# 6. Merkle-Based Post-Quantum OT: Mathematical Framework

## 6.1 Key Definitions

Security Parameters:

n: Number of secrets held by the Sender.

k: Security parameter (e.g., the bit-length of hashes and salts).

Merkle Tree:

Leaves: L = { Hk(m1), Hk(m2), ..., Hk(mn) }, where Hk(x) is a quantum-safe hash function (e.g., SHA-3) and mi are the secrets.

Tree Construction: Internal nodes are defined as Hk(Li || Lj), where Li, Lj are sibling nodes, and || denotes concatenation.

Root: The root R is the top-most hash of the tree, uniquely identifying the set of secrets.

Merkle Proof:

A proof Pi for leaf i is a sequence of hashes that reconstructs R when combined with Hk(mi).

## 6.2 Protocol Phases

### Phase 1: Setup by Sender

Input:

M = { m1, m2, ..., mn }: The secrets.

Hk: A quantum-safe hash function.

Output:

R: The root of the Merkle tree.

Pi: The Merkle proof for each i in {1, 2, ..., n}.

Procedure:

1. Compute Hk(mi) for each secret mi.

2. Construct a Merkle tree from { Hk(m1), ..., Hk(mn) }.

3. Share R with the Receiver.

### Phase 2: Oblivious Transfer

Receiver’s Selection:

i: The index of the desired secret.

Generate a query mask Q such that Q(i) is indistinguishable from random noise.

Sender’s Response:

Compute Pi: The Merkle proof for the i-th leaf.

Return { Hk(mi), Pi }.

Verification by Receiver:

Verify the Merkle proof: R = Reconstruct(Hk(mi), Pi).

If valid, extract mi using additional context (e.g., decoding the hash).

## 6.3 Properties

Correctness:

If the Receiver follows the protocol, they will retrieve mi such that:

Hk(mi) is valid under Pi → R.

Privacy:

The Sender cannot determine i due to the masking of the query Q(i).

Quantum Resistance:

The security relies on the hardness of inverting Hk, a quantum-safe hash function.

## 6.4 Optional Extensions

Add Salting for Hk(mi):

Hk(mi || si), where si is a unique salt per secret to prevent replay attacks.

Use Adaptive Masking for Q(i):

Ensuring it resists statistical attacks.