CHALLENGE 3: MERKLE TREES

30/08/2019
European Cyber Security Challenge 2019
Bucharest, Romania

1. Initial Write-Up

Description:

A Merkle tree is a tree consisting of leaves and nodes. Each non-leaf node is defined with a value or a label and contains a hash of its children. This builds a hash tree, where the root of the tree is a value distributed by a trusted-third party, used to provide a verification of large-scale data structures. Competing teams need to understand the concept of Merkle trees, calculate certain hash values, and finally compute a signature in a hash-based tree construction.

2. Challenge specifications

- Category: Crypto
- Difficulty : Medium
- Expected time to solve: 2h

3. Technical specifications

Merkle Trees

Definitions and Goal

A Merkle tree is a tree consisting of leaves and nodes. Each non-leaf node is defined with a value or a label and contains a hash of its children. This builds a hash tree, where the root of the tree is a value distributed by a trusted-third party, used to provide a verification of large-scale data structures.

By using a tool to calculate hash values, the participants will be asked to verify if two values belong in the tree.

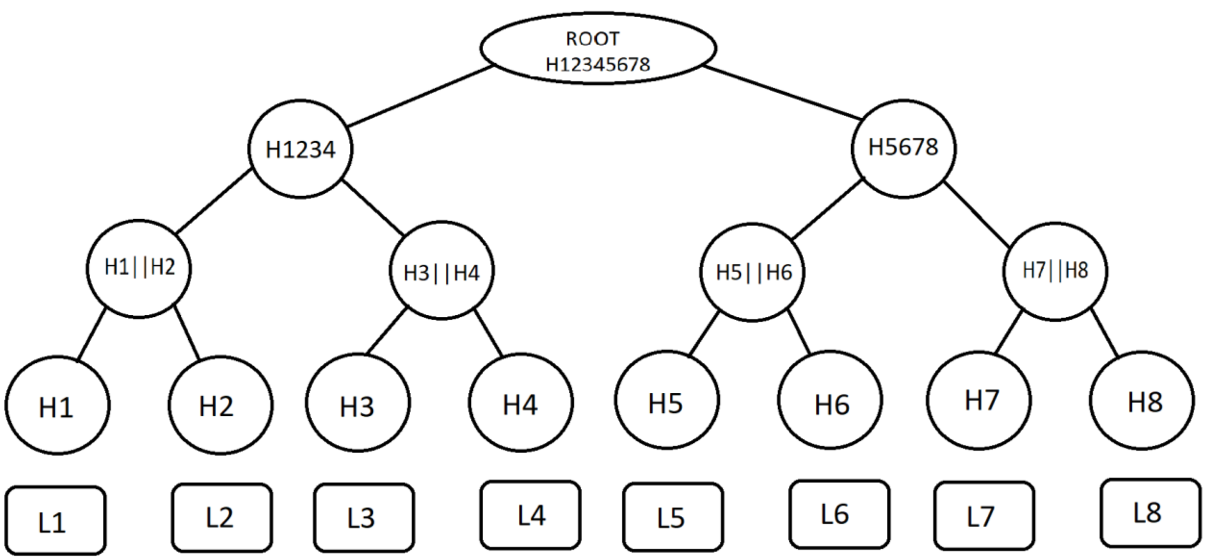
Goal of the challenge is to understand the concept of Merkle trees, their benefits compared to hash lists/hash tables and how the verification procedure works (with the knowledge of sibling nodes). A simplified extension to a signature scheme will be described and the participants will have to calculate the signature of a node.

4. Questions and answers

An online server for saving users' files, uses Merkle tree structures to store values related to specific documents. We know that this server uses the hash function SHA-256 for hashing the nodes. The files, which are represented by the nodes, are marked with a unique number, in order to make the hashing procedure easier (so we don't have to hash the whole file, but the corresponding number to the file). Given the figure below, the public root node:

R= 8b80cda3f604e6406501a427e4ee699a2b33bd30a0ee0c05e3c406cee667c570

and some intermediate values, answer the following questions:



H1 = H(L1) = 2a57042a43991d2ca310938e6802d7283954e38c825a548c4bee89c45238b43b$
H2 = H(L2) = 02dc668144d5bceea63129dbf78d853aafbadddf8c4b1e7909965e62422f2ebf
H3 = H(L3) = 023849c38925e2af028a2eb4e1dc41afd7dc7a238195c1c2ae00438d1dae00e1
H4 = H(L4) = c76b405781134be1dab7fe45adfb8c32104805a01de7b863e1004b66d56edf9f
H5 = H(L5) = 227445a988500528d7826c6921d2e3b4a79ccf3a94cc3bcf7b667e3ae4990b36
H6 = H(L6) = e52d08747b9d7a6d04551bb86ee3f7ee6c49f7477c8cd66f77448378cc30b92b

H7 = H(L7) = d4e33e2934280979f580a63f992daa7d0de2cd64a145d5c403a75c3dc5c0004e
H8 = H(L8) = b4944c6ff08dc6f43da2e9c824669b7d927dd1fa976fadc7b456881f51bf5ccc
H12 = H(1)||H(2) = 25962b8724bb0b5e64390f35bda4e9fc12a88eed2c88c39c59247521a730155a
H56 = H(5)||H(6) = 795aa46e2f616151f345ec5d0e7e72d28354d8805d9c35f057fa032b58f1600e
H78 = H(7)||H(8) = 3ed5c614e59b93f111b4ee74d5201d6f23deac0ac2f917ca73c10e30f07cf8d1
H1234 = H(12)||H(34) = 42885424e75b1c367bef442ed398c46cdb4f57e7ea93240dde279f082d9ef77b
R = H((H1234)||(H5678)) = 8b80cda3f604e6406501a427e4ee699a2b33bd30a0ee0c05e3c406cee667c570

Question 1:

Check whether the values 24 and 593 belong in the files' numbers.

Solution:

We have to check if the values H(24) and H(593) belong in the leaves of the tree, we do this using the online calculator:

H(24) = sha256(24) = c2356069e9d1e79ca924378153cfbbfb4d4416b1f99d41a2940bfdb66c5319db

H(593) = sha256(593) = d4e33e2934280979f580a63f992daa7d0de2cd64a145d5c403a75c3dc5c0004e

So 24 does not belong, but 593=L7, since H(593) = H(L7).

Question 2:

Compute the value H5678.

Solution:

First, we need to compute H56, which is
H56=H(227445a988500528d7826c6921d2e3b4a79ccf3a94cc3bcf7b667e3ae4990b36e52d08747b9d7a6d04551bb86ee3f7ee6c49f7477c8cd66f77448378cc30b92b) =
07bbc5f6ec2d52126962c5e4e9047fba8905c9d3345eccfa5734470d6033ca90 .

Then, we can compute:

H(H56 || H78) =
H(795aa46e2f616151f345ec5d0e7e72d28354d8805d9c35f057fa032b58f1600e3ed5c614e59b93f111b4ee74d5201d6f23deac0ac2f917ca73c10e30f07cf8d1) which is:

H5678 = 07bbc5f6ec2d52126962c5e4e9047fba8905c9d3345eccfa5734470d6033ca90

Question 3:

Signature Generation

Assume a simplified signature generation scheme using Merkle trees and One Time Signatures, where a signature on a message m has the following format:

Sig(m) = (H(m)|| pub_key || auth_path)

with pub_key = H_i, i={1,...,8} the leave node and auth_path is the authentication path of sibling nodes starting from the chosen public key. Generate the signature of the message {ECSC2019 challenges are fun} using H1 as public key.

Answer:

Starting with this message, we calculate the hash over it, which is {ab1cd93962ae70b2eab5a42b3ae7f2acd440b58b862cd5e31c628523168408a4}. Then we have from the table that H1 = 2a57042a43991d2ca310938e6802d7283954e38c825a548c4bee89c45238b43b. The authentication path should contain the sibling nodes: H2, H34, H5678, which are calculated in the previous sub-questions. So the signature is:

Sig = {ab1cd93962ae70b2eab5a42b3ae7f2acd440b58b862cd5e31c628523168408a4||

2a57042a43991d2ca310938e6802d7283954e38c825a548c4bee89c45238b43b||

02dc668144d5bceea63129dbf78d853aafbadddf8c4b1e7909965e62422f2ebf1f981f2057a2d336fe78ff7e0bb86581c0e52546cf39f3ebd3483f8c28afd22407bbc5f6ec2d52126962c5e4e9047fba8905c9d3345eccfa5734470d6033ca90}

5. Attack Scenario

Description:

The attacker can be able to change the content of certain files in the server (integrity) or add new files and claim that they are valid files and they there from the beginning. This cannot be done with Merkle trees and proper hash functions, such as SHA-256. A possible implementation question could be taken from the hippiehug documentation, by installing the library in python (pip install hippiehug) and asking the participants to develop a specific functionality of Merkle trees [4].

6. Installation instructions

No special installation instruction needed

7. Tools needed

1 Online SHA calculator

SHA values calculator tool	**Online SHA tool https://emn178.github.io/online-tools/sha256.html**

8. Artefacts Provided

N/A

9. Walkthrough (writeup)

N/A

10. References

1. Merkle trees https://www.codementor.io/blog/merkle-trees-5h9arzd3n8
2. Merkle Signatures for Real-World Use http://www.mit.edu/~rio/merkle.pdf
3. Merkle tree - hash tables: Check if node belong in Merkle tree https://asecuritysite.com/encryption/merkle
4. Hippiehug Documentation by George Danezis https://buildmedia.readthedocs.org/media/pdf/hippiehug/latest/hippiehug.pdf
5. Encryption Merkle: https://asecuritysite.com/encryption/merkle