Experiments with B92 Quantum Key Distribution Algorithm Implementation

Ayush Gopal[1]
Email: ayush.gopal8@gmail.com

Abstract: This study presents the B92 Quantum Key Distribution scheme and its optimization and implementation considerations. Simulation software was developed to model and analyze the B92 protocol under varying conditions. Another contribution of this work is investigating how this protocol runs on real quantum computing hardware. New eavesdropping schemes were proposed and modeled.

Keywords: quantum cryptography, quantum physics, quantum key distribution (QKD), B92 protocol, eavesdropper detection method, qubit, photon polarization

I. Introduction

Quantum cryptography is an encryption technique that uses quantum properties to carry out cryptographic operations. Much of public-key cryptography is premised on the presumption that evaluating the factors of a large integer is difficult for a classical computer. For even the most powerful classical computers, factoring large integers can take centuries using traditional factorization algorithms. Quantum computing poses a challenge to this encryption approach. Shor's quantum algorithm hypothetically calculates the prime factors of a large integer in polynomial time—much more efficiently than a classical computer—which could break this method of encryption. Due to present quantum hardware limitations, Shor's algorithm cannot currently seriously threaten contemporary encryption practices, but with further development of quantum devices, quantum algorithms like Shor's algorithm could conceivably pose a substantial danger to digital security in the future. As a result, encryption techniques based on quantum mechanical properties have been proposed and devised rather than relying on the difficulty of large integer factorization. The goal of quantum cryptography is to produce and distribute a secret key between two users, conventionally known as Alice and Bob, using a technique known as Quantum Key Distribution (QKD). Since the state of a qubit cannot be measured without collapsing the qubit, an eavesdropper, Eve, may not be able to exact the information from a particular qubit. To avoid being discovered, Eve would have to detect the qubit and then re-send it. However, she will likely eventually send a qubit in the incorrect state, resulting in errors that will expose the presence of the eavesdropper. The BB84 scheme, the E91 scheme, and the B92 scheme are three of the most common QKD schemes. The somewhat newer and less well-researched B92 scheme is the subject of this study.

B92 Protocol:

The B92 QKD protocol uses polarized photons to allow two parties, Alice and Bob, to create a secret shared key, as well as identify any eavesdropper, Eve, who may have intercepted the quantum channel. The steps of the B92 protocol are as follows:

1. Alice generates a random string of bits (either 0 or 1).
2. Alice chooses one of two polarization states at random: 0° or 45°. She then creates a photon polarization state corresponding to the bit value that she has chosen.
3. Alice sends her qubits to Bob over the quantum channel.
4. Bob receives the qubits and selects a basis to measure each qubit in at random (rectilinear or diagonal).
5. If Bob detects a photon, he knows the polarization and bit value of the photon Alice transmitted, but if he doesn't, he has no indication which state Alice sent. As a result, Bob will only keep the values where the photon was detected.
6. Bob communicates with Alice through a classical channel for which qubits he was able to detect a photon, and those corresponding bit values will form the key.

[1] High School Student (Unaffiliated)

7. Alice and Bob exchange a random sample of the bits and check for errors.

II. Theoretical Analysis of B92 Protocol

I conceptually studied the B92 protocol using the theories of optics and quantum mechanics to establish the probabilities of possible outcomes under varying circumstances. Alice can send two distinct types of qubits over the quantum channel, denoted by → and ↗, where → and ↗ represent photons in the H-polarization state and the +45°-polarization state respectively. To measure the photons he receives, Bob chooses at random one of two bases, either 90° or -45°, orthogonal from Alice's directions. These are designated by the rectilinear basis "R" and the diagonal basis "D". Bob's measurement is made using a photodetector after a polarizer that is polarized in the basis he chose for that particular measurement.

According to Malus's law, the intensity of plane-polarized light transmitted through a polarizer is proportional to the square of the cosine of the angle between the plane of the polarized light and the polarizer's transmission axis. The following formula describes Malus's law: $I = I_0 \cos^2\theta$. In this equation, I represents the intensity of the light transmitted through the polarizer, I_0 represents the intensity of light incident on the polarizer, and θ represents the angle between the axes of the polarizer and the light. According to quantum mechanics, light consists of discrete packets called photons. Each polarizer therefore has a given probability of blocking or allowing a single photon to pass through. This probability directly corresponds to θ. Malus's law establishes that no light passes through the polarizer when the transmission axis of the polarizer and the polarization of the photon are perpendicular to one another, while when there is a 45° angle between them, there is a 0.5 probability that the photon will pass through and a 0.5 probability that the photon will not pass through.

No Eavesdropper:

The B92 protocol is analyzed under ideal circumstances (without an eavesdropper) to determine each potential scenario and its probability. If Alice sends a → qubit and Bob measures it in the R basis, Bob will not detect a photon. If Alice sends a ↗ qubit and Bob measures it in the R basis, there is a 0.5 probability that Bob will not detect the photon and a 0.5 probability that Bob will detect the photon. If Bob does detect a photon while measuring in the R basis, he can be certain that he received a ↗ photon from Alice. If he does not detect the photon, the qubit Alice sent could have either been ↗ or →, and Bob has no means of determining exactly which state Alice sent. As a result, Bob will discard all measurements in which he failed to detect a photon and only use the values in which he did. If Alice sends a ↗ qubit and Bob measures it in the D basis, Bob will not detect a photon. If Alice sends a → qubit and Bob measures it in the D basis, there is a 0.5 probability that Bob will not detect the photon and a 0.5 probability that Bob will detect the photon. Bob can conclude that he received a → photon from Alice if he detects a photon while measuring in the D basis. Bob should theoretically discard 75% of the qubits Alice delivers, according to probability. Figure 1 illustrates each potential outcome.

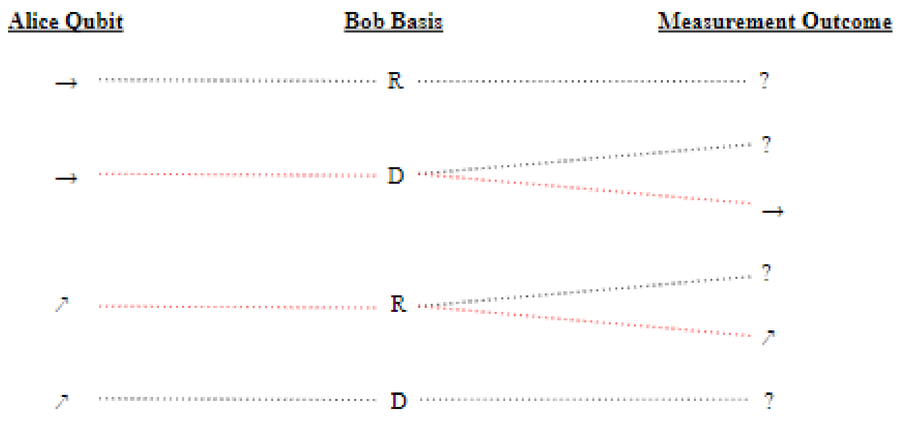

Figure 1. No eavesdropper

With Eavesdropper:

Before the qubits Alice sends reach Bob, Eve intercepts them. She must make a measurement in order to extract information from these qubits. Performing a measurement, however, would only allow Eve a 25% chance of determining the state of each photon Alice sent. Eve still has to send some qubit to Bob in the other cases. Eve's method for transmitting a qubit to Bob if she cannot identify the state of the qubit she received is unclear based on the reviewed scientific literature. Therefore, I have formulated a few different approaches Eve could adopt.

When Eve is unable to identify the state of the qubit she has received, she can simply transmit an arbitrary qubit, such as → for example. This relatively simple method would result in 6.25% leaked bits and 9.375% corrupted bits. Eve has a 0.625^N probability of evading detection, with N indicating the number of bits checked in the final step of the B92 protocol.

Each time Eve is unable to identify the qubit she has received, it would be advantageous for her to transmit a strategic qubit rather than an arbitrary qubit. Eve could send a qubit polarized in the measurement basis to maximize her chances of evading detection whenever she is unable to identify the photon state she obtained from Alice. This approach results in 6.25% leaked bits and 6.25% corrupted bits, lowering the risk of Eve being caught with respect to the previous scheme but still leaving a fair chance of detection. Eve would have a 0.75^N chance of going undetected using this method. Figure 2 shows various potential outcomes for which Eve sends the measurement basis each time she is unable to determine the state of the intercepted qubit.

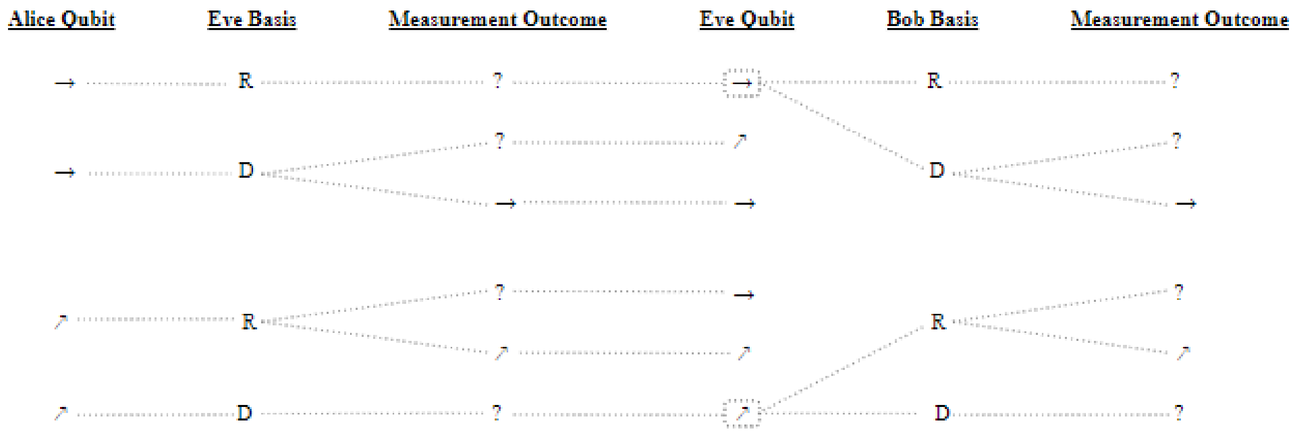

Figure 2. Eve sends basis

In Table 1 below, the theoretical values obtained from this conceptual case-by-case analysis of the B92 protocol can be seen.

Theoretical Values (Percentage)			
Eve Scheme	**Discarded**	**Leaked**	**Corrupted**
No Eve	75	0	0
Sends →	75	6.25	9.375
Sends ↗	75	6.25	9.375
Sends Basis	75	6.25	6.25

Table 1. Theoretical values for each eavesdropper scheme

III. Implementation of B92 Protocol

This study assesses the behavior of the B92 protocol on real quantum hardware, as well as how the B92 protocol responds to variations in key length and different eavesdropper schemes.

Methodology:

For each key length and eavesdropper method, the program runs a certain number of times depending on the feasibility of the compute time. Each time, a percentage of the original key length discarded, leaked, or corrupted is recorded, producing the data that will be analyzed. For each key length or eavesdropper scheme, the three outcomes from each trial will be averaged and the standard deviation will be determined.

The program developed to implement the B92 protocol is presented below:

```
import random
import secrets
from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from numpy.random import randint
import numpy as np
from qiskit import IBMQ
prov = IBMQ.load_account()

from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor

qc_backend = least_busy(prov.backends(filters=lambda b: b.configuration().n_qubits >= 4 and
                                   not b.configuration().simulator and
b.status().operational==True))
aer_sim = Aer.get_backend('aer_simulator')

use_QC = 1

def build_circuit(qc):
    if (use_QC == 1):
        t_qc = transpile(qc, qc_backend, optimization_level=3)
    else:
        t_qc = assemble(qc)
    return t_qc
```

```
def get_backend():
    if (use_QC == 1):
        return qc_backend
    return aer_sim

def xmit_success(bit, basis):
    qc = QuantumCircuit(1,1)
    if (bit):
        qc.h(0)
    qc.barrier()
    if (basis):
        qc.h(0)
    qc.measure(0,0)
    t_qc = build_circuit(qc)
    backend = get_backend()
    job = backend.run(t_qc,shots=1, memory=True)
    job_monitor(job)
    exp_result = job.result()
    memory = exp_result.get_memory()
    if (memory[0] == '0'):
        return 0
    else:
        return 1

dbg = 1

N = 16
valid = 0
leaked = 0
corrupted = 0
errs = 0
print(get_backend())
print("N: ",N)

for i in range(N):
               bit = secrets.randbelow(2)
               eve_basis = secrets.randbelow(2)
               bob_basis = secrets.randbelow(2)
               e_val = False
               e_cor = False
               res = xmit_success(bit, eve_basis)
               if dbg:
                              print("Res 1:",res)
               if res:
                              new_bit =  1-eve_basis
                              if bit != new_bit: #Not known by Eve, Alice, or Bob
                                             errs += 1
                                             e_cor = True
                                             print("Eve silent error at bit:",i)
                              e_val = True
               else:
                              new_bit = eve_basis  #Changes based on eavesdropper scheme

               res = xmit_success(new_bit, bob_basis)
               if dbg:
                              print("Res 2:",res)
               if res:
                              valid += 1
                              bob_bit = 1-bob_basis
                              ok = (bob_bit == bit) #Not known by Eve, Alice, or Bob
                              lkd = ok and (e_val == True) and (e_cor == False)
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2022 doi:10.20944/preprints202207.0279.v1

```
                              if dbg:
                                             print("sent:",bit,"got:",1-bob_basis,"\tSuccess:",
ok,"\tLeaked:",lkd,"\tbit:",i)
                              if lkd:
                                             leaked += 1
                              if (not ok):
                                             corrupted += 1

Discarded = (N - valid)/N * 100
leaked = leaked/N * 100
corrupted = corrupted/N * 100



print("Discarded:\t\t",Discarded)
print("leaked:\t\t",leaked)
print("corrupted:\t",corrupted)
print("EveErr:\t",100*errs/N)
```

This program can switch between operating on a real quantum device and simulating it on the AerSimulator backend by changing the use_QC variable. The program was initially run on a classical system utilizing the Qiskit AerSimulator backend in order to effectively study the effects of key length and eavesdropper scheme on the performance of the B92 protocol under ideal conditions, as well as check for any defects in the program.

Data:

The averages and standard deviation figures (expressed as a percentage of the initial key length) derived from data collected while simulating the B92 protocol on the AerSimulator backend are presented below.

The data is presented below with varying eavesdropping schemes for a constant key length of 1,024 bits.

No Eavesdropper			
	Discarded	**Leaked**	**Corrupted**
Average	75.12695313	0	0
Standard Deviation	1.180029858	0	0

Table 2. No eavesdropper on AerSimulator backend

Sends →			
	Discarded	**Leaked**	**Corrupted**
Average	74.30664063	6.5234375	9.599609375
Standard Deviation	1.36831086	1.076485141	1.0767312

Table 3. Eve sends → on AerSimulator backend

Sends ↗			
	Discarded	**Leaked**	**Corrupted**
Average	74.97070313	6.416015625	9.12109375
Standard	1.439707659	0.8413942205	0.8845545742

Deviation			

Table 4. Eve sends ↗ on AerSimulator backend

Sends Basis			
	Discarded	**Leaked**	**Corrupted**
Average	75.4296875	6.142578125	6.103515625
Standard Deviation	0.7705979546	0.6419449108	0.7469266866

Table 5. Eve sends basis on AerSimulator backend

The following data maintains a constant eavesdropper scheme of sending the basis, but varies key length.

	32 bit		
	Discarded	**Leaked**	**Corrupted**
Average	77.1875	5.3125	5.3125
Standard Deviation	6.258674536	3.623443152	4.896941364

Table 6. Original key length 32 bits on AerSimulator backend

	64 bit		
	Discarded	**Leaked**	**Corrupted**
Average	73.75	5.78125	6.40625
Standard Deviation	3.668086846	4.169920604	2.598953115

Table 7. Original key length 64 bits on AerSimulator backend

	128 bit		
	Discarded	**Leaked**	**Corrupted**
Average	74.609375	6.484375	6.5625
Standard Deviation	3.953704805	1.33042081	2.246235276

Table 8. Original key length 128 bits on AerSimulator backend

	256 bit		
	Discarded	**Leaked**	**Corrupted**
Average	76.6796875	5.625	5.9765625
Standard Deviation	2.255650337	1.2380067	1.766711656

Table 9. Original key length 256 bits on AerSimulator backend

	512 bit		
	Discarded	**Leaked**	**Corrupted**
Average	75.68359375	6.30859375	5.76171875
Standard Deviation	1.657920778	1.225100586	1.07076069

Table 10. Original key length 512 bits on AerSimulator backend

	1024 bit		
	Discarded	**Leaked**	**Corrupted**
Average	75.44921875	6.494140625	6.46484375
Standard Deviation	1.886448699	0.9437297535	1.182496713

Table 11. Original key length 1024 bits on AerSimulator backend

Analysis:

The data appears to be fairly consistent with the theoretical predictions. The percent error for each set of trials was computed by comparing the averages to the theoretical values. In relation to the key length, the percent error values are displayed in the table and graph below:

Percent Error vs Key Length			
Key Length	**Discarded**	**Leaked**	**Corrupted**
32	2.916666667	15	15
64	1.666666667	7.5	2.5
128	0.5208333333	3.75	5
256	2.239583333	10	4.375
512	0.9114583333	0.9375	7.8125
1024	0.5989583333	3.90625	3.4375

Table 12. Percent error vs. key length

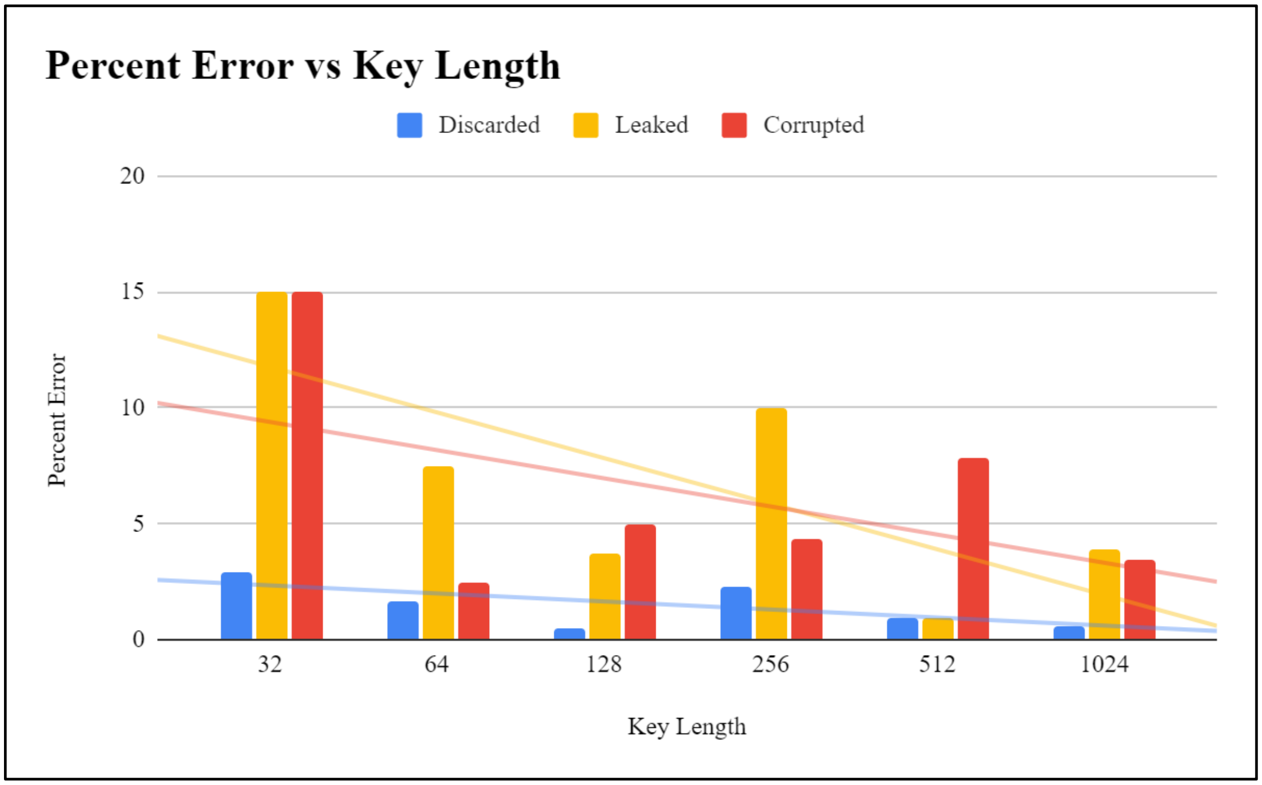

Figure 3. Percent error vs. key length

While there is an overall downward trend as expected, the percent error figures appear to be noisy and prone to random inconsistencies. As a general rule, increasing the key length tends to result in values that are closer to the theoretical values. Nevertheless, due to the probabilistic nature of the algorithm, this is simply a generalization and there appear to be several exceptions.

The standard deviation values for each set of trials may be analyzed in a similar fashion to observe how consistent the outcomes are with varying key lengths.

Standard Deviation vs Key Length			
Key Length	**Discarded**	**Leaked**	**Corrupted**
32	6.258674536	3.623443152	4.896941364
64	3.668086846	4.169920604	2.598953115
128	3.953704805	1.33042081	2.246235276
256	2.255650337	1.2380067	1.766711656
512	1.657920778	1.225100586	1.07076069
1024	1.886448699	0.9437297535	1.182496713

Table 13. Standard deviation vs. key length

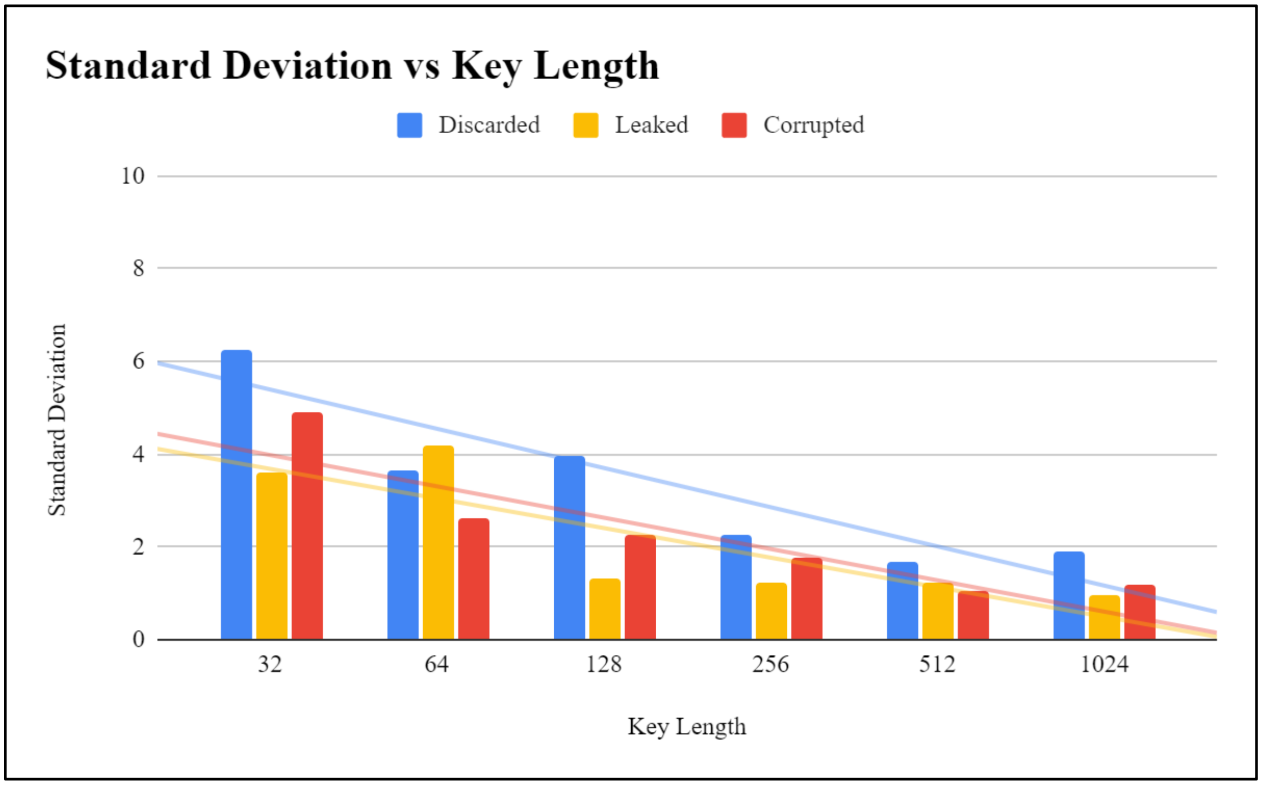

Figure 4. Standard deviation vs. key length

The standard deviation and key length have a strong decreasing correlation, demonstrating that increasing the amount of qubits Alice delivers leads to significantly improved overall consistency. With larger key length values, the B92 simulation seems to perform quite well, but with smaller key length values, the B92 protocol is clearly not dependable enough to be trusted as a method of reliable, secure communication. As a result, in order to ensure optimal security, the B92 protocol should be carried out with the maximum number of bits feasible.

Eve delivering a photon polarized in the same state as her basis evidently gave the lowest risk of detection out of all the tested eavesdropper schemes. The other two strategies, in which Eve either sends only a → photon or only a ↗ photon whenever she cannot figure out the state of the photon she received, fared poorly, with 9.60 and 9.12 percent of qubits corrupted respectively. Even with the basis transmission method, with roughly 6.10% of qubits corrupted, there still remains a distinct possibility of detection.

The majority of the experimental values closely reflect the theoretically expected values, confirming the validity of my B92 QKD implementation for each of my proposed eavesdropper schemes. Based on the simulation findings, it was found that the ideal circumstances for the B92 protocol are when Eve delivers a photon polarized in the same basis as her measurement basis and when she uses larger key lengths. As a result, further study will be conducted under those parameters.

B92 Implementation on Quantum Hardware:

The B92 protocol was implemented on actual quantum hardware using the same basic program. The program constructs an individual quantum circuit for each bit, making simulation for larger bit values relatively unfeasible due to the compute time and memory constraints. However, this approach allowed for an in-depth look at each bit as it underwent each stage of the B92 protocol.

The following sample output is provided:

```
ibmq_belem
N:  16
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 1
sent: 0 got: 0 Success: True  Leaked: False  bit: 6
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 1
sent: 0 got: 0 Success: True  Leaked: True   bit: 8
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Job Status: job has successfully run
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 19 July 2022 doi:10.20944/preprints202207.0279.v1

```
Res 1: 1
Job Status: job has successfully run
Res 2: 1
sent: 0 got: 0 Success: True  Leaked: True   bit: 13
Job Status: job has successfully run
Res 1: 1
Job Status: job has successfully run
Res 2: 1
sent: 0 got: 0 Success: True  Leaked: True   bit: 14
Job Status: job has successfully run
Res 1: 0
Job Status: job has successfully run
Res 2: 0
Discarded:      75.0
leaked:         18.75
corrupted:      0.0
EveErr:  0.0
```

The instance shown is a rare case in which there were no corrupted qubits, meaning Eve was able to eavesdrop without being detected. This did not occur for any larger key length values, demonstrating the unreliability of the B92 protocol for smaller bit values.

For large key length values, ideal for optimal performance from a security standpoint, the quantum simulation using the present technique was not sustainable, potentially taking days to run the B92 protocol for 1,024 bits using the current methodology. As a result, a new approach for modeling the data was devised in which each possible case was pre-built using the quantum device and then correlated to each outcome, requiring the quantum device to run only one (4,4) quantum circuit rather than construct a (1,1) quantum circuit for each individual bit. This strategy allows a single job to run 1,024 shots, significantly more efficient than the prior method that required two jobs per bit. In general, the program was observed to process key lengths of 1,024 bits in periods ranging from 2 minutes to half an hour.

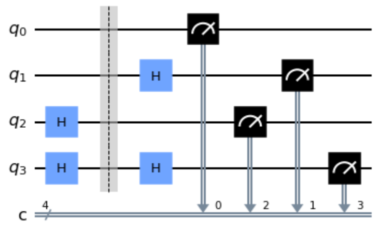

Figure 5. Quantum circuit diagram for updated approach

Presented here is the code for the more efficient approach:

```
import random
import secrets
from qiskit import QuantumCircuit, Aer, transpile, assemble
from qiskit.visualization import plot_histogram, plot_bloch_multivector
from numpy.random import randint
import numpy as np
from qiskit import IBMQ
prov = IBMQ.load_account()

from qiskit.providers.ibmq import least_busy
from qiskit.tools.monitor import job_monitor

qc_backend = least_busy(prov.backends(filters=lambda b: b.configuration().n_qubits >= 4 and
                                   not b.configuration().simulator and
b.status().operational==True))
aer_sim = Aer.get_backend('aer_simulator')
```

doi:10.20944/preprints202207.0279.v1

```
use_QC = 1

def build_circuit(qc):
    if (use_QC == 1):
        t_qc = transpile(qc, qc_backend, optimization_level=3)
    else:
        t_qc = assemble(qc)
    return t_qc

def get_backend():
    if (use_QC == 1):
        return qc_backend
    return aer_sim

def xmit_success(bit, basis):
    if (bit == basis) and (N > 32):
        return 0
    qc = QuantumCircuit(1,1)
    if (bit):
        qc.h(0)
    qc.barrier()
    if (basis):
        qc.h(0)
    qc.measure(0,0)
    t_qc = build_circuit(qc)
    backend = get_backend()
    job = backend.run(t_qc,shots=1, memory=True)
    job_monitor(job)
    exp_result = job.result()
    memory = exp_result.get_memory()
    if (memory[0] == '0'):
        return 0
    else:
        return 1

dbg = 1

N = 16
valid = 0
leaked = 0
corrupted = 0
errs = 0
print(get_backend())
print("N: ",N)

# pre-build all memories using the QC
qc = QuantumCircuit(4,4)
qc.h(2)
qc.h(3)
qc.barrier()
qc.h(1)
qc.h(3)
qc.measure(0,0)
qc.measure(1,1)
qc.measure(2,2)
qc.measure(3,3)
display(qc.draw())

t_qc = build_circuit(qc)
backend = get_backend()

job = backend.run(t_qc,shots=N, memory=True)
```

```
job_monitor(job)  # displays job status under cell
exp_result = job.result()
e_memory = exp_result.get_memory()
print("A->E channel results:")
print(e_memory)
job = backend.run(t_qc,shots=N, memory=True)
job_monitor(job)  # displays job status under cell
# Get the results and display them
exp_result = job.result()
b_memory = exp_result.get_memory()
print("E->B channel results:")
print(b_memory)

def xmit_success_ae(bit, basis, index):
    j = 2*bit + basis # convert into 00, 01, 10, 11
    if (e_memory[index][3-j] == '0'): # memory[] is laid out as q3q2q1q0
        return 0
    else:
        return 1

def xmit_success_eb(bit, basis, index):
    j = 2*bit + basis # convert into 00, 01, 10, 11
    if (b_memory[index][3-j] == '0'): # memory[] is laid out as q3q2q1q0
        return 0
    else:
        return 1


for i in range(N):
               bit = secrets.randbelow(2)
               eve_basis = secrets.randbelow(2)
               bob_basis = secrets.randbelow(2)
               e_val = False # b
               e_cor = False
               res = xmit_success_ae(bit, eve_basis, i)
               if dbg:
                              print("Res 1:",res)
               if res:
                              new_bit =  1-eve_basis
                              if bit != new_bit: #Not known by Eve, Alice, or Bob
                                             errs += 1
                                             e_cor = True
                                             print("Eve silent error at bit:",i)
                              e_val = True
               else:
                              new_bit = eve_basis  #Changes based on eavesdropper scheme

               res = xmit_success_eb(new_bit, bob_basis, i)
               if dbg:
                              print("Res 2:",res)
               if res:
                              valid += 1
                              bob_bit = 1-bob_basis
                              ok = (bob_bit == bit) #Not known by Eve, Alice, or Bob
                              lkd = ok and (e_val == True) and (e_cor == False)
                              if dbg:
                                             print("sent:",bit,"got:",1-bob_basis,"\tSuccess:",
ok,"\tLeaked:",lkd,"\tbit:",i)
                              if lkd:
                                             leaked += 1
                              if (not ok):
                                             corrupted += 1
```

```
Discarded = (N - valid)/N * 100
leaked = leaked/N * 100
corrupted = corrupted/N * 100

print("Discarded:\t\t",Discarded)
print("leaked:\t\t",leaked)
print("corrupted:\t",corrupted)
print("EveErr:\t",100*errs/N)
```

Data:

This is the data gathered from the 1,024-bit program that was executed on the real quantum device:

	Discarded	**Leaked**	**Corrupted**	**Quantum Error**
Average	75.49805688	5.8496094	6.6729844	0.302734375
Standard Deviation	1.02185028	0.524896666	0.60207276	0.166015625

Table 14. Eve sends basis for initial key length of 1024 bits on real quantum hardware

On the real quantum hardware, the B92 protocol performed exceptionally well, with average values close to theoretical values. Quantum errors had little influence on the program, only within a percent of the actual key length. The percentages of bits leaked and corrupted appear to be off by about the same amount as the percentage of quantum errors recorded.

IV. Conclusions

In the long term, quantum key distribution appears to be highly promising. Quantum errors, according to current scientific consensus, are a key hindrance to QKD's possible practical implementation. Because QKD relies largely on error detection, many believe that performing this encryption technique on quantum technology would prove far too noisy to produce usable results. However, my work seems to indicate that quantum errors may actually not be as major a concern as many seem to believe. Given the immense potential of quantum key distribution, I would advise cybersecurity experts and cryptographers to revisit quantum key distribution and refine it further. However, since the quantum devices used are kept within small labs with highly regulated ambient conditions, there is the distinct possibility that the quantum hardware utilized offers substantially better overall performance and considerably smaller errors than long-distance photon transmission. Even in theory, the B92 protocol is not as safe as other contemporary cryptography methods, and it should likely be developed and optimized further before eventually replacing present encryption systems. Unless current cryptographic approaches are rendered obsolete in some way, the B92 QKD protocol does not appear to be a viable replacement until it has been further researched and refined.

The current method for detecting eavesdropping and errors in the B92 protocol entails Alice and Bob exchanging a random sample of bits over a classical channel and checking for errors. This strategy, however, gives Eve an undue advantage as she would only corrupt roughly 6.25% of the bits and, depending on the size of the random sample, may go undetected if the random sample somehow does not include any of the corrupted bits. This is improbable, depending on the size of the random sample. Nonetheless, it presents Eve with an unnecessary additional possibility of escaping undetected that could easily be eliminated by a change in methodology. I propose that Bob encrypts and sends a message to Alice over a traditional channel using the key he generated. Alice would try to decode the message with her version of the key. Failure to decrypt would suggest that the keys are incompatible, which could be attributed to the presence of an eavesdropper. Rather than using a random sample, this approach would incorporate all of the bits used in the final key, reducing the likelihood of Eve escaping unnoticed. Because the error rate is so low, Alice and Bob could repeat this process a few times until success. If there is no eavesdropper present, this should likely work in less than or around ten attempts.

A comparative study of alternative QKD schemes, considerations for further optimization of the B92 protocol based on the suggested eavesdropper schemes and gathered data, and the formulation and simulation of additional eavesdropper schemes are all possible extensions of this research.

Keywords: quantum cryptography, quantum physics, quantum key distribution (QKD), B92 protocol, eavesdropper detection method, qubit, photon polarization

Works Cited

Anghel, C. A Comparison of Several Implementations of B92 Quantum Key Distribution Protocol. Preprints 2021, 2021020486 (doi: 10.20944/preprints202102.0486.v1).

D. Bouwmeester, A. Ekert, A. Zeilinger (Eds.), The Physics of Quantum Information.

Project, QuVis Quantum Mechanics Visualization. *QKD (B92 Protocol)*, https://www.st-andrews.ac.uk/physics/quvis/simulations_html5/sims/cryptography-b92/B92_photons.html.

Quantum Computing Group, IIT Roorkee. "Fundamentals of Quantum Key Distribution-BB84, B92 & E91 Protocols." *Medium*, Medium, 6 Sept. 2021, https://medium.com/@qcgiitr/fundamentals-of-quantum-key-distribution-bb84-b92-e91-protocols-e1373b683ead.

Team, The Qiskit. "Quantum Key Distribution." *Qiskit*, Data 100 at UC Berkeley, 25 Jan. 2022, https://qiskit.org/textbook/ch-algorithms/quantum-key-distribution.html.

Kelley, Sean. “Measuring Photon Polarization.” *The Quantum Atlas*, Joint Quantum Institute, quantumatlas.umd.edu/entry/measuring-polarization. Accessed 2 July 2022.