

EE597: Hardware Security

Project Update

Quantum Hash Function

1. Problem Statement

In the rapidly evolving landscape of digital security, the advent of quantum computing presents both a significant challenge and an opportunity for cryptographic technologies. Traditional cryptographic methods, which form the backbone of data security and integrity in current digital systems, are increasingly vulnerable to the computational capabilities of quantum computers. As technology advances, especially with quantum computing, we're looking at new ways to keep our digital information safe. Quantum hash functions can be one such promising tools that can help with a lot of things like keeping passwords safe, making sure digital signatures are secure, and protecting all kinds of data from being tampered with. This project focuses on addressing the critical need for developing and implementing quantum hash functions.

2. Motivation

The advancements in quantum computing presents both a challenge and an opportunity for innovation.

a) Quantum computing's threat to existing cryptography: The exceptional processing power of quantum computers poses a significant risk to current cryptographic protocols. For instance, Shor's algorithm is capable of decrypting established encryption methods, such as RSA, by efficiently breaking down large numbers. This ability compromises the security infrastructure that protects today's digital communications and information.

b) The potential of quantum algorithms for enhanced security: Facing the quantum computing challenge, there is a substantial prospect for security innovations. Quantum algorithms, leveraging phenomena such as superposition and entanglement, offer the potential to create more secure and efficient cryptographic systems. These quantum-based solutions present a promising avenue for bolstering data protection against sophisticated cyber threats.

The motivation behind exploring quantum hash functions stems from the imminent threat quantum computing poses to classical cryptographic schemes. Quantum hash functions, leveraging quantum circuits and algorithms' unique properties, offer a promising solution. These functions can enhance data integrity verification and collision resistance, addressing vulnerabilities exposed by quantum computing advancements.

3. Proposed Approach

Our approach to developing a quantum hash function centers around the use of Parameterized Quantum Circuits (PQCs). These circuits encode input data using rotation angles to generate a hash string or value, effectively utilizing the high-dimensional Hilbert spaces provided by quantum computing. This method is distinguished by its potential to uniformly address the Hilbert space, thereby improving the distribution and quality of hash values, which are critical for the performance of the quantum hash function.

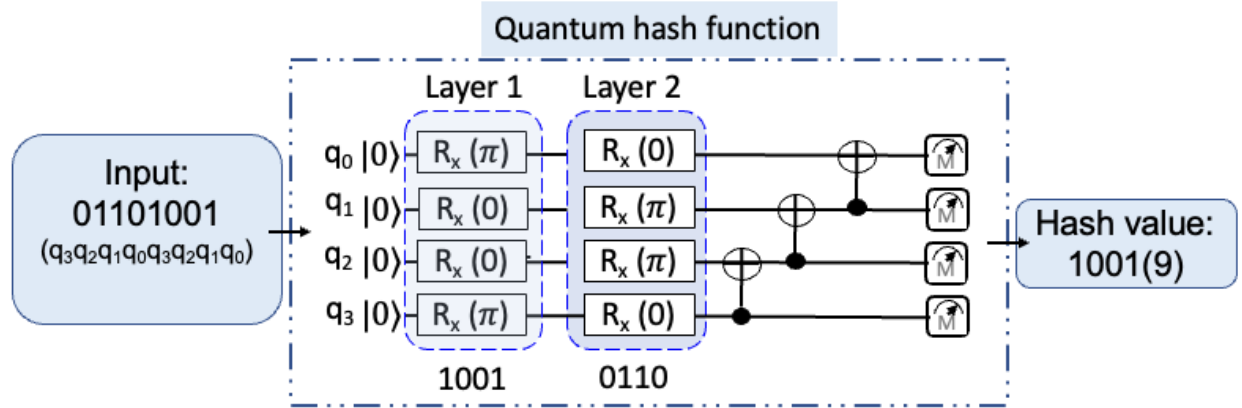


Fig1. Generating a 4-bit numerical hash value from an 8bit input bit string using PQC.

A. Basic Idea

The foundation of our proposed quantum hash function is the innovative use of PQCs. By encoding input data through carefully chosen rotation angles, we can generate hash values that are both unique and secure (Fig 1). This approach ensures that the quantum circuit efficiently explores the Hilbert space, enhancing the hash function's accuracy and reliability.

B. Implementation and Novelty

We plan to employ well-established PQCs, which have been extensively researched within the quantum computing field, to guarantee the robustness of our approach. Novel feature of our proposed method is the assignment of different angles to input bits based on a weighting scheme that considers the probability distribution of input bit values. This approach offers a more refined control over the distribution of output hash values, making it particularly effective for data with non-uniform bit distributions. By weighting the rotation angles, our method can improve the performance of the quantum hash function in terms of collision resistance and cluster prevention. Additionally, we recognize the challenge posed by noise in quantum operations. To combat this, we will focus on selecting quantum circuits and encoding methodologies that are particularly effective at mitigating noise effects, ensuring the deterministic nature of our quantum hash function remains intact.

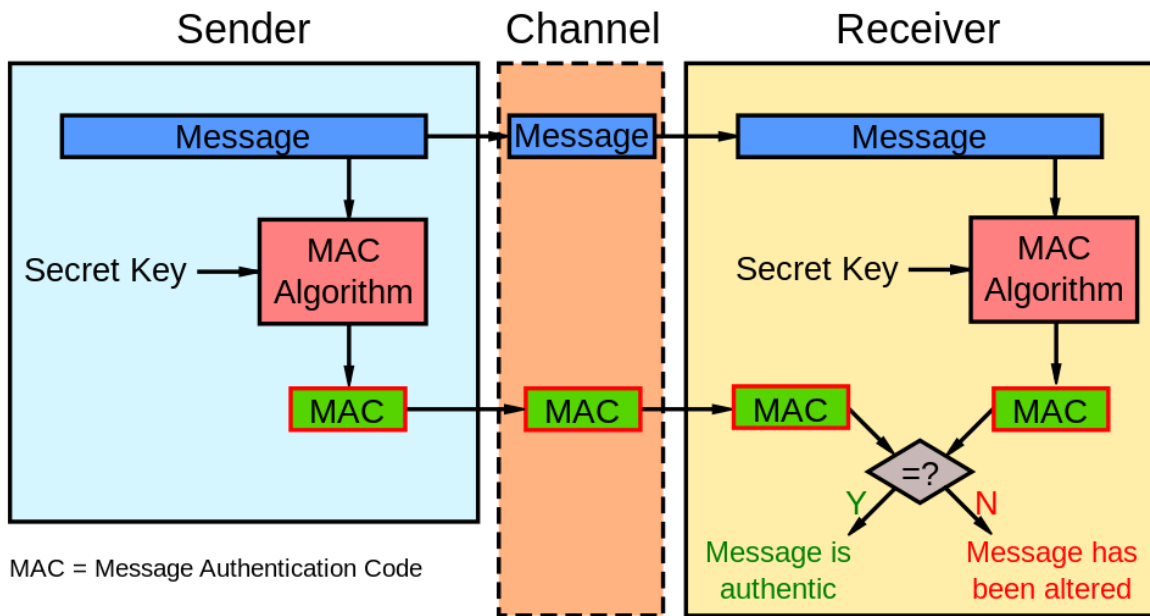


Fig2. Generating MAC based on message and a key.

C. Use Case

Problem Statement: As we know, communication over the network enables rapid development and ease of transferring information, and sometimes misinformation. Typically, when we want to send some information to the other end of our connection, we need to ensure that the data is Confidential, and that its Integrity is maintained. The former is done through encryption and the latter is done through a

form of hashing called, Message Authentication Code (MAC, not to be confused with MAC Address of network interfaces). As we see in Fig2, MAC is done based on the plaintext that is sent. MAC also uses a secret key to XOR to the message for more security. However, we will not be using a secret key just for simplicity and to focus on the quantum hashing. We use the fact that, only the sender and the receiver have the keys for encryption, and if the adversary tries to completely change the data and generate a fresh valid MAC for it, they will not be able to encrypt the data correctly as they don't possess the key, and hence at the receiver end there will be a garbage decrypted data which will be discarded.

Basic Idea: when data is in transmission, the adversary cannot extract the plaintext, and hence cannot modify the data. If they try to modify, they must also adjust the MAC accordingly, which again is impossible since the data is encrypted. If the data is messed with, the receiver can find out that the decrypted data and the MAC appended to it don't match. We use PQC to generate a Quantum MAC. Quantum advancements emphasize the need to integrate quantum it into our solutions, both, to deter quantum attacks against classical hashing and cryptography, and to leverage the powerhouse that is quantum computing.

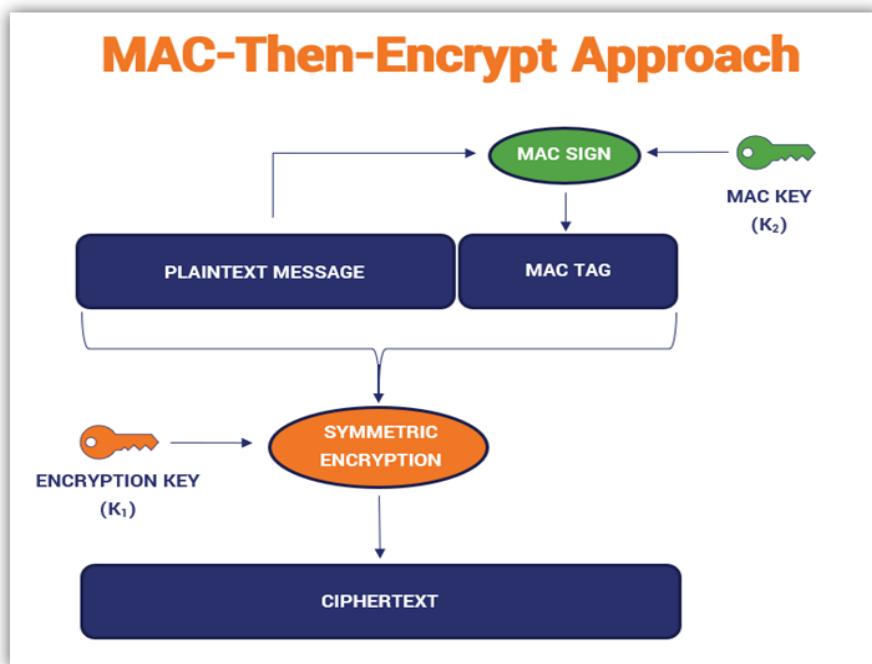


Fig3. MAC-Then-Encrypt.

Methods: This project starts with a simple data sender, receiver and a Man-in-the-Middle (MitM) running. Without the MitM, the sender prepares a data packet by generating a MAC based on quantum hashing (QMAC), then encrypts the data (this can be based on quantum encryption), combines the encrypted data and the QMAC, and finally transmits it to the receiver. In fact, there are different ways one can go about generating the data to send, Fig3 shows a MAC-Then-Encrypt approach, but there can be an Encrypt-Then-MAC approach as well. We follow the former, and we don't encrypt the MAC. The only thing that would change is the order of encryption/decryption and the combining/separating of data. The receiver extracts the data and the QMAC, unencrypts the data and runs the QMAC algorithm on the unencrypted data to check if this matches the sender's QMAC. If they do, then they accept that data. This means that the packet likely wasn't corrupted or modified in between.

Adversarial objective: objective of the MitM is to eavesdrop on all data that the sender sends and modify it before sending it to the receiver.

But unfortunately for them, this data is not only encrypted but also contains a QMAC appended to it. So even though they can sniff the data, any modification to it will be detected at the receiver. Fig4, shows the idea. Eve cannot figure out the key, and hence the generated QMAC is wrong, or the encryption is done with a wrong key, both causing problems at the receiver end.

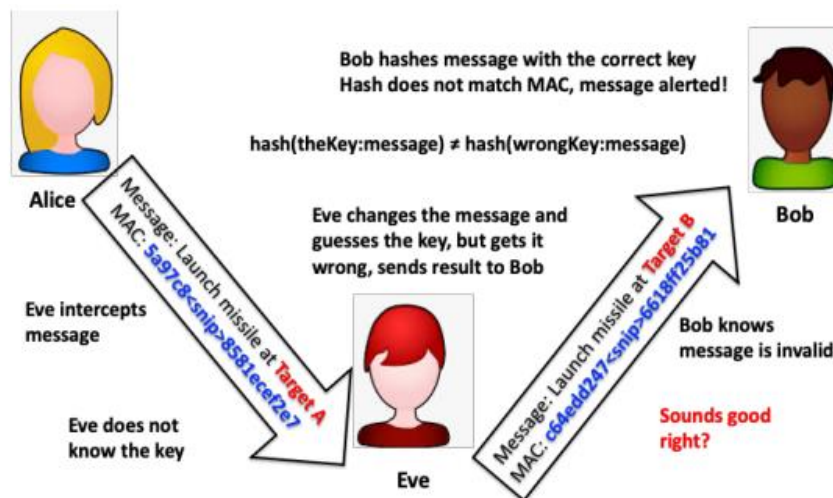


Fig4. MitM cannot correctly produce a correct MAC.

4. Implementation

a) Quantum hash function implementation

Preliminary Results: Proposed Quantum Hashing Algorithm

Implementation Basis: Utilizes established Parameterized Quantum Circuits (PQCs) recognized in quantum computing research. For our case we are using the following PQC:

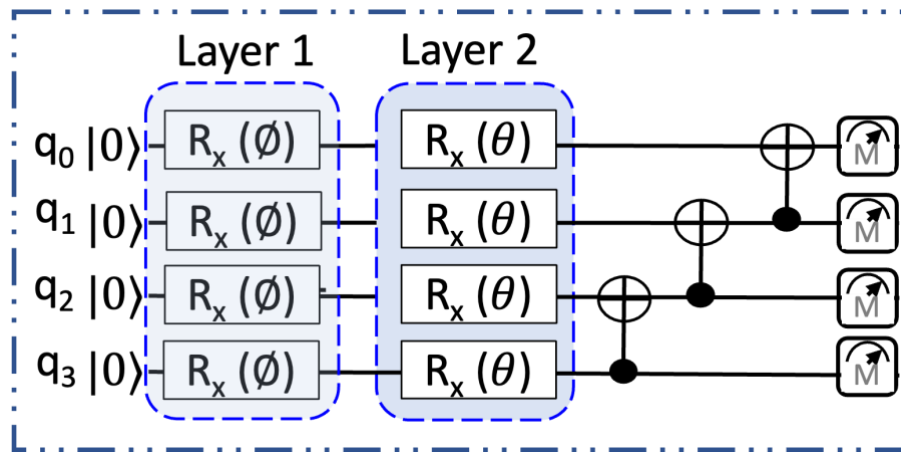


Fig 5. PQC used for Quantum hash function implementation

Data Encoding: Transforms any input data into bit strings; bits determine the rotation angles in the circuit's gates (θ_1 for '1', ϕ_1 for '0').

Encoding Process:

1. Convert input to binary bit string.
2. Loop through each bit in the binary bit string:
 - For the first $n/2$ bits:
 - If bit = '1', set Rx gate on qubit n (Layer 1) to θ_1 .
 - If bit = '0', set Rx gate on qubit n (Layer 1) to 0 (or ϕ_1 if needed).
 - For the last $n/2$ bits:
 - If bit = '1', set Rx gate on qubit n (Layer 2) to θ_2 .
 - If bit = '0', set Rx gate on qubit n (Layer 2) to 0 (or ϕ_2 if needed).
3. Output hash values of the input.

Performance Metric:

Collision Rate (CR):

- Defined as $CR = (favg + stdev) / 2^{\text{qubits}}$.
- Measures the frequency of collisions within the hash function for a given input dataset.
- favg represents the average frequency of hash values; stdev is the standard deviation.
- 2^{qubits} is the total number of distinct hash values that can be generated by an n-qubit quantum hash circuit.
- A lower CR indicates a more effective quantum hash function for the dataset in question.

Preliminary Results:

1) Hash Values:

Validation of the PQC as a quantum hash function on A 4-qubit PQC circuit variant processes a limited 100-input batch of 8-bit bitstrings (0 to 99) with 1000 shots each as shown in Fig6.

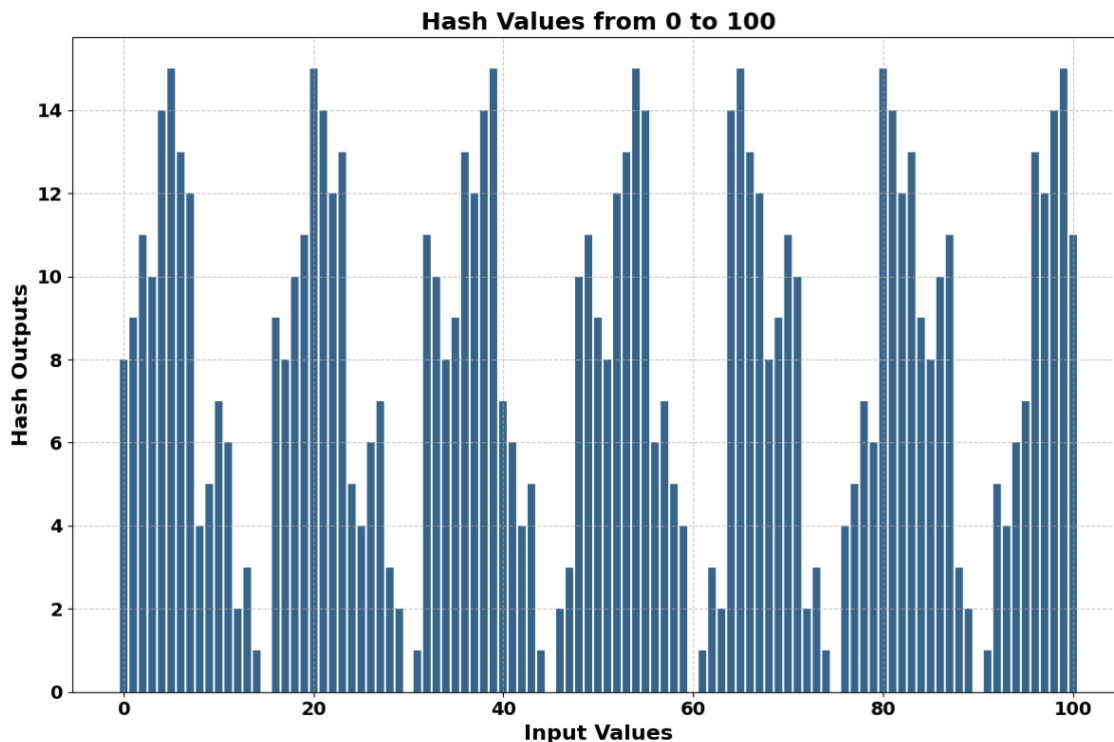


Fig 6. HASH values for different inputs

2) Batch size and Collision rates:

As the batch size increases, the collision rate tends to rise proportionally. When batch sizes are expanded, more data inputs are processed simultaneously. This increased input volume introduces a higher probability of two or more inputs coincidentally generating identical hash values, thus elevating the collision rate.

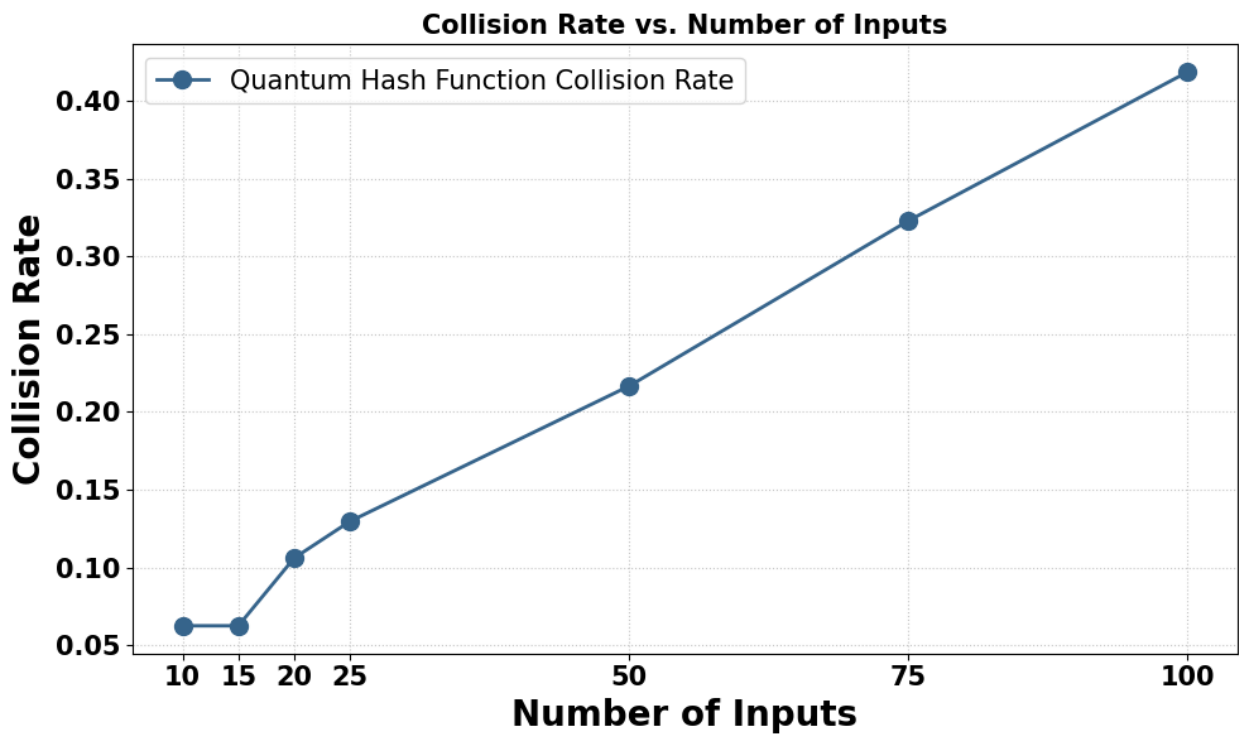


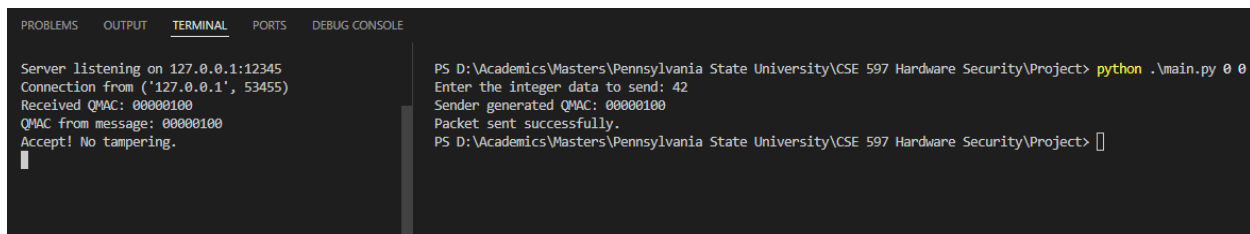
Fig 7. Collision Rate

b) Quantum MAC

We now look at the executions of running the simulation without MitM and with MitM. The code was written in python and uses sockets to send data on the same host, but different ports (12345 on the receiver, 12346 on the MitM). The sender connects to the appropriate port depending on whether the MitM is active or not. Usually this is not how MitM works, but just for simulation we distinguish them by ports.

Modes: Mode 0 is sender, Mode 1 is receiver, Mode 2 is MitM. Further, for Mode 0, we distinguish presence of MitM by another command line argument, 0 indicating no MitM. Also note that the receiver is nothing but a server, the sender is a benign client and MitM is an attacker.

Case 1: *where there is no MitM* (Fig 8). The data is transmitted by the sender along with the QMAC generated for it. In this case, the integer “42” converts to an 8-bit QMAC of ‘00000100’. The receiver verifies this by checking the received QMAC against recalculation of the QMAC from the message. And since both matches, the receiver accepts it.

A screenshot of a terminal window with a dark background and light text. The terminal is divided into two panes. The left pane shows the server's perspective: 'Server listening on 127.0.0.1:12345', 'Connection from ('127.0.0.1', 53455)', 'Received QMAC: 00000100', 'QMAC from message: 00000100', and 'Accept! No tampering.' The right pane shows the client's perspective: 'PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 0 0', 'Enter the integer data to send: 42', 'Sender generated QMAC: 00000100', 'Packet sent successfully.', and 'PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project>'.

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

Server listening on 127.0.0.1:12345
Connection from ('127.0.0.1', 53455)
Received QMAC: 00000100
QMAC from message: 00000100
Accept! No tampering.

PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 0 0
Enter the integer data to send: 42
Sender generated QMAC: 00000100
Packet sent successfully.
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project>
```

Fig 8. Sender transmits data without tampering.

Case 2: *the MitM makes a connection to the server, and the server doesn't know better, thinking it's a benign client* (Fig 9). The client, due to some misconfiguration or vulnerability connects to the MitM instead of the actual server. The client tries sending “16” and “15” with QMAC values of ‘00001001’ and ‘00000000’ respectively. However, both times, the attacker can't append the correct QMAC for the tampered data because they aren't aware of the secret key shared between the client and server. The server detects both these attempts and discards the data packets.

```
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 1
Server listening on 127.0.0.1:12345
Connection from ('127.0.0.1', 59946)
Received QMAC: 00001001
QMAC from message: 00000101
Discard! Data has been tampered.
Connection from ('127.0.0.1', 59955)
Received QMAC: 00000000
QMAC from message: 00000101
Discard! Data has been tampered.
Connection from ('127.0.0.1', 59960)
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 2
MITH listening on 127.0.0.1:12346
Connection from ('127.0.0.1', 59954)
Connection from ('127.0.0.1', 59959)
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 0 1
Enter the integer data to send: 16
Sender generated QMAC: 00001001
Packet sent successfully.
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project> python .\main.py 0 1
Enter the integer data to send: 15
Sender generated QMAC: 00000000
Packet sent successfully.
PS D:\Academics\Masters\Pennsylvania State University\CSE 597 Hardware Security\Project>
```

Fig 6. Sender transmits data with tampering.

4. Work Division

The project will be undertaken by Suryansh and Rahul. Most of the work will be carried out in the form of a collaborative effort from both the students, including discussions, design, and programming. However, roughly -

Suryansh will focus on the theoretical groundwork, specifically the development and fine-tuning of the weighting scheme for input bit angles. Rahul will take on the practical implementation aspect, which includes programming the PQCs and conducting experiments to evaluate the quantum hash function's effectiveness against noise and benchmarking the hash functions.