.2 Pauli-X gate represented in a circuit[3]

Therefore, $|0\rangle = X|1\rangle$ and $|1\rangle = X|0\rangle$. And to see the effect the gate has on qubit:

$$X|0\rangle = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} = |1\rangle \qquad (2.6)^3$$

Pauli-Y Gate

Y gate changes the quantum state $|0\rangle$ to $i|1\rangle$ and $|1\rangle$ to $-i|1\rangle$. It is the equivalent of rotating Bloch sphere by the Y-axis.

$$Y = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} = -i|0\rangle\langle 1| + i|1\rangle\langle 0| \qquad (2.7)^3$$

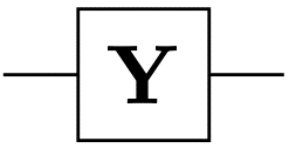

Figure 2.3 Pauli-Y gate represented in a circuit[3]

Therefore, $-i|0\rangle = Y|1\rangle$ and $i|1\rangle = Y|0\rangle$. And to see the effect the gate has on qubit:

$$Y|0\rangle = \begin{bmatrix} 0 & -i \\ i & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} -i0 \\ i1 \end{bmatrix} \quad (2.8)^3$$

Pauli-Z Gate

Basis state $|0\rangle$ is left unchanged by Z gate but changes $|1\rangle$ to $-|1\rangle$ and $-|1\rangle$ to $|1\rangle$. It is the equivalent of rotating Bloch sphere by the Z-axis.

$$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} = |0\rangle\langle 0| - |1\rangle\langle 1| \quad (2.9)^3$$

Figure 2.4 Pauli-Z gate represented in a circuit[3]

Therefore, $|0\rangle = Z|0\rangle$ and $-|1\rangle = Z|1\rangle$. And to see the effect the gate has on qubit:

$$Z|1\rangle = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \end{bmatrix} \quad (2.10)^3$$

Hadamard Gate

H gate will map $|0\rangle$ to $\frac{|0\rangle+|1\rangle}{\sqrt{2}}$ and $|1\rangle$ to $\frac{|0\rangle-|1\rangle}{\sqrt{2}}$ which indicates that the qubit is in superposition state with $|0\rangle$ and $|1\rangle$ having equally weighted probabilities.

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \quad (2.11)^3$$

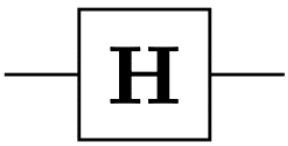

Figure 2.5 Hadamard gate represented in a circuit[3]

Therefore, $\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle) = H|0\rangle$ and $\frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) = H|1\rangle$. And to see the effect the gate has on qubit:

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix} \quad (2.12)^3$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\ 1 & -1\end{bmatrix}\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix} \quad (2.13)^3$$

2.1.4 Controlled Gates

Controlled gates manipulate more than one qubit. If gate $U = \begin{bmatrix}u_{00} & u_{01}\\ u_{10} & u_{11}\end{bmatrix}$ is a general gate that performs on a single qubit, then basis states will be mapped as follows:

Table 2.1 Shows what basis states will map to by controlled-U gate[3]

$\|00\rangle\ to\ \|00\rangle$
$\|01\rangle\ to\ \|01\rangle$
$\|10\rangle\ to\ \|1\rangle\otimes U\|0\rangle = \|1\rangle\otimes(u_{00})\|0\rangle + (u_{10})\|1\rangle$
$\|11\rangle\ to\ \|1\rangle\otimes U\|1\rangle = \|1\rangle\otimes(u_{01})\|0\rangle + (u_{11})\|1\rangle$

$$C(U) = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & u_{00} & u_{01}\\ 0 & 0 & u_{10} & u_{11}\end{bmatrix} \quad (2.14)^3$$

Equation (2.14) shows the controlled-U gate in matrix representation. Some of the controlled gates that are used are controlled-X gate (or CX), controlled-Y gate (or CY) and controlled-Z gate (or CZ). For example, CX gate in matrix representation would be:

$$CX = CNOT = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.15)^{3}$$

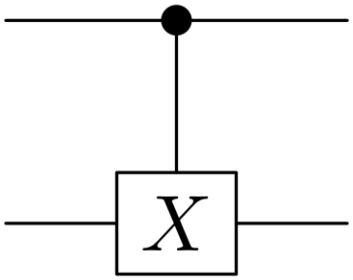

Figure 2.6 CX gate represented in a circuit[3]

2.1.5 Measurement

Measurement or observation is non-reversible, so it is not a gate. Observing a qubit will collapse or change its state to be a single classical value like $|0\rangle$ or $|1\rangle$ instead of a superposition state. Why, how, or even if the measurement collapses the qubit state to be $|0\rangle$ or $|1\rangle$ is known as the measurement problem. One example of the measurement problems is Schrodinger's cat (Schrödinger, 1935). There are no definitive answers yet but there are some interpretations.

Figure 2.7 Measurement represented in a circuit[3]

For example, $\alpha|0\rangle + \beta|1\rangle = \begin{bmatrix} \alpha \\ \beta \end{bmatrix}$ implies that $|\alpha|^2$ is a probability of a qubit state being in $|0\rangle$ and $|\beta|^2$ is a probability of a qubit state being in $|1\rangle$. If a number $\frac{1}{\sqrt{2}}$ is

plugged into coefficients α and β, the equation becomes $\frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}$. This means that the probability has equal weight for the Z basis. Measuring the state on a Z basis will give $+Z$ or $-Z$ to the observer with a 50% chance. If X basis is used instead for measurement, then it will always give $+X$.

2.1.6 Entanglement

More than one qubit is in an entangled state when they interact or share spatial proximity in a uniform way that each qubit state cannot be independent of the state of other qubits. This means that measuring the state of one qubit in entangled pair will result in another qubit collapsing as well regardless of the physical distance between them (Bancal, et al., 2012).

Bell States

One common example of an entangled state is the Bell pair or EPR pair (Einstein-Podolsky-Rosen pair) (Einstein, et al., 1935). The simplest way to create an entangled Bell state is to use the CNOT gate and H gate as follows:

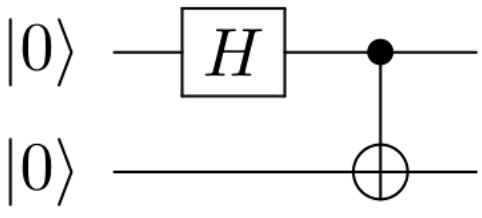

Figure 2.8 Circuit to create Bell state[5]

[5] https://en.wikipedia.org/wiki/Bell_state - Accessed 15th April 2021

The circuit shown above will take two input qubits $|00\rangle$ and transform it to one of the Bell states (equation (2.16)). Explicitly, the first qubit will transform to superposition state $\frac{(|0\rangle+|1\rangle)|0\rangle}{\sqrt{2}}$. This will act as an input to the CNOT gate. When the first qubit is $|1\rangle$, only the second qubits will be inverted. The output of the CNOT gate would be $\frac{(|00\rangle+|11\rangle)}{\sqrt{2}} = |\Phi^+\rangle$.

The following shows the four entangled states that can be used as basis sets. Also known as Bell states.

$$|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle) \quad (2.16)^5$$

$$|\Phi^-\rangle = \frac{1}{\sqrt{2}}(|00\rangle - |11\rangle) \quad (2.17)^5$$

$$|\Psi^+\rangle = \frac{1}{\sqrt{2}}(|01\rangle + |10\rangle) \quad (2.18)^5$$

$$|\Psi^-\rangle = \frac{1}{\sqrt{2}}(|01\rangle - |10\rangle) \quad (2.19)^5$$

2.1.7 Quantum Networking

A Quantum network is like a classical network but is useful for certain kinds of problems. For example, qubits can be sent through the quantum network so quantum computers can solve certain problems faster by connecting several quantum computers (quantum computing cluster) (Tabia, 2011). Another example could be security. Quantum network can be more secure because QKD allows the detection of eavesdropper and theoretically unbreakable cryptography that can be used to encrypt communication (Tomamichel & Leverrier, 2017) and (Portmann & Renner, 2014).

Quantum Superdense Coding

Superdense coding allows one person to send two classical bits to another person by sending only one qubit if each person has a qubit pre-shared in an entangled state (Nielsen & Chuang, 2010).

The protocol involves five steps[6]:

1. Preparation
2. Sharing
3. Encoding
4. Sending
5. Decoding

Preparation requires that two qubits be in one of Bell states. Such as $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle)$. After preparation, one qubit needs to be sent to Alice (qubit B) and another qubit to Bob (qubit A). If Alice wants to send Bob two classical bits, then she needs to encode her local qubit by applying a quantum gate. This will alter the entangled state $|\Phi^+\rangle$ into one of the four Bell states.

$$I|\Phi^+\rangle = |\Phi^+\rangle \coloneqq B_{00} = \frac{1}{\sqrt{2}}(|0_A 0_B\rangle + |1_A 1_B\rangle) \quad (2.20)^6$$

$$X|\Phi^+\rangle = |\Psi^+\rangle \coloneqq B_{01} = \frac{1}{\sqrt{2}}(|0_A 1_B\rangle + |1_A 0_B\rangle) \quad (2.21)^6$$

$$Z|\Phi^+\rangle = |\Phi^-\rangle \coloneqq B_{10} = \frac{1}{\sqrt{2}}(|0_A 0_B\rangle - |1_A 1_B\rangle) \quad (2.22)^6$$

[6] https://en.wikipedia.org/wiki/Superdense_coding - Accessed 15th April 2021

$$Z * X|\Phi^+\rangle = |\Psi^-\rangle \coloneqq B_{11} = \frac{1}{\sqrt{2}}(|0_A 1_B\rangle - |1_A 0_B\rangle) \quad (2.23)^6$$

Now, Alice just needs to send her qubit to Bob. Bob will decode the qubit B by applying CNOT unitary operation with qubit A as a control qubit and qubit B as a target qubit. Then he will perform $H \otimes I$ unitary operation on qubit A. This means the H gate is only applied to qubit A.

For example, if Alice got state B_{01} after encoding her qubit, when Bob applies CNOT gate with qubit A as control bit and B as target then B_{01} will transform to

$$B'_{01} = \frac{1}{\sqrt{2}}(|0_A 1_B\rangle - |1_A 0_B\rangle) \quad (2.24)^6$$

Now, when the H gate is only applied to qubit A:

$$B''_{01} = \frac{1}{\sqrt{2}}\left(\frac{1}{\sqrt{2}}(|01\rangle + |11\rangle) + \frac{1}{\sqrt{2}}(|01\rangle - |11\rangle)\right) \quad (2.25)^6$$
$$= \frac{1}{2}|01\rangle + \frac{1}{2}|11\rangle + \frac{1}{2}|01\rangle - \frac{1}{2}|11\rangle = |01\rangle$$

Now that Bob has basis state $|01\rangle$, Bob knows that Alice wanted to send classical bits 01.

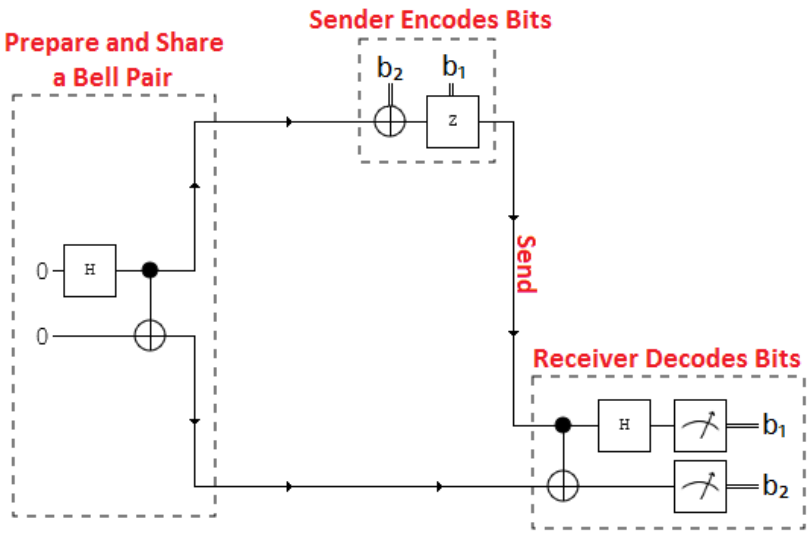

Figure 2.9 Circuit representation of superdense coding[6]

Quantum Teleportation

It is possible to use gate operation to send quantum information between short distances and long distances. It can be thought of as the opposite of superdense coding. Superdense coding sends two classical bits using one qubit, but teleportation sends one qubit using two classical bits (Bennet, et al., 1993). The requirement of sending two classical bits forbids the transmission of information faster than light.

The protocol involves five steps[7]:

1. Preparation
2. Sharing
3. Encoding
4. Sending
5. Decoding

Step 1 and 2 is the same as superdense coding. If Alice has q1 of Bell pair and Bob has q2 of Bell pair. And if Alice wants to send a qubit ($|\Psi\rangle$) to Bob, then she needs to apply a CNOT gate to q1 controlled by $|\Psi\rangle$. Then she applies H gate to $|\Psi\rangle$. Next, she will measure both qubit q1 and $|\Psi\rangle$ which the result will be stored as two classical bits. Now, Alice will perform step 4 and send the classical bits to Bob through classical communication. After Bob has received two classical bits, he will apply quantum gates on his qubit (q2) depending on the classical bits as follows:

Table 2.2 Shows which quantum gate to use based on classical bits.

$00 \rightarrow Do\ nothing$[7]
$01 \rightarrow Apply\ X\ gate$[7]

[7] https://en.wikipedia.org/wiki/Quantum_teleportation - Accessed 15th April 2021

$10 \rightarrow Apply\ Z\ gate$[7]
$11 \rightarrow Apply\ XZ\ gate$[7]

After applying quantum gates, Bob will have the qubit that Alice has sent.

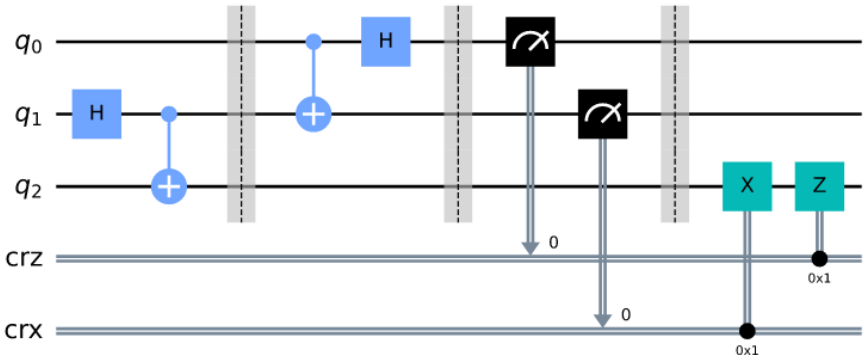

Figure 2.10 Circuit representation of quantum teleportation[8]

2.2 Attacks on BB84 QKD

In this section, the author will introduce both theoretical attacks and experimentally verified attacks.

2.2.1 Intercept and Resend

This is the simplest type of attack. When Alice sends qubits to Bob, Eve can intercept and measure the state [9]. Then Eve can send a replacement state to Bob. However, Eve can be detected by Alice and Bob as Eve must guess which sending basis Alice used. And Eve will have a 50% chance to guess correctly.

[8] https://qiskit.org/textbook/ch-algorithms/teleportation.html - Accessed 15th April 2021

[9] https://en.wikipedia.org/wiki/Quantum_key_distribution - Accessed 15th April 2021

For example, Alice generates basis +×+× and qubits 0101 to send it to Bob. Then Bob will generate basis ×++× and measures the bits which will be 1001. When Alice and Bob compare their measured bits, they should get the same bit for the same basis. However, if Eve intercepts their communication, Eve will generate a basis ++×+ and measure the bits before Bob receives the qubits. Eve measures 0011 and when Bob measures, he will get 1000. So, Alice and Bob know that since both got basis × but the result was different, there must be an interceptor.

The probability that Eve guesses incorrectly is 50%. And Bob also has a probability of 50% to guess correctly which means that the probability of an intercepted qubit generating an error in the key is $\frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$. Therefore, if Alice and Bob compare n bits, the probability of identifying the presence of Eve is

$$P_d = 1 - \left(\frac{3}{4}\right)^n \qquad (2.26)^9$$

2.2.2 Man in the Middle Attack

This is like classical networks. If a communication channel is used without authentication, then someone could be in the middle of communication without anyone knowing[9]. This is because quantum mechanics cannot distinguish different people with different intentions just like in a classical network.

2.2.3 Photon Number Splitting Attack

Implementation of quantum computer that send photons use laser pulses to send quantum states. Each laser pulses contain a small number of photons which is distributed according to Poisson distribution[9]. This means that there may be no photons,

1 photon or 2 photons per pulse. When Alice sends a quantum state using photons, extra photons can be split off by Eve, stored in a quantum memory, and the remaining will go to Bob. Eve can later measure the qubits on the correct basis when Alice reveals the encoding basis.

Even with this attack, the generation of the secure key is possible by having a higher number of qubits to send which reduces the rate of generation of secure keys exponentially. The rate of secure key generation is reduced by t^2 with the attack. The rate of secure key generation is reduced by t without the attack, where t is the transmittance of the quantum channel.

2.2.4 Denial of Service Attack

By cutting off or blocking the line that connects two nodes could route the communication elsewhere[9]. Such as another line or classical communication line which could be used to do further attacks.

2.2.5 Trojan Horse Attack

Eve can send bright light through the quantum channel to probe the system by analysing the back-reflections[9]. Eve has a higher than 90% probability to discern Bob's secret basis choice (Jain, et al., 2014).

2.2.6 Random Number Generator Attack

Almost all random number generator attacks require access to the system before performing the attack whether it is remotely or physically[9]. One example is when the

attacker obtains a stream of random bits and uses this to predict the future output. This could be used to do man in the middle attack on the classical communication channel. Another example could be modified to the input of a random number generator. Emptying the entropy and putting it in a known state. This could be used to know the random encoding basis or random bits that Alice sends to Bob to create a secure key.

2.2.7 Physical Attack

Physical attacks involve gaining access to the victim's device physically[9]. With this attack, the attacker can impersonate the victim, get random encoding basis and random bits, create a backdoor and other things which can compromise the communication between the victim and another person.

3 Methodology

This section focuses on the discussion, data collection, analysis and evaluation of the methodology used for this research.

3.1 Discussion of Methodology

The aim of this research is to design and simulate attacks against the QKD algorithm and to create a mitigation plan against such attacks as stated in section 1.3. To fulfil this aim, the first step is to design attacks whether they are existing or new. Then the effectiveness of each attack will be measured by using various criteria. The collected data will then be analysed to ensure that they are correct by confirming with their respective formula. Lastly, each attack will be put into categories. The steps will be repeated with and without a mitigation plan.

Overall, the methodology follows the waterfall model as shown in Figure 3.1. The reason the waterfall model (Royce, 1970) was chosen is that there are no changing requirements that need to be accounted for in this research.

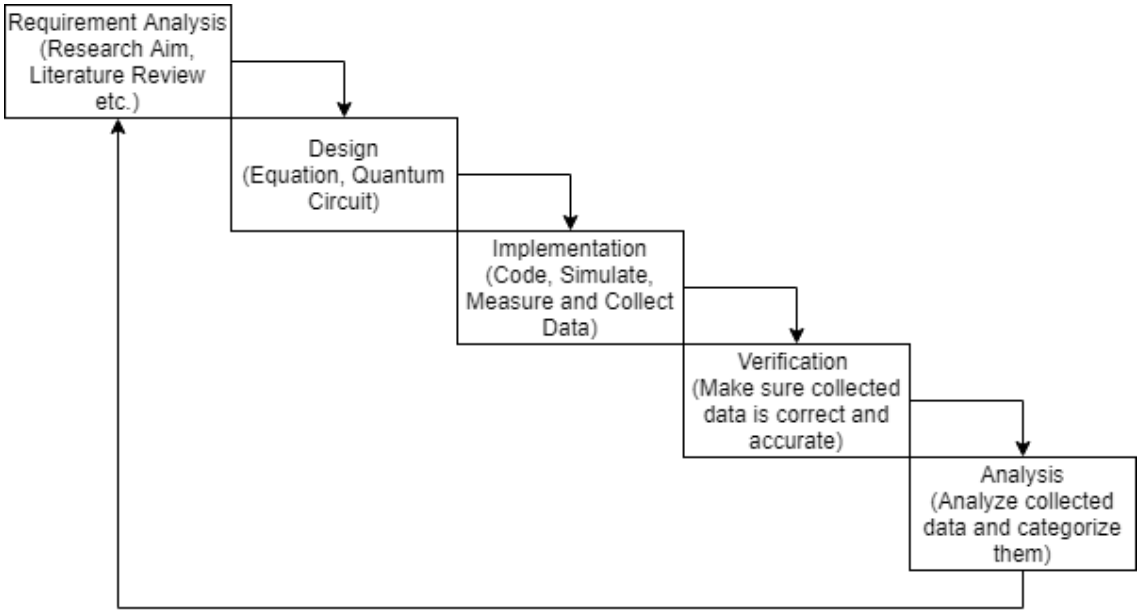

Figure 3.1 Showing the waterfall model of this research methodology.

3.2 Data collection strategy

This section explains the process up to the collection of measurement. Primary data will be collected from experiments where possible and secondary data will be collected for the rest. However, some attacks may not have secondary data. In this case, the author will provide a reasonable guess based on arguments that will be stated.

3.2.1 Proposed Design

The attacks will be designed in form of the equation, quantum circuit diagram then the code for simulation.

Equation

By designing the attacks in form of an equation, it is easier to verify if the attacks or measurement are correct. Equations also help with seeing patterns which can help with

relationships or correlation between different numbers. As an example, equation (3.1) shows the probability of Bob guessing the basis correctly. This will be the probability without the attack. When Eve tries to intercept the communication, Eve will have a 50% chance of guessing the basis incorrectly as shown in equation (3.2). In order to get the probability of a qubit generating an error in secure key, both equation (3.1) and (3.2) are combined as shown in equation (3.3). Therefore, the probability of identifying the presence of Eve per qubit is $1 - \frac{3}{4}$ as shown in equation (3.4). Finally, if Alice and Bob compare *n* bits then the probability of detecting the presence of Eve is shown in equation (3.5).

$$B_g = 1 - \frac{1}{2} \tag{3.1}$$

$$E_g = 1 - \frac{1}{2} \tag{3.2}$$

$$Q_e = B_g \times E_g = \left(1 - \frac{1}{2}\right)\left(1 - \frac{1}{2}\right) = \frac{1}{4} \tag{3.3}$$

$$Q_e = \frac{1}{4} = 1 - \frac{3}{4} \tag{3.4}$$

$$P_d = 1 - \left(\frac{3}{4}\right)^n \tag{3.5}$$

Quantum Circuit Diagram

After the equation, a quantum circuit diagram will be created based on the equation and descriptions of the attack. This will help to see how the qubits will move and with creating the code for simulation. A Quantum circuit diagram will also show where or when the measurement is being made which will be helpful. For example, the quantum circuit diagram for intercept and resend attack is shown below.

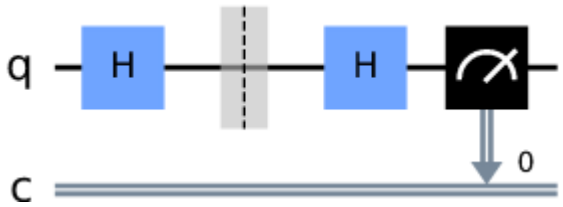

Figure 3.2 Circuit representation of simulation of BB84 protocol without interception[10]

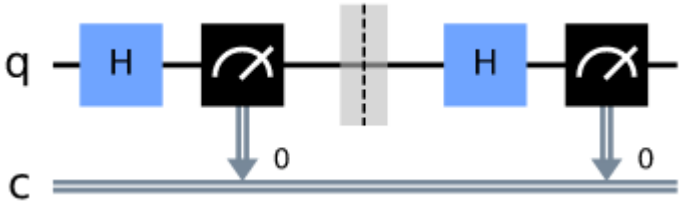

Figure 3.3 Circuit representation of simulation of BB84 protocol with interception[10]

The attacks will be designed with a quantum circuit diagram without interception then with an interception as shown in Figure 3.2 and Figure 3.3.

Code

Then, the code will be implemented based mostly on a quantum circuit diagram while verifying with the equation that the code does what it is supposed to do. Since the code must be accurate, portable, and measurable the code itself will follow the waterfall model to make sure that the code meets the requirements.

[10] https://qiskit.org/textbook/ch-algorithms/quantum-key-distribution.html - Accessed 15th April 2021

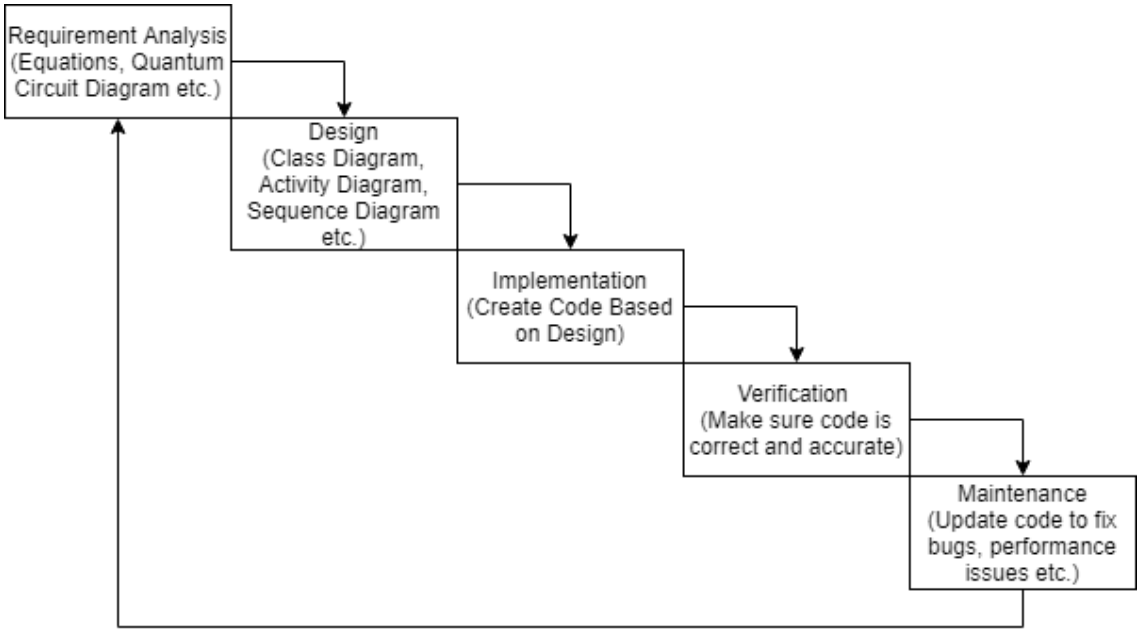

Figure 3.4 Waterfall model for creating code.

Since, the code will be based on the description, equation and quantum circuit diagram, the requirement analysis has mostly been done. The second step is to design the code, and this will involve creating multiple UML diagrams to ensure that code is portable, accurate and performant. Then the code will be implemented based on the UML diagrams. The code will then be run multiple times with different cases to ensure that it is working as intended. The final stage in the waterfall model is to maintain the code and to fix any issues that may arise.

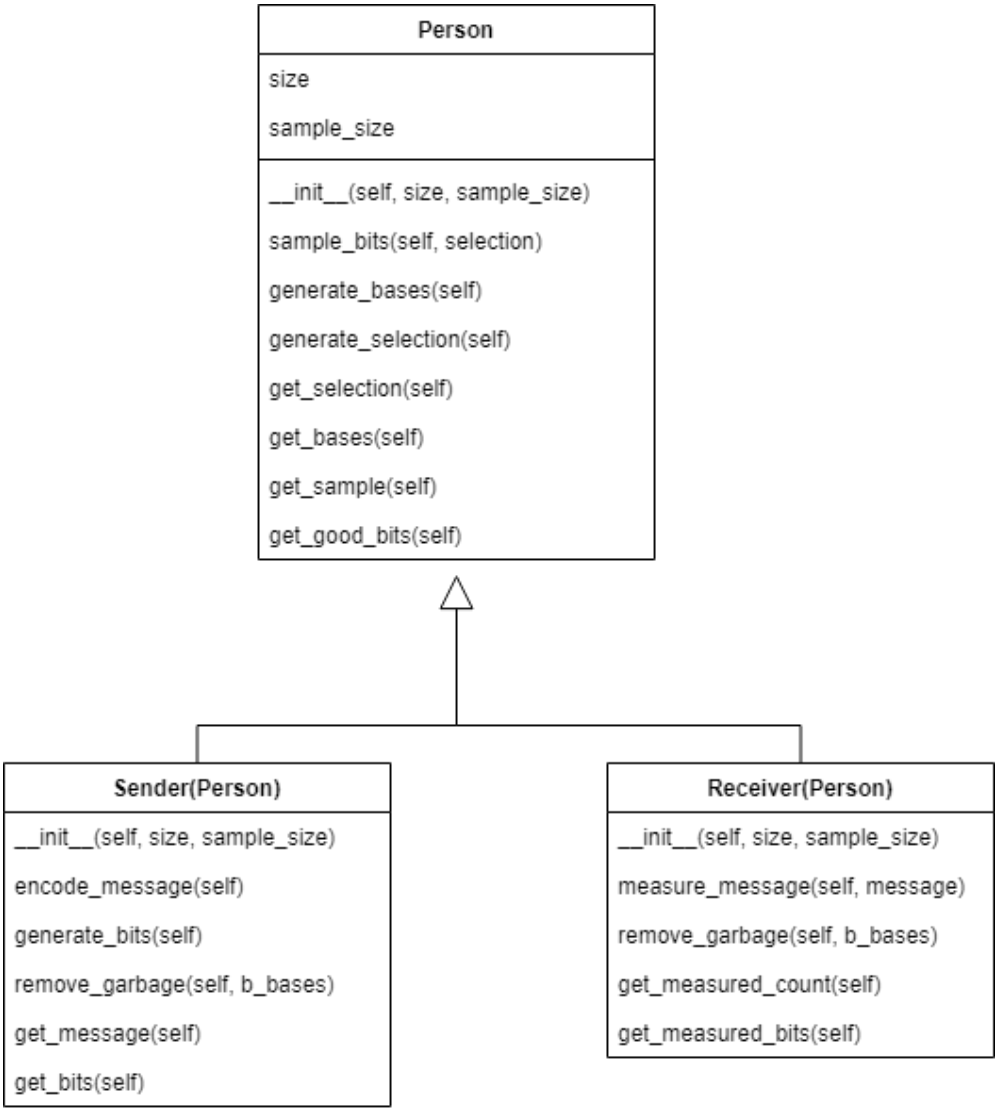

Figure 3.5 Class diagram for simulation both with and without interception

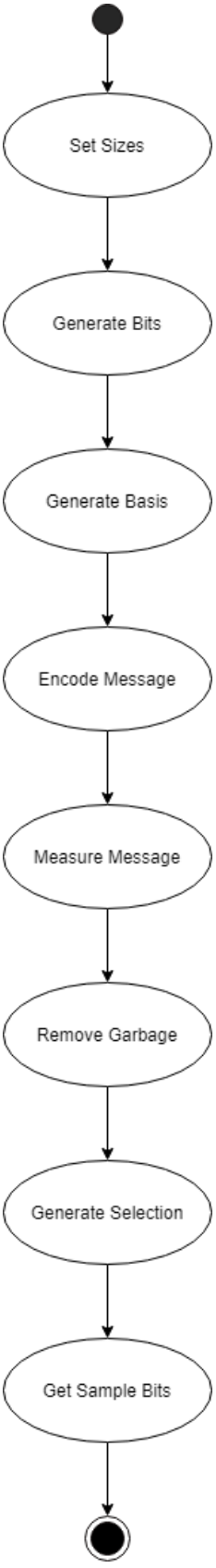

Figure 3.6 Activity diagram for simulation without interception

Only class diagram and activity diagram are produced because the other UML diagrams are not necessary for this research. Class diagram and activity diagram provides most of the necessary information and quantum circuit diagram acts similarly to use case diagram and sequence diagram for this research.

Sources of Code

The final code will be produced using many sources such as the internet and other research papers as too much time could be devoted to the development of the code that may be freely available on the internet. However, most sources except the Qiskit website have outdated code that does not work anymore. This implies that the code produced from this research will be partly from Qiskit website [11] which will be referenced in the code.

3.2.2 Measurement

The Quantum circuit diagram will have a measurement icon (see Figure 3.2 and Figure 3.3) that shows when the measurement is being made. The code will show the output of these measurements to confirm if the attack has been successful or not. The code will contain extra measurements that will be made internally which will be used to determine that code is correct and how the attack performs. Examples of internal measurements are the basis, bits and other numbers used to generate basis and bits.

[11] https://qiskit.org/textbook/ch-algorithms/quantum-key-distribution.html - Accessed 21st April 2021

3.3 Analysis of Collected Data

This section explains the analysis method of collected data.

3.3.1 Confirmation with Equation

The first step is to confirm the collected data against the equation which is known to be correct. This will ensure that the data is sufficiently accurate. The same input that is used in the code will be given to the equation and the equation should output the same result as the code.

3.3.2 Proposed Categorization Method

After the data is found to be correct, it will get put into three groups that are ordered from easiest to hardest attacks, highest to lowest success rate and success rate of attacks from easiest to hardest attacks.

Table 3.1 Example table for organising the attacks from easiest to hardest.

Rank	Name of Attack	Points
1		
2		
3		
4		
5		
6		

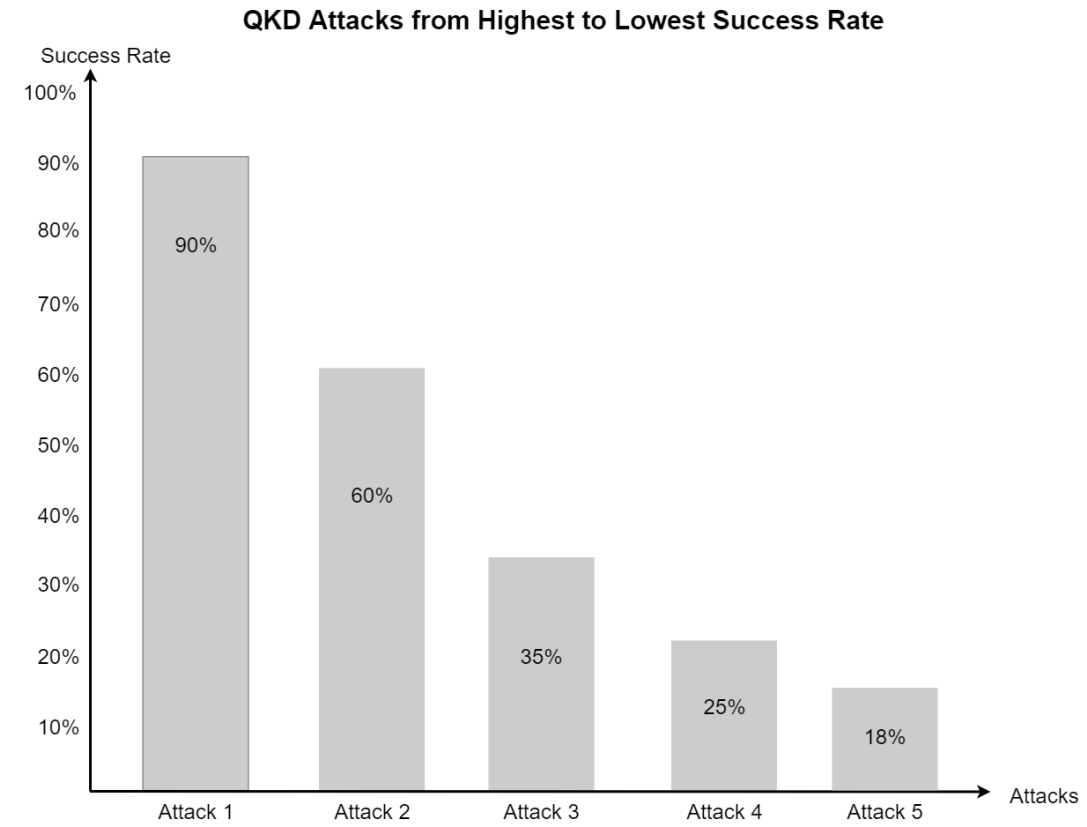

Figure 3.7 Example graph for organising attacks from highest to lowest success rate.

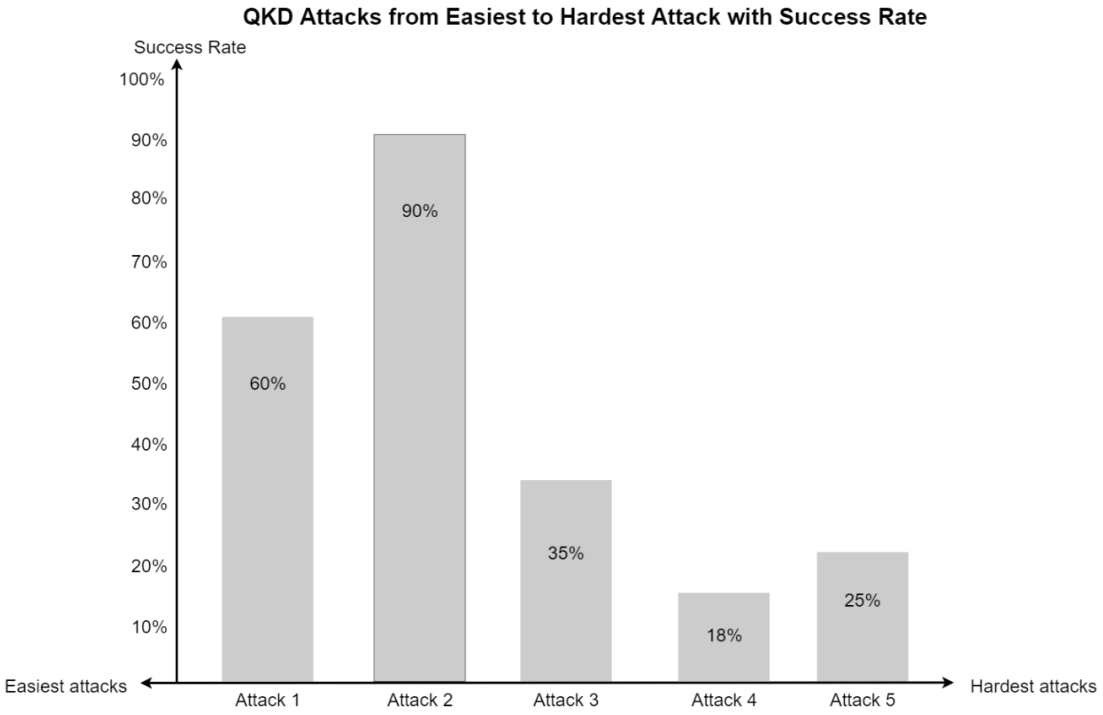

Figure 3.8 Example graph for organising attacks from easiest to hardest attacks with success rate.

These three categories were chosen because they are simple and easy to see the variations in data. For example, even if an attack is easy, it might not be of concern because of its low success rate. However, if there are attacks that have a very high success rate with moderate difficulty can have significant consequences.

Criteria of Difficulty of Attacks

The following are the criteria that will be used to determine if an attack is easy or hard.

- Proximity
- Privilege
- knowledge

Proximity is the distance that is required to conduct an attack. An attack could be fully remote, partly remote (for example, outside the house or in a local area network only) or physical access to the device required. It will be measured on a scale from 0 to 10.

Privilege is the level of control that the attacker gets. The attacker could get metadata, access to the browser or full access to the device. It will be measured on a scale from 0 to 10.

Knowledge is the knowledge required to conduct an attack. In case of attacks involving quantum computers, an attacker may need to know some quantum mechanics and mathematics in addition to computer science. Certain attacks such as denial of service attack are easier than interception, man in the middle attacks or photon number

splitting attack because the former requires less knowledge compared to the latter attacks. It will be measured on a scale from 0 to 10.

Criteria of Success Rate of Attacks

Success rate or percentage will be determined purely by empirical observation of simulation of the attacks for those that can be simulated. Both with and without the mitigation plan. For those that cannot be simulated, the author will provide a reasonable guess based on arguments that will be stated.

3.4 Tools Used

In this section, the author will introduce tools that are used in this research.

3.4.1 Python

Python is a general-purpose high level interpreted language. Python's design philosophy is code readability and requires that the code be indented properly[12], or the program will not run. This programming language was chosen because of its ease of use and extensive support for many libraries, frameworks and modules that can be used to analyse data and create diagrams. In this research, Python version 3.8 is used.

3.4.2 Qiskit

Qiskit is an SDK that has all the tools that are needed to create a quantum circuit, run attack simulations, analyse data, and create diagrams. Qiskit allows working with

[12] https://en.wikipedia.org/wiki/Python_(programming_language) – Accessed 20th April 2021

pulses, circuits, and application modules on a real quantum computer with both trapped ion architecture and superconducting architecture. In this research, Qiskit version 0.25.0 is used. The specific versions are as follows:

- Qiskit-terra – 0.17.0
- Qiskit-aer – 0.8.0
- Qiskit-ignis – 0.6.0
- Qiskit-ibmq-provider – 0.12.2
- Qiskit-aqua – 0.9.0

3.4.3 IBM Quantum Lab

IBM Quantum Lab is a cloud programming environment that allows the building of quantum applications and experiments. Without any installation, scripts that combine Qiskit code, equations, visualizations, and narrative text can be created in a Jupyter Notebook environment[13]. The Notebook can then be run on a real quantum computer or a quantum computer simulator. The Notebook can be stored, accessed, and managed from IBM Quantum Lab. It requires an account that is free to sign up. Qasm simulator is used to run the simulation. It is not a real quantum computer and does not have qubit errors.

3.4.4 Miscellaneous

To create UML diagrams and graphs, a website called draw.io has been used.

[13] https://quantum-computing.ibm.com/lab/docs/iql/ - Accessed 20th April 2021

3.5 Evaluation of Methodology

The waterfall approach is the most effective because there are no changing requirements. It is simple and ensures that all the process are followed exactly to be correct and accurate. It also covers everything that is required from start to finish which acts as a guide. This makes the development and experiments faster with accurate data.

4 Implementation and Results

This section introduces the testing of code, results of the experiment and result of the categorization.

4.1 Testing

Testing will be done with measurements of simulation and equation. Both with and without an interception. The seed for the pseudo-random number generator will be set to 5 for reproducibility. The test is also done without the seed being set at runtime. However, this produces different output every run so is not recorded in the tables below.

4.1.1 Testing without Interception

Table 4.1 Testing table for simulation of attack without an interception.

Test No.	Description	Measurements	Confirm mathematically	Passed Test
1	Alice generates a random basis.	See Figure 4.2 labelled “Alice basis”.	Since there is a 50% chance of getting 0 or 1, the basis will have almost 50% 0s and almost 50% 1s. See Figure 4.2.	Yes
2	Bob generates a random basis.	See Figure 4.3 labelled “Bob basis”.	Since there is a 50% chance of getting 0 or 1, the basis will have almost 50% 0s and almost 50% 1s. See Figure 4.3.	Yes
3	Alice generates a random message.	See Figure 4.2 labelled “Alice random message”.	Since there is a 50% chance of getting 0 or 1, the message will have almost 50% 0s and almost 50% 1s. See Figure 4.2.	Yes
4	Bob measures the message from Alice based on Bob's basis.	See Figure 4.3 labelled “Result of Bob measurement”.	Since there is a 50% chance of getting 0 or 1, the measured bits will have almost 50% 0s and almost 50% 1s. See Figure 4.3.	Yes
5	Alice and Bob generate good bits.	See Figure 4.4 labelled “Alice good bits” and “Bob good bits”.	Since there is a 50% chance of getting 0 or 1, the good bits will have almost 50% of the total number of message bits. See Figure 4.4.	Yes
6	Alice and Bob choose random bits for key bits selection.	See Figure 4.5 labelled “Alice selection” and “Bob selection”.	Not applicable	Yes
7	Alice and Bob get sample bits.	See Figure 4.5 labelled “Bob sample” and “Alice sample”.	Not applicable	Yes

```
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
The M attribute was deprecated in Matplotlib 3.4 and will be removed two minor releases later. Use self.axes.M instead.
  x_s, y_s, _ = proj3d.proj_transform(xs3d, ys3d, zs3d, renderer.M)
/opt/conda/lib/python3.8/site-packages/qiskit/visualization/bloch.py:69: MatplotlibDeprecationWarning:
```

Figure 4.1 **States of the qubit cannot be shown because of error. It is possible to fix the error however, it is outside the scope of this research.**

```
Alice basis: [0 1 0 1 0 1 0 0 0 1 0 1 0 0 0 1 1 0 1 0 0 0 0 0 0 1 0 1 1 1 0 0 1 0 0 1 1
 0 0 1 0 1 0 1 1 0 1 1 0 0 0 1 0 1 1 0 0 1 0 0 1 0 1 0 1 1 0 0 1 0 1 0 0 0
 1 1 0 0 0 1 1 0 0 0 0 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0]
Alice number of 0s in basis: 59
Alice number of 1s in basis: 41
Alice random message: [1 0 1 1 0 0 0 1 0 0 1 0 1 0 0 0 1 0 1 1 1 1 0 1 0 1 1 1 1 0 0 0 1 0 0 1 1
 0 1 1 0 1 0 1 0 0 1 0 0 0 0 0 0 1 1 1 1 0 0 0 1 1 1 0 1 1 1 0 0 1 0 0 1 1
 1 0 1 1 1 1 0 1 1 1 1 1 1 0 0 0 0 1 1 0 1 0 1 0 1 1]
```

Figure 4.2 **Showing the internal measurements made for sending a message.**

```
Bob basis: [0 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 1 1 1 1 1 0 0 1 0 0 0 1 1 1 1 0 0 1 1 1 0
 1 1 1 0 0 1 1 1 0 0 0 0 0 1 1 0 1 1 1 0 0 1 0 1 0 0 1 0 0 0 1 0 1 1 0 1 0
 0 1 1 1 0 1 1 1 0 1 0 0 0 1 0 0 1 1 1 0 1 1 0 1 1 0]
Bob number of 0s in basis: 46
Bob number of 1s in basis: 54
Result of Bob measurement: [1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 1,
1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 1, 0, 1]
Bob number of 0s in result: 46
Bob number of 1s in result: 54
```

Figure 4.3 **Showing the internal measurements made for measuring message.**

```
Alice good bits: [1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1]
Bob good bits:   [1, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1,
0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1]
Alice good bits = Bob good bits: True
Number of good bits: 51
```

Figure 4.4 **Showing the internal measurements made for checking good bits.**

```
Alice selection = [83 55  3 23  3 95  2 54 93 38 18 71 64 35 37]
Bob selection =   [83 55  3 23  3 95  2 54 93 38 18 71 64 35 37]
Alice sample = [1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1]
Bob sample =   [1, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1]
Alice sample = Bob sample: True
```

Figure 4.5 **Showing the internal measurements made for selecting sample bits.**

4.1.2 Testing with Interception

Table 4.2 Testing table for simulation of attack with an interception.

Test No.	Description	Measurements	Confirm mathematically	Passed Test
1	Alice generates a random basis.	See Figure 4.2 labelled “Alice basis”.	Since there is a 50% chance of getting 0 or 1, the basis will have almost 50% 0s and almost 50% 1s. See Figure 4.2.	Yes
2	Bob generates a random basis.	See Figure 4.3 labelled “Bob basis”.	Since there is a 50% chance of getting 0 or 1, the basis will have almost 50% 0s and almost 50% 1s. See Figure 4.3.	Yes
3	Eve generates a random basis.	See Figure 4.6 labelled “Eve basis”.	Since there is a 50% chance of getting 0 or 1, the basis will have almost 50% 0s and almost 50% 1s. See Figure 4.6.	Yes
4	Alice generates a random message.	See Figure 4.2 labelled “Alice random message”.	Since there is a 50% chance of getting 0 or 1, the message will have almost 50% 0s and almost 50% 1s. See Figure 4.2.	Yes
5	Eve measures the message from Alice based on Eve's basis.	See Figure 4.6 labelled “Result of Eve measurement”.	Since there is a 50% chance of getting 0 or 1, the measured bits will have almost 50% 0s and almost 50% 1s. See Figure 4.6	Yes
6	Bob measures the message from Alice based on Bob's basis. However, Eve has already measured so Bob should get a different outcome.	See Figure 4.7 labelled “Result of Bob measurement”.	Since there is a 25% chance of getting 0 or 1 because of interception, the measured bits will have almost 25% correct compared to Alice. See Figure 4.7.	Yes
7	Alice and Bob generate good bits.	See Figure 4.4 labelled “Alice good bits” and “Bob good bits”.	Since there is a 50% chance of getting 0 or 1, the good bits will have almost 50% of the total number of message bits. However, the error rate	Yes

			should be almost 50%. See Figure 4.4.	
8	Alice and Bob choose random bits for key bits selection.	See Figure 4.5 labelled “Alice selection” and “Bob selection”.	Not applicable	Yes
9	Alice and Bob get sample bits.	See Figure 4.5 labelled “Bob sample” and “Alice sample”.	Not applicable	Yes

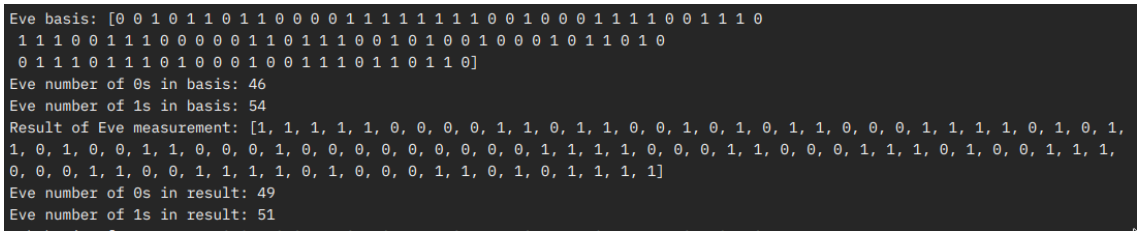

Figure 4.6 Showing the internal measurements made for Eve.

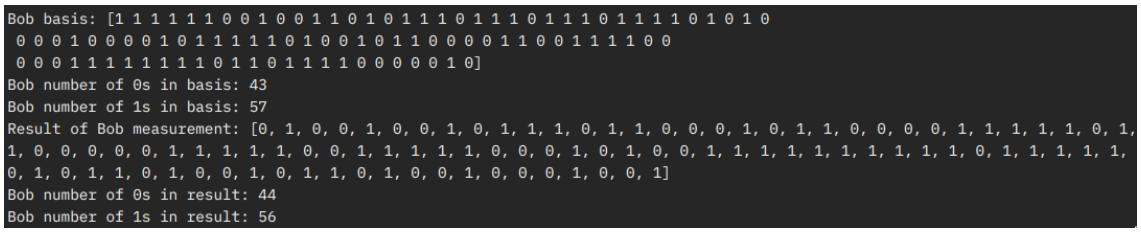

Figure 4.7 Showing the internal measurements made for Bob during the intercept attack.

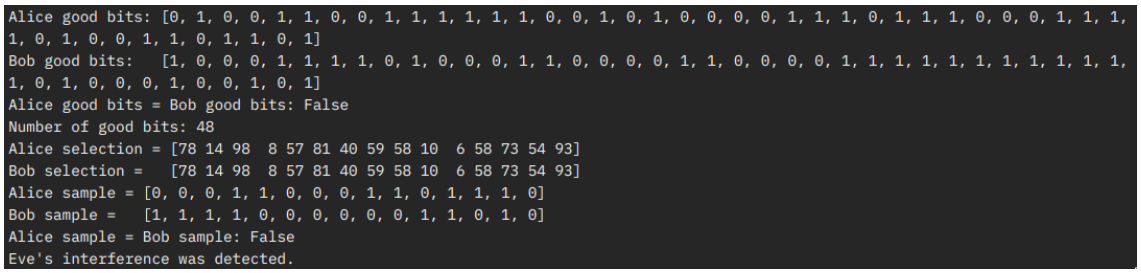

Figure 4.8 Showing the internal measurements made for selecting sample bits with an interception.

4.2 Results

In this section, the result of the experiment will be introduced then the result of the categorization of various attacks.

4.2.1 Result of the Experiment

Without Interception

First, simulation without an interception is shown.

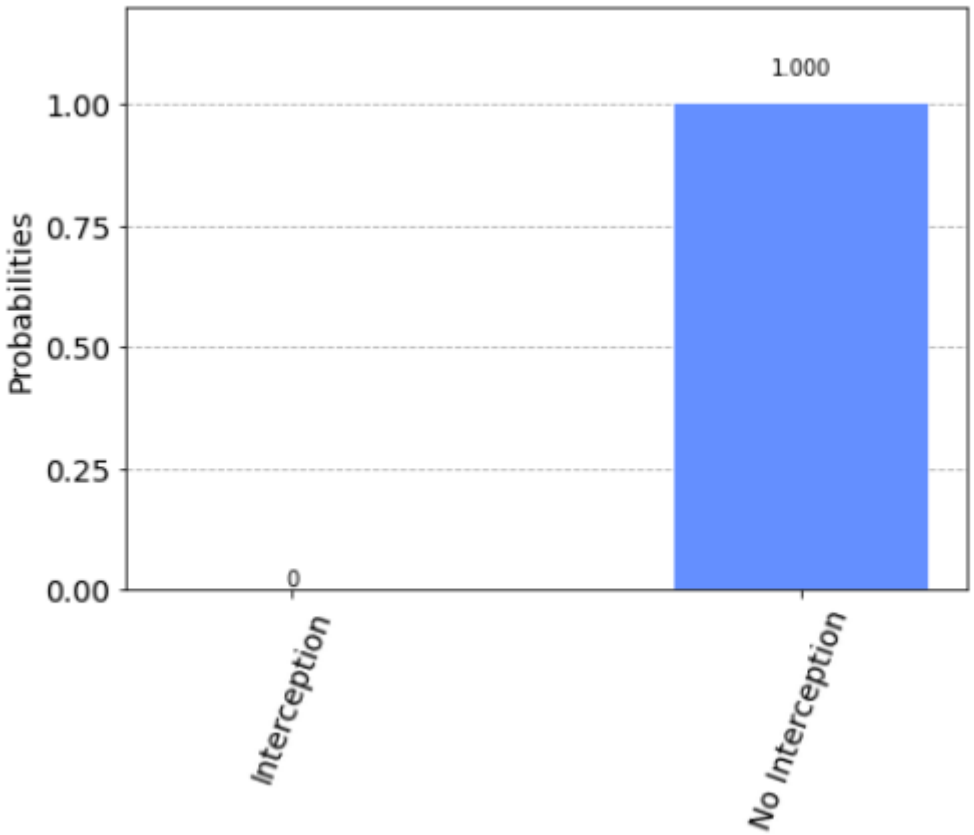

Figure 4.9 Showing that without an interception, there is no interception.

Figure 4.9 shows that there are no interceptions because Eve is not intercepting. If Eve were to intercept, then the interception bar would not be 0.

With Interception

Secondly, simulation with interception is shown below.

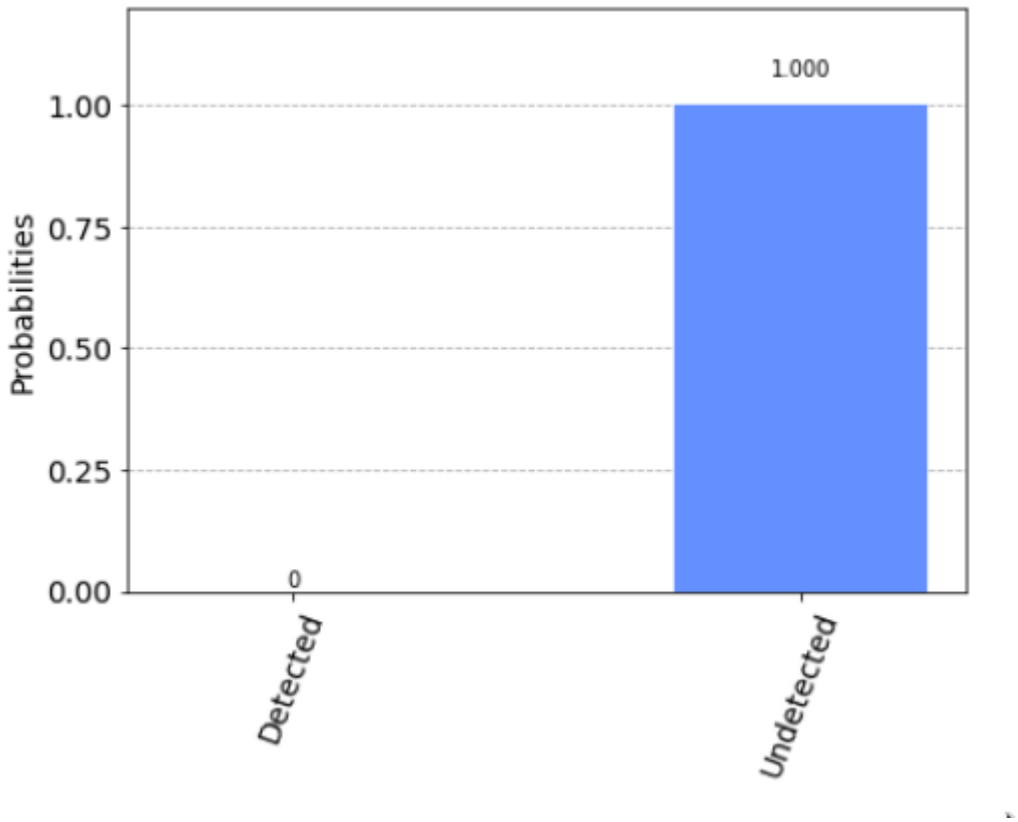

Figure 4.10 Probability of detecting Eve for one qubit.

The result is like equation (3.5). However, it is not exactly like equation (3.5). The probability of Eve going undetected is around 100% for one qubit. Whereas equation (3.5) shows that the probability should be 75% for one qubit. See Figure 4.10.

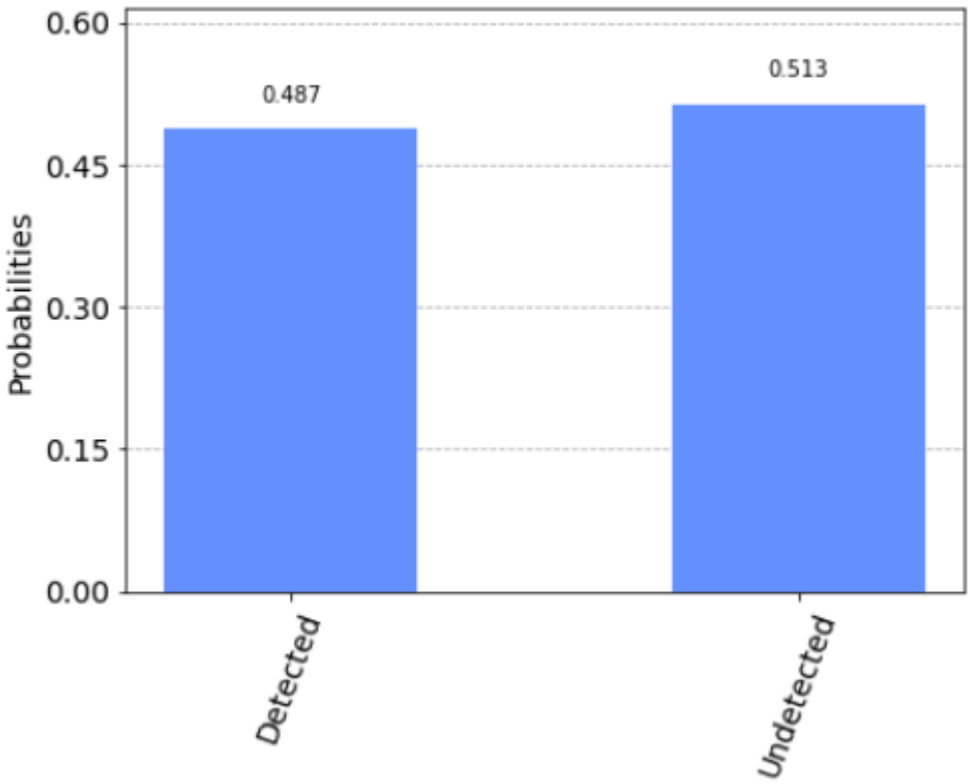

Figure 4.11 Probability of detecting Eve for two qubits.

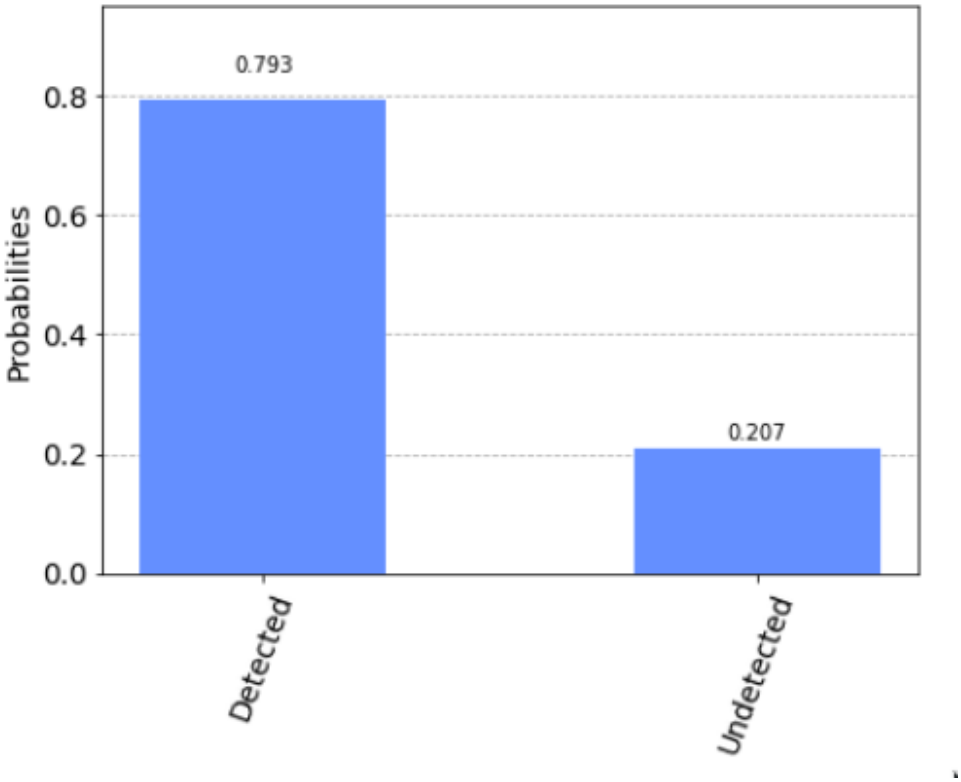

Figure 4.12 Probability of detecting Eve for three qubits.

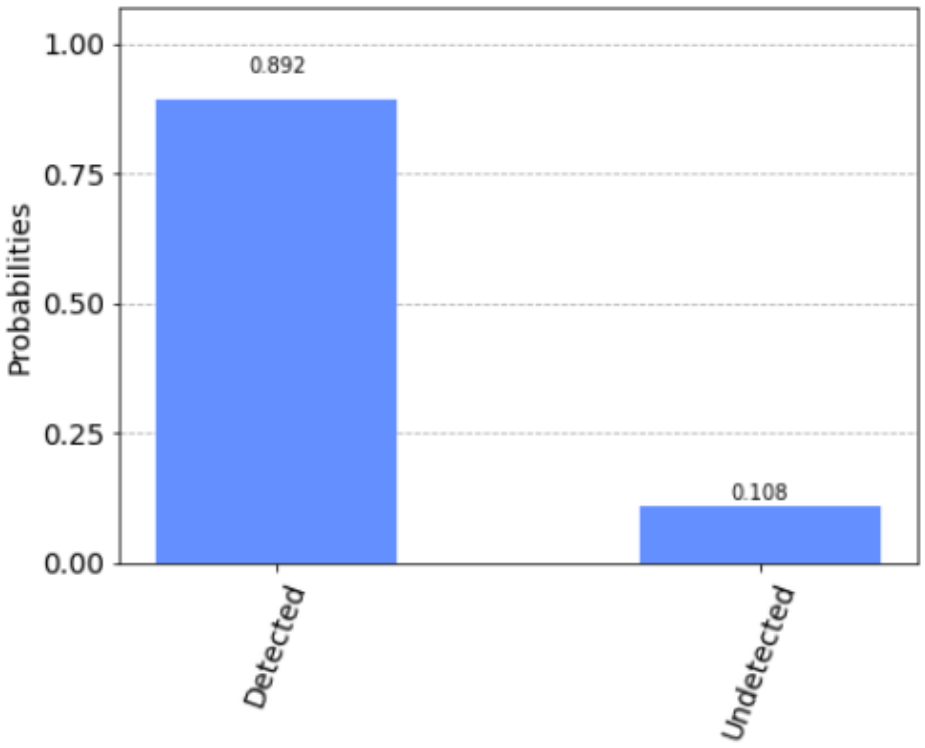

Figure 4.13 Probability of detecting Eve for four qubits.

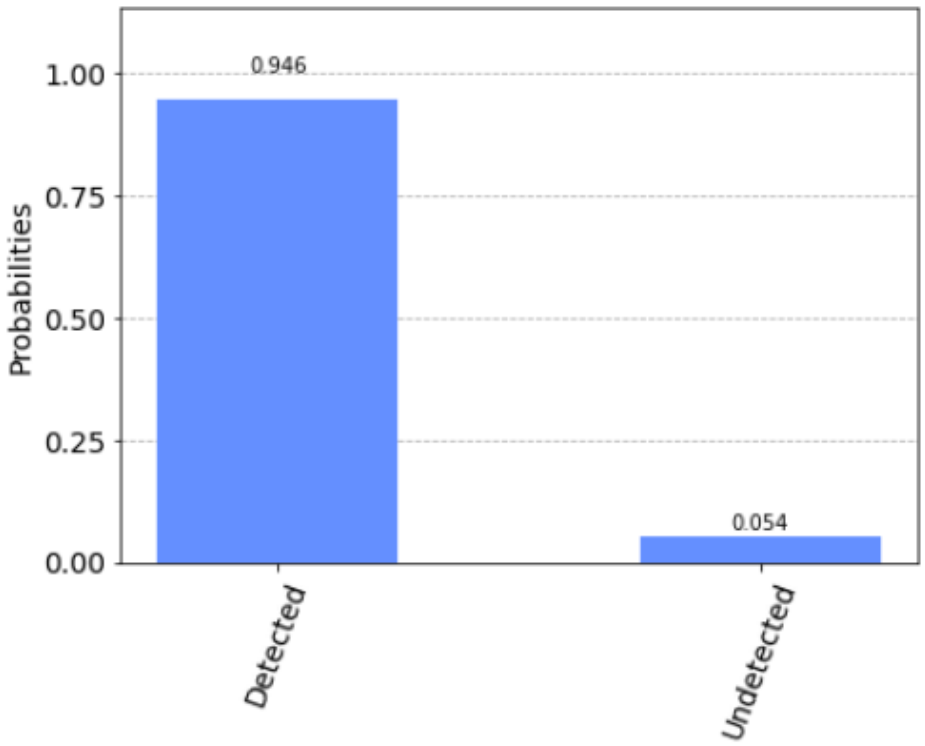

Figure 4.14 Probability of detecting Eve for five qubits.

However, the probability of Eve being undetected goes down exponentially correlated with the number of qubits see Figure 4.11 to Figure 4.14. This is the same as what the equation has predicted.

Since the intercept and resend attack relies on guessing and measuring qubits based on a basis, this attack cannot be tested without a mitigation plan. This is due to how quantum mechanics work and QKD will automatically mitigate against these types of attacks. QKD is the mitigation plan against these types of attacks. However, attacks that require a mitigation plan will be introduced in the next section.

4.2.2 Result of the Categorization

First, tables or graphs will be shown. This will be the result of this research. Then explanation will be provided for each attack and their rank in tables or graphs.

Attacks from Easiest to Hardest

Table 4.3 Attacks ranked from easiest to hardest.

Rank	Name of Attack	Points
1	Denial of Service	20
2	Intercept and Resend	20
3	Man in the Middle	19
4	Trojan Horse	16
5	Photon Number Splitting	15

6	Random Number Generator	15
7	Physical	12

Intercept and Resend Attack

- Proximity – 10 points were given because it can be fully remote.
- Privilege – 3 points were given because this attack does not give lots of access to the victim's computer. Only the reading of data can be done.
- Knowledge – 7 points were given because it does not require in-depth knowledge to do this attack.

The total is 20 points.

Man in the Middle Attack

- Proximity – 10 points were given because it can be fully remote.
- Privilege – 4 points were given because this attack does not give lots of access to the victim's computer. However, manipulation of a victim may provide more privilege.
- Knowledge – 5 points were given because it requires some in-depth knowledge to do this attack. Such as psychology to manipulate victims. Or knowledge to send convincing data.

The total is 19 points.

Photon Number Splitting Attack

- Proximity – 7 points were given because it can be mostly remote. Further research is needed however since the attack needs to be directly connected to the victim, this attack may not be doable over the internet.
- Privilege – 4 points were given because this attack does not give lots of access to the victim's computer. Only the reading of data can be done. However, it does not reveal the attacker's presence to the victims.
- Knowledge – 4 points were given because it requires some in-depth knowledge to do this attack. Such as the probability of two photons being emitted which depends on the hardware.

Total is 15 points.

Denial of Service

- Proximity – 10 points were given because it can be fully remote.
- Privilege – 2 points were given because this attack does not give any access to the victim's computer by itself.
- Knowledge – 8 points were given because it does not require any in-depth knowledge to do this attack.

The total is 20 points.

Trojan Horse

- Proximity – 5 points were given because it can be somewhat remote. This attack requires that the attacker be in a local area network or have a direct connection.
- Privilege – 8 points were given because this attack gives lots of access to the victim's computer. Since the attacker can read data directly from the victim's machine, there could be other valuable data that can be obtained. There is a possibility to modify the state of the victim's machine which would provide the attacker with a lot of privilege.
- Knowledge – 3 points were given because it requires in-depth knowledge to do this attack for now.

The total is 16 points.

Random Number Generator

- Proximity – 5 points were given because it can be mostly remote. However, doing this attack remotely requires that the attacker have access to the victim's machine. Or the attacker needs to have physical access to the machine.
- Privilege – 3 points were given because this attack does not give lots of access to the victim's computer by itself. However, this can be used in combination with other attacks to have almost total control over the victim's machine. For example, have the victim generate a password with a known initial random number generator state.
- Knowledge – 7 points were given because it requires in-depth knowledge to do this attack.

Total is 15 points.

Physical

- Proximity – 1 point was given because an attacker needs physical access.
- Privilege – 10 points were given because this attack gives lots of access to the victim's computer. There can be many things that an attacker can do.
- Knowledge – 1 point was given because it requires in-depth knowledge to do this attack. For example, knowledge of physical security, location, knowledge about the victim etc.

The total is 12 points.

Attacks from Highest to Lowest Success Rate without Mitigation

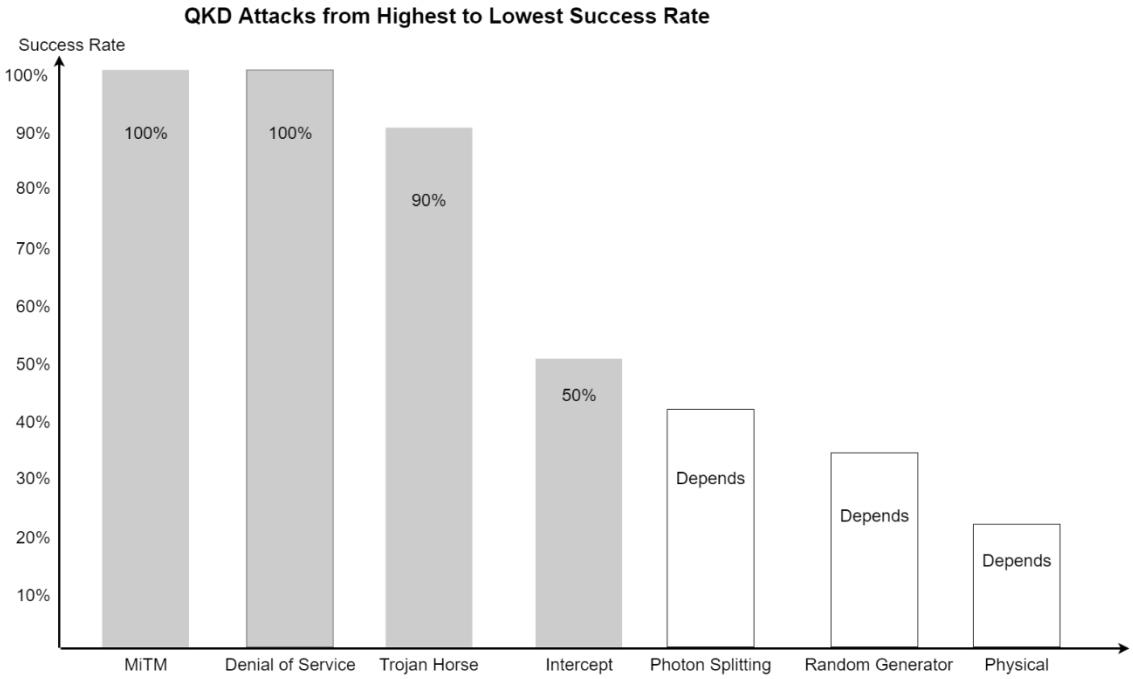

Figure 4.15 Attacks ranked from highest to lowest success rate without mitigation plan.

Man in the middle attack has a 100% success rate plan because if Alice did not authenticate and did not know that Bob is who he is claiming to be then Alice will not know that someone is in the middle.

Denial of service attack has a 100% success rate because there is no way to know if traffic is legitimate or not. This means that anyone can ping the machine and if the machine is pinged too frequently with a large size, then the machine will start to slow down and unable to serve legitimate requests.

Trojan horse attack has a 90% success rate because according to a group of researchers, they have managed to do the experiments and found out that this attack has a 90% success rate (Jain, et al., 2014).

Intercept and resend attack have about a 50% success rate for two qubits. See Figure 4.10.

As for photon number splitting, random number generator and physical attacks, they all depend on the implementation and the person doing the attack. For example, for photon number splitting, laser pulses may send 3 or 4 photons per pulse which will raise the success rate of this attack significantly. Whereas laser pulses that send 2 photons occasionally will have a very low success rate. Similarly, a random number generator attack's success rate could be zero if the victim is using a true quantum random number generator.

Attacks from Highest to Lowest Success Rate with Mitigation

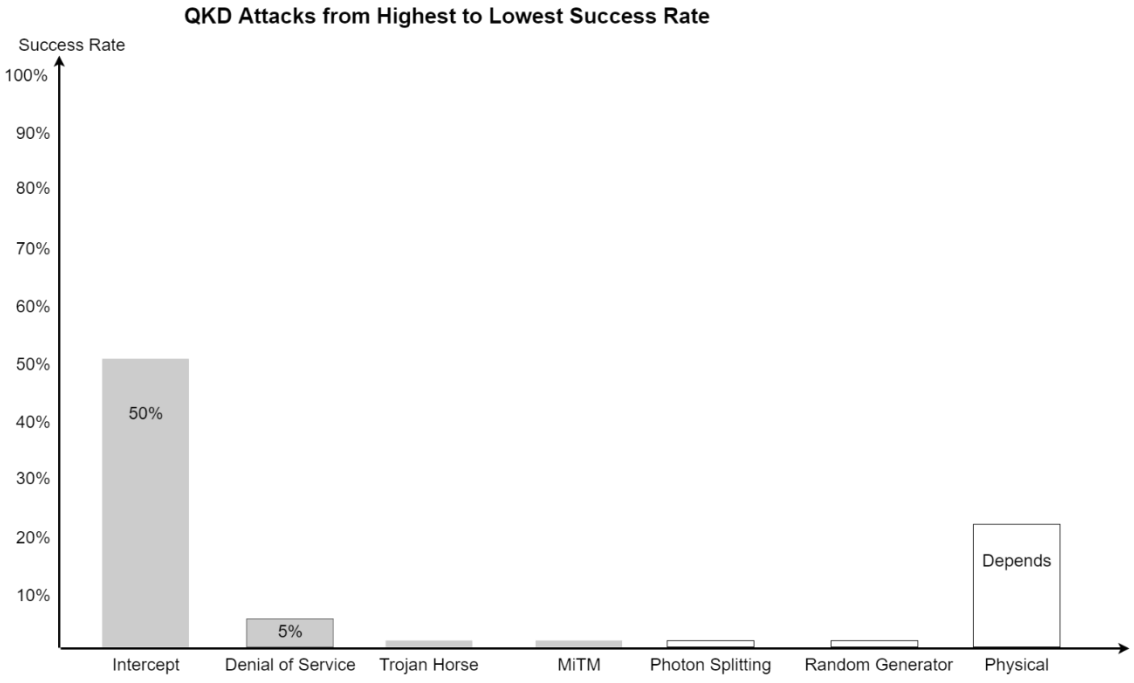

Figure 4.16 Attacks ranked from highest to lowest success rate with a mitigation plan.

Intercept and resend attack are mitigated by default so there is no change in the success rate. However, increasing the number of bits used for comparison will reduce the probability of Eve going undetected exponentially.

Denial of service attack drops to 5% because, with a decent configuration of the machine, the amount of ping and size of data can be reduced. For example, if an attacker tries to send a thousand pings per second, they could be rate-limited and banned for a short time.

Trojan horse drops to 0% because with a classical light detector to check for illegitimate light or another hardware that acts similarly to a firewall, this attack can be detected and mitigated 100% of the time.

Man in the middle attack drops to 0% because, with proper authentication and identification, this attack can be detected 100% of the time. For example, Alice and Bob could create a pre-shared key in-person to be used as an authentication method.

Photon number splitting attack drops to 0% because using a true single-photon source will render this attack useless (Intallura, et al., 2007).

Random number generator attack drops to 0% because using a quantum random number generator will make any kind of attacks useless (Herrero-Collantes & Garcia-Escartin, 2017).

As for physical attacks, there are too many factors to consider so it depends on the situation.

Attacks from Easiest to Hardest with Success Rate without Mitigation

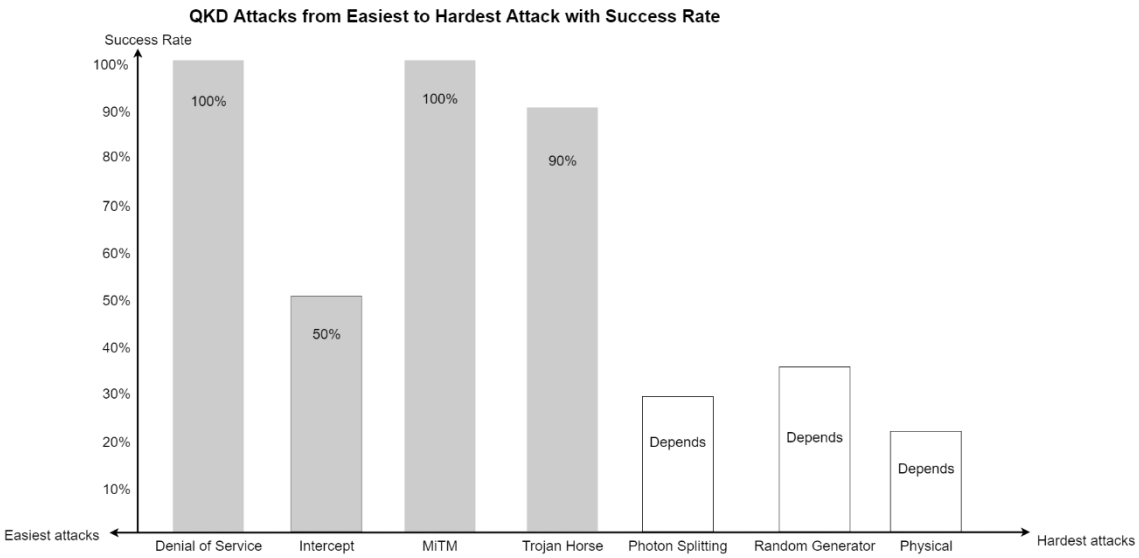

Figure 4.17 Attacks ranked from easiest to hardest with success rate without mitigation plan.

Figure 4.17 shows that some attacks that are easy does not necessarily have high success rate.

Attacks from Easiest to Hardest with Success Rate with Mitigation

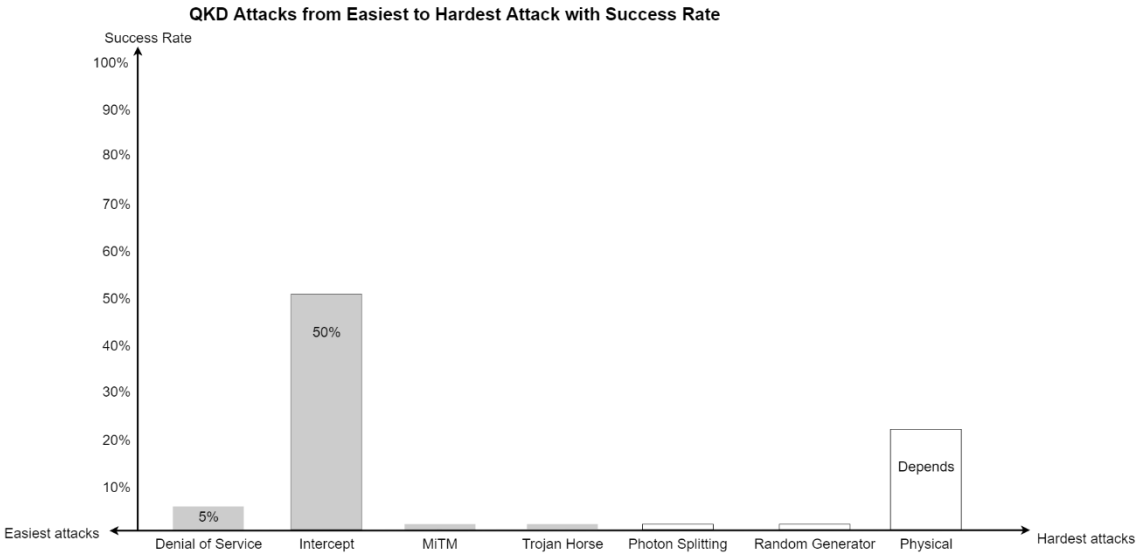

Figure 4.18 Attacks ranked from easiest to hardest with success rate with a mitigation plan.

Figure 4.17 and Figure 4.18 shows the attacks from easiest to hard with their success rate. This can be interpreted in various ways which will be discussed in section 5.

5 Discussion

In this section, interpretation of results, evaluation of approach and implications of this research will be discussed.

5.1 Interpretation of Results

5.1.1 Interpreting Experiment

The result of the experiment in section 4.2.1 shows that it is like what the equation (3.5) predicts but it does not follow the exact pattern as is the case with the real world. For a lower number of qubits transmitted, there is a possibility that Eve can have a higher than 75% success rate at intercepting the communication. This can be a significant threat to those that have security requirements. The closes thing to the mitigation plan is to generate more than 72 random qubits to transmit. This is because according to the equation, with 72 qubits, the probability of detecting Eve is about 0.9999999989897916. However, assuming those who need this kind of security will also use a quantum random number generator to generate random qubits, the rate of entropy it produces might be too slow to be practical.

The result has been generated by running the code 100 times, counting the number of times Eve was detected and by plotting them as a histogram. However, the code has not been audited or formally reviewed, which means that there could be some inconsistencies that produce slightly different result compared to the equation. Or the simulator that was used has some biases which may explain the difference. In this case, this could be argued that in the real world, the probability of detection of Eve could be

99% or 1% for one qubit. This could make the interception and resend the most dangerous or the least dangerous attack. Further research is needed to confirm.

5.1.2 Interpreting Easiest to Hardest Attacks

The result of categorization can be used in many ways. For example, the Table 4.3 shows the attacks that are easiest to hardest to do. This can be interpreted as the attacks that are most likely to occur once the quantum internet is realised. This is because most people will not have the necessary knowledge to conduct advanced attack as is the case with the classical internet. Also, attackers will want to automate as much as possible. This implies that attackers will build bots that will randomly attack whatever the bots find. And since the bots cannot do too advanced attacks, the bots will do the easiest attacks as they will be sure to find devices with the improper configuration that can be attacked using an easy method.

5.1.3 Interpreting Highest to Lowest Success Rate without Mitigation

The Figure 4.15 shows the success rate of attacks from highest to lowest. This can be used to know which attacks have the most severity once the attacker succeeds. For example, without any mitigation plan, the man in the middle attack will have a 100% success rate. This means that attackers can compromise the communication channel 100% of the time. This must be mitigated because if it is not mitigated then any secure communication in quantum internet is susceptible to compromise. For example, communication between banks, governments, or some other sensitive data. Another example could be a denial of service which can be used to disrupt a communication entirely or to route the communication to another channel where there is a higher

chance of compromise. This can be a very problematic attack without a mitigation plan as seen in the early days of the internet.

5.1.4 Interpreting Highest to Lowest Success Rate with Mitigation

The Figure 4.16 shows that after mitigation, most of the attacks have a 0% or almost 0% success rate. This is because most of the attacks can be detected and entirely blocked. The intercept and resend attack became first while the denial of service remained second. However, the only real threat after mitigation is intercept and resend attack. As discussed previously, the only real problem to rendering this attack useless is to have a quantum random number generator that can generate random numbers at a higher rate than the number of random qubits sent. The figure shows that for QKD to become more widely used, mitigation plans must be used to make QKD safer.

The exception is the physical attack. Physical attacks have too many factors to accurately get a success rate.

Summary of mitigation plans are as follows:

- Intercept and Resend – Use a higher number of random qubits to send.
- Denial of Service – Use a rate blocking software or device.
- Man in the Middle – Proper identification and authentication.
- Trojan Horse – Use an illegitimate signal detecting device. Or some kind of firewall.
- Photon Number Splitting – Use a true single-photon source.
- Random Number Generator – Use a quantum random number generator.

5.2 Evaluation of Approach

Overall, the waterfall model approach to this research has been successful. The requirements were set at the beginning and then according to requirements, the research was designed, implemented, and verified step by step. Any other approach would have been counterproductive as many steps would be repeated without any change.

By gathering both primary and secondary data, this research was able to produce a mitigation plan against many attacks. This research also identified a problem for future research. Such as the real-world data of intercept and resend attack. Another problem is the unavailability of a real quantum computer. Attacks such as photon number splitting, denial of service and trojan horse attack all require at least two quantum computers to experiment. This leads to the limitations of this research. Only one attack could be simulated whereas, for other attacks, the categorization relied on secondary data or by speculation. Another limitation of this research is time. To get more accurate data, the simulation needs to be run many times. However, each run takes a long time so the data may not be as accurate as possible.

The future plan is to review the code and to run it more than 100 times to get more accurate data and to find out if the real-world data correlates with equation (3.5). If possible, access to at least two quantum computers would be desirable to experiment with other attacks also and get real-world data. Then the categorization will be redone based on new data.

5.3 Implication of this Research

This research has identified six mitigation plans against attacks on QKD. It also identified one problem that needs to be researched further. The result of categorization

also predicts the most frequent attacks against QKD and the severity of each attack with and without a mitigation plan.

European Telecommunications Standards Institute (ETSI) is working on the standardization of attacks on QKD [14]. And they are collecting and evaluating QKD security proofs based on International Organisation for Standardization (ISO) security evaluation standards. This research will help ETSI by providing them with an example of real-world data with a mitigation plan for attacks which can be used to further develop the standardization of attacks. The data can also be used to create the best security practices standard for QKD. This research also contains various criteria that were used to categorize each attack. ETSI can use this data to further develop their accuracy on standardization of attacks on QKD.

[14] https://ec.europa.eu/info/funding-tenders/opportunities/portal/screen/opportunities/horizon-results-platform/29227 - Accessed 21st April 2021

6 Conclusion

This research has introduced the basic theory of quantum information which explained the theory behind seven attacks on BB84 QKD and their mitigation plan. One of the attacks, intercept and resend, was simulated on a quantum simulator which has shown to be slightly different to the equation that predicts the probability of undetected interception occurring. The data from categorization predicts the frequency of each attack in the future. For example, without mitigation plan, man in the middle, trojan horse and denial of service attacks are going to be frequently used because of their high success rate and low knowledge barrier. With the mitigation plan, intercept and resend and denial of service attacks will be frequently used because there is a chance to succeed. This implies that a mitigation plan should be put in place when using BB84 QKD and QKD will be much more secure than traditional methods. So, QKD must be widely adopted when quantum supremacy is realised.

This research has partly achieved the aims which were to design and simulate attacks against the QKD algorithm and to create a mitigation plan against such attacks. This is because no new attacks or mitigation plan has been created because of time constraints. This implies that the scope was too wide and out of reach so it will be considered appropriately in the future. However, intercept and resend attack has been simulated which found new data. Existing data have been categorised in novel ways which shows the effect each mitigation plan has on their respective attacks. And the categorization demonstrated a pattern that can occur in the future. So, the research has been partly successful.

In the future, simulations will be run many times more to make sure that the data cannot have biases. For future research, creating new attacks and mitigation plan, correlation between equation and real-world data can be considered,

References

Anon., 1992. Rapid solution of problems by quantum computation. *Royal Society,* 439(1907).

Ashokkumar, C., Giri, R. P. & B., M., 2016. Highly Efficient Algorithms for AES Key Retrieval in Cache Access Attacks. *2016 IEEE European Symposium on Security and Privacy (EuroS&P),* pp. 261-275.

Bancal, J.-D.et al., 2012. Quantum non-locality based on finite-speed causal influences leads to superluminal signalling. *Nature Physics,* 8(12), pp. 867-870.

Bell, J. S., 1964. ON THE EINSTEIN PODOLSKY ROSEN PARADOX*. *Physics Publishing Co.,* 1(3), pp. 195-200.

Benioff, P., 1980. The computer as a physical system: A microscopic quantum mechanical Hamiltonian model of computers as represented by Turing machines. *Journal of Statistical Physics,* 22(5), pp. 563-591.

Bennet, C. H. & Brassard, G., 1984. *Quantum Cryptography, public key distribution and coin tossing.* Bangalore, International conference on computers, systems and signal processing.

Bennet, C. H. et al., 1993. Teleporting an Unknown Quantum State via Dual Classical and Einstein-Podolsky-Rosen Channels. 70(13).

Bennett, C. H. & Brassard, G., 1989. Experimental quantum cryptography: the dawn of a new era for quantum cryptography: the experimental prototype is working. *ACM SIGACT News,* 20(4), p. 78.

Bloch, F., 1946. Nuclear Induction. *Phys. Rev,* 70(7-8), pp. 460-474.

Chien, C.-H., Meter, R. V. & Kuo, S.-Y., 2015. Fault-Tolerant Operations for Universal Blind Quantum Computation. *ACM Journal on Emerging Technologies in Computing Systems (JETC),* 12(1).

Chuang, I., 2000. Quantum Algorithm for Distributed Clock Synchronization. *Physical review letters,* Volume 85.

Deutch, D., 1984. Quantum theory, the Church-Turing principle and the universal. *Proceedings of the Royal Society of London,* pp. 97-117.

Dirac, P. A. M., 1939. A new notation for quantum mechanics. *Cambridge University Press,* 35(3), pp. 416-418.

Einstein, A., Podolsky, B. & Rosen, N., 1935. Can Quantum-Mechanical Description of Physical Reality Be Considered Complete?. 47(777).

Feynman, R. P., 1981. Simulating Physics with Computers. *International Journal of Theoretical Physic,* Volume 21.

Genkin, D., Shamir, A. & Tromer, E., 2013. RSA Key Extraction via Low-Bandwidth Acoustic Cryptanalysis∗.

Hernandez, M. a. L. G. G. et al., 2019. A Quantum-Inspired Method for Three-Dimensional Ligand-Based Virtual Screening. *Journal of Chemical Information and Modeling,* 59(10), pp. 4475-4485.

Herrero-Collantes, M. & Garcia-Escartin, J. C., 2017. Quantum random number generators. *Rev. Mod. Phys.,* 89(1), p. 48.

Intallura, P. M. et al., 2007. Quantum key distribution using a triggered quantum dot source emitting near 1.3µm. *Applied Physics Letters,* 91(16).

Jain, N. et al., 2014. Trojan-horse attacks threaten the security of practical quantum cryptography. *arxiv,* 16(12), p. 22.

K, I., 2018. *qBitcoin: A Peer-to-Peer Quantum Cash System.* s.l., Springer, Cham.

Kimble, H. J., 2008. The quantum internet. *Nature,* 453(7198).

Krco, M. & Paul, P., 2001. Quantum Clock Synchronization: a Multi-Party Protocol. *Physical Review A,* Volume 66.

Nielsen, M. A. & Chuang, I. L., 2010. Application: Superdense coding. In: *Quantum Computation and Quantum Information.* s.l.:Cambridge University Press, 2010, p. 97.

O'Malley, P. J. J. et al., 2016. Scalable Quantum Simulation of Molecular Energies. *Phys. Rev. X,* 6(3).

Portmann, C. & Renner, R., 2014. Cryptographic security of quantum key distribution. *arxiv.*

Royce, W. W., 1970. *ANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS,* s.l.: s.n.

Schrödinger, E., 1935. Die gegenwärtige Situation in der Quantenmechanik. *Naturwissenschaften,* 23(48), pp. 807-812.

Schumacher, B., 1995. Quantum coding. *Phys. Rev. A,* 51(4), pp. 2738-2747.

Shannon, C. E., 1949. Communication theory of secrecy systems. *The Bell System Technical Journal,* 28(4).

Shor, P. W., 1994. Algorithms for quantum computation: discrete logarithms and factoring. *Proceedings 35th Annual Symposium on Foundations of Computer Science,* pp. 124-134.

Tabia, G. N. M., 2011. Quantum Computing with Cluster States. *Perimeter Institute for Theoretical Physics.*

Tomamichel, M. & Leverrier, A., 2017. A largely self-contained and complete security proof for quantum key distribution. *Quantum,* Volume 1, p. 14.

Appendix I

This section will explain the process to run the code used in this project and the code will be provided.

Steps to run the code

1. Log in to IBM Quantum Lab using IBMid.
2. Create new file.
3. Copy the code and put them in a specified cell (Marked "**Cell** " followed by a number).
4. Run the code.

Code without interception

Cell 1

```
# QKD Without Interception
```

Cell 2

```
# imports based on qiskit website code
from qiskit import QuantumCircuit, Aer, assemble
from qiskit.visualization import plot_histogram
from numpy.random import randint
import numpy as np

class Person:
    size = 0
    sample_size = 0

    def __init__(self, size, sample_size):
        self.size = size
        self.sample_size = sample_size
```

```
        #np.random.seed(seed=5)

    def sample_bits(self, selection): # method based on qiskit website code
        self.sample = []
        for i in selection:
            i = np.mod(i, len(self.good_bits))
            self.sample.append(self.good_bits.pop(i))

    def generate_bases(self):
        self.bases = randint(2, size=self.size) # code based on qiskit
website code

    def generate_selection(self):
        self.selection = randint(size, size=self.sample_size) # code based on
qiskit website code

    def get_selection(self):
        return self.selection

    def get_bases(self):
        return self.bases

    def get_sample(self):
        return self.sample

    def get_good_bits(self):
        return self.good_bits
```

Cell 3

```
class Sender(Person):

    def __init__(self, size, sample_size):
        super().__init__(size, sample_size)

    def encode_message(self): # method based on qiskit website code
        self.message = []
        for i in range(size):
            qc = QuantumCircuit(1,1)
            if self.bases[i] == 0:
                if self.bits[i] == 0:
```

```
                    pass
                else:
                    qc.x(0)
            else:
                if self.bits[i] == 0:
                    qc.h(0)
                else:
                    qc.x(0)
                    qc.h(0)
            qc.barrier()
            self.message.append(qc)

    def generate_bits(self):
        self.bits = randint(2, size=self.size) # code based on qiskit website
code

    def remove_garbage(self, b_bases): # method based on qiskit website code
        self.good_bits = []
        for q in range(self.size):
            if self.bases[q] == b_bases[q]:
                self.good_bits.append(self.bits[q])

    def get_message(self):
        return self.message

    def get_bits(self):
        return self.bits
```

Cell 4

```
class Receiver(Person):

    def __init__(self, size, sample_size):
        super().__init__(size, sample_size)

    def measure_message(self, message): # method based on qiskit website code
        self.measurements = []
        for q in range(self.size):
            if self.bases[q] == 0: # measuring in Z-basis
                message[q].measure(0,0)
```

```
            if self.bases[q] == 1: # measuring in X-basis
                message[q].h(0)
                message[q].measure(0,0)
            qasm_sim = Aer.get_backend('qasm_simulator')
            qobj = assemble(message[q], shots=1, memory=True)
            result = qasm_sim.run(qobj).result()
            self.measured_counts = result.get_counts()
            measured_bit = int(result.get_memory()[0])
            self.measurements.append(measured_bit)

    def remove_garbage(self, b_bases): # method based on qiskit website code
        self.good_bits = []
        for q in range(self.size):
            if self.bases[q] == b_bases[q]:
                self.good_bits.append(self.measurements[q])

    def get_measured_count(self):
        return self.measured_counts

    def get_measured_bits(self):
        return self.measurements
```

Cell 5

```
def count_zero_one(count):
    a = 0
    b = 0
    ab = []
    for i in count:
        if i == 0:
            a = a + 1
        if i == 1:
            b = b + 1
    ab.append(a)
    ab.append(b)
    return ab
```

Cell 6

```
# 1. Get internal measurements
```

Cell 7

```
size = 100 # message size
sample_size = 15

alice = Sender(size, sample_size)
bob = Receiver(size, sample_size)

alice.generate_bits()
alice.generate_bases()
print("Alice basis: " + str(alice.get_bases()))
ab = count_zero_one(alice.get_bases())
print("Alice number of 0s in basis: " + str(ab[0]))
print("Alice number of 1s in basis: " + str(ab[1]))

alice.encode_message()
print("Alice random message: " + str(alice.get_bits()))

bob.generate_bases()
print("Bob basis: " + str(bob.get_bases()))
ab = count_zero_one(bob.get_bases())
print("Bob number of 0s in basis: " + str(ab[0]))
print("Bob number of 1s in basis: " + str(ab[1]))

bob.measure_message(alice.get_message())
print("Result of Bob measurement: " + str(bob.get_measured_bits()))
ab = count_zero_one(bob.get_measured_bits())
print("Bob number of 0s in result: " + str(ab[0]))
print("Bob number of 1s in result: " + str(ab[1]))

alice.remove_garbage(bob.get_bases())
bob.remove_garbage(alice.get_bases())

print("Alice good bits: " + str(alice.get_good_bits()))
print("Bob good bits:   " + str(bob.get_good_bits()))
print("Alice good bits = Bob good bits: " + str(alice.get_good_bits() ==
bob.get_good_bits()))

print("Number of good bits: " + str(len(alice.get_good_bits())))
```

```
alice.generate_selection()
print("Alice selection = " + str(alice.get_selection()))
print("Bob selection =   " + str(alice.get_selection()))
alice.sample_bits(alice.get_selection())
print("Alice sample = " + str(alice.get_sample()))
bob.sample_bits(alice.get_selection())
print("Bob sample =   " + str(bob.get_sample()))
print("Alice sample = Bob sample: " + str(alice.get_sample() == bob.get_sample()))
if alice.get_sample() != bob.get_sample(): # code based on qiskit website code

    print("Eve's interference was detected.")
else:
    print("No interception")
```

Cell 8

```
# 2. Gather data
```

Cell 9

```
def calculate_data(size, sample_size):
    alice = Sender(size, sample_size)
    bob = Receiver(size, sample_size)

    alice.generate_bits()
    alice.generate_bases()
    alice.encode_message()

    bob.generate_bases()
    bob.measure_message(alice.get_message())

    alice.remove_garbage(bob.get_bases())
    bob.remove_garbage(alice.get_bases())

    alice.generate_selection()
    alice.sample_bits(alice.get_selection())
    bob.sample_bits(alice.get_selection())

    if alice.get_sample() != bob.get_sample(): # code based on qiskit website code
```

```
        return True
    else:
        return False
```

Cell 10

```
def collect_data_based_on_runs(sample_size, size, num_runs):
    plot_hist = {'Interception':[], 'No Interception':[]}
    interception_count = 0
    no_interception_count = 0

    for x in range(num_runs):
        if calculate_data(size, sample_size) == True:
            interception_count = interception_count + 1
        elif calculate_data(size, sample_size) == False:
            no_interception_count = no_interception_count + 1

    plot_hist['Interception'].append(interception_count)
    plot_hist['No Interception'].append(no_interception_count)

    display(plot_histogram(plot_hist))
```

Cell 11

```
sample_size = 5
size = 100 # message size
num_runs = 100

for x in range(sample_size):
    collect_data_based_on_runs(x, size, num_runs)
```

Code with interception

Cell 1

```
# QKD With Interception
```

Cell 2

```
# imports based on qiskit website code
from qiskit import QuantumCircuit, Aer, assemble
from qiskit.visualization import plot_histogram
```

```
from numpy.random import randint
import numpy as np

class Person:
    size = 0
    sample_size = 0

    def __init__(self, size, sample_size):
        self.size = size
        self.sample_size = sample_size
        #np.random.seed(seed=5)

    def sample_bits(self, selection): # method based on qiskit website code
        self.sample = []
        for i in selection:
            i = np.mod(i, len(self.good_bits))
            self.sample.append(self.good_bits.pop(i))

    def generate_bases(self):
        self.bases = randint(2, size=self.size) # code based on qiskit
website code

    def generate_selection(self):
        self.selection = randint(size, size=self.sample_size) # code based on
qiskit website code

    def get_selection(self):
        return self.selection

    def get_bases(self):
        return self.bases

    def get_sample(self):
        return self.sample

    def get_good_bits(self):
        return self.good_bits
```

Cell 3

```
class Sender(Person):
```

```
    def __init__(self, size, sample_size):
        super().__init__(size, sample_size)

    def encode_message(self): # method based on qiskit website code
        self.message = []
        for i in range(size):
            qc = QuantumCircuit(1,1)
            if self.bases[i] == 0:
                if self.bits[i] == 0:
                    pass
                else:
                    qc.x(0)
            else:
                if self.bits[i] == 0:
                    qc.h(0)
                else:
                    qc.x(0)
                    qc.h(0)
            qc.barrier()
            self.message.append(qc)

    def generate_bits(self):
        self.bits = randint(2, size=self.size) # code based on qiskit website
code

    def remove_garbage(self, b_bases): # method based on qiskit website code
        self.good_bits = []
        for q in range(self.size):
            if self.bases[q] == b_bases[q]:
                self.good_bits.append(self.bits[q])

    def get_message(self):
        return self.message

    def get_bits(self):
        return self.bits
```

Cell 4

```
class Receiver(Person):

    def __init__(self, size, sample_size):
        super().__init__(size, sample_size)

    def measure_message(self, message): # method based on qiskit website code
        self.measurements = []
        for q in range(self.size):
            if self.bases[q] == 0: # measuring in Z-basis
                message[q].measure(0,0)
            if self.bases[q] == 1: # measuring in X-basis
                message[q].h(0)
                message[q].measure(0,0)
            qasm_sim = Aer.get_backend('qasm_simulator')
            qobj = assemble(message[q], shots=1, memory=True)
            result = qasm_sim.run(qobj).result()
            self.measured_counts = result.get_counts()
            measured_bit = int(result.get_memory()[0])
            self.measurements.append(measured_bit)

    def remove_garbage(self, b_bases): # method based on qiskit website code
        self.good_bits = []
        for q in range(self.size):
            if self.bases[q] == b_bases[q]:
                self.good_bits.append(self.measurements[q])

    def get_measured_count(self):
        return self.measured_counts

    def get_measured_bits(self):
        return self.measurements
```

Cell 5

```
def count_zero_one(count):
    a = 0
    b = 0
    ab = []
    for i in count:
        if i == 0:
```

```
            a = a + 1
        if i == 1:
            b = b + 1
    ab.append(a)
    ab.append(b)
    return ab
```

Cell 6

```
# 1. Get internal measurements
```

Cell 7

```
size = 100 # message size
sample_size = 15

alice = Sender(size, sample_size)
eve = Receiver(size, sample_size)
bob = Receiver(size, sample_size)

alice.generate_bits()
alice.generate_bases()
print("Alice basis: " + str(alice.get_bases()))
ab = count_zero_one(alice.get_bases())
print("Alice number of 0s in basis: " + str(ab[0]))
print("Alice number of 1s in basis: " + str(ab[1]))

alice.encode_message()
print("Alice random message: " + str(alice.get_bits()))

eve.generate_bases()
print("Eve basis: " + str(eve.get_bases()))
ab = count_zero_one(eve.get_bases())
print("Eve number of 0s in basis: " + str(ab[0]))
print("Eve number of 1s in basis: " + str(ab[1]))

eve.measure_message(alice.get_message())
print("Result of Eve measurement: " + str(eve.get_measured_bits()))
ab = count_zero_one(eve.get_measured_bits())
print("Eve number of 0s in result: " + str(ab[0]))
print("Eve number of 1s in result: " + str(ab[1]))
```

```
bob.generate_bases()
print("Bob basis: " + str(bob.get_bases()))
ab = count_zero_one(bob.get_bases())
print("Bob number of 0s in basis: " + str(ab[0]))
print("Bob number of 1s in basis: " + str(ab[1]))

bob.measure_message(alice.get_message())
print("Result of Bob measurement: " + str(bob.get_measured_bits()))
ab = count_zero_one(bob.get_measured_bits())
print("Bob number of 0s in result: " + str(ab[0]))
print("Bob number of 1s in result: " + str(ab[1]))

alice.remove_garbage(bob.get_bases())
bob.remove_garbage(alice.get_bases())

print("Alice good bits: " + str(alice.get_good_bits()))
print("Bob good bits:   " + str(bob.get_good_bits()))
print("Alice good bits = Bob good bits: " + str(alice.get_good_bits() ==
bob.get_good_bits()))

print("Number of good bits: " + str(len(alice.get_good_bits())))

alice.generate_selection()
print("Alice selection = " + str(alice.get_selection()))
print("Bob selection =   " + str(alice.get_selection()))
alice.sample_bits(alice.get_selection())
print("Alice sample = " + str(alice.get_sample()))
bob.sample_bits(alice.get_selection())
print("Bob sample =   " + str(bob.get_sample()))
print("Alice sample = Bob sample: " + str(alice.get_sample() ==
bob.get_sample()))
if alice.get_sample() != bob.get_sample():
    print("Eve's interference was detected.")
else:
    print("Eve went undetected!")
```

Cell 8

```
# 2. Gather data
```

Cell 9

```
def calculate_data(size, sample_size):
    alice = Sender(size, sample_size)
    eve = Receiver(size, sample_size)
    bob = Receiver(size, sample_size)

    alice.generate_bits()
    alice.generate_bases()
    alice.encode_message()

    eve.generate_bases()
    eve.measure_message(alice.get_message())

    bob.generate_bases()
    bob.measure_message(alice.get_message())

    alice.remove_garbage(bob.get_bases())
    bob.remove_garbage(alice.get_bases())

    alice.generate_selection()
    alice.sample_bits(alice.get_selection())
    bob.sample_bits(alice.get_selection())

    if alice.get_sample() != bob.get_sample():
        return True
    else:
        return False
```

Cell 10

```
def collect_data_based_on_runs(sample_size, size, num_runs):
    plot_hist = {'Detected':[], 'Undetected':[]}
    detected_count = 0
    undetected_count = 0

    for x in range(num_runs):
        if calculate_data(size, sample_size) == True:
            detected_count = detected_count + 1
        elif calculate_data(size, sample_size) == False:
            undetected_count = undetected_count + 1
```

```
    plot_hist['Detected'].append(detected_count)
    plot_hist['Undetected'].append(undetected_count)

    display(plot_histogram(plot_hist))
```

Cell 11

```
sample_size = 5
size = 100 # message size
num_runs = 100

for x in range(sample_size):
    collect_data_based_on_runs(x, size, num_runs)
```