**EMCS2020: Advanced Topics in Computer Security**

Assignment: Explaining the Merkle Tree in Layperson's Terms

Brian Russel Davis, [brian\_davis@brown.edu](mailto:brian_davis@brown.edu)

***What is a Merkle tree?***

A Merkle tree is a way of associating a group of elements such that their identity, qualities, and association cannot be altered or forged. By using a mathematical algorithm known as a hashing function and the qualities of the items ( such as their file name, file size, or any other special or unique properties ), a Merkle tree binds the elements together through the creation of an equation that serves as a special unforgeable signature for that specific group of elements. For example, the elements of [“Baked Alaska”](https://en.wikipedia.org/wiki/Baked_Alaska) ( my favorite dessert ) are quite simple, it includes Milk, Cream, Eggs, Sugar, Flour, etc. However, Baked Alaska is one of the more difficult desserts to create because of the expertise required to assemble the ingredients. Any small change in the amount of one ingredient or in the timing of the way it is created usually results in a catastrophe. Furthermore, if you forget something or try to add something “extra” it will also ruin the recipe. It doesn’t work for example to make the dish without sugar and then try to add it later. The process of cooking the dish binds the ingredients into a special configuration that cannot be reversed or added to once it is done. In a Merkle tree, the ingredients are the elements and the hashing function is the cooking process. A skilled engineer ( like a skilled cook, i.e. Gordon Ramsay ) would be able to instantly tell if the number or quality of the elements changed in any way.

***What is an example of the usage of a Merkle tree in an application?***

An application that needs to keep track of daily stock trading activity is a perfect example where a Merkle tree might be useful. An organization that facilitates stock trades relies heavily on the absolute accuracy of the daily record. The details in the activity are incredibly important, down the millisecond since large transactions executed within seconds of each other could mean the difference in millions of dollars. Creating a new Merkle tree every time a new transaction record is created could be one way to ensure that the set of records is not tampered with ( internal or externally ). Additionally, a Merkle tree can be used to keep a log of any time anyone accesses the records. Every time a new log record was added, the application would create a new Merkle tree with all of the log entries for that day. To take the example even further, the root of the transaction records and the root of the access log trees could be used to create a “Mega Meta” tree[[1]](#footnote-0), so that that the access logs and the transaction records are inextricably bound together in one unalterable, mathematically verifiable record. To reference my analogy above this would mean that each tree would produce a different unique dish and each mega-meta tree would produce a different unique full course meal. Trying to replace or change the ingredients would ruin/invalidate the dish, and the ruined/invalidated dish would ruin/invalidate the meal. In this application, the Merkle tree helps to bind important elements together in a way that preserves their properties and timing so the records can be trusted.

***Why does the above use of a Merkle tree make the application more secure?***

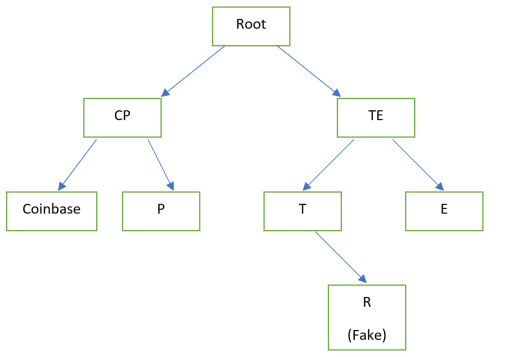
In the example above the trustworthiness of the records relies on the inability of anyone, including the company that owns the application, to tamper with the records. An attacker seeks to tamper with the files and to make their record of access to the files undetectable, so they records seem creidible. In other data structures where confidentiality is more important securing the data means hiding it from view. In order for data in a Merkle tree to be more secure it better that it is public so there is several sources of independent consensus on the record of the tree itself. The layers of Merkle tree data structure secures the intergrity of the records and creativity links them with the the log of who accessed the records. If were easy to just change a few records and reconstitute the Merkle tree, then the intergity of the information wouldn’t be very safe from internal and external attackers. However, since the Merkle tree makes it near impossible to change anything about any of the elements without the risk of invalidating all the dependent records up the chain, it could be said that the trustworthiness of the records as a set is 99.99% percent safe. Trying to forge a record without being dectected would mean finding a collision of a hash of what might be millions of records. Furthermore a forgery of a full day of records would be caught when the mega meta tree was made and verified. The security that is given with the use of a Merkle tree allows the admins and users to rest assured that the records they are viewing are accurate.

***Addtional Observations***

While the Merkle tree protects from forgery or alteration after a record is created, it cannot protect from inaccurate records that are entered into the tree at the time of creation.

Merkle trees are not indestructable. In fact weaknesses have been found that can compromise their integrity specifically in the way BTC Wallets uses them to verify transaction.[[2]](#footnote-1) The attack relies on the fact that in BTC merkle trees it not easy, and sometimes impossible to tell the difference between the hash of a inner node and a leaf node, since both are created by concatenating a 32 bit hash. If a BTC wallet uses a more simplified form of verification, one that just looks at a branch of the Merkle tree, an attacker can trick the wallet into thinking that a leaf node is an inner node with a fake proof.

Lastly, I am not so sure about the speed at which this data structure could be traversed. In order to verify a smaller body of data I can see the Merkle tree perform moderately well, but not nearly as fast as a heap or a stack. If I were constructing a database that needed the verification of Merkle nodes then I would build a hybrid structure, one that used a heap to store pointers to the “next node” so the Merkle tree can be traverses O(n) rather than O(log(n)). As each record is created I would store a the pointer to the record in a stack so the code traversing the Tree didn’t have to query for or loop to the next node or the end structure. Instead it would simply read directly from each not with a need for processing power to figure what comes next. It would be like folding a complex paper snowflake that was created from 10,000 layers into an accordion that revealed it chunk by chunk, rather than having to traverse the paths of the layers.



1. “Mega Meta Merkle tree”, copyright Brian Russel Davis, 2019 [↑](#footnote-ref-0)
2. Lerner, S. D. (2018, June 13). Leaf-Node weakness in Bitcoin Merkle Tree Design. Retrieved November 3, 2019, from https://bitslog.com/2018/06/09/leaf-node-weakness-in-bitcoin-merkle-tree-design/. [↑](#footnote-ref-1)