Implementation Study — Merkle Trees for Large-Scale Integrity Verification in Review Systems

Project Title

Integrity Verification and Existence Proof in Large-Scale Review Systems Using Merkle Trees

Overview

This project is designed as an applied case study to develop, analyze, and evaluate a **Merkle Tree–based cryptographic verification system** for large-scale product review datasets. Using Amazon's publicly available review datasets—which span categories such as Books, Electronics, Home & Kitchen, Clothing, and Toys—students will implement a complete pipeline for preprocessing data, constructing Merkle Trees, generating root hashes, verifying record existence, and detecting tampering.

The project focuses on both **algorithmic correctness** and **scalability**, requiring the system to handle a minimum of **1 million review records** and to provide verification results in **under 100 milliseconds**. Students will also conduct performance tests, analyze hashing costs, evaluate memory usage, and report findings.

Learning Objectives

• Implement a Merkle Tree from scratch for large-scale datasets.
• Understand and analyze cryptographic hashing, inclusion proofs, and integrity verification.
• Design a scalable system capable of handling ≥1M Amazon review records.
• Generate existence proofs for individual review IDs or product IDs.
• Detect data tampering through Merkle Root comparison.
• Evaluate performance metrics such as hash computation time, proof latency, and memory usage.
• Produce analytical insights comparing Merkle-based verification to other integrity-checking approaches.

Case Study Background

Online platforms such as **Amazon** store hundreds of millions of user-generated product reviews across diverse categories. These reviews influence purchasing decisions and hold significant commercial value. Ensuring their authenticity is crucial to prevent manipulation, fake reviews, or unauthorized modifications.

A data security team is developing a system that guarantees **tamper-evident storage** of large-scale review datasets using **Merkle Trees**. Since Merkle Trees allow efficient hashing, proof generation, and quick detection of altered records, they are widely used in blockchain technologies and distributed databases.

Students are tasked to:

1. Load and preprocess at least **1 million Amazon review records**.
2. Construct a complete Merkle Tree using SHA-256 hashing.
3. Generate the Merkle Root representing dataset integrity.
4. Provide inclusion proofs for review IDs or product IDs.
5. Detect tampering, modification, deletion, or insertion of records.
6. Measure verification time and memory consumption.
7. Produce a detailed performance and analysis report.

Dataset Description

Dataset Source: Amazon Product Data (Jianmo Ni)

https://nijianmo.github.io/amazon/index.html

The Amazon Product Review Dataset is a large-scale collection of user-written reviews and product metadata. Data is divided into **multiple category-specific subsets**, including:

- Books
- Electronics
- Home & Kitchen
- Clothing, Shoes & Jewelry
- Toys & Games
- Beauty and Personal Care

Each review includes fields such as:

rating, title, text, images, asin (product ID), parent_asin,

user_id, timestamp, helpful_vote, verified_purchase

Metadata files supplement product-level details such as:

main_category, features, description, price,

average_rating, rating_count, brand, images, videos

Minimum Requirement:
Students must use a dataset that contains **at least 1,000,000 review records** (from one category or combined categories).

Implementation Requirements

Core Modules

The system must include the following components:

1. Data Preprocessing Module

- Load JSON review files (≥1 million records).
- Clean, parse, normalize, and extract relevant fields.
- Generate unique IDs if missing.

2. Merkle Tree Construction Module

- Implement Merkle Tree insertion from scratch.
- Hash each review using SHA-256.
- Build all parent nodes and compute the final Merkle Root.

3. Integrity Verification Module

- Store the root hash.
- Compare roots to detect tampering or dataset updates.

4. Existence Proof Module

- Return Merkle proof paths for any review ID or product ID.
- Verify proof within **≤100 ms**.

5. Tamper Detection Module

- Detect modification, deletion, or injection of records.
- Report mismatched hashes or altered nodes.

6. Performance Measurement Module

Measure:
- Hashing speed
- Proof generation latency

• Memory usage
• Effect of dataset size

Interface Requirements

CLI Interface (Mandatory)

• Display dataset in tabular form with unique review IDs.
• Menu options for:

- Building the Merkle Tree
- Saving/Comparing Merkle Roots
- Generating proof paths
- Simulating tampering
- Running performance tests

GUI Interface (Optional — Bonus Marks)

• Visual tree representation
• Animated proof paths
• Interactive integrity check dashboard

Test Data

Use Amazon review subsets.
At least **1M** records must be used during evaluation.

Test Cases

Action	**Expected Result**
Load 1 million reviews from Amazon “Electronics” dataset	System constructs Merkle Tree and displays computed Merkle Root
Save generated Merkle Root	Root hash saved successfully
Query existence of Review ID = “R12345”	Returns “Verified – Exists” with proof path
Query non-existing Review ID = “R99999”	Returns “Not Found in Dataset”

Action	Expected Result
Modify one review text in dataset	System detects hash mismatch → “Data Tampering Detected”
Modify one character in a review (e.g., change “good” → “god”).	Merkle Root changes; system flags “Data Integrity Violated.”
Delete a review record	Root mismatch detected → “Data Integrity Violated”
Insert a fake record into dataset	Root mismatch detected → “Data Integrity Violated”
Compare current root vs stored root	Displays either “Integrity Verified” or “Integrity Violated”
Measure proof generation time for 100 random review IDs	Each query completes in < 0.1 seconds
Measure memory and hash computation performance	Performance metrics logged successfully
Build trees for multiple categories (Books, Electronics, Beauty) and compare roots	Each category generates independent Merkle roots; integrity verified individually
Rebuild Merkle tree after dataset update	New Merkle root computed and stored; previous root retained for historical comparison
Load the same dataset twice (without modification).	Merkle Root must remain identical both times.
Add a new review entry dynamically, rebuild the partial tree.	New Merkle Root generated; proof paths update efficiently (not full rebuild).

Metrics to Record

Metric	Description
Hash Time (avg)	SHA-256 per record
Proof Generation Time	Time to verify existence
Total Build Time	Time to construct Merkle Tree
Memory Usage	Peak memory during build

Tamper Detection Accuracy	Ability to detect modifications
Root Consistency	Root stability on unchanged data

Sample Experiments

Experiment A — Static Verification

Compute Merkle Root for a 1M-record dataset.

Experiment B — Tamper Simulation

Modify, delete, or insert entries and measure detection accuracy.

Experiment C — Proof Benchmarking

Test 1000 random existence proofs and average their latency.

Experiment D — Multi-Category Comparison

Generate roots for multiple datasets and analyze consistency and scaling patterns.

Deliverables

1. **Complete Source Code** (all modules)
2. **Test Suite** (correctness + performance)
3. **Performance Report** containing:
    - Time and memory metrics
    - Proof path analysis
    - Dataset statistics
4. **Final Analytical Report** including:
    - Implementation explanation
    - Efficiency improvements
    - Comparison with other hashing/tree methods
    - Complexity analysis
    - Results & interpretation
    - Limitations & future work

Suggested Report Outline

1. Introduction & Motivation
2. Background: Merkle Trees & Hash Functions
3. Dataset Description
4. System Architecture & Implementation
5. Experimental Setup
6. Results & Analysis
    - Time Complexity Analysis
    - Build times
    - Tamper detection
    - Proof latency
    - Memory usage
7. Discussion: Scalability & Practical Use
8. Conclusion & Recommendations