Analysis of Post Quantum Cryptography

Lee Dale

May 7, 2023

1 Introduction

Currently the world of cryptography is solely based on what classical computers can achieve with well know Turing machine based algorithms. Quantum computers don't come into the picture when it comes to general use encryption. This is set to change as quantum computers start to gain enough Qubits to successfully implement general use quantum algorithms that can break the encryption of our current state of the art public key exchange mechanisms [Bel+21]. Once this happens there will be a need for robust quantum key exchange methods that allow privacy of message exchange to be maintained, even when subjected to a quantum computing algorithms. Quantum computers with new quantum based algorithms will also make it possible to brute force current encryption keys in polynomial time. This puts any data that is currently encrypted with todays state of the art encryption algorithms in jeopardy of being cracked and made available for anyone with a quantum computer to view. Once a quantum computer with enough Qubits becomes available for general use it will mean they can also be used to keep communications between two parties secret.

This paper outlines the current weaknesses in todays cryptography schemes including public key exchange methods that rely on the a mathematically hard to compute private key, and why these will be vulnerable to quantum computers in the future. It will then outline some key solutions to this problem that will be robust in a post quantum world.

2 Literature Review

2.1 Overview of current key exchange mechanisms

As it stands pretty much all of communication methods used over the internet are based on asymmetric cryptography

The methods currently available to us for key exchange fall into three main categories that all rely on the premise that there is publicly available information used for the key exchange between two parties and there is also private information only known to the individual. The secrecy of the public key exchange relies on the fact that it is hard to compute using a typical Turing state machine approach of classical computers

The two main mathematical problems that most public key exchange mechanisms are built on are prime factorization and discrete logarithms [And20]. Discrete logarithms can be further broken down into two categories, with the addition of discrete logarithms that use elliptic curves. I will outline how these are implemented and how each one provides security guarantees based on it's particular mathementical problem.

2.1.1 Prime Factorization

In current key exchange methods such as the RSA method, prime factorization is the main basis for the algorithm's security.

A factorization problem hinges on the fact that it is easy to compute a number N from two primes, but it is difficult to compute which prime numbers are factors of the product N. Imagine two 64 bit prime numbers used to create a 128bit modulus. A 64 bit number can be in the range

$$2^{64}$$

which is a large number. To brute force the factorization you would have to compute all possible primes that factorize with another 64 bit prime to make N

The RSA algorithm works by selecting two large prime numbers, p and q, and multiplying them together to obtain a product, $N = p * q$. N is called the modulus and is used for generating the private and public keys.

The private key in RSA cryptography is derived from the factors p and q of n, which are kept secret. The public key, on the other hand, is the modulus n and an exponent e and are shared between the two parties.

When we encrypt a message with RSA, we convert the message into a number and then raise that number to the power of the exponent e, we then take the modulo N. The resulting ciphertext can only be decrypted using the private key. To obtain this private key would involve factoring the modulus n back into its two prime factors p and q.

The hardness of the prime factorization problem comes from the fact that to try and mathematically compute which prime factors make up the modulus would need a classical computer to execute an algorithm that tries all possible factors. This would fall into exponential time complexity and therefore, for any large enough modulus becomes an impossible problem to solve for a classical computer in any reasonable amount of time. In contrast to this a quantum computer with the right algorithm would be able to easily try all possible factors in polynomial time.

2.1.2 Discrete Logs

Discrete logarithms are used in various common public key cryptography algorithms, such as ElGamal encryption and the Diffie-Hellman key exchange which are both used in popular web browser implementations. Discrete logs can also be used in digital signature schemes like the Digital Signature Algorithm (DSA).

In discrete log algorithms the difficulty comes from the fact that computing a discrete logarithm in certain mathematical groups is a hard problem and is used as the guarantee of security.

In the context of public key cryptography, the groups used are typically elliptic curves, and the discrete logarithm problem refers to the difficulty of finding the exponent required to generate a given element of the group from a fixed base element.

For example, in the Diffie-Hellman key exchange protocol, two parties agree on a generator and a prime number, and publicly exchange their exponentiation of the generator with a randomly generated value modulus the prime number. The shared secret key is computed from their respective exponentiation of each others public key with their random value modulus the prime number. The difficulty for an attacker is determining the secret exponents used by both parties to generate the public key.

The discrete logarithm problem is a key component of many public key cryptography algorithms, where it is used to provide security guarantees by making it computationally infeasible for an attacker to determine a secret key from its corresponding public key. As with the RSA method, any classical computer would end up using an algorithm with

exponential time complexity, where a quantum computer would execute in polynomial time.

2.1.3 Discrete logs with Elliptic Curves

Elliptic curve cryptography is a type of public-key cryptography that is based on the mathematics of elliptic curves. Elliptic Curve Diffie-Hellman (ECDH) is an example of a public key exchange based on elliptic curves and is widely used in web browsers.

In ECDH, two parties, first agree on a publicly known elliptic curve. They then choose a base point on the curve, which is also publicly known.

Each party, then chooses a secret value, which is kept private. They use these secret values to compute their respective public keys.

Lets say that two parties called Alice and Bob want to communicate using ECDH. They compute their public keys as follows:

Alice generates her public key

$$A = ag$$

Bob generates his public key

$$B = bg$$

Alice and Bob then exchange their public keys A and B over an insecure channel.

Where Alice's secret value is a, Bob's is b and g is the generator which is a value chosen to lie on the elliptic curve.

Once Alice and Bob have exchanged their public keys, they can derive a shared secret key using the following steps:

Alice generates the point

$$K = aB$$

Bob generates the point

$$K = bA$$

The shared secret key is then derived from the x-coordinate of the point K which matches for both Alice and Bob.

The security of ECDH comes from the fact that it is difficult to compute the private keys from the public keys, even if the elliptic curve and the generator are known.

All the methods above are vulnerable to quantum computers by the fact that they rely on the exponential time complexity of the mathematically hard problem. As we will understand next quantum computers can drastically reduce the execution time of finding out the secret keys which could make all current cryptography methods obsolete.

2.2 How current mechanisms are vulnerable to quantum computers

In a 1984 paper David Deutch [Deu85] discussed a universal quantum computer in which he shows that a quantum computer can evolve computational basis states into linear superposition's of each other. Building on this work a decade later it was then shown by Peter Shor [Sho94] that it was possible to design algorithms to factor large integer values and find discrete logarithms in polynomial time. These algorithms threaten the viability of the key exchange mechanisms discussed above and render them unusable should a quantum computer be built with enough Qubits to successfully run Shor's proposed algorithms.

It is considered that an algorithm is efficient when the number of steps grow as a polynomial to the size of its input. [Sho94] However there exists no algorithm for factoring large integers that meets this criteria based on a classical computer. Montgomery [Mon94]

states "No polynomial-time algorithm for solving this problem is known ." Shor however in his 1994 paper outlined [Sho94] such an algorithm that can be executed on a quantum computer that does meet this criteria.

The issue for any asymmetric cryptography that relies on large integer factorization such as the popular RSA method is that instead of taking exponential time to find the prime factors of a large integer, a quantum computer will carry out this calculation in polynomial time.

If we use a brute force a method that takes 2 seconds per key to try all the keys for a 4 bit encryption key using an algorithm that has exponential time complexity we get:

$$Keys = 2^4 = 16$$
$$f((2^2)^4)) = 65536$$

Instead with polynomial time we get:

$$f((2^4)^2) = 256$$

We can see the polynomial time algorithm will be much faster to brute force an integer factorization problem. The time taken will also grow less quickly with the polynomial time complexity than with the exponential time complexity.

This means with a quantum computer it will become feasible for an attacker to find out the prime factors that make up a public key in a cryptographic encryption scheme that relies on the hardness of the prime factorization problem, and by extension discrete logs. If there came a time when a quantum computer could execute Shor's algorithms for 2048 bit encryption keys, this would render all current methods for public key cryptography useless.

2.3 Quantum robust methods

Various methods have been devised to deal with a world where quantum computers are able to crack encryption keys up to 2048 bits long. Kumar and Garhwal state that quantum protocols are based on either the Heisenberg Uncertainty principle or quantum entanglement [KG21] I will outline the most notable methods in this section and then focus on quantum key distribution based on quantum entanglement.

2.3.1 Lattice based cryptography

Lattice based cryptography is a promising area of post quantum cryptography. At its heart lattice based cryptography is built on the premise that finding a mathematical solution to a lattice based problem is said to be a hard problem to solve. Lattices are generated by starting with a set of basis vectors and generating a lattice by adding new vectors to the basis vectors. An example of a problem based around a lattice is the closest vector problem. The closest vector problem might ask us to find the set of vectors that give us the point closest to the origin but not equal to the origin. Brute forcing this problem is extremely hard computationally and the hardness can be increased to a point where solving it falls into exponential time complexity by increasing the dimensions of the lattice to a higher number such as 100. It is said that there exists no polynomial time algorithm that can approximate this lattice based problem within polynomial factors [MR08]. Lattice based problems are also said to be hard problems for quantum computers as there doesn't yet exist any known algorithm for a quantum computer that can solve higher dimensional lattice based problems in less than exponential time [MR08].

Even though there is currently no polynomial time algorithm for solving higher dimensional lattice based problems this doesn't prove that lattice based cryptography will never be susceptible to a new algorithm for quantum computers.

Lattice based cryptography is therefore a way to generate encryption keys that are hard to crack for quantum computers in the same way as current large encryption keys are hard to crack for classical computers. Lattice based cryptography is great for encryption in a post quantum world but we still need a secure method of key exchange when dealing with asymmetric public key based forms of communication.

2.3.2 Quantum Key Distribution

The no cloning theorem in quantum mechanics leads us to an interesting way of distributing secret keys between two parties. If we use a secret key that exists in a superposition of entangled states and subsequently share this secret key between two parties, there would always be a way to detect whether the key had been tampered with in transit. This is explained by the fact that if an eavesdropper had modified the key in any way (in quantum mechanics we say that the entangled state was observed) the secret key would have de-cohered into one of the many possible superposition of states and therefore each party would be able to detect the eavesdroppers attempt at intercepting the secret key.

2.3.3 Quantum key distribution using entangled photons

From Kumar and Garhwal [KG21] quantum key distribution protocols can be classified as either; Discrete variable QKD or Continuous variable QKD. Discrete variable QKD is based around the spin of an electron or the polarization of a single photon. Continuous variable QKD is mainly based around the use of a laser to store information in the form of a continuous light source.

We will concentrate here on a discrete variable QKD example in the BB84 protocol which uses the polarization of single photons.

The BB84 protocol was first introduced by Charles Bennet and Gilles Brassard in 1984 [BB20] . In the BB84 protocol the two parties, say Alice and Bob establish two communication channels. The first is a classical communication channel that conforms to the laws of classical physics and the second is a quantum communication channel which conforms to the laws of quantum mechanics. Any eavesdropping on communications over the classical channel is undetectable, in contrast communications over the quantum channel by virtue of the laws of quantum mechanics can highlight with high probability, any interference in the set of bits transmitted . An encryption key is established over the quantum communication channel and this cryptographic key is encoded using the polarization states of individual photons sent over the channel. Furthermore the polarization states of the photons are entangled into a single quantum wave function. Bob and Alice use the a set of random bits sent over the quantum communication channel as a one time pad for encryption. They agree over the classical channel whether their bits have been interfered with and discard any bits that they deem compromised. Once they have enough bits to form an encryption key, they can use this key to encrypt messages over the classical channel, happy in the knowledge that they have used a random, one-time encryption key that nobody else has knowledge of.

The interesting thing about the BB84 protocol is that both channels can be public channels and the two parities do not need to share any prior information before exchanging a symmetric key. The symmetric key exchanged can be guaranteed to be secret as any form in eavesdropping will be detected by either party. No amount of computing power can avoid detection over the quantum channel and this is why QKD is a quantum robust

method of communication. The technical details of the BB84 protocol are outlined in the implementation section.

2.3.4 Quantum key distribution using Muon detection

Quantum key distribution based on entanglement and the Heisenberg's uncertainty principle have been shown to have weakness that could allow an eavesdropper to go undetected in the communication process. It has been shown possible to shine bright light on single-photon detectors within the quantum communication channel turning them into classical devices. This is the basis of a faked-state attack [Gus+09] in which Eve is able to eavesdrop on the quantum key exchange between Alice and Bob and stay undetected. Instead of using entangled photos as a source of quantum bits in which to transfer between Alice and Bob, a recent article [CCE23] proposed the use of Cosmic rays that rain down on Alice and Bob from deep space as a source of quantum randomness in which to form an encryption key. This method was created by Hiroyuki Tanaka at the University of Tokyo and is called Cosmic Coding and Transfer (COSMOCAT). The way this works is as the cosmic rays hit the Earths atmosphere they decay into Muons, and each participant in the communication can detect these Muons with a detector. They can then generate the same secret keys from the timestamp of when the individual Muons hit the detectors and thus never have to exchange a secret key or send each other bits over a quantum communication channel.

The advantage of this method is that the source of bits comes from an external source and can never be interfered with by an eavesdropper. Currently limitations in the hardware accuracy mean that only about 20% of the Muons detected are able to be used for a secret key.

3 Implementation

The code [Dal23] in listing 1 is a Q# implementation of the BB84 protocol outlined above and runs through the steps of Alice and Bob establishing a quantum channel to send bits they can later use to encrypt a message. If Eve interferes with the quantum state of the Qubits then Alice and Bob will obtain measurement values that differ when compared to random bases, with a difference of more than 50% being an indication that someone had eavesdropped on the communication channel.

Listing 1: Q# Microsoft Quantum Development Kit

```
namespace Cryptography.BB84.Quantum {

    open Microsoft.Quantum.Convert;
    open Microsoft.Quantum.Random;
    open Microsoft.Quantum.Canon;
    open Microsoft.Quantum.Intrinsic;

    @EntryPoint()
    operation RunBB84Example(length: Int): Result {
        Message("Running␣BB84...");

        // Create qubits for key length
        Message("Creating␣Qubits␣with␣length␣" + IntAsString(↩
            length) + "...");
        use qubits = Qubit[length];
```

```
    // Alice creates bits based on random bases.
    Message("Alice encoding bits...");
    let (aliceBases, aliceBitValues) = EncodeBits(qubits);


    // ------------------------------------------------
    // Qubits sent across quantum channel to Bob
    // ------------------------------------------------
    //
    Message("Sending bits across quantum channel...");

    // Eve interferes with the qubits
    Message("Eve interferring with qubits...");
    let (eveBases, eveResult) = Eavesdrop(qubits, 1.0);

    // Bob now selects random basis and measures qubit ↩
        using either Pauli X or Pauli Z basis.
    Message($"Bob measuring qubits...");
    let (bobBases, bobResults) = GetResults(qubits);

    // Alice and Bobs bases are now compared and the ↩
        shared values are returned.
    Message("Getting shared basis results...");
    let (sharedValues, sharedResults) = GetSharedResults(↩
        aliceBitValues, bobResults, aliceBases, bobBases);

    // We now check for eavesdropping by comparing Alice ↩
        and Bobs results and determining the percentage of ↩
        differences.
    Message("Getting percentage difference in results...")↩
        ;
    let (indicesToUse, percentage) = CheckForEavesdropping↩
        (sharedValues, sharedResults);
    Message("Difference " + DoubleAsString(percentage) + "↩
        %");

    Message("Getting shared key...");
    let key = GetSharedKey(indicesToUse, sharedValues);
    Message("Key length is " + IntAsString(Length(key)));

    return Zero;
}

operation GetSharedKey(indicesToUse: Int[], sharedValues: ↩
    Bool[]) : Bool[] {
    mutable keyBits = [false, size = 0];

    for index in indicesToUse {
        set keyBits += [sharedValues[index]];
    }
```

```
    return keyBits;
}

operation CheckForEavesdropping(values: Bool[], results: ↩
    Bool[]) : (Int[], Double) {
    mutable differences = 0;
    mutable indicesToUse = [0, size = 0];

    for index in 0..Length(values) -1{
        if values[index] == results[index] {
            set indicesToUse += [index];
                    }
        else {
            set differences += 1;
        }
    }

    return (indicesToUse, (IntAsDouble(differences)/↩
        IntAsDouble(Length(values))) * 100.0);
}

operation EncodeBits(qubits: Qubit[]) : (Bool[], Bool[]) {
    mutable values = [false, size = 0];
    mutable bases = [false, size = 0];

    for qubit in qubits {
        let index = 0;
        // Select a random bit and bases with 50% ↩
            probabilbity.
        let randomBasis = DrawRandomBool(0.5);
        let randomBit = DrawRandomBool(0.5);

        // If basis = 1 apply the Hardamard transformation↩
             to the qubit.
        if randomBasis == true {
            H(qubit);
        }

        // If bit = 1 apply the Pauli X gate to the qubit.
        if randomBit == true {
            X(qubit);
        }

        set bases += [randomBasis];
        set values += [randomBit];
    }

    return (bases, values);
}

operation GetResults(qubits: Qubit[]) : (Bool[], Bool[]) {
    mutable results = [false, size = 0];
```

```
    mutable bases = [false, size = 0];

    for qubit in qubits {
        let index = 0;
        // Select a random bases with 50% probabilbity.
        let randomBasis = DrawRandomBool(0.5);
        set bases += [randomBasis];

        // If Basis = 1 measure with the Pauli X else ↩
            measure Pauli Z
        let result = Measure([randomBasis ? PauliX | ↩
            PauliZ], [qubit]);
        set results += [ResultAsBool(result)];
        Reset(qubit);
    }

    return (bases, results);
}

operation GetSharedResults(values: Bool[], results: Bool↩
    [], bases1: Bool[], bases2: Bool[]) : (Bool[], Bool[]) ↩
    {
    mutable sharedResults = [false, size = 0];
    mutable sharedValues = [false, size = 0];

    for index in 0..Length(values) -1 {
        if bases1[index] == bases2[index] {
            set sharedValues += [values[index]];
            set sharedResults += [results[index]];
        }
    }

    return (sharedValues, sharedResults);
}

operation Eavesdrop(qubits: Qubit[], eavesdropProbability:↩
    Double) : (Bool[], Bool[]) {
    mutable results = [false, size = 0];
    mutable bases = [false, size = 0];

    for qubit in qubits {
        let shouldEavesdrop = DrawRandomBool(↩
            eavesdropProbability);

        if shouldEavesdrop == true {
            // Select a random bit and bases with 50% ↩
                probabilbity.
            let randomBasis = DrawRandomBool(0.5);

            // If Basis = 1 measure with the Pauli X else ↩
                measure Pauli Z
```

```
                let result = Measure([randomBasis ? PauliX | ↩
                    PauliZ], [qubit]);
                set results += [ResultAsBool(result)];
            }
                }

        return (bases, results);
    }
}
```

4 Evaluation

Listing 2 and 3 below show the execution of the process with a 6 bit key. The driver code written in C# passes in 4n bits to the BB84 protocol. This results in Qubits with a length of 24.

The BB84 protocol is first executed with Eve interfering with the Qubits. On this run a difference of 65% is calculated for the measurements that differ when Bob measures from his random bases.

Listing 2: C# Driver code executes process with a 6 bit key

```
using Cryptography.BB84;
BB84.Run(6);
```

Listing 3: C# Driver code sets the number of qubits to 4n

```
using Cryptography.BB84.Quantum;
using Microsoft.Quantum.Simulation.Simulators;

namespace Cryptography.BB84;

public class BB84
{
    public static void Run(int n)
    {
        // Use a quantum computer to get bit values.
        using QuantumSimulator simulator = new();

        // Get 4n random bits.
        int bitLength = n * 4;

        var result = RunBB84Example.Run(simulator, bitLength);

        Console.ReadKey();
    }
}
```

4.1 Execution

We execute the BB84 program twice, once with Eve interfering with the Qubits and once without Eve interfering. The results can be seen in the outputs below.

```
Running BB84...
Creating Qubits with length 24...
Alice encoding bits...
Sending bits across quantum channel...
Eve interferring with qubits...
Bob measuring qubits...
Getting shared basis results...
Getting percentage difference in results...
Difference 64.70588235294117%
Getting shared key...
Key length is 6
```

As you can see from the above execution, when there was an eavesdropper in the communication channel the calculated difference between Alice and Bob's measurements was 65%. This is an expected result because if there was no eavesdropper then the expected result would be no more than a 50% difference between Alice and Bob's measurements. The fact that Eve changed the quantum state of the Qubits means that there will be a statistically higher difference percentage than usual. Alice and Bob know there was a eavesdropper in the communication channel, they discard the bits that have been interfered with and try sending the bits again. They do this until they have enough bits to use for their encryption key.

The second execution is then run without an eavesdropper.

```
Running BB84...
Creating Qubits with length 24...
Alice encoding bits...
Sending bits across quantum channel...
Bob measuring qubits...
Getting shared basis results...
Getting percentage difference in results...
Difference 25%
Getting shared key...
Key length is 6
```

The result above shows the process being executed a second time but this time without Eve interfering with the Qubits. As you can see this results in a percentage difference of 25%.

4.2 Results

When Eve interfered with the Qubits the result of the difference measurement was 65%. When no eavesdropping occurred the differences percentage dropped to only 25%. This shows that the amount of difference in bases comparisons increases when there is an eavesdropper interfering with the qubits. A percentage threshold can be set to determine when Alice and Bob should retry their communication based on a set percentage of difference of measurements.

5 Conclusions

I have outlined the problems with cryptography that relies on hard to compute mathematical properties and the issues this creates for current asymmetric public key exchange

mechanisms such as RSA and elliptic curves. The current state of post quantum cryptography includes new methods of encryption such as lattice based cryptography and new public key exchange methods such as quantum key distribution. I showed that a protocol called BB84 can be used to share bits that can be later used to encrypt messages and which crucially doesn't rely on a key that is hard to crack computationally. BB84 relies on the properties of quantum mechanics that allow two actors to know if their communication channel has been compromised. I showed an example of the BB84 protocol using a 6 bit key, due to the nature of the protocol this means that we would need a quantum computer that can support 24 qubits. Scaling this up to larger keys would means a larger amount of qubits. A 128 bit key for example would need 512 qubits and as of November 2022 the largest quantum computer produced by IBM has 433 qubits [22], falling short of the required amount for even a 128 bit key. For generating encryption keys it seems that lattice based cryptography methods are showing the most promise as protection against the computation power of quantum computers and this is most likely the area where the most benefits will be shown in the early stages of a post quantum world. In terms of communications channels then QKD methods will likely be adopted with more advanced methods like using Muons or other sources of quantum randomness to generate secret keys being used when methods become more refined and hardware is able to support the methods.

References

[Deu85] D Deutsch. "Quantum theory, the Church–Turing principle and the universal quantum computer". In: *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences* 400.1818 (Apr. 1985), pp. 97–117. ISSN: 00804630. DOI: 10.1098/RSPA.1985.0070. URL: https://royalsocietypublishing.org/doi/10.1098/rspa.1985.0070.

[Mon94] Peter L Montgomery. "A Survey of Modern Integer Factorization Algorithms". In: 7.4 (1994), pp. 337–365.

[Sho94] Peter W. Shor. "Algorithms for quantum computation: Discrete logarithms and factoring". In: *Proceedings - Annual IEEE Symposium on Foundations of Computer Science, FOCS* (1994), pp. 124–134. ISSN: 02725428. DOI: 10.1109/SFCS.1994.365700.

[MR08] Daniele Micciancio and Oded Regev. "Lattice-based Cryptography *". In: (2008).

[Gus+09] Julia Guskind et al. "Controlling passively quenched single photon detectors by bright light Circular Semi-Quantum Secret Sharing Using Single Particles This content was downloaded from IP address New Journal of Physics Controlling passively quenched single photon detectors by bright light". In: *New Journal of Physics* 11.18pp (2009), p. 65003. DOI: 10.1088/1367-2630/11/6/065003. URL: http://www.idquantique.com/;.

[And20] Ross Anderson. "Security engineering: a guide to building dependable distributed systems". In: 2020, pp. 170–170.

[BB20] Charles H. Bennett and Gilles Brassard. "Quantum cryptography: Public key distribution and coin tossing". In: *Theoretical Computer Science* 560.P1 (Mar. 2020), pp. 7–11. DOI: 10.1016/j.tcs.2014.05.025. URL: http://arxiv.org/abs/2003.06557%20http://dx.doi.org/10.1016/j.tcs.2014.05.025.

[Bel+21] Davide Bellizia et al. "Post-Quantum Cryptography: Challenges and Opportunities for Robust and Secure HW Design". In: *Proceedings - IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT* 2021-October (2021). ISSN: 2765933X. DOI: 10.1109/DFT52944.2021.9568301.

[KG21] Ajay Kumar and Sunita Garhwal. "State-of-the-Art Survey of Quantum Cryptography". In: *Archives of Computational Methods in Engineering 2021 28:5* 28.5 (Apr. 2021), pp. 3831–3868. ISSN: 1886-1784. DOI: 10.1007/S11831-021-09561-2. URL: https://link.springer.com/article/10.1007/s11831-021-09561-2.

[22] *IBM unveils world's largest quantum computer at 433 qubits — New Scientist.* 2022. URL: https://www.newscientist.com/article/2346074-ibm-unveils-worlds-largest-quantum-computer-at-433-qubits/.

[CCE23] Edwin Cartlidge, Cartlidge, and Edwin. "Muons used for cryptography system". In: *PhyW* 36.3 (Mar. 2023), pp. 5–5. ISSN: 0953-8585. DOI: 10.1088/2058-7058/36/03/05. URL: https://ui.adsabs.harvard.edu/abs/2023PhyW...36....5C/abstract.

[Dal23] Lee Dale. *leedale1981/msc-applied-crypto-coursework: Coursework repository for MSc Applied Cryptography document and code.* 2023. URL: https://github.com/leedale1981/msc-applied-crypto-coursework.

6 Declaration

I declare that all work in this submission is my own work.