Quantum Computing: Eavesdropping Detection of Encrypted Messages Over Optical Channels With Spatiotemporal Information Loss

Jai Veilleux
Department of Physics
Edmonds Community College,
20000 68th Ave. Lynnwood, WA 98036
(Dated: June 16, 2023)

As a computer science major the problem of modern encryption systems being vulnerable to attack by quantum computers is extremely interesting. The math underlying the algorithms can be "solved" by a mature quantum computer using Shor's algorithm in logarithmic time, as opposed to the exponential time required by classical computers. By harnessing the power of quantum information systems, we can guarantee that no observer is intercepting our communications due to the no-cloning property of quantum states and mitigate the risk of future decryption. In the event of an observer, significantly more errors will occur during key transmission ensuring detection. At what point does this become indistinguishable from information loss due to distance of signal transmission? I was able to determine that a 5mW laser was capable of transmitting a stable signal up to at least 3m with a 100% laser detection rate and 56% valid base pairs rate.

Keywords: Quantum Computing, Encryption, Eavesdropping Detection, Optical Channel

I. INTRODUCTION

Quantum states can be mapped to classical states using polarized light and Bra-Ket *Dirac* notation. We can use this to represent binary bits from polarized light.

$$|v\rangle \tag{1}$$

$$\langle f| \tag{2}$$

$$\langle f|v\rangle \tag{3}$$

1. Ket - a columnar vector representing a quantum state
2. Bra - a row vector and linear map mapping a vector to a number in the complex plane
3. Bra-Ket - a scalar product of two states

A scalar product of two states

$$\langle 90^\circ|90^\circ\rangle = 1 \tag{4}$$

While orthogonal states

$$\langle 90^\circ|0^\circ\rangle = 0 \tag{5}$$

From this, we can represent states as combinations of other bases:

$$|45^\circ\rangle = \frac{1}{\sqrt{2}}|0^\circ\rangle + \frac{1}{\sqrt{2}}|90^\circ\rangle \tag{6}$$

Why $\frac{1}{\sqrt{2}}$?

$$\langle 45^\circ|45^\circ\rangle = \begin{bmatrix}\frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}}\end{bmatrix}\begin{bmatrix}\frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}}\end{bmatrix} = 1 \tag{7}$$

Squaring the absolute value of a scalar product gives the probability of the expected result from a Ket in decimal form. For the purposes of this experiment, that the received bit matches the transmitted bit. The 0 from orthogonal states is used to detect the 0-bit.

$$\langle 45^\circ|0^\circ\rangle = |\frac{1}{\sqrt{2}}\langle 45^\circ|45^\circ\rangle + \frac{1}{\sqrt{2}}\langle 45^\circ|-45^\circ\rangle|^2 = \frac{1}{2} \tag{8}$$

This has important consequences, when the incident of polarized light does not match there is only a 50% chance that the data received is correct. Introducing a third-party observer which must randomly select a base decreases this probability to 25%. This will be the threshold for the experiment at which data loss would be indistinguishable from an observer.

II. METHODOLOGY

This experiment utilizes four different Bra's as bases, each corresponding to either the + or - base:

|+: 0°& 90°polarizations|X: -45°& 45°polarizations|

Either base can represent a binary bit using a specific angle of polarization:

|0: 90°& -45°polarizations|1: 0°& 45°polarizations|

Due to a lack of polarizing lenses, this is simulated computationally on an Arduino UNO board. The cryptographic algorithm utilized is a single use key being generated consisting of binary digits the same length as the message. If the message were to be transmitted, binary addition of the key to the message would be applied by Alice for encryption. Subsequently, Bob would again apply binary addition of the key to the encrypted message to decrypt it. This would be guaranteed to be safe due to any observer being detected when generating the key. For each trial, distance between transmitter (Alice) and sensor (Bob) is increased. Intervals used: 1m, 2m, 3m

III. MATERIALS AND APPARATUS

1. Arduino Uno
2. KY-008 Laser
3. Laser sensor (https://www.amazon.com/Acxico-Arduino-Transmitter-Receiver-Non-modulator/dp/B082PFX8LK/)
4. Arduino wiring

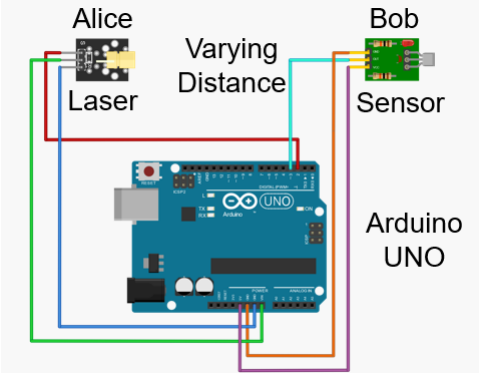

FIG. 1. Arduino schematic

IV. COMPUTATION AND PROCEDURE

All steps in procedure completed using code available on github: `https://github.com/Didgety/QKD_Principles_Demo/`.

1. Alice generates a random set of bases: (0°, 90°, -45°, or 45°) and corresponding bits. 52 bases were generated for a 20-bit message to ensure a long enough key.
2. Bob generates a random set of bases (0°or 45°).
3. Alice then transmits each bit via pulsed laser using the generated base to Bob.
4. Bob records the bit received, if the base does not match or there is signal loss a random bit is chosen, this simulates the 50% probability in a quantum system
5. Alice and Bob then compare their bases and keep bits with matching bases to be used as the key.
6. If only 25% of bases and bits match, there is an eavesdropper or signal loss so significant as to be indistinguishable, and the key is discarded.

V. DATA

See Figure 2 - Figure 13 in Appendix B.

VI. ANALYSIS & CONCLUSIONS

Each trial had 100% signal receipt, and an expected ratio of valid base pairs around 50% when generating the encryption key. One trial had 52%, but that is within a reasonable bound. This indicates that transmission is extremely stable over short distances, even with a low power light source. With no errors, the primary goal for future research would be scaling the experiment up to a more practical size.

VII. FUTURE WORK

In the future, my primary goal would be to scale this experiment up to real world sizes. A signficant increase in transmission distance is required for practical use cases. Additionally, I would like to physically polarize the light instead of relying on software. I would also want to use fiber optic cables since that would best mirror real world scenarios. Finally, I would like to use a trapped photon source for true randomness.

VIII. ACKNOWLEDGEMENTS

Thank you to Professor Tom Fleming for being supportive and helping me obtain the necessary materials for this experiment, and to the Edmonds College Foundation for their generous support.

IX. APPENDIX

A. CODE

```
#define laser 2 // port 2 on arduino
#define sensor 3 // port 3 on arduino
unsigned long signalPeriod = 1000;
    //milliseconds
bool state; // 1: laser detected, 0:
    no laser detected

const unsigned int messageLength = 20;
    // num of bits in message
const unsigned int keyLength = 52; //
    num of transmissions with random
    basis (longer than message to
    ensure enough matching bases for a
    long enough key)

/*
[+ basis: 0, 2], [x basis: 1, 3], [0 =
    0 deg, 1 = 45 deg, 2 = 90 deg, 3 =
    -45 deg]
*/
int aliceBasisList[keyLength];
int bobBasisList[keyLength];

int aliceKeyBits[keyLength];

int laserDetected = 0;
int keyDetectionList[keyLength];

int validBases[messageLength];
int validKeys[messageLength];
int validIndex[messageLength];

int encryptedMessage[messageLength];
int receivedMessage[messageLength];
int decryptedMessage[messageLength];

// using 5 bit binary for
    simplification, 1 = A, 2 = B, etc.
// ref:
    https://www.binary-code.org/bits/5
int message[] = {0, 0, 0, 1, 0,
                 0, 1, 1, 1, 1,
                 0, 1, 0, 0, 0,
                 1, 0, 0, 1, 0}; //"BOHR"

void setup() {
Serial.begin(9600); // 9600 baud
```

```
randomSeed(random(1251, 9001));

pinMode(laser, OUTPUT);
pinMode(sensor, INPUT);

genBases();

delay(signalPeriod); // give program
    time to think before printing,
    unusual artifacts in output
    otherwise
// print initial values
//[0 = 0 deg, 1 = 45 deg, 2 = 90 deg,
    3 = -45 deg]
Serial.print("\nAlice Bases, ");
for(int i = 0; i < keyLength; i++) {
    switch (aliceBasisList[i]) {
    case 0:
        Serial.print("0 deg, ");
        break;
    case 1:
        Serial.print("45 deg, ");
        break;
    case 2:
        Serial.print("90 deg, ");
        break;
    case 3:
        Serial.print("-45 deg, ");
        break;
    }
}
Serial.print("\nBob Bases, ");
for(int i = 0; i < keyLength; i++) {
    switch (bobBasisList[i]) {
    case 0:
        Serial.print("0 deg, ");
        break;
    case 1:
        Serial.print("45 deg, ");
        break;
    }
}
Serial.print("\nAlice Key Bits, ");
for(int i = 0; i < keyLength; i++) {
    Serial.print(aliceKeyBits[i]);
    Serial.print(", ");
}
}

void loop() {
digitalWrite(laser, LOW); // start
    with laser off

int numErrors = basisTest();

bool validKey = keyGen();
while (validKey == false) {
    Serial.println("Insufficient
        matching values for message
        size, regenerating");

    double errorRate =
        (double)(keyLength - numErrors)
        / (double)keyLength;

    Serial.print("\nAccuracy: ");
    Serial.println(errorRate);
    Serial.print("\nLaser detections:
        ");
    Serial.println(laserDetected);

    genBases();
    basisTest();

    // print new values
    Serial.print("New Alice Bases, ");
    for(int i = 0; i < keyLength; i++) {
    switch (aliceBasisList[i]) {
    case 0:
        Serial.print("0 deg, ");
        break;
    case 1:
        Serial.print("45 deg, ");
        break;
    case 2:
        Serial.print("90 deg, ");
        break;
    case 3:
        Serial.print("-45 deg, ");
        break;
    }
    //Serial.print(aliceBasisList[i]);
    //Serial.print(", ");
    }
    Serial.print("\nNew Bob Bases, ");
    for(int i = 0; i < keyLength; i++) {
    switch (bobBasisList[i]) {
    case 0:
        Serial.print("0 deg, ");
        break;
    case 1:
        Serial.print("45 deg, ");
        break;
    }
    //Serial.print(bobBasisList[i]);
    //Serial.print(", ");
```

```
    }
    Serial.print("\nNew Alice Key Bits,
        ");
    for(int i = 0; i < keyLength; i++) {
    Serial.print(aliceKeyBits[i]);
    Serial.print(", ");
    }
}

double errorRate = (double)(keyLength
    - numErrors) / (double)keyLength;

Serial.print("\nAccuracy, ");
Serial.println(errorRate);
Serial.print("\nLaser detections, ");
Serial.println(laserDetected);
// print the bits generated
Serial.print("Bits generated, ");
for (int i = 0; i < keyLength; i++) {
    Serial.print(keyDetectionList[i]);
    Serial.print(", ");
}
// print the index value of valid base
    and key from initially generated
    values
Serial.print("\nValid base/key index,
    ");
for (int i = 0; i < messageLength;
    i++) {
    Serial.print(validIndex[i]);
    Serial.print(", ");
}
// print the valid bases
Serial.print("\nValid bases, ");
for (int i = 0; i < messageLength;
    i++) {
    Serial.print(validBases[i]);
    Serial.print(", ");
}
// print the key generated
Serial.print("\nKey bits, ");
for (int i = 0; i < messageLength;
    i++) {
    Serial.print(validKeys[i]);
    Serial.print(", ");
}


delay(signalPeriod); // delay before
    halting processor to complete any
    serial communications
exit(0); // halts the processor
    preventing [void loop()] from
    repeating the experiment
}

// generate random bases and bits for
    key generation
//[+ basis: 0, 2], [x basis: 1, 3], [0
    = 0 deg, 1 = 45 deg, 2 = 90 deg, 3
    = -45 deg]
void genBases() {
for (int i = 0; i < keyLength; i++) {
    aliceBasisList[i] = random(0, 4);
    bobBasisList[i] = random(0, 2);
    switch (aliceBasisList[i]) {
    case 0:
        aliceKeyBits[i] = 1;
        break;
    case 1:
        aliceKeyBits[i] = 1;
        break;
    case 2:
        aliceKeyBits[i] = 0;
        break;
    case 3:
        aliceKeyBits[i] = 0;
        break;
    }
}
}

// generates bits based on random
    bases for both alice and bob and
    an initial random selection of
    bits by alice
//[+ basis: 0, 2], [x basis: 1, 3], [0
    = 0 deg, 1 = 45 deg, 2 = 90 deg, 3
    = -45 deg]
int basisTest() {
int errors;
for (int i = 0; i < keyLength; i++) {
    digitalWrite(laser, HIGH); // laser
        on
    state = 1;
    // if laser is not detected assign
        a random value to the key list
    // rework this? but if no value is
        assigned the array will have a
        null value causing problems.
    // a random value could be
        considered the same as an error
        for the purposes of this
        experiment
    if (state == 0) {
```

```
    keyDetectionList[i] = random(0, 2);
        // random(low, high) low -
        inclusive, high - exclusive
    errors++;
    }
    // if alice polarized at 0 deg or
        90 deg and bob polarized at 0
        deg record the bit (should be 0)
    else if (state == 1 &&
        (((aliceBasisList[i] == 0) ||
        (aliceBasisList[i] == 2)) &&
        (bobBasisList[i] == 0))) {
    keyDetectionList[i] =
        aliceKeyBits[i];
    laserDetected++;
    }
    // if alice polarized at 45 or -45
        deg and bob polarized at 45 deg
        record the bit (should be 1)
    else if (state == 1 &&
        (((aliceBasisList[i] == 1) ||
        (aliceBasisList[i] == 3)) &&
        (bobBasisList[i] == 1))) {
    keyDetectionList[i] =
        aliceKeyBits[i];
    laserDetected++;
    }
    // if the bases do not match assign
        a random bit (0 or 1) to the
        key list as there is 50% chance
        of error in a physical system
    else {
    keyDetectionList[i] = random(0, 2);
    laserDetected++;
    errors++;
    }
    delay(signalPeriod); // 1 second
        delay between laser pulses
    digitalWrite(laser, LOW); // laser
        off
    delay(signalPeriod);
}
return errors;
}

// assembling the key, same base
    checking logic as basisTest()
bool keyGen() {
int keyCount = 0;
for (int i = 0; i < keyLength; i++) {
    if (((aliceBasisList[i] == 0) ||
        (aliceBasisList[i] == 2)) &&
        (bobBasisList[i] == 0)) {
        validKeys[keyCount] =
            keyDetectionList[i];
        validBases[keyCount] =
            aliceBasisList[i];
        validIndex[keyCount] = i;
        keyCount++;
        } else if (((aliceBasisList[i] ==
            1) || (aliceBasisList[i] == 3))
            && (bobBasisList[i] == 1)) {
        validKeys[keyCount] =
            keyDetectionList[i];
        validBases[keyCount] =
            aliceBasisList[i];
        validIndex[keyCount] = i;
        keyCount++;
        }
        if (keyCount == messageLength) {
        return true; // returns true when a
            number of bits with matching
            bases are found
        }
    }
    return false; // returns false when
        there are insufficient matching
        bits, new bases and bits need to
        be chosen using genBases() and
        basisTest()
    }
```

B. Data

- + basis: 0, 2
- X basis: 1, 3,
- 0 = 0°, 1 = 45°, 2 = 90°, 3 = -45°

FIG 2: Trial 1 (1m) Pt. 1

Alice Bases	Bob Bases	Alice Bits	Bits generated
-45 °	45 °	0	0
45 °	0 °	1	1
90 °	0 °	0	0
0 °	0 °	1	1
-45 °	45 °	0	0
0 °	45 °	1	0
0 °	0 °	1	1
-45 °	45 °	0	0
-45 °	45 °	0	0
0 °	0 °	1	1
-45 °	45 °	0	0
45 °	45 °	1	1
0 °	45 °	1	1
-45 °	0 °	0	1
0 °	45 °	1	0
45 °	0 °	1	1
0 °	45 °	1	0
90 °	45 °	0	1
-45 °	45 °	0	0
-45 °	45 °	0	0
-45 °	0 °	0	1
45 °	0 °	1	1
45 °	45 °	1	1
0 °	45 °	1	1
90 °	45 °	0	0
45 °	45 °	1	1
0 °	0 °	1	1

FIG 3: Trial 1 (1m) Pt. 2

Alice Bases	Bob Bases	Alice Bits	Bits generated
0 °	0 °	1	1
0 °	0 °	1	1
90 °	45 °	0	0
90 °	0 °	0	0
45 °	0 °	1	0
45 °	45 °	1	1
45 °	0 °	1	1
0 °	45 °	1	1
90 °	45 °	0	0
-45 °	45 °	0	0
45 °	0 °	1	0
-45 °	0 °	0	0
45 °	45 °	1	1
-45 °	0 °	0	1
45 °	45 °	1	1
45 °	45 °	1	1
45 °	0 °	1	0
-45 °	0 °	0	1
0 °	45 °	1	1
45 °	0 °	1	1
0 °	0 °	1	1
0 °	45 °	1	1
90 °	0 °	0	0
45 °	45 °	1	1
-45 °	0 °	0	0

FIG 4: Trial 1 (1m) Pt. 3

Valid base/key index	Valid bases	Key bits
0	3	0
2	2	0
3	0	1
4	3	0
6	0	1
7	3	0
8	3	0
9	0	1
10	3	0
11	1	1
18	3	0
19	3	0
22	1	1
25	1	1
26	0	1
27	0	1
28	0	1
30	2	0
32	1	1
36	3	0

FIG 5: Trial 1 (1m) Pt. 4

Laser detections	Valid base pairs
52/52	26/52
100%	50%

FIG 6: Trial 2 (2m) Pt. 1

Alice Bases	Bob Bases	Alice Bits	Bits generated
90 °	0 °	0	0
0 °	0 °	1	1
90 °	45 °	0	1
-45 °	45 °	0	0
0 °	0 °	1	1
45 °	0 °	1	0
90 °	45 °	0	1
-45 °	0 °	0	0
0 °	0 °	1	1
0 °	45 °	1	1
0 °	0 °	1	1
-45 °	0 °	0	1
45 °	45 °	1	1
-45 °	0 °	0	1
-45 °	45 °	0	0
-45 °	45 °	0	0
0 °	0 °	1	1
45 °	0 °	1	1
0 °	0 °	1	1
90 °	45 °	0	1
90 °	45 °	0	1
-45 °	0 °	0	1
-45 °	45 °	0	0
90 °	0 °	0	0
0 °	0 °	1	1
45 °	45 °	1	1
0 °	45 °	1	0

FIG 7: Trial 2 (2m) Pt. 2

Alice Bases	Bob Bases	Alice Bits	Bits generated
0 °	45 °	1	1
45 °	45 °	1	1
45 °	0 °	1	0
45 °	0 °	1	1
0 °	0 °	1	1
-45 °	45 °	0	0
-45 °	0 °	0	0
90 °	45 °	0	0
0 °	0 °	1	1
0 °	45 °	1	1
45 °	45 °	1	1
-45 °	0 °	0	1
90 °	0 °	0	0
-45 °	0 °	0	1
45 °	0 °	1	1
-45 °	0 °	0	0
0 °	0 °	1	1
-45 °	0 °	0	1
0 °	45 °	1	0
90 °	0 °	0	0
90 °	45 °	0	0
0 °	0 °	1	1
90 °	0 °	0	0
0 °	0 °	1	1
-45 °	0 °	0	0

FIG 8: Trial 2 (2m) Pt. 3

Valid base/key index	Valid bases	Key bits
0	2	0
1	0	1
3	3	0
4	0	1
8	0	1
10	0	1
12	1	1
14	3	0
15	3	0
16	0	1
18	0	1
22	3	0
23	2	0
24	0	1
25	1	1
28	1	1
31	0	1
32	3	0
35	0	1
37	1	1

FIG 9: Trial 2 (2m) Pt. 4

Laser detections	Valid base pairs
52/52	26/52
100%	50%

FIG 10: Trial 3 (3m) Pt. 1

Alice Bases	Bob Bases	Alice Bits	Bits generated
90 °	0 °	0	0
90 °	45 °	0	1
90 °	0 °	0	0
0 °	0 °	1	1
-45 °	0 °	0	0
0 °	0 °	1	1
45 °	0 °	1	0
90 °	0 °	0	0
90 °	45 °	0	0
0 °	0 °	1	1
45 °	0 °	1	1
0 °	0 °	1	1
0 °	45 °	1	1
45 °	0 °	1	1
0 °	45 °	1	0
90 °	45 °	0	0
-45 °	45 °	0	0
90 °	45 °	0	0
0 °	0 °	1	1
45 °	0 °	1	0
45 °	45 °	1	1
90 °	45 °	0	1
0 °	45 °	1	0
-45 °	45 °	0	0
45 °	0 °	1	0

FIG 11: Trial 3 (3m) Pt. 2

Alice Bases	Bob Bases	Alice Bits	Bits generated
90 °	0 °	0	0
0 °	45 °	1	0
90 °	45 °	0	1
0 °	0 °	1	1
45 °	0 °	1	0
90 °	0 °	0	0
90 °	0 °	0	0
45 °	45 °	1	1
90 °	0 °	0	0
0 °	0 °	1	1
0 °	0 °	1	1
90 °	45 °	0	0
45 °	0 °	1	1
45 °	0 °	1	1
45 °	0 °	1	0
-45 °	0 °	0	1
90 °	45 °	0	1
90 °	0 °	0	0
45 °	45 °	1	1
90 °	0 °	0	0
0 °	45 °	1	1
-45 °	0 °	0	0
0 °	0 °	1	1
90 °	0 °	0	0
0 °	0 °	1	1
0 °	0 °	1	1
0 °	0 °	1	1

FIG 12: Trial 3 (3m) Pt. 3

Valid base/key index	Valid bases	Key bits
0	2	0
2	2	0
3	0	1
5	0	1
7	2	0
9	0	1
11	0	1
16	3	0
18	0	1
20	1	1
23	3	0
25	2	0
28	0	1
30	2	0
31	2	0
32	1	1
33	2	0
34	0	1
35	0	1
42	2	0

FIG 13: Trial 3 (3m) Pt. 4

Laser detections	Valid base pairs
52/52	27/52
100%	52%

X. BIBLIOGRAPHY

[1] Bloom, Y., Fields, I., Maslennikov, A., Rozenman, G. (2022) *Quantum Cryptography - A Simplified Undergraduate Experiment and Simulation*. Physics, 4, 104-123.

[2] "Bra-Ket Notation." *Math Is Fun, Bra-Ket Notation. Math Is Fun.* Available at: `www.mathsisfun.com/physics/bra-ket-notation.html`.

[3] Libretexts (2023) *Bra-Ket Notation, Chemistry LibreTexts.* Available at: `chem.libretexts.org/Bookshelves/Physical_and_Theoretical_Chemistry_Textbook_Maps/Supplemental_Modules_(Physical_and_Theoretical_Chemistry)/Quantum_Mechanics/03._The_Tools_of_Quantum_Mechanics/Bra-Ket_Notation`.

[4] Tsai C., Yang, C., Lin, J., Chang, Y., Chang, R. (2021) *Quantum Key Distribution Networks: Challenges and Future Research Issues in Security.* Applied Science, 11, 3767.

[5] Sharma, P., Agrawal, A., Bhatia, V., Prakash, S., Misha, A. (2021) *Quantum Key Distribution Secured Optical Networks: A Survey.* IEEE Open Journal of the Communications Society, 2, 2049-2083

[6] Kumar, Manish. (2022) *Post-quantum cryptography Algorithm's standardization and performance analysis.* Array, 15, 100242.

[7] Shapoval, I., and Duplij, S. (2007). *Quantum Computations: Fundamentals and Algorithms.* Problems of Atomic Science and Technology, 3, 230-235.

[8] R. Alleaume, C. Branciard, J. Bouda, T. Debuisscher, M. Dianati, N. Gisin, M. Godfrey, P. Grangier, T. Langer, N Lutkenhaus, C. Monyk, P. Paincahult, M. Peev, A. Poppe, T. Pornin, J. Rarity, R. Renner, G. Ribordy, M. Riguidel, L. Salvail, A. Shields, H. Weinfurter, A. Zeilinger. (2009) *Using quantum key distribution for cryptographic purposes: A survey.* Theoretical Computer Science, 560, 62-81.