Subject Code : CS32101

Merkel Tree

Team : Aimers

Mathematics and Scientific Computing (M.Sc.)
MNNIT ALLAHABAD

April 21, 2023

Members

1. Amit Kumawat 2022MSC15
2. Madhav Khant 2022MSC03
3. Mukund Kushwaha 2022MSC18
4. Narasingha Mahananda 2022MSC25
5. Ritik 2022MSC02
6. Rupesh Kumar 2022MSC13
7. Sainjit 2022MSC22
8. Saurabh Gupta 2022MSC23
9. Shubham Chauhan 2022MSC21

Outline

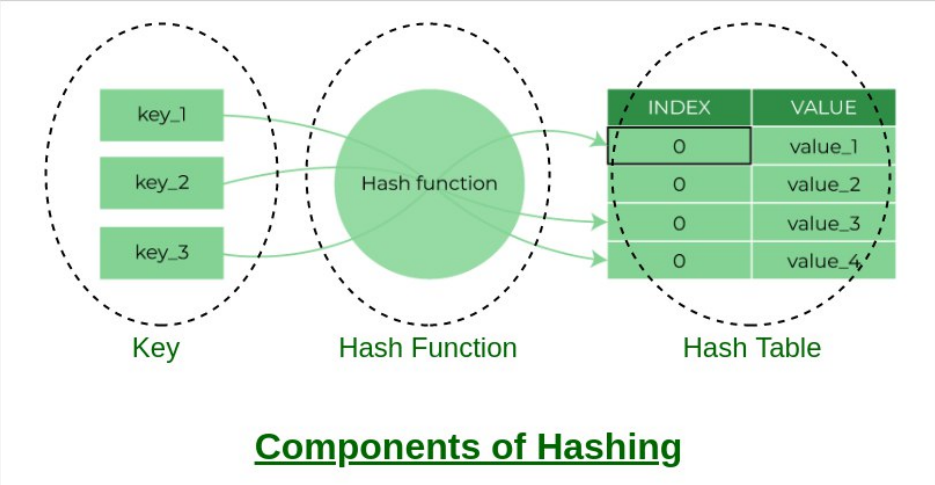
What is Merkle Tree (1/1)

A Merkle tree is a hash-based data 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.

- It is also known as Hash tree.
- It allows efficient and secure verification of the contents of a large data structure.
- It is a generalization of a hash list[1] and a hash chain[2].
- The concept of a hash tree is named after Ralph Merkle, who patented it in 1979.
- Merkle trees are typically implemented as binary trees. However, a Merkle tree can be created as an n-ary tree, with n children per node.

Hashing (1/5)

- Hashing is a technique used to store and retrieve data in a database very quickly and efficiently.
  - Hashing involves mapping data to a unique value, called a hash code.
  - The hash code is then used to index into an array, where the data is stored.



file:Hashing.jpeg

Hashing (2/5)

- A **hash function** is a mathematical formula which, when applied to a key, produces an integer which can be used as an index for the key in the hash table.
  - The main aim of a hash function is that elements should be relatively, randomly, and uniformly distributed.
  - It produces a unique set of integers within some suitable range in order to reduce the number of collisions.
- **Hash table** is a data structure in which keys are mapped to array positions by a hash function.
  - In a hash table, an element with key k is stored at index h(k) not k. It means a hash function, h is used to calculate the index at which the element with key k will be stored.

Hashing (3/5)

Data		Hash Values	
Amit Kumawat	2022MSC15	15	2
Madhav	2022MSC03	03	3
Mukund	2022MSC18	18	5
Narshing	2022MSC25	25	12
Ritik	2022MSC02	02	2
Rupesh	2022MSC13	13	0
Sainjit	2022MSC22	22	9
Saurabh	2022MSC23	23	10
Shubham	2022MSC21	21	8

Hash table →

0	Rupesh
1	
2	Amit Kumawat
3	Madhav
4	
5	Mukund
6	
7	
8	Shubham
9	Sainjit
10	Saurabh
11	
12	Narasingh

file:Hash.png

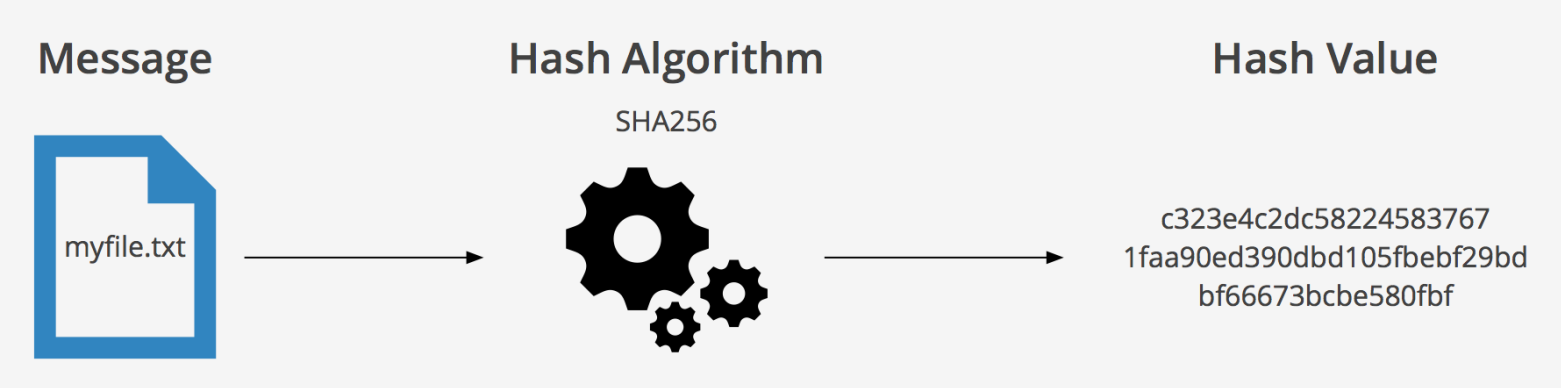
- **Collision** is a situation when two or more keys map to the same memory location.

file:Hashing.pdf

Hashing (4/5)

- A cryptographic hash function such as SHA-256[5] is used for the hashing.
- **SHA-256** is a part of the SHA-2 family of algorithms, where SHA stands for Secure Hash Algorithm.
  - Published in 2001, it was a joint effort between the NSA and NIST to introduce a successor to the SHA 1 family, which was slowly losing strength against brute force attacks.

Hashing (5/5)



- The significance of the 256 in the name stands for the final hash digest value, i.e. irrespective of the size of plaintext/cleartext, the hash value will always be 256 bits.
- It cannot be read or decrypted, as it only allows for a one-way cryptographic function.

file:SHA.pdf

History (1/2)

Behind the Merkel Tree

- The history of Merkle trees in data structures dates back to the late 1970s when Ralph Merkle first introduced the concept of a cryptographic hash function based on a tree structure.

Figure: Ralph Merkel

History (2/2)

Behind the Merkel Tree

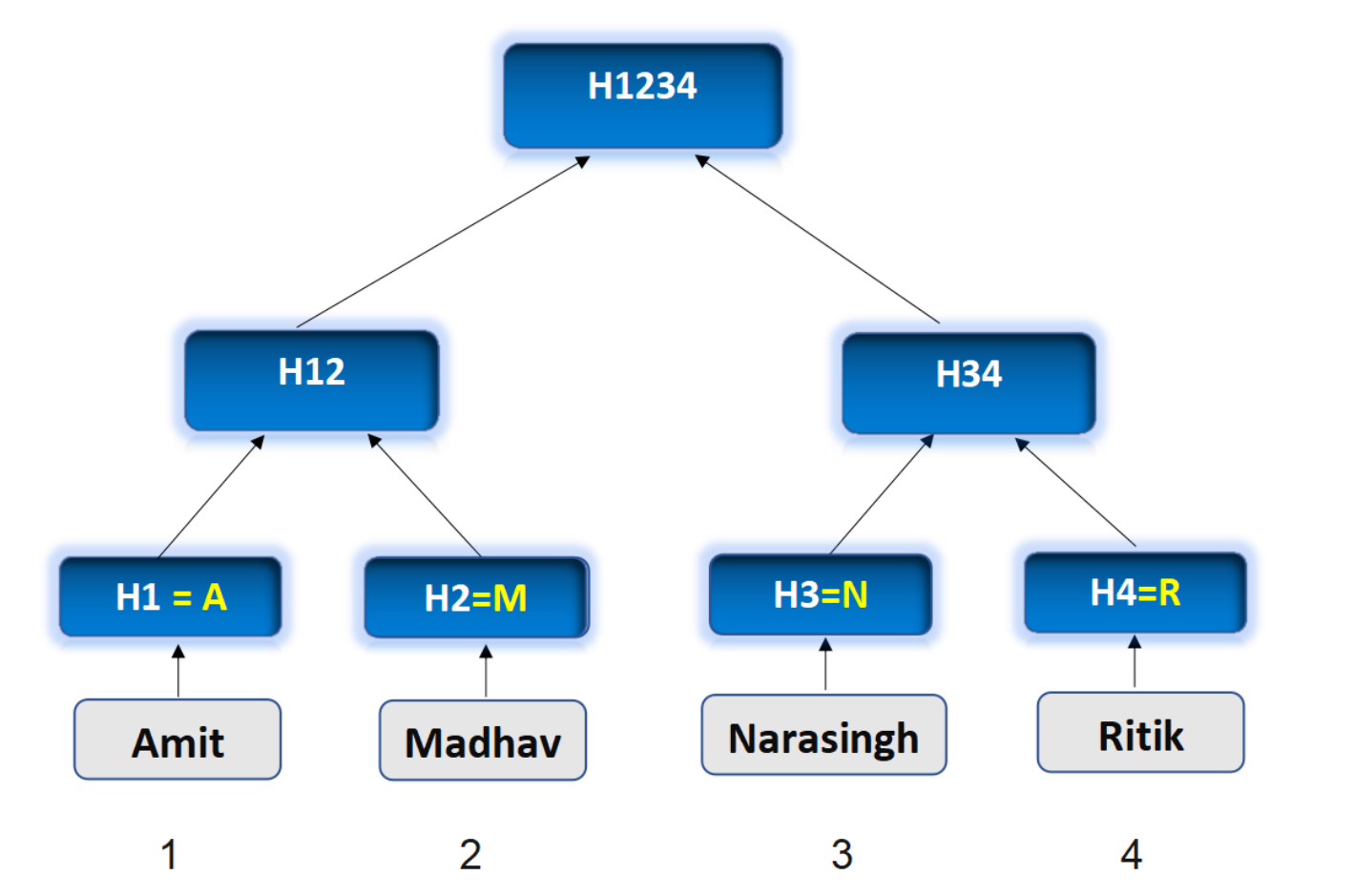
- At the time, hash functions were primarily used for message authentication, but Merkle's tree-based approach allowed for more efficient and secure verification of large datasets.
- Merkle's original paper,"A Certified Digital Signature," [3] proposed a hash function that used a binary tree to organize the hashed data.
- In 1991, Stuart Haber and W. Scott Stornetta built on Merkle's work and proposed the use of Merkle trees for creating tamper-evident log files. Their paper,"How to Time-Stamp a Digital Document," [4] introduced the idea of linking hashed data blocks in a chain, forming a blockchain.

Creation of Merkel tree (1/6)

- Merkle trees are created by repeatedly calculating hashing pairs of nodes until there is only one hash left. This hash is called the **Merkle Root**, or the **Root Hash**.
- The Merkle Trees are constructed in a **bottom-up approach**.
- Every leaf node is a hash of transactional data, and the non-leaf node is a hash of its previous hashes.

Let us assume that we have a set of data as **Amit, Madhav, Narasingh** and **Ritik**. The Merkle tree will look like below:

Creation of Merkel tree (2/6)

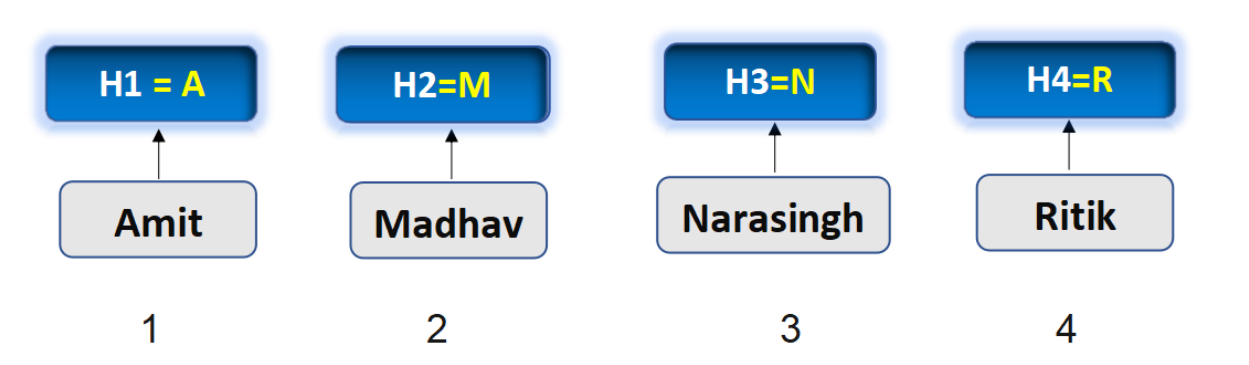


file:In.png

Creation of Merkel tree (3/6)

Let us now see how the Merkle tree is constructed[6].

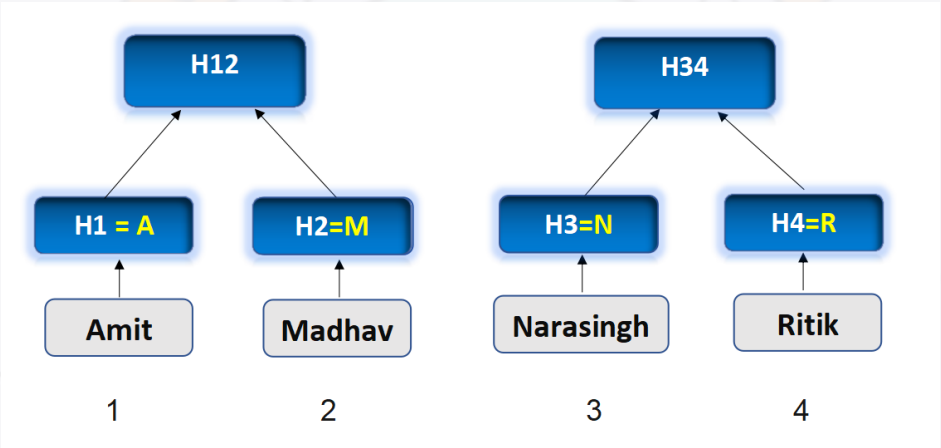
- Find the hash value of every data item. These hash values will be the leaf nodes of our Merkle Tree as shown below .



- Group the leaf nodes into pairs, starting from left to right.

Creation of Merkel tree (4/6)

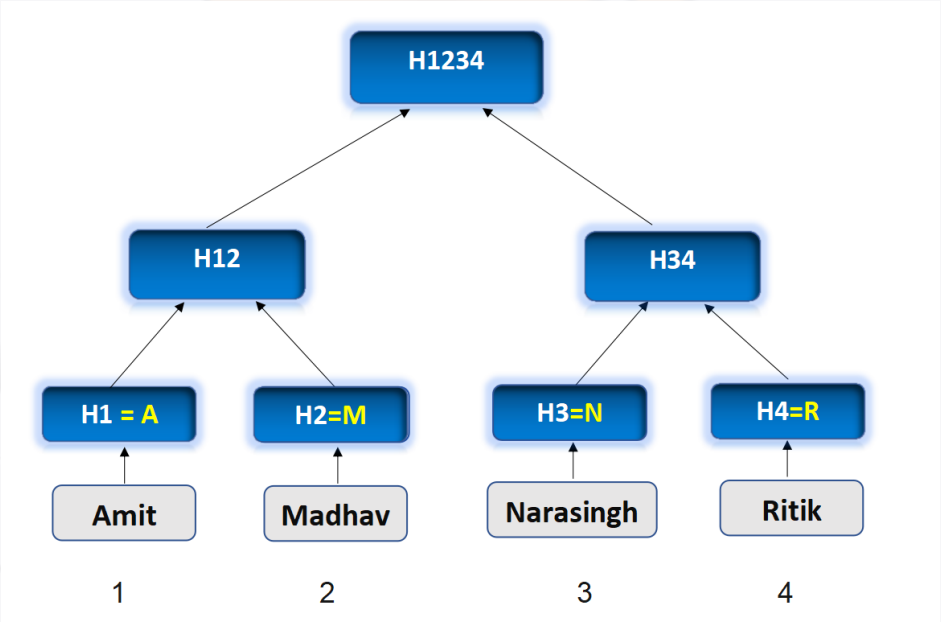
- Create a parent node for each pair. Combine the hash values of the leaf nodes and apply the hash function to the combined value. This will be the label of the parent node.



- Next group the parent nodes into pairs and repeat the above step.

Creation of Merkel tree (5/6)

- Continue the pairing until we are left with a single node which is the Merkle root or root hash.



Creation of Merkel tree (6/6)

Now you may have a question in mind. What if we need to pair an odd number of nodes?

Just create a duplicate of the last node. We now have an even number of nodes in the Merkle Tree. As in the above example, the last element is duplicated.

For more details `file:Iodd.png`

Code :`file:code.pdf`

Deletion (1/3)

In a Merkle tree, deletion is not a straightforward operation as it can compromise the integrity of the tree.

- The reason for this is that each node in the Merkle tree is constructed based on the hash values of its children nodes, and any deletion will result in the modification of the hash values of the affected nodes.

Therefore, the deletion process must be carefully designed to ensure the Merkle tree remains consistent and trustworthy.

1. One way to handle deletion in a Merkle tree is to mark the node as deleted instead of actually removing it from the tree.

Deletion (2/3)

- To mark a node as deleted, we can set its value to a special marker value (e.g., null) and recompute the hash value of the affected node and its ancestors.
- This approach ensures that the integrity of the Merkle tree is maintained because the hash values of the unaffected nodes remain the same.

2. Another approach is to keep track of the deleted nodes separately and use a separate Merkle tree for them.
   - This approach ensures that the integrity of the original Merkle tree is preserved, but it requires additional storage and computational resources.

Deletion (3/3)

In general, the approach to deletion in a Merkle tree will depend on the specific use case and the requirements for data integrity and security.
Lets go through them in detail

Approach 1 (1/4)

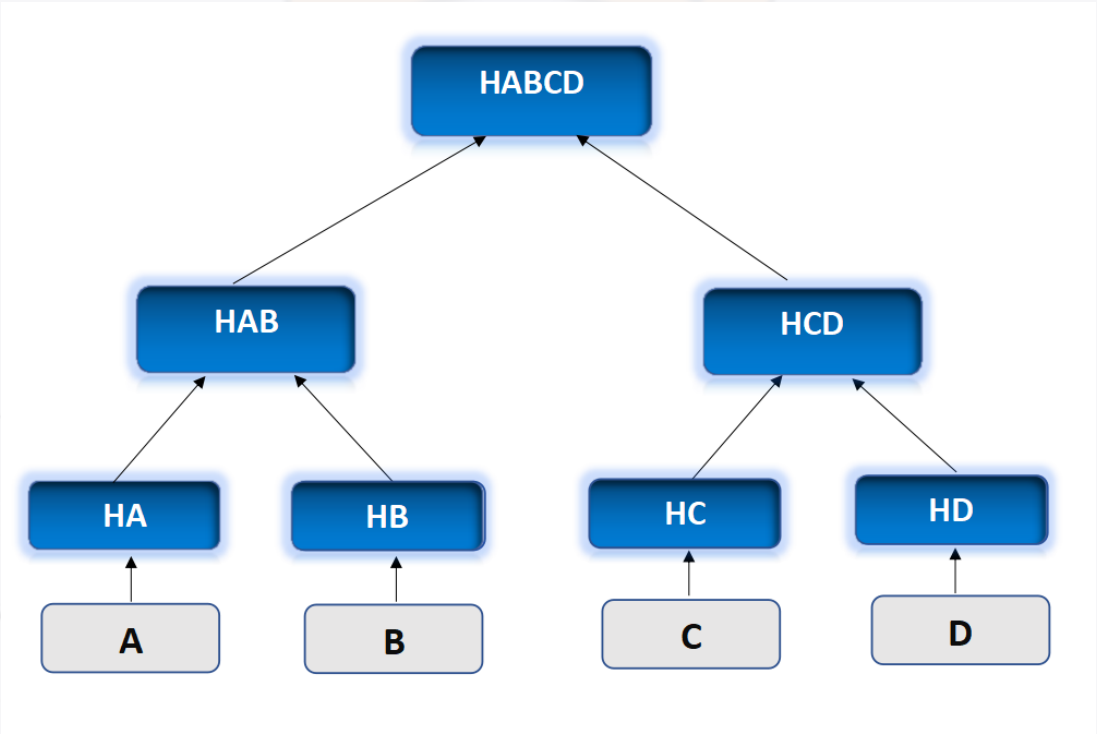
To delete a data element from a Merkle Tree, the following algorithm can be used:

- Find the leaf node corresponding to the data element that needs to be deleted.
- Make that the leaf node as NULL.
- Recalculate the hash of the parent node of the deleted leaf node. This is done by taking the hash of the remaining child node of the parent node.
- Continue to recalculate the hash of each parent node on the path from the deleted leaf node to the root of the tree, until the root hash is reached.

Here is a step-by-step example of how this algorithm works:

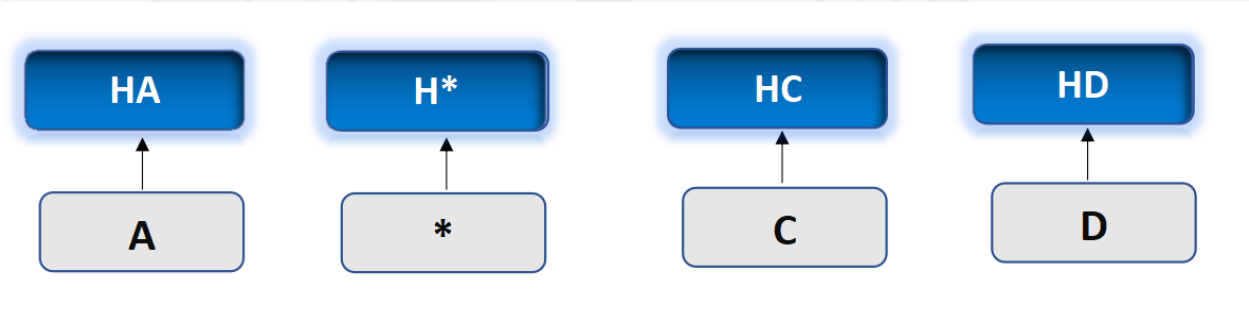
Approach 1 (2/4)

1. Suppose we have a Merkle Tree with four data elements, A, B, C, and D, and the corresponding hash tree is



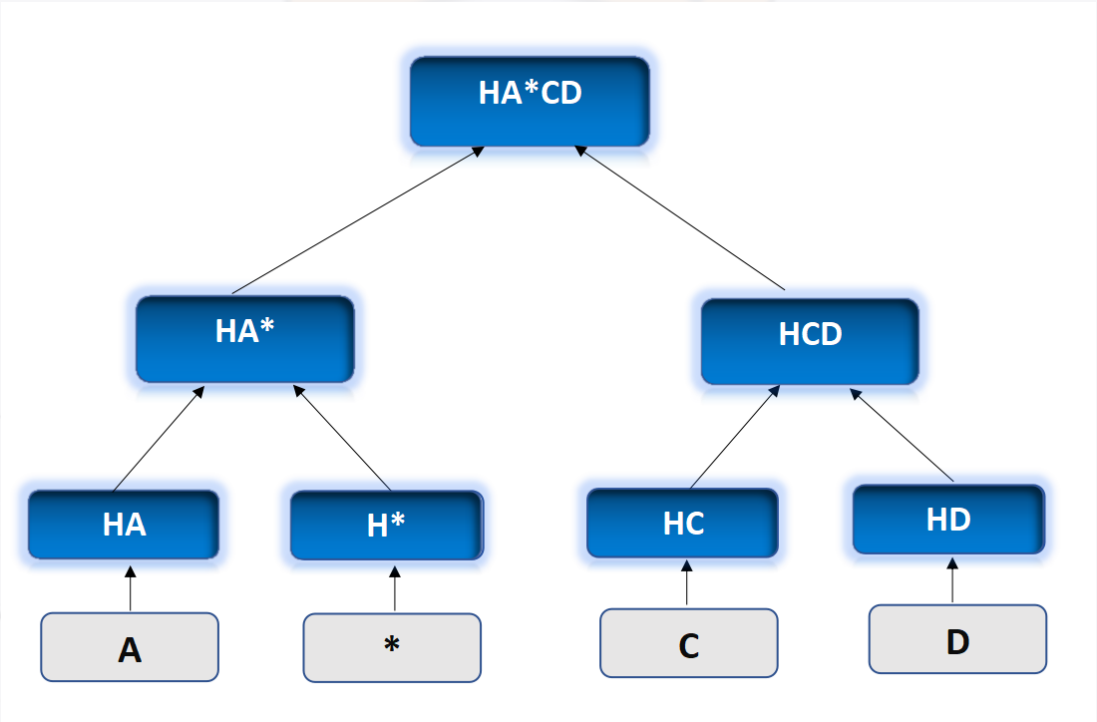
Approach 1 (3/4)

2. Now suppose we want to delete data element B from the tree. First, we locate the leaf node corresponding to B and remove it from the tree(i.e Make it NULL).
   - Here, the required leaf node is node having hash value **HB**
   - Now, make the hash value as NULL
3. Recalculate the hash of the parent node of the deleted leaf node.



Approach 1 (4/4)

4. Continue to recalculate the hash of each parent node on the path from the deleted leaf node to the root of the tree.



So, this is the Merkel tree after deletion.

Approach 2 (1/4)

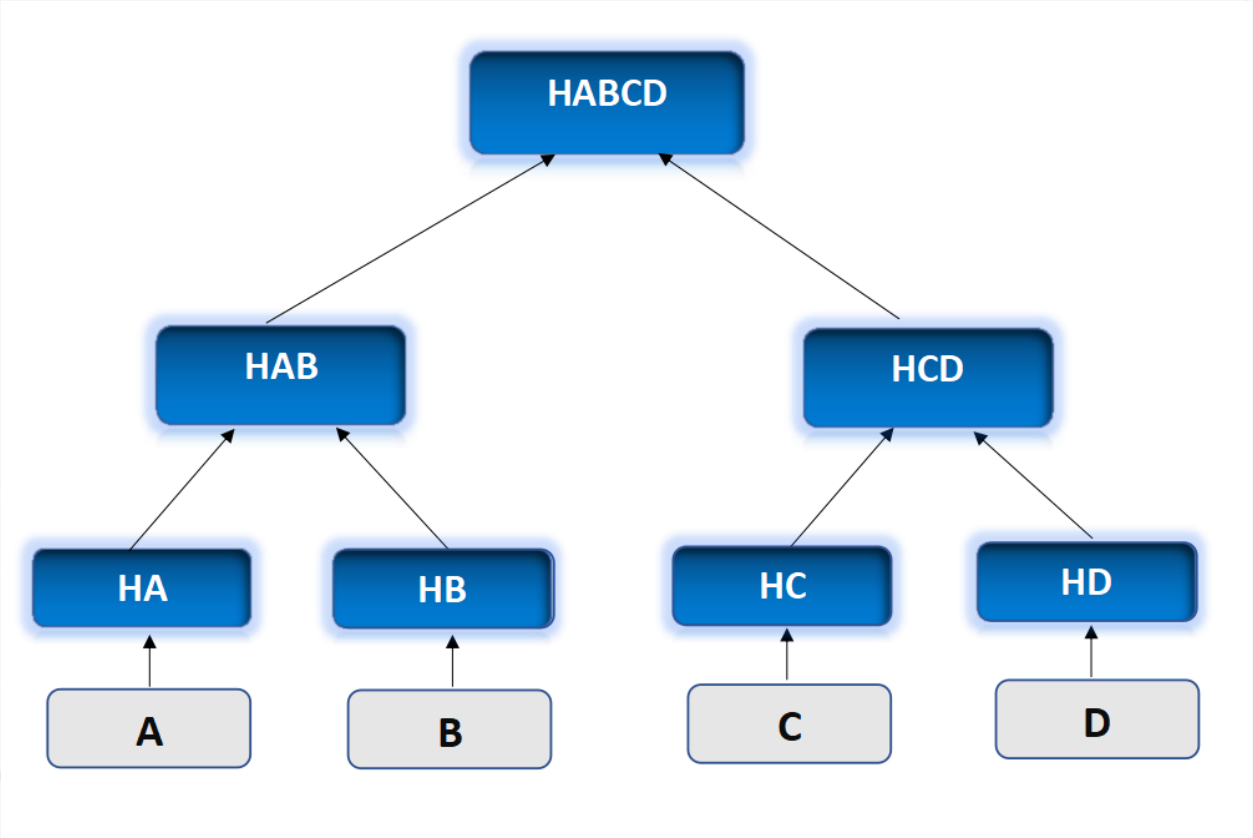
In this approach, the following algorithms can be used :

1. Create an additional storage having the non-deleted data.
2. Now, just create the Merkel tree of this new set of data using the same algorithm used before.

Here is a step-by-step example of how this algorithm works:

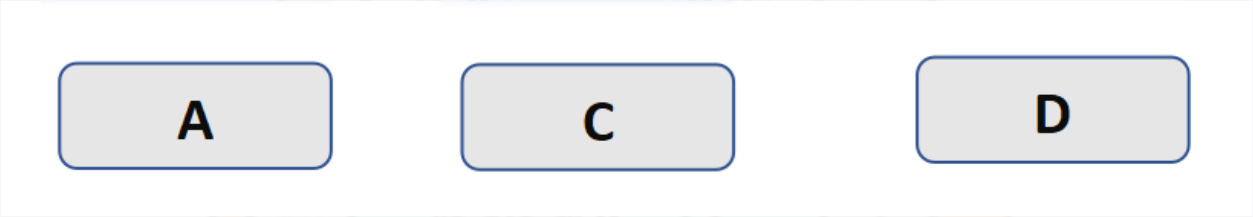
1. Suppose we have a Merkle Tree with four data elements, A, B, C, and D, and the corresponding hash tree is

Approach 2 (2/4)



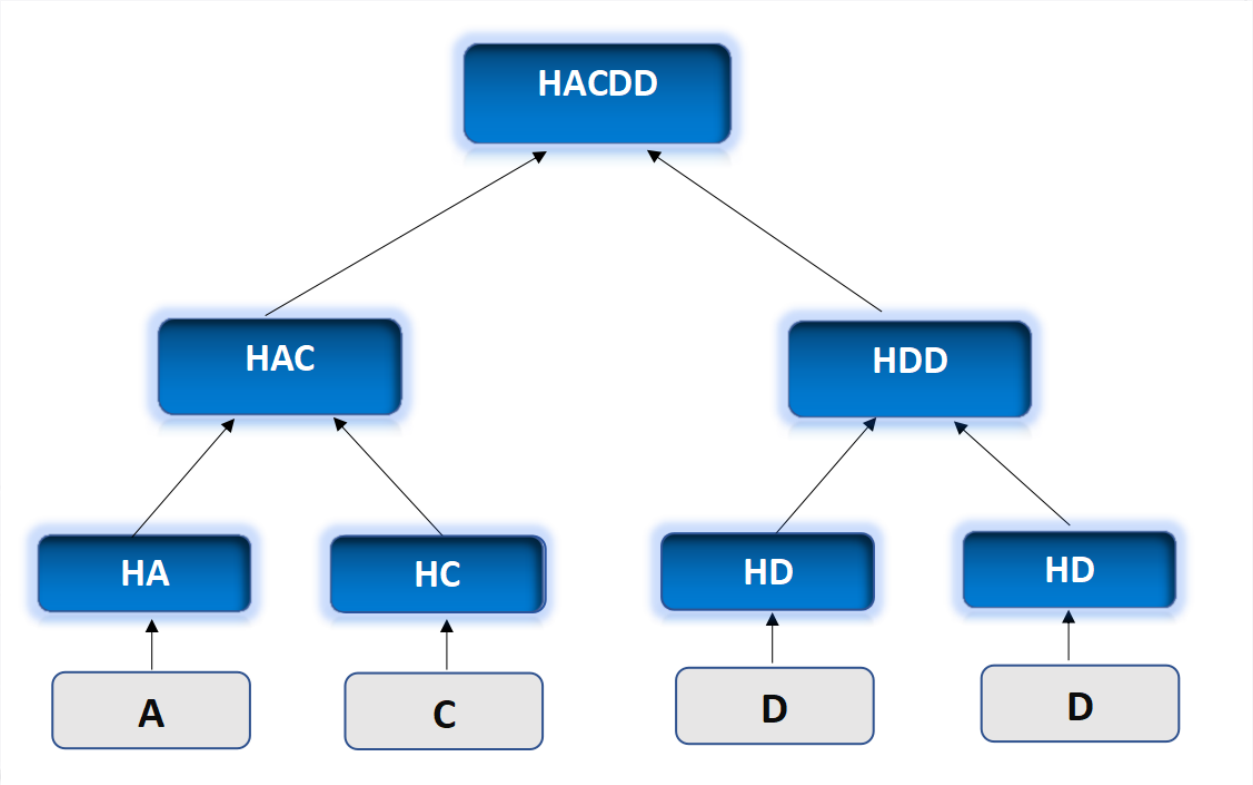
Approach 2 (3/4)

2. Now suppose we want to delete data element B from the tree. First, copy the entire data (except the deleted ones) into new array .



3. Now,just create the merkel tree as described in the insertion section.

Approach 2 (4/4)



So, this is the Merkel tree after deletion.

Data Verification (1/4)

In various distributed and peer-to-peer systems, data verification is very important.

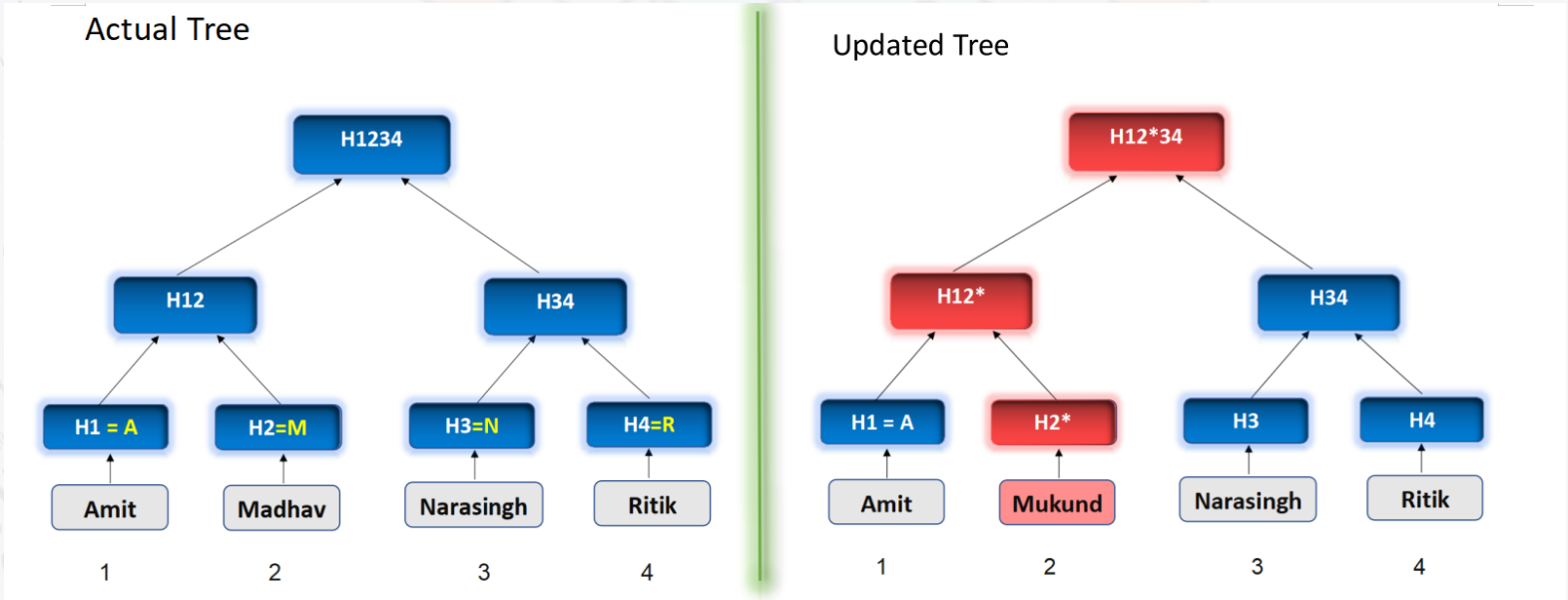
- This is because the same data exists in multiple locations. So, if a piece of data is changed in one location, it's important that data is changed everywhere.
- Data verification is used to make sure data is the same everywhere.

However, it is time-consuming and computationally expensive to check the entirety of each file whenever a system wants to verify data.

- So, this is why Merkle trees are used.

Data Verification (2/4)

Basically, we want to limit the amount of data being sent over a network (like the Internet) as much as possible. So, instead of sending an entire file over the network, we just send a hash of the file to see if it matches.



file:Dv.png

Data Verification (3/4)

The protocol goes like this[7]:

1. Computer A sends a hash of the file to computer B.
2. Computer B checks that hash against the root of the Merkle tree.
3. If there is no difference, we're done! Otherwise, go to step 4.
4. If there is a difference in a single hash, computer B will request the roots of the two subtrees of that hash.
5. Computer A creates the necessary hashes and sends them back to computer B.
6. Repeat steps 4 and 5 until you've found the data blocks(s) that are inconsistent.

Data Verification (4/4)

Remarks :

- It's possible to find more than one data block that is wrong because there might be more than one error in the data.
- Because the computers are only sending hashes over the network (not the entire file), this process can go very quickly.
- Moreover, if an inconsistent piece of data is found, it's much easier to insert a small chunk of fixed data than to completely rewrite the entire file to fix the issue.

Why merkle trees ? (1/2)

Advantages of Merkel tree

- It provides a means to maintain the integrity and validity of data[8].
- Their proofs and management require tiny amounts of information to be transmitted across networks.
- Merkle trees are efficient data structures in terms of storage and time.
- They store hash values of data that require lesser storage space.
- It helps in saving the memory or disk space as the proofs, computationally easy and fast.

Why merkle trees ? (2/2)

Advantages of Merkel tree

- You may note that Merkle trees are used in various distributed applications where data is replicated and stored by multiple parties in a network. If anyone tries to alter data at one location, the hash values will change, leading to a change in the root hash. Thus others in the network can detect this based on the mismatch in the root hash.

Disadvantages (1/2)

When and Where Merkel tree fails

- They require additional computation and storage resources.
  - Each node in the tree needs to compute and store a hash value, which can increase the size of the data being transmitted or stored.
  - Additionally, computing the hash values can be computationally intensive, particularly for large datasets or complex data structures.if the data is constantly changing or being updated, the Merkle tree may need to be frequently recalculated, which could impact performance and add additional overhead.

Disadvantages (2/2)

When and Where Merkel tree fails

- Additionally, if the data is highly structured or interdependent, it may not be possible to efficiently break it into small, independent pieces that can be hashed and organized into a Merkle tree.

- They are vulnerable to collision attacks.
  - While hash functions are designed to minimize the likelihood of collisions, it is still possible for two different pieces of data to have the same hash value.
  - If an attacker is able to create a piece of data with the same hash value as the original data, they could potentially substitute the fake data for the real data without being detected by the Merkle tree.

Applications of Merkel Tree (1/2)

Where Hash trees are Used?

The short answer is, merkle trees can be an integral part of the systems which require:

- Data verification[9]
- Consistency verification [9]
- Data Synchronization [9]
- Authetication [10]

So, based on these concepts, There are many real-life application of Merkel tree. Some of them are

- Blockchain
- Version Control System

file:VCS.pdf

Applications of Merkel Tree (2/2)

Where Hash trees are Used?

- Database System file:Database.pdf
- Cloud Storage file:Cloud.pdf

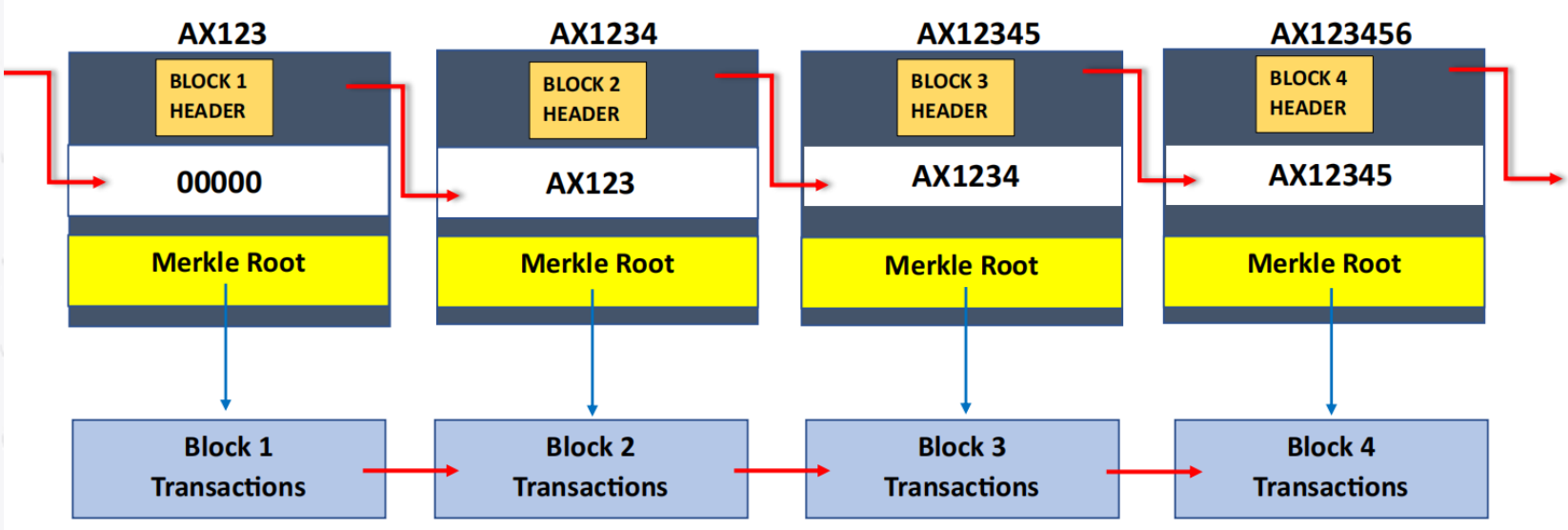
Blockchain (1/4)

- Blockchain[11] is a secure and transparent digital ledger that records transactions across a network of computers, making it resistant to tampering and fraud.
  - It is commonly associated with cryptocurrencies but has broader applications in various industries.

Merkle Tree in Blockchain :-

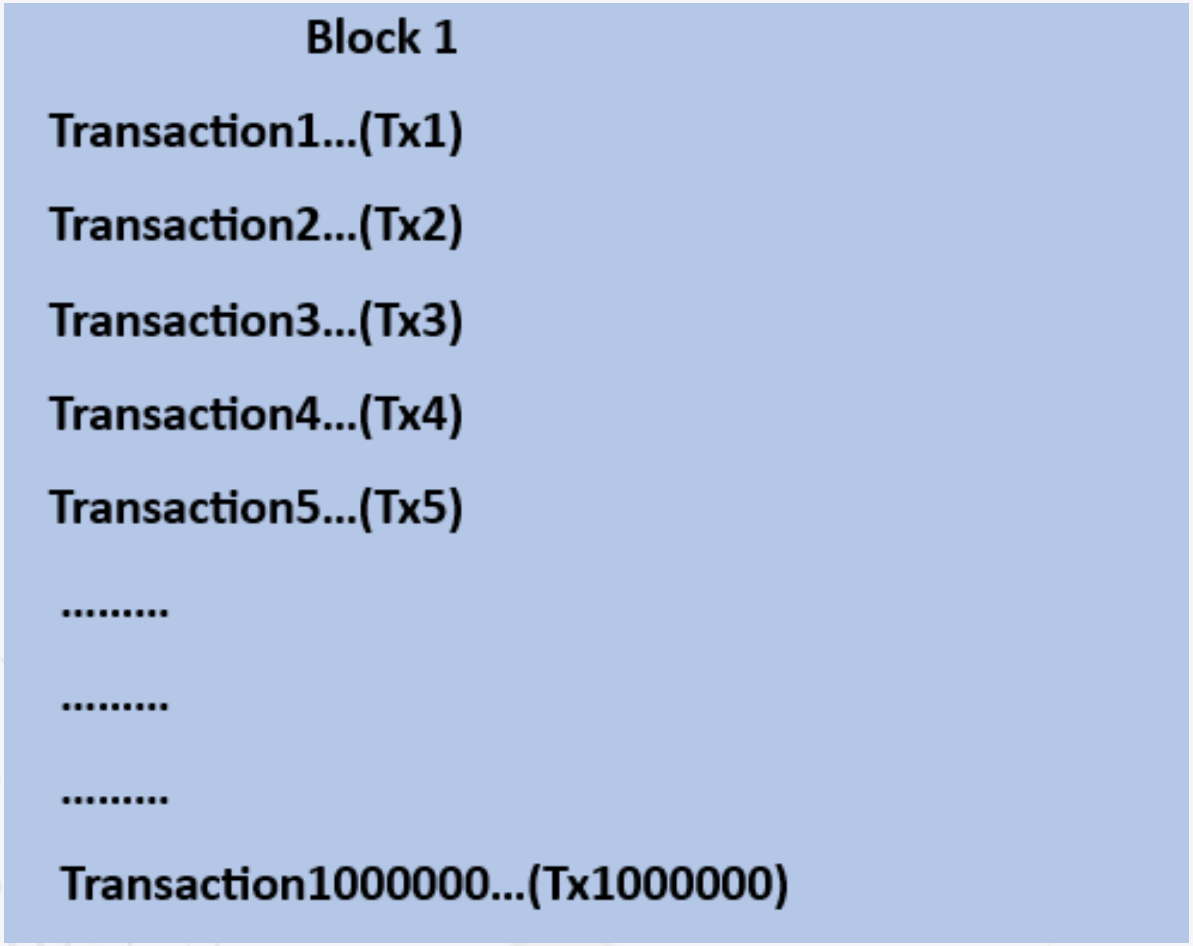
- There are four blockchain as shown in diagram

Blockchain (2/4)



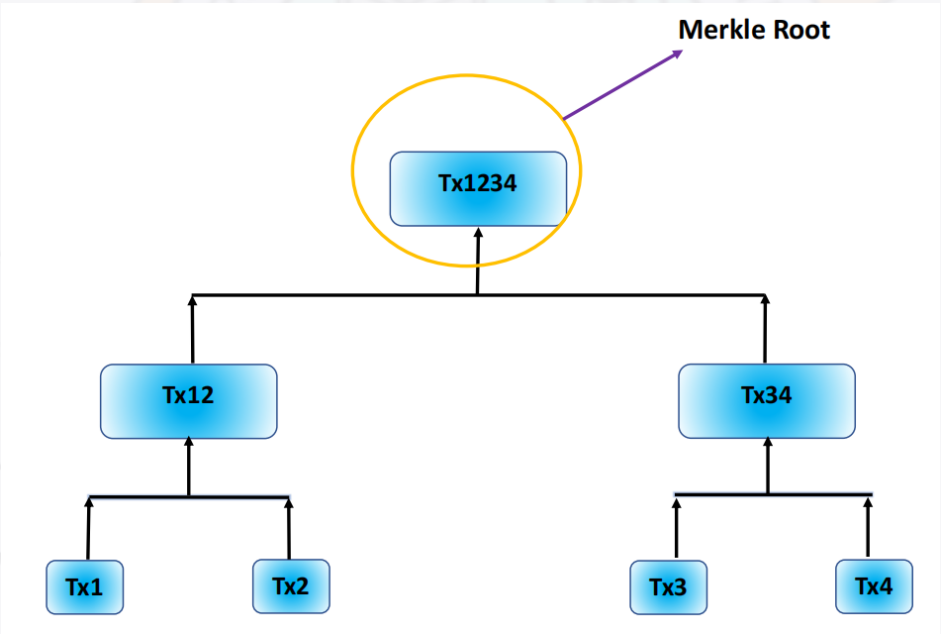
- Each block contains previous block header and lots of transaction.
- Consider one blockchain chain to see how exactly merkle tree works in blockchain.

Blockchain (3/4)



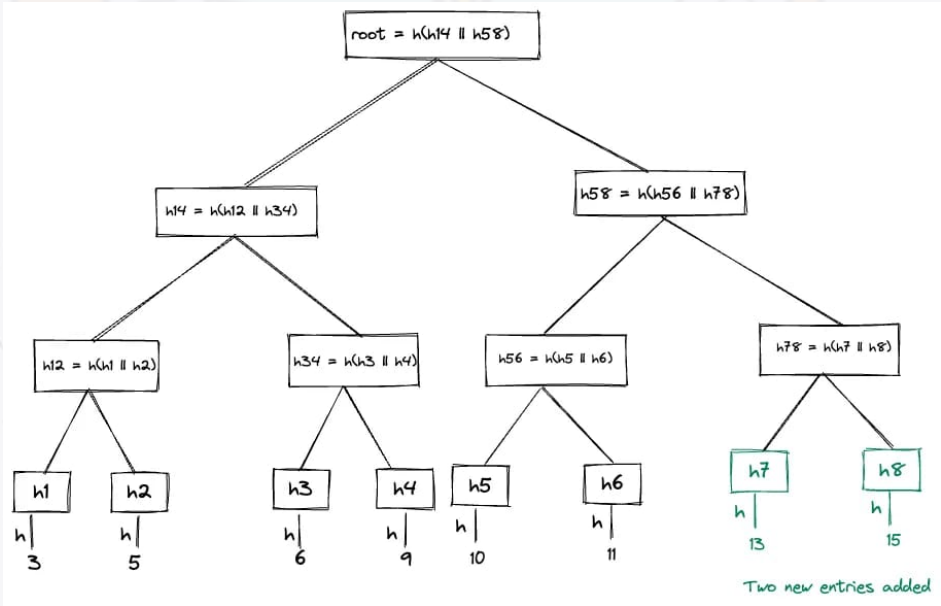
Blockchain (4/4)

- As we can see in given block, there are many transactions. To make more easy to understand, we take only four transactions.
- Applying the merkle tree creation algorithm, we get the merkle root for the blockchain.



Consistency Verification

- Consistency verification is desired in systems maintaining immutable (and hence append-only) log of data.
- It is used to verify that the entire log is untampered, which means verifying that the newer version at any time frame includes all the data of the older version and in the same order, i.e. no data at any stage has been into the history of the log.



Data Synchronization

Merkle trees can be used in synchronizing data across multiple nodes (peers) in a distributed system.

- With merkle trees, we don't need to compare the entire data to figure out what changed — we can just do a hash comparison of the trees.
- Once we figure out which leaves have been changed, the corresponding data chunk can be sent over the network and synced across all the nodes.

References (1/3)

[1] Hash List, *Wikipedia* https://en.wikipedia.org/wiki/Hash_list

[2] Hash Chain, *Wikipedia* https://en.wikipedia.org/wiki/Hash_chain

[3] Ralph C. Merkel, A Certified Digital Signature *CRYPTO* 1989: 218-238

[4] Stuart Haber and W. Scott Stornetta, "How to Time-Stamp a Digital Document,"

[5] SHA-256, "Simplilearn"

[6] Creation of tree, Medium.com

[7] Brilliant.org https://brilliant.org/wiki/merkle-tree/#:~:text=Merkle

References (2/3)

[8] Advantages,Medium.com. https://medium.com/blockchain-stories/merkle-tree-a-beginners-guide-5c53a7defeb9

[9] Data Verification, Consistency verification and Data synchronization,Codementor. https://www.codementor.io/blog/merkle-trees-5h9arzd3n8#data-verification

[10] Authentication,Gene Tsudik,University of California Irvine https://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkleodb.pdf

[11] Blockchain. https://www.simplilearn.com/tutorials/blockchain-tutorial/merkle-tree-in-blockchain

References (3/3)

[12] Codementor
https://www.codementor.io/blog/merkle-trees-5h9arzd3n8

[13] Keccak,GFG https://www.geeksforgeeks.org/difference-between-sha-256-and-keccak-256/

[14] Database System https://medium.com/coinmonks/merkle-trees-concepts-and-use-cases-5da873702318

[15] For Latex :

- Colors
- Overleaf
- Wikibook
- Stackoverflow

Thanks for your attention.