Quantum-Key Exchange

HK Transfield

1.0 Introduction

This report documents a QKE (Quantum Key Exchange) algorithm designed in Python3. Throughout the explanation of the algorithm and testing, this report will refer to the cryptographic characters.

- Alice (as a transmitter)
- Bob (as a receiver)
- Eve (as an undetected third-party eavesdropper)

The first section details the design of the program and the steps involved in completing a QKE and symmetric encryption. The following sections demonstrate confidential MITM (man-in-the-middle) attacks in obtaining a secret message exchanged between two legit parties and deciphering its content. Finally, there is a discussion on the testing conducted, highlighting critical factors and results.

2.0 Design

The program utilizes a custom QKE package to facilitate the various operations required to execute a QKE. The `QKE.Qubit` module represents a quantum bit, taking in a measurable value and polarization. The `QKE.XOR` (Exclusive-OR) module cipher a message using some key.

The `QKE.Channel` module defines a Quantum Channel class containing operations and attributes needed to establish a stream to communicate, encode qubits to send through the stream, measure transferred qubits, and find shared keys. A Python list represents the stream where qubits can transfer and be retrieved since they can store multiple items as a single variable, utilized in storing qubits.

As the algorithm requires, both the transmitter and receiver or other parties participating in the exchange generate values used to determine the direction and polarization randomly. The `randint()` function from the `numpy.random` module[a] produces a randomized array of '1's and '0's, then converted into a list.

2.1 QKE Emulation

A final module, `Emulation`, puts Everything together in a `QKEEmulator` class. Here, the class takes a `run_type,` which determines if the program should run a QKE without interception or MITM attack, and `qubit_length,` which sets how long the qubit stream length will be.[b] These

[a] This was the simplest module to use for a pseudo-random number generator. Refer to the NumPy Python3 docs at https://numpy.org/doc/stable/reference/random/generated/numpy.random.randint.html

[b] The program gives options to run the emulation at 16, 256, or 1048 qubit stream length. As will be discussed later, the higher the qubit value, the greater the chance will be at a successful QKE (without interception).

parameters are determined by the user's input when starting the program. There are two main functions necessary to perform a successful QKE algorithm contained in the class:

1) `run_QKE()`: Performs the QKE algorithm between transmitter and receiver. Depending on the `run_type` flag, it will also introduce a malicious third party.

2) `run_symmetric_encryption()`: Uses the newly generated keys to encrypt and decrypt arbitrary messages between transmitter and receiver. If there are no keys, it will raise the appropriate `TypeError`.

Here is how a typical QKE and encryption would carry out between Alice and Bob, following the outline of the algorithm.

Step 1: Alice Prepares Qubits

Alice generates a random list of values and a random list of polarizations that only she knows about at this stage of the QKE. Alice encodes each value and polarization into a qubit that now transfers into the stream.

`Alice’s Values`	`[0, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0]`
`Alice’s Polarization`	`[1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0]`

Step 2: Bob's Measurements

Bob retrieves the qubits from the stream and generates his random list of polarizations. For Every Qubit currently in the stream, Bob uses the corresponding bit at that position in his polarization to measure it.

If the polarizations match, the `Qubit` will return that value associated with the `Qubit`; else, `Qubit.Measure()` will set the polarization to Bob's polarization and randomly choose a new value with a 50/50 random chance. Bob then stores the values returned from the measurements.

`Bob’s Polarization`	`[0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1]`
`Bob’s Measurements`	`[0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0]`

Step 3: Shared Secret Key

Bob and Alice exchange their polarization types used for the stream. Any polarization that Bob measured and matched from the polarizations Alice prepared means that anything in Bob's results will correspond with that bit in Alice's values. These matching bits generate the shared secret key while discarding all other bits.

`Alice’s Key`	`[0, 1, 0, 0, 0, 1, 0, 1]`
`Bob’s Key`	`[0, 1, 0, 0, 0, 1, 0, 1]`

Step 4: Symmetric Encryption

Once the key generates, Alice creates a new message which is encrypted with that key and ciphered with the XOR module.[c] The incoming ciphered message decrypts with Bob's key. If both keys match, the message should be the same.

Alice’s Message	11100111100111010011000010011000
Alice’s Key Repeated	01000101010001010100010101000101
XOR Cipher to be Sent	10100010110110000111010111011101
Bob’s Incoming Message	10100010110110000111010111011101
Bob’s Key Repeated	01000101010001010100010101000101
Decrypted message	11100111100111010011000010011000

3.0 Man-In-The-Middle Attacks

Generally, MITM attacks exploit the way digital data transmits between two parties. Usually, digital requests, messages, or instructions are sent, which are then acknowledged by the receiving party, who then sends information back as per the requests. However, if a third-party actor positions themselves between the transmitter and receiver, they can intercept those requests, messages, or instructions by pretending to be a legitimate party. It is a swift, complex attack and difficult to detect [1].

Using the three protagonists as an example, Alice sends a message to Bob.

Alice → **message** → Bob

With a man-in-the-middle attack, Eve can either eavesdrop, deny, intercept, or alter those messages. such as:

Alice → **message** → Eve → **message** → Bob (Eavesdrop)

Alice → **message** → Eve → ***altered* message** → Bob (Interception)

The emulation assumes that the channel in which Alice and Bob are communicating is insecure and, therefore, open for malicious actors to eavesdrop on their conversation as they exchange information. The algorithm executes as before; however, a new character, Eve, is introduced as an unknown third party.

Here are two types of attacks Eve performs.

Attack 1: Intercept and Resend

The first type of attack conducted is known as an intercept-resend attack.

Here, Eve intercepts the stream of qubits sent from Alice before they can reach Bob.

Using her own random set of polarizations, Eve tries to measure the qubits, producing her results. The qubits pass on to Bob, who also measures them [2].

[c] This applies XOR mask using some message and a key. This implementation works specifically with any message in a binary representation. The strength of the encryption is determined by the length of the key; keys shorter than the provided message are repeated to match the length of the key, which is considered less secure [3].

Eve's measurements may have changed some of the qubit's states; if Bob randomly chose polarizations to measure in the same polarizations that Alice prepared, the value associated with that polarization would return without an interception.

Since Eve has tampered with the qubits by reading the stream, those qubits now have a 50% chance of returning the desired value.

The consequence of this interception is that Bob and Alice's keys will not match, and Eve will fail to retrieve any sensitive information. The only viable way Alice and Bob could know of Eve's presence is to calculate the erroneous results to identify Eve as the culprit.

However, the consequence of not doing this is that Bob and Alice would not know Eve is also receiving their message:

Alice → **message** → Eve → **message** → Bob

Attack 2: Confidential MITM

It is also possible to assess if Eve accessed both strings of polarization types exchanged by Alice and Bob. Eve no longer measures the intercepted stream with her polarizations; instead, she waits until Alice and Bob's polarizations have been exchanged and uses them to measure the qubits.

In this instance, Eve uses Bob's polarization to obtain the same results Bob did, consequently allowing her to generate the same key as Alice and Bob. Therefore, when Alice sends Bob an encrypted message over this insecure channel, Eve intercepts it and uses her key to decrypt it and read it.

Unlike the previous attack, which saw Eve create errors with the keys by measuring the qubits with her polarizations, this type of attack is much harder to detect.

Alice and Bob would have no way of knowing that Eve is also receiving or altering the data the exchange polarizations and messages:

Alice → **message** → Eve → ***altered* message** → Bob

3.2 Protection

The successful execution of these Man-in-middle attacks can attribute to a lack of authentication[d] between Alice and Bob — they have no way of verifying that they are talking to the right person. A straightforward solution would be to use some form of a digital certificate to confirm that the transmitter is whom they say they are [4].

[d] Authentication, within Cyber Security, is concerned with having a user prove their identity when accessing some system or information, assuring they have the correct permissions to do so. This is usually with passwords, digital certificates, or some passcode etc. [5]

Another possible solution would be to secure the channel they are communicating on using a solid firewall, or Even a VPN (Virtual Private Networks)[e], as they are standard methods in defending against MITM attacks.

4.0 Testing

The most critical factors requiring testing were to ensure that Bob and Alice generated identical keys from the exchanged polarizations and measured values in a standard QKE without an interception. If keys matched, the QKE algorithm was a success. Furthermore, the testing ensures that messages encrypted with the key and sent between the two parties resulted in the same message.

Testing also revealed flaws in the design of the Qubit and XOR modules. Within the Qubit module, the `Qubit.Measure()` function generated random values as expected; there were cases where Alice and Bob (without an interception) were not producing identical keys. This bug was due to the function not correctly setting the new value and polarizations upon a mismatch.

Within the XOR class, exception handlers had to be added, as there were cases where qubits stream would by chance not generate any matching polarization, thereby not generating a key.[f] The absence of a key breaks the program due to there being `ZeroDivisionError` from the modulo operator.

When testing the MITM attacks, tests asserted that if Eve were to intercept the qubit stream and attempt to measure them using her polarizations, the resultant keys generated by Alice and Bob would not match, leading to messages incorrectly encrypted. Furthermore, when having access to the exchanged polarizations, tests checked to see if Eve's key and deciphered message would produce the same results as Alice and Bob.

5.0 Evaluation

Overall, while the QKE algorithm works as expected. The greater the qubit stream length, the higher the chance of generating some shared key was, leading to a correctly executed symmetric encryption. This program also highlights the flaws found in QKE, as testing conducted under the assumption that the channel in which Alice and Bob are communicating over is insecure, leading to MITM attacks from Eve.

Therefore, the implementation of some authentication would be the next step in creating a secure QKE.

[e] VPNs are used to encrypt connections between you and the network. Therefore, even if the attacker retrieves what is shared, they will have difficultly deciphering traffic on the VPN [6].

[f] Only low qubit lengths of `qubit_length <= 16` was susceptible to producing no keys, hence the forementioned `ZeroDivisionError` exceptions occurring. This issue did not happen with higher length values.

References

[1] *Man in the Middle (MITM) Attack*. (n.d.). Retrieved from VERACODE: https://www.veracode.com/security/man-middle-attack

[2] G. Mogos, "Intercept-Resend Attack on Quantum Key Distribution Protocols with Two, Three and Four-State Systems: Comparative Analysis," 2015 2nd International Conference on Information Science and Security (ICISS), 2015, pp. 1-4, doi: 10.1109/ICISSEC.2015.7371010.

[3] *XOR Encryption Algorithm*. (2020). Retrieved from 101 Computing: https://www.101computing.net/xor-encryption-algorithm/

[4] J. Huang, Y. Wang, H. Wang, Z. Li and J. Huang, "Man-in-the-middle attack on BB84 protocol and its defence," 2009 2nd IEEE International Conference on Computer Science and Information Technology, 2009, pp. 438-439, doi: 10.1109/ICCSIT.2009.5234678.

[5] *The Basics of Authentication in Cyber Security*. (2021). Retrieved from Sangfor: https://www.sangfor.com/en/info-center/blog-center/cyber-security/the-basics-of-authentication-in-cyber-security

[6] *Man in the Middle (MITM) Attacks, MITM Techniques, Detection, and Best Practices for* Prevention. (n.d.). Retrieved from RAPID: https://www.rapid7.com/fundamentals/man-in-the-middle-attacks/