*Prof. Dr. Schönhuth*
*Johannes Schlüter*

# Privacy in Healthcare

(Possibly achievable number of points: 20 points + 13 BONUS points)

**Deadline: November 27th, 23:59.**
**We agreed on postponing our meetings by one week. Therefore, next Meeting is the 6th of December.**

November 15, 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 how transactions work in Bitcoin, how proof-of-work is implemented and how we can sign and verify transactions. Unlike the other exercises, this exercise sheet has no theoretical questions, because the programming exercise is more demanding than usual. The details of exercise 1 and 2 can be found in the file 'Exercise 3 skeleton.ipynb'. This document merely provides some background information to set the stage for the exercises. Please submit your solution as a **zip file**.

**A Bitcoin transaction**

As you can see in Figure 1, a Bitcoin transaction consists of metdadate, input(s) and output(s).

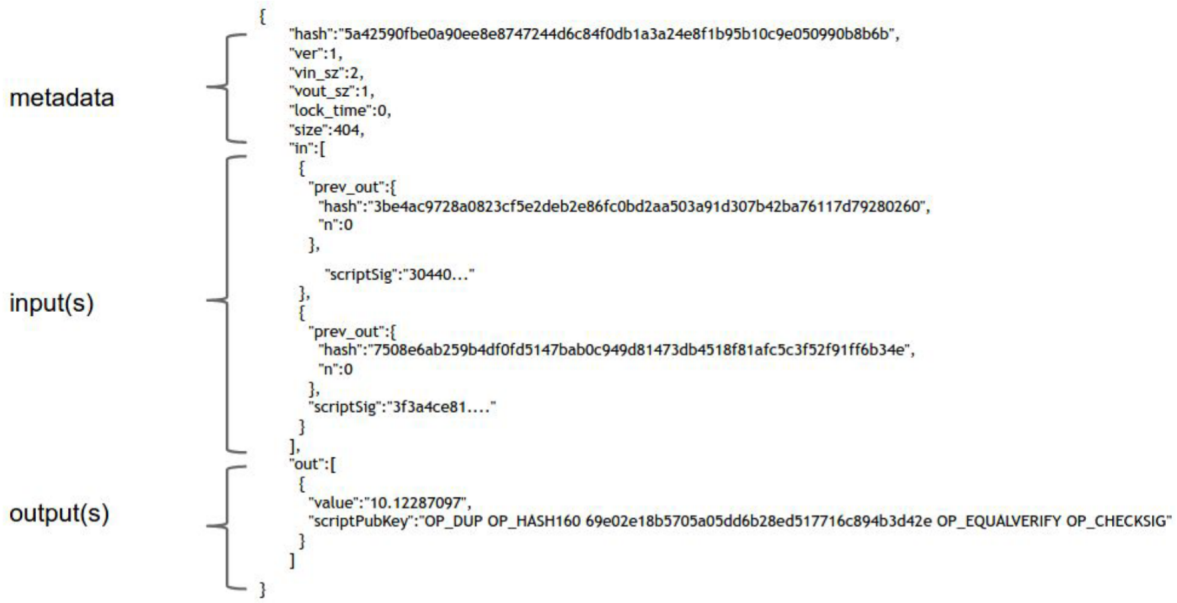


**Figure 1: An actual Bitcoin transaction. Figure by Narayanan et al.**

- Metadata: There's some housekeeping information — the size of the transaction, the number of inputs, and the number of outputs. There's the hash of the entire transaction which serves as a unique ID for the transaction. There's also a"lock time" field, which we are going to ignore here.
- Inputs: The transaction inputs form an array, and each input has the same form. An input specifies a previous transaction, so it contains a hash of that transaction (stored in the metadata part of a transaction), which acts as a hash pointer to it. The input also contains the index of the previous transaction's outputs that's being claimed. And then there's a signature. Remember that we have to sign to show that we actually have the ability to claim those previous transaction outputs.

- Outputs: The outputs are again an array. Each output has just two fields. They each have a value, and the sum of all the output values has to be less than or equal to the sum of all the input values. If the sum of the output values is less than the sum of the input values, the difference is a transaction fee to the miner who publishes this transaction. And then there's a funny line that looks like what we want to be the recipient address. Each output is supposed to go to a specific public key, and indeed there is something in that field that looks like it's the hash of a public key. But there's also some other stuff that looks like a set of commands. Indeed, this field is a Bitcoin script.

In this exercise, we are not going to work with Bitcoin scripts for two reasons: First of all, it would break the scope of the exercise sheet to ask you to implement scripts, and second, the vast majority (99.9 percent according to Narayanan et al.) of scripts being used, are scripts that just specify one public key and require a signature for that public key in order to spend the coins. Therefore, instead of implementing scripts, we are just going to assume, that the output section of a transaction includes a value field (as specified above) and the public key of the recipient of the transaction.

**A Coinbase transaction**

In Bitcoin, the node that is allowed to propose a new block, is also entitled to create a coin creation transaction and to enclose it in the newly created block. We call it a 'coinbase' transaction. A coinbase transaction creates new coins. It does not redeem a previous output, and it has a null hash pointer indicating this. It has a coinbase parameter which can contain arbitrary data. The value of the coinbase transaction is the block reward plus all of the transaction fees included in this block.

```
"in":[
       {
         "prev_out":{
           "hash":"000000.....0000000",
           "n":4294967295
         },
         "coinbase":"..."
       },
     ]
"out":[
       {
         "value":"25.03371419",
         "scriptPubKey":"OPDUP OPHASH160 ... ”
       }
      ]
```

**Figure 2: Coinbase transaction. Figure by Narayanan et al.**

In this exercise, we are not gong to take the transaction fees into account. Instead, the value of a coinbase transaction is always going to be the block reward, which is set to 25. Note, that a coinbase transaction also contains meta data, figure 2 only shows the input and output part of a coinbase transaction.

# 1 Exercise 1: Programming exercise

In the file 'Exercise 3 skeleton.ipynb' you can find the class blockchain(). Implement the class. Everything you need to know to implement the class and its methods is written in the .ipynb file. The exercise is worth 23 points in total, while the last 3 points are BONUS points.

*Prof. Dr. Schönhuth*
*Johannes Schlüter*

## 2 Exercise 2: For interested only: A little use case

In the file 'Exercise 3 skeleton.ipynb' you can find a little use case, in which we use the newly implemented class blockchain(). Follow the instructions as specified in the .ipynb file. The exercise is worth 5 BONUS points in total.