Analysis of Post Quantum Cryptography and Comparison of Quantum Key Distribution mechanisms

Lee Dale

March 26, 2023

Abstract

Your abstract.

1 Introduction

At some point in the future quantum computers will have enough processing power or Qubits to successfully implement quantum algorithms that can break the encryption of our current state of the art public key exchange mechanisms. 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.

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

2 Literature Review

2.1 Overview of current key exchange mechanisms

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 three main categories are outlines below.

2.1.1 Prime Factorization

In the RSA method of cryptography, prime factorization is used as the basis for the algorithm's security. The RSA algorithm works by selecting two large prime numbers, p and q, and multiplying them together to obtain a product, $n = p * q$. This product n is then used as the modulus for the public and private keys.

The security of RSA relies on the fact that it is difficult to factorize the large composite number n back into its two prime factors, p and q. Specifically, if someone knows the value of n, it is difficult for them to find p and q without trying all possible factors, which becomes computationally infeasible for large enough values of p and q.

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, which is typically a small odd integer such as 3 or 65537.

To encrypt a message using RSA, the message is first converted into a number and then raised to the power of the public exponent e modulo n. The resulting ciphertext can only be decrypted using the private key, which involves factoring the modulus n back into its two prime factors p and q.

Thus, prime factorization plays a crucial role in the security of RSA cryptography. The larger the prime factors used in the algorithm, the more secure it is against attacks that attempt to factorize the modulus n.

2.1.2 Discrete Logs

Discrete logarithms are used in various public key cryptography algorithms, such as Diffie-Hellman key exchange, ElGamal encryption, and digital signature schemes like the Digital Signature Algorithm (DSA) and Schnorr signature.

In these algorithms, the difficulty of computing discrete logarithms in certain mathematical groups is exploited to provide security guarantees.

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

For instance, in the Diffie-Hellman key exchange protocol, two parties agree on a shared group, a generator of that group, and publicly exchange their exponentiations of the generator. The shared secret key is computed from their respective exponentiations, which are then equal to each other, but difficult for an attacker to determine without knowledge of the secret exponents.

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.

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, say Alice and Bob, first agree on a publicly known elliptic curve E over a finite field F. They also choose a base point P on the curve, which is also publicly known.

Each party, Alice and Bob, then chooses a secret integer, which is kept private. Alice's secret integer is denoted by a and Bob's secret integer is denoted by b. They use these secret integers to compute their respective public keys as follows:

Alice computes her public key A = aP on the elliptic curve E. Bob computes his public key B = bP on the elliptic curve E. Alice and Bob then exchange their public keys A and B over an insecure channel.

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

Alice computes the point Q = aB on the elliptic curve E. Bob computes the point Q' = bA on the elliptic curve E. The shared secret key is then derived from the x-coordinate of the point Q (or Q') using a one-way hash function.

The security of ECDH comes from the fact that it is difficult to compute the private keys a and b from the public keys A and B, even if the elliptic curve E and the base point P are known.

ECDH allows two parties to derive a shared secret key without revealing their private keys over an insecure channel, and its security is based on the difficulty of computing discrete logarithms on the elliptic curve.

2.2 How current mechanisms are vulnerable to quantum computers

2.3 Quantum robust methods

2.4 Quantum key distribution using entagled photons

2.5 Quantum key distribution using Muon detection

3 Implementation

Using Q or Qiskit to implement key exchange using entangled photons.

https://learn.microsoft.com/en-us/azure/quantum/tutorial-qdk-explore-entanglement?pivots=ide-azure-portal

4 Evaluation

5 Conclusions