



## **CMSE353 Security of Software Systems**

### **Lab 2 Report**

#### **Team members**

**Seyit Ahmet Inci – 19331143**

**Sinem Imge Turgut - 19001316**

**Emre Çakır – 20001797**

**Jamshid Artykov – 19700076**

**Group N0: 02**

**Lab Topic: Merkle Hash Tree**

**Semester Term: 2021-2022 Fall Term**

## Outline

<b>Problem Definition</b> .....	2
<b>Teamwork Distribution and Meetings</b> .....	3
<b>Meetings</b> .....	3
<b>Word break-down</b> .....	3
<b>Data structures and algorithms used</b> .....	4
<b>Implementation Tools</b> .....	4
<b>Program function and class descriptions</b> .....	5
<b>generateHash</b> .....	5
<b>toHexString</b> .....	5
<b>generateTree</b> .....	6
<b>buildTree</b> .....	6
<b>printTree</b> .....	7
<b>main</b> .....	8
<b>Node class</b> .....	8
<b>User Guide</b> .....	9
<b>Conducted Tests</b> .....	10
<b>Conclusion</b> .....	11
<b>References</b> .....	11

## Problem Definition

A Merkle tree is a hash-based data structure that is a generalization of the hash list. It is a tree structure in which each leaf node is a hash of a block of data, and each non-leaf node is a hash of

its children. Typically, Merkle trees have a branching factor of 2, meaning that each node has up to 2 children. Merkle trees can be used to validate integrity of data, peer-to-peer network transactions (*Merkle Tree / Brilliant Math & Science Wiki*, n.d.).

## Teamwork Distribution and Meetings

### Meetings

Date	Meeting No	Participants	What was done
6.12.2021	1	Emre, Seyit, Sinem, Jamshid	Discussion about how the Merkle tree works and which language to use.
11.12.2021	2	Emre, Seyit, Sinem, Jamshid	Work break-down and distribution of the work. Finding suitable algorithms and similar programs.
14.12.2021	3	Emre, Seyit, Sinem, Jamshid	Integration of components, writing main function, testing and documentation.

### Word break-down

	Emre	Sinem	Seyit	Jamshid
<b>generateHash</b>	<b>X</b>	<b>X</b>		
<b>toHexString</b>	<b>X</b>	<b>X</b>		
<b>main</b>	<b>X</b>	<b>X</b>		
<b>generateTree</b>			<b>X</b>	<b>X</b>

<b>buildTree</b>			<b>X</b>	<b>X</b>
<b>printTree</b>			<b>X</b>	<b>X</b>
<b>Node class</b>			<b>X</b>	<b>X</b>

## Data structures and algorithms used

**SHA-256 Hashing algorithm:** SHA-256 is a cryptographic hash function. Several cryptocurrencies use this algorithm for verifying transaction and calculating proof of work. The algorithm is included inside of java built-in security library.

**Bytes to hex:** Converts hashed value to hexadecimal string.

**Binary Tree:** A binary tree is a data structure whose elements have at most 2 children. It contains pointer to the left child, pointer to the right child and data or hash of the data. If the node numbers are odd, then duplicate node added to node which does not have left or right node.

## Implementation Tools

We chose JAVA programming language for implementation of this tool. We used latest version of Eclipse IDE since it has built-in hashing algorithm and it is easy to implement binary tree.

To be able to run the source code, user must create a JAVA project in Eclipse IDE and create 2 packages named “MerkleTree” and “Node”. Afterwards user must put MerkleTree.java file under MerkleTree package and Node.java file under Node package. The both files are available in ZIP file under src folder.

## Program function and class descriptions

### generateHash

Takes an input of string and returns hash value in bytes.

```
public static byte[] generateHash(String input) throws NoSuchAlgorithmException
{
    //SHA-256 Hashing Algorithm

    MessageDigest md = MessageDigest.getInstance("SHA-256");
    return md.digest(input.getBytes(StandardCharsets.UTF_8));
}
```

### toHexString

Takes an input of byte values and returns string of hexadecimal values.

```
//Converts byte into hexadecimal value
public static String toHexString(byte[] hash)
{
    BigInteger number = new BigInteger(1, hash);
    StringBuilder hexString = new StringBuilder(number.toString(16));

    while (hexString.length() < 32)
    {
        hexString.insert(0, '0');
    }

    return hexString.toString();
}
```

## generateTree

Generates most lower nodes(leaves) with their hashes and calls buildTree function to make a binary tree.

```
public static Node generateTree(ArrayList<String> dataBlocks) throws NoSuchAlgorithmException
{
    ArrayList<Node> childNodes = new ArrayList<>();

    for (String message : dataBlocks) {
        childNodes.add(new Node(null, null, toHexString(generateHash(message))));
    }

    return buildTree(childNodes);
}
```

## buildTree

Builds a binary tree out of leaf nodes. Connects 2 child node to make a parent node, if the child nodes does not contain two nodes duplicates the last node. Continues connecting until function reaches root(children.size()==1). Returns root.

```
private static Node buildTree(ArrayList<Node> children) throws NoSuchAlgorithmException
{
    ArrayList<Node> parents = new ArrayList<>();

    while (children.size() != 1) {
        int index = 0, length = children.size();
        while (index < length) {
            Node leftChild = children.get(index);
            Node rightChild = null;

            //add duplicate if the tree nodes are odd
            if ((index + 1) < length) {
                rightChild = children.get(index + 1);
            } else {
                rightChild = new Node(null, null, leftChild.getHash());
            }

            String parentHash = toHexString(generateHash(leftChild.getHash() + rightChild.getHash()));
            parents.add(new Node(leftChild, rightChild, parentHash));
            index += 2;
        }
        children = parents;
        parents = new ArrayList<>();
    }
    return children.get(0);
}
```

## printTree

Prints hash values of each node starting from the root

```
//Print hash contents of tree starting from root to leaves
private static void printLevelOrderTraversal(Node root) {
    if (root == null) {
        return;
    }

    if ((root.getLeft() == null && root.getRight() == null)) {
        System.out.println(root.getHash());
    }
    Queue<Node> queue = new LinkedList<>();
    queue.add(root);
    queue.add(null);

    while (!queue.isEmpty()) {
        Node node = queue.poll();
        if (node != null) {
            System.out.println(node.getHash());
        } else {
            System.out.println();
            if (!queue.isEmpty()) {
                queue.add(null);
            }
        }

        if (node != null && node.getLeft() != null) {
            queue.add(node.getLeft());
        }

        if (node != null && node.getRight() != null) {
            queue.add(node.getRight());
        }
    }
}
```

## main

Prompts the user to enter choice. First choice is to take any number of inputs and store them in array. Second choice takes the input array builds the tree and displays hash values of the tree starting from the root until  $2^n$  children where  $n$  is depth (level) of the tree.

```
public static void main(String[] args) throws NoSuchElementException {
    ArrayList<String> dataBlocks = new ArrayList<>();

    Scanner sc= new Scanner(System.in);
    Scanner sc2= new Scanner(System.in);

    System.out.println("-----Welcome-----\n"
        +"Press '1' to input\n"
        +"Press '2' to build and display merkle tree\n"
        +"Press '3' to exit program\n");

    boolean t=true;
    while (t){
        System.out.println("Enter your choice:");
        int ch = sc.nextInt();

        if (ch==1){
            int i;
            System.out.println("Enter number of elements you want to enter");
            int num = sc.nextInt();

            for (i=0;i<num;i++) {
                System.out.println("Enter element " + (i+1));
                String s = sc2.nextLine();

                dataBlocks.add(s);
            }
        }
        if (ch==2){
            System.out.println("\nMerkle Hash Tree\n");
            Node root = generateTree(dataBlocks);
            printTree(root);
        }

        if (ch==3){
            t=false;
        }
    }
}
```

## Node class

A structure in java. Each node has data block or hash value and pointers to left and right children. The class contains some get and set methods.



```

public class Node {

    private Node left;
    private Node right;
    private String hash;

    public Node(Node left, Node right, String hash) {
        this.left = left;
        this.right = right;
        this.hash = hash;
    }

    public Node getLeft() {
        return left;
    }

    public void setLeft(Node left) {
        this.left = left;
    }

    public Node getRight() {
        return right;
    }

    public void setRight(Node right) {
        this.right = right;
    }



    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }
}

```

## User Guide

To directly run the program user should extract folder 'application' and run 'app.bat' file. The program will launch in a windows console (cmd). In order for program to work the folder must contain MerkleTree.jar and app.bat files as shown below.

 app	14/12/2021 10:07	Windows Batch File	1 KB
 MerkleJava	14/12/2021 10:24	JAR File	4 KB

After the launch the user will be confronted with 3 options. User must enter the array inputs before building and printing the tree hash values.

User must follow steps as follows:

1. Extract 'Application' folder from the RAR file
2. Run app.bat file
3. Press '1'
4. Enter the number of inputs
5. Enter value for each input array
6. Press '2' to build and print a hash tree
7. Press '3' to exit the program

## Conducted Tests

Let's test our program with following outputs: 'Hello', 'World', '!', 'CMSE', '353', 'Security' and the the result.

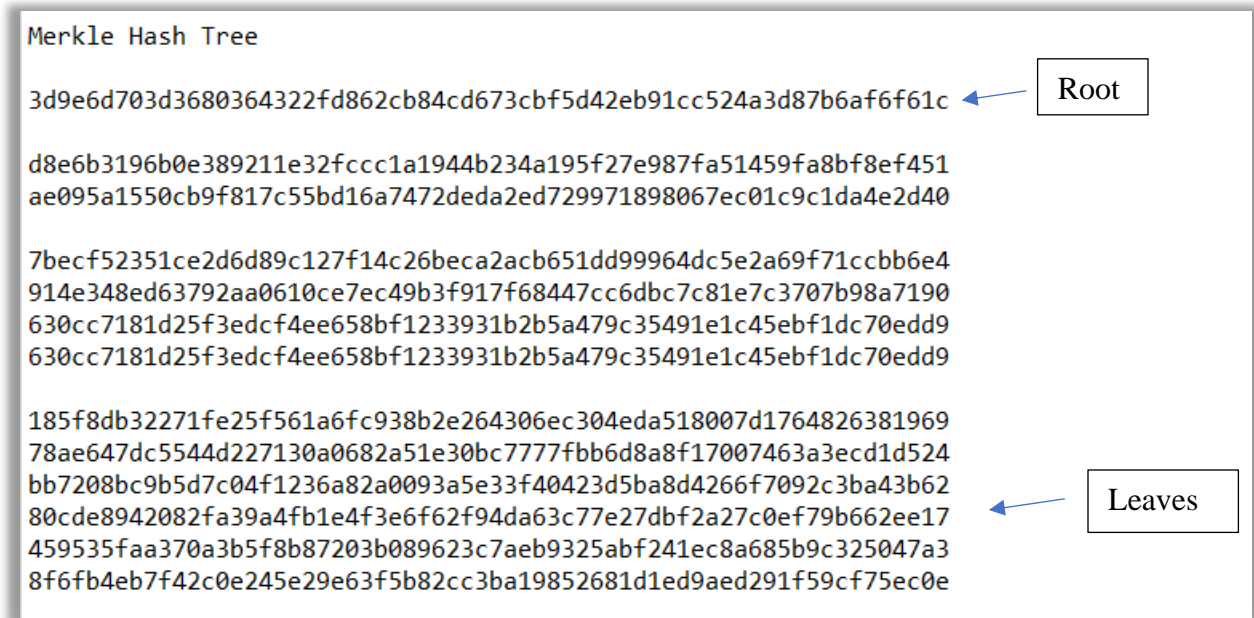
```

|-----Welcome-----
Press '1' to input
Press '2' to build and display merkle tree
Press '3' to exit program

Enter your choice:
1
Enter number of elements you want to enter
6
Enter element 1
Hello
Enter element 2
World
Enter element 3
!
Enter element 4
CMSE
Enter element 5
353
Enter element 6
Security

```

Merkle hash tree of our attempt:



As you can see 3rd and 4th hash values in 3rd level are copies of each other.

## Conclusion

To conclude, Merkle trees are very useful programs in many ways. Nowadays, demand in cryptocurrencies are very high and Merkle tree can be used as an instrument of secure transactions. It also can be used for data integrity of file sharing systems.

## References

- GeeksforGeeks. (2019, August 7). SHA-256 Hash in Java.

<https://www.geeksforgeeks.org/sha-256-hash-in-java/>

- *Merkle Tree / Brilliant Math & Science Wiki*. (n.d.). Brilliant.

<https://brilliant.org/wiki/merkle-tree/>

- Open Source Merkle Tree Implementation

<https://gist.github.com/pranaybathini/e3bb4e6ca6bc387b58e534e370033c05>