Prof. Dr. Schönhuth
Johannes Schlüter

Privacy in Healthcare

(Possibly achievable number of points: 20)

Deadline: November 13th, 23:59.
Please, choose your group in the Moodle. Without a group, submitting your solutions is not possible.

November 1, 2023

If you have any questions regarding the exercises, send an e-mail to j.schlueter@uni-bielefeld.de. I will give my best to respond as fast as possible. The goal of this exercise sheet is to give you a better idea of some of the foundations of the Bitcoin blockchain. Exercises consist of theoretical questions and programming exercises in python. Please submit your solution as a **zip file**.

1 Exercise 1: Theoretical questions

1.1 Exercise 1.1:

Recall the basic cryptocurrency ScroogeCoin from the lecure. In ScroogeCoin, suppose Mallory tries to generate (sk, pk) pairs until her secret key matches someone else's. What will she be able to do and how long will it take before she succeeds on average? What if Alice's random number generator has a bug and her key generation procedure produces only 1,000 distinct pairs? (3 points)

1.2 Exercise 1.2:

How does Bitcoin achieve consensus? How does consensus prevent double-spending? (4 points)

2 Exercise 2: Merkle Trees

Each block in the Bitcoin blockchain consists of a header and a main body. The main body contains all transactions, whereas the header contains the current version of the Bitcoin client, the hash of the previous block, the Merkle root, a timestamp, the number of bits of the block, the nonce and the number of transactions, refer to Table 1.

Version	02000000
Hash of the Previous Block	0000000000000001ef42d93177gf. . .
Merkle root	46e7a942fb425c3837d52f198452. . .
Timestamp	358b0553
Number of bits	535f0119
Nonce	48750833
Number of Transactions	1493
Transactions	

Table 1: Structure of a Bitcoin block in the style of Bhaskar, N. D./Lee, D. K. C. (2015), S.48

Definition by Narayanan et al.: Merkle tree. A binary tree with hash pointers is known as a Merkle tree. Suppose we have a number of blocks containing data. These blocks comprise the leaves of our tree. We group these data blocks into pairs of two, and then for each pair, we build a data structure that has two hash pointers, one to each of these blocks. These data structures make the next level up of the tree. We in turn group these into groups of two, and for each pair, create a new data structure that contains the hash of each. We continue doing this until we reach a single block, the root of the tree. As before, we remember just the hash pointer at the head of the tree.

Prof. Dr. Schönhuth
Johannes Schlüter

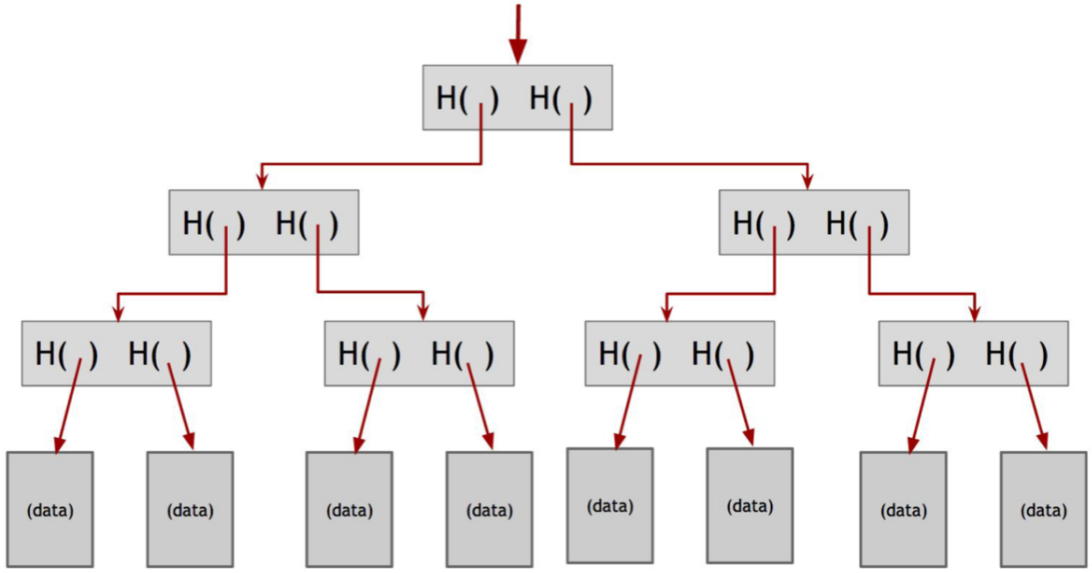

Figure 1: In a Merkle tree, data blocks are grouped in pairs and the hash pointer to each of these blocks is stored in a parent node. The parent nodes are in turn grouped in pairs and their corresponding hash pointers are stored one level up the tree. This continues all the way up the tree until we reach the root node. Figure by Narayanan et al.

Note, that the Merkle tree in 1 can only handle a total number of leaves that is a power of two. If we had nine leaves for example, we would not be able to group all leaves into pairs of two. The Merkle tree that we are going to implement should be able to handle an uneven number of leaves. One way to do so is to create a parent node that stores two hash pointers that both point to the same child.

2.1 Exercise 2.1: In the file exercise 2.ipynb, you can find the classes MerkleTree() and Node(). Implement the class Node() with the following properties:

- A node stores two hash pointers to its two children, which are either leaves or other nodes. (1 point)
- Implement the method get left child(). The method returns the left child, the child is either a node or a leaf. The method does not return the hash pointer to the child, but the entire node object or leaf. (1 point)
- Implement the method get right child(). The method returns the right child, the child is either a node or a leaf. The method does not return the hash pointer to the child, but the entire node object or leaf. (1 point)

2.2 Exercise 2.2: Implement the class MerkleTree() with the following properties:

- Implement the method create node to create new Node objects. This method receives either two leaves or two Node objects as input. Use the newly implemented class Node(). (2 points)
- Implement the method parents of leaves(). This method returns the parent nodes of the leaves. (2 points)
- Implement the method get parents from node objects(). This method returns the parent nodes of other node objects. Note: You can implement a single method that returns parent nodes regardless of the input (leaves or node objects), but splitting the task in two might be helpful, the choice is up to you though. (2 points)
- Implement the method get root(). This method receives some sample transactions as input and returns the corresponding Merkle root. (2 points)

2.3 Exercise 2.3: Use the newly implemented classes and the sample transactions at the top of the file exercise_2.ipynb to build your own Merkle tree. Show, that your Merkle tree can handle an uneven number of inputs. Then use the Merkle root to go trough the tree and retrieve the left and rigth leaf. (2 points)