UNIVERSITY of NICOSIA

Session 3

Merkle Tree Demo

MSc in Digital Currency

Merkle Tree and Transaction Verification

Step-by-step exercise

This is a step-by-step exercise designed to visualize Merkle trees, Merkle root, and the ability to prove the verity of the data held, without revealing what the actual information is.

The concept of hash trees is named after Ralph Merkle, who patented it in 1979. Merkle tree uses hashes in order to calculate a summary of all information, in a way that you can prove that something is part of the tree in a very efficient way.

We will simulate a list of four data elements with the following data elements; words in our example:

Data Elements:

- “University”
- “of”
- “Nicosia”
- “Cyprus”

https://en.wikipedia.org/wiki/Merkle_tree Ralph Merkle - Wikipedia

Merkle Tree and Transaction Verification

Step One: Hash the initial data elements

To construct a Merkle Tree, it is required that we hash the initial data elements using a hash function, to generate the ***leaf-***nodes of the Merkle tree.

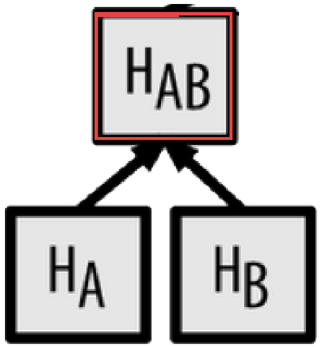
You can choose any hash function available, in this example we will be using SHA-256.

Data	SHA-256	
University	9D38443E2220DED5F60BA39B85ACDAD3AF96F810E359BB741F6D273470C7BF9B	**Hash A**
of	28391D3BC64EC15CBB090426B04AA6B7649C3CC85F11230BB0105E02D15E3624	**Hash B**
Nicosia	7BEE85A2D89FE2978D484FD57513EA6C19734A6C2F651C79D4C3B5FBB840C1FE	**Hash C**
Cyprus	5F1162C57ECF998190D473FF238B6E311D7C52F5F81CA7AC3FCA9267832B585C	**Hash D**

https://passwordsgenerator.net/sha256-hash-generator/

Merkle Tree and Transaction Verification

Step Two: Concatenate H_A+H_B to Hash(H_AH_B)



For each **Parent Node**, the resulting hashed data are subsequently hashed together in pairs to create the parent nodes of the leaf nodes. The example below shows how this is calculated:

Concatenate H_A+H_B

9D38443E2220DED5F60BA39B85ACDAD3AF96F810E359BB741F6D273470C7BF9B28391D3BC64EC15CBB090426B04AA6B7649C3CC85F11230BB0105E02D15E3624

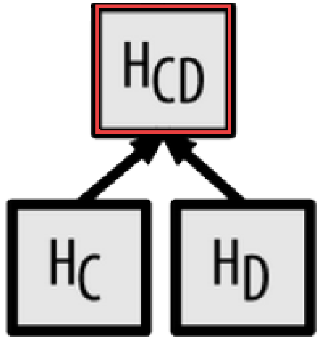
Hash(H_AH_B)

E2E4A6AFDB1332E2CB6E1634ED76BB9D3C8BFBBE19FED545E6A0347A93DA89AE

https://passwordsgenerator.net/sha256-hash-generator/

Merkle Tree and Transaction Verification

Step Three: Concatenate H_C+H_D to Hash($H_C H_D$)



For each **Parent Node**, the resulting hashed data are subsequently hashed together in pairs to create the parent nodes of the leaf nodes. The example below shows how this is calculated:

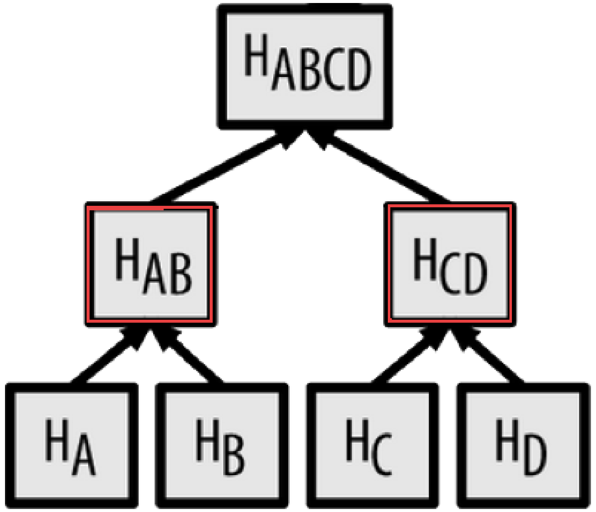
Concatenate H_C+H_D
7BEE85A2D89FE2978D484FD57513EA6C19734A6C2F651C79D4C3B5FBB840C1FE5F1162C57ECF998190D473FF238B6E311D7C52F5F81CA7AC3FCA9267832B585C

Hash($H_C H_D$)
211B3393EDBB7C3E35E24B9885C238B8BA3394A21A4E71824EB0F335DD7A95EC

https://passwordsgenerator.net/sha256-hash-generator/

Merkle Tree and Transaction Verification

Step Four: Concatenate H_{AB}+H_{CD} to Hash(H_{ABDC})



Concatenate H_{AB}+H_{CD}

E2E4A6AFDB1332E2CB6E1634ED76BB9D3C8BFBBE19FED545E6A0347A93DA89AE211B3393EDBB7C3E35E24B9885C238B8BA3394A21A4E71824EB0F335DD7A95EC

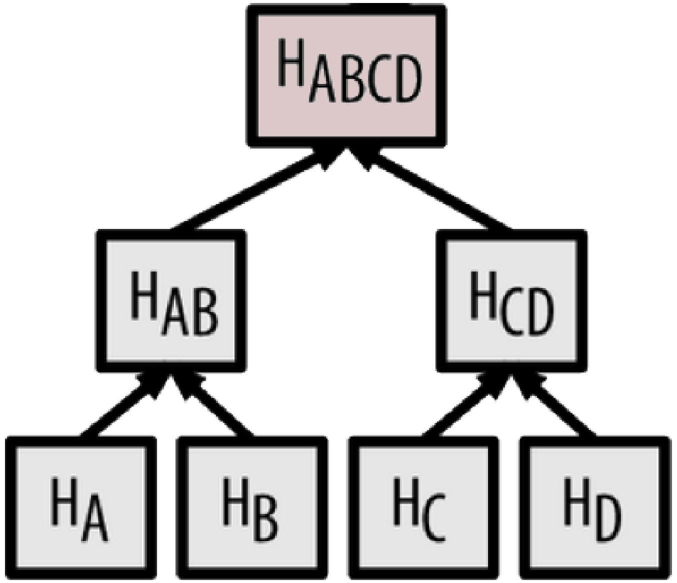
Hash(H_{ABCD})

505A08CC95D523E66300D07F523F8061DFA0673AD4B1ADEA2998CD4E7CC533CF **<= Merkle Root**

https://passwordsgenerator.net/sha256-hash-generator/

Merkle Tree and Transaction Verification

Final Step: Find the Merkle Root: Hash(H_{ABDC})



MERKLE ROOT
Hash(H_{ABCD}): 505A08CC95D523E66300D07F523F8061DFA0673AD4B1ADEA2998CD4E7CC533CF

https://passwordsgenerator.net/sha256-hash-generator/

Merkle Tree and Transaction Verification

Why is it important

- For Blockchains an important usage of a Merkle tree is the ability to construct a Merkle proof that verifies the inclusion of a transaction ***T*** in a specific block ***B***

Example

- Let us assume the block B with block header the previous Merkle root:

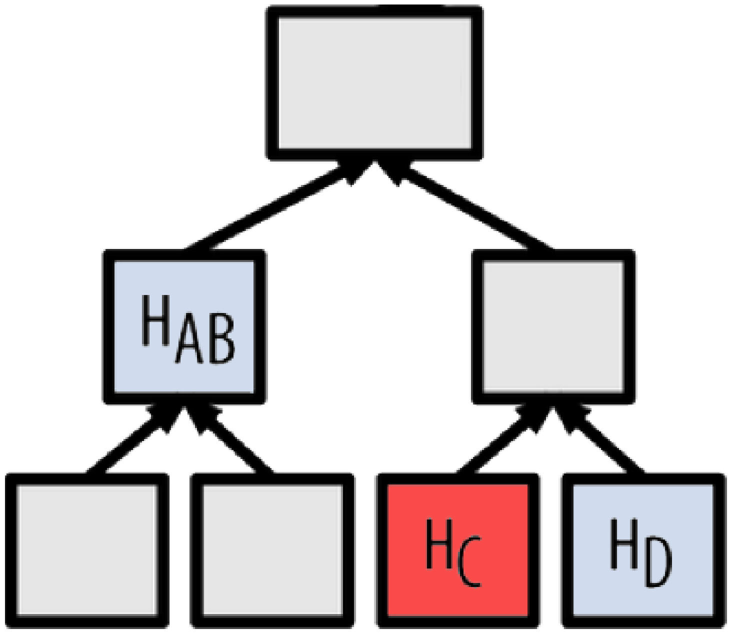
- **Hash(H_{ABCD})**
- 505A08CC95D523E66300D07F523F8061DFA0673AD4B1ADEA2998CD4E7CC533CF

- **Question:** We would like to verify that a transaction - for example, that with hash $\mathbf{H_C}$ is part of the transaction list, thus a ***valid*** transaction.

Merkle Tree and Transaction Verification

Why is it important - proof

Answer: It is sufficient to verify that $\mathbf{H_C}$ leads to the correct **Merkle root (which is currently known, being included in the block).** This verification happens by revealing only ***parts*** of the tree.



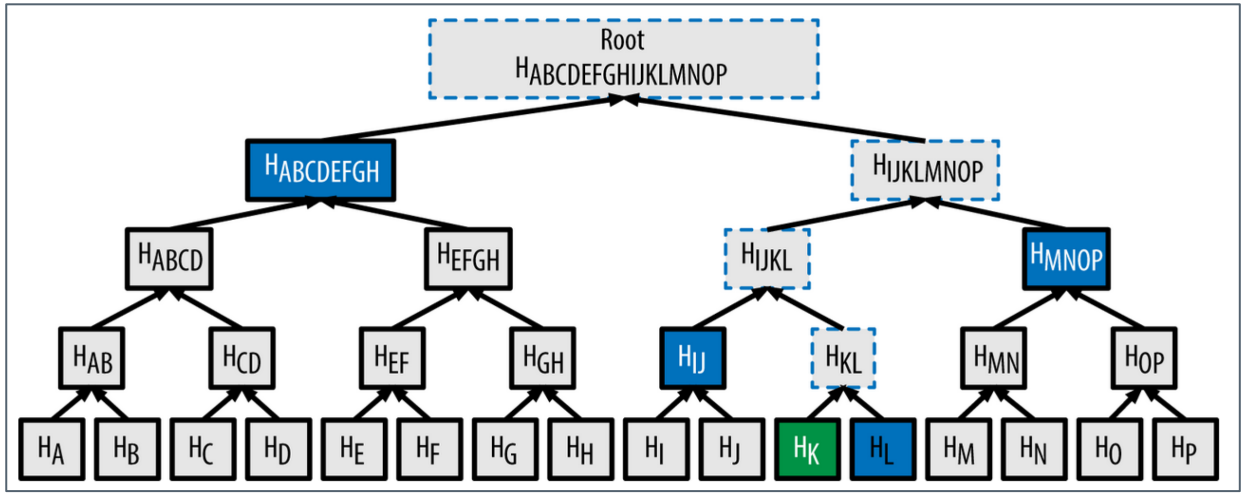
For example to verify that element C is part of the tree, we can reveal hash H_C and hash H_D (thus we can calculate hash H_{CD}) plus reveal H_{AB} (thus we can calculate H_{CABDC} whish is the Merkle root).

Eventually by revealing only three hashes (in this example) from the tree we have proven that element C is part of the tree.

The verification of a Merkle path proceed from the leaf node to the Merkle root.

Merkle Tree and Transaction Verification

Why is it important - proof



'In A merkle path used to prove inclusion of a data element, a node can prove that a transaction K is included in the block by producing a merkle path that is only four 32-byte hashes long (128 bytes total). The path consists of the four hashes (noted in blue in A merkle path used to prove inclusion of a data element) H_L, H_{IJ}, H_{MNOP} and $H_{ABCDEFGH}$. With those four hashes provided as an authentication path, any node can prove that H_K (noted in green in the diagram) is included in the merkle root by computing four additional pair-wise hashes H_{KL}, H_{IJKL}, $H_{IJKLMNOP}$, and the merkle tree root (outlined in a dotted line in the diagram). If H_K leads to the correct Merkle root, then T_K was in the transaction list.'

Mastering Bitcoin https://github.com/bitcoinbook/bitcoinbook/blob/14a25a6fd9e97002c57a46f1fe9faf48cd5d6e46/ch07.asciidoc#merkle-trees (image and explanation)

Questions?

Contact Us:

Twitter: **@mscdigital**
Course Support: **digitalcurrency@unic.ac.cy**
IT & Live Session Support: **dl.it@unic.ac.cy**