Aufgabe 3.1

a) Siehe Abbildung 1-3:

```
kuhmist123@kuhmist123-VirtualBox:~$ bitcoin-cli -regtest getblockchaininfo
{
  "chain": "regtest",
  "blocks": 3433,
  "headers": 3433,
  "bestblockhash": "2aa3cbcde877651a6046abad0a61c0650caca1a8ad5027b566c6b9a3c3cc49c5",
  "difficulty": 4.656542373906925e-10,
  "time": 1715860546,
  "mediantime": 1715859240,
  "verificationprogress": 1,
  "initialblockdownload": false,
  "chainwork": "0000000000000000000000000000000000000000000000000000000001ad4",
  "size_on_disk": 1184019,
  "pruned": false,
  "warnings": ""
}
```

Abbildung 1: Verbindung zur Blockchain

```
kuhmist123@kuhmist123-VirtualBox:~$ bitcoin-cli -regtest addnode "144.91.79.46:1800" add
kuhmist123@kuhmist123-VirtualBox:~$ bitcoin-cli -regtest getaddednodeinfo '144.91.79.46:1800'
[
  {
    "addednode": "144.91.79.46:1800",
    "connected": true,
    "addresses": [
      {
        "address": "144.91.79.46:1800",
        "connected": "outbound"
      }
    ]
  }
]
```

Abbildung 2: Verbindung zur Node 1

b) 293 Bytes zu Beginn 2048322 Bytes heute
c) ID von Block 21: 099027aad3ce9e733b4cc326ef85aae6471a99c83e6e9457fca730a357f63337
d) Text im Block 2100: Gründlagen der Blockchain-Technologie: öbungsaufgabe
e)
f)
Das Skript wurde als Powershell-Skript entwickelt:

```
$blockCount = .\bitcoin-cli -regtest getblockcount
for ($i = 0; $i -le $blockCount; $i++) {
    $blockHash = .\bitcoin-cli -regtest getblockhash $i
    $block = .\bitcoin-cli -regtest getblock $blockHash
    $convBlock = $block | ConvertFrom-Json
    foreach ($txid in $convBlock.tx) {
        $txDetails = .\bitcoin-cli -regtest getrawtransaction $txid 1
```

```
kuhmist123@kuhmist123-VirtualBox:~$ bitcoin-cli -regtest getaddednodeinfo '62.171.152.173:1800'
[
  {
    "addednode": "62.171.152.173:1800",
    "connected": true,
    "addresses": [
      {
        "address": "62.171.152.173:1800",
        "connected": "outbound"
      }
    ]
  }
]
```

Abbildung 3: Verbindung zur Node 2

```
        $convTxDetails = $txDetails | ConvertFrom-Json
        foreach($vout in $convTxDetails.vout) {
        $scriptPubKey = $vout.scriptPubKey
            if ($scriptPubKey -and $scriptPubKey.asm){
                if($scriptPubKey.hex -match '1[0-9a-fA-F]+'){
                    $pubKeys = $scriptPubKey.asm -split " "
                    Write-Output "Oeffentlicher Schluessel: $pubKeys"
                }
            }
        }
    }
}

        $timestamp = Get-Date -Format "yyyy-MM-dd HH:mm:ss"
        Write-Output "Skript ausgefuehrt am: $timestamp"
```

Aufgabe 3.2

a) $p_1 : 61, p_2 : 97, q_1 : 23, q_2 : 29[1; 100]$
Zum Verschlüsseln müssen wir zunächst $n = p_1 * q_1 = 61 * 23 = 1403$ berechnen. Wir berechnen ausserdem: $\phi(n) = (p_1 - 1) * (q_1 - 1) = 60 * 22 = 1320$. Nun wählen wir eine Zahl e_1, die teilerfremd zu ϕ ist. Die Zahl 7 ist teilerfremd, weshalb wir diese verwenden werden. Nun bestimmen wir ein d_1, sodass $e_1 * d_1 \equiv 1 \mod \phi(n)$ gilt -> (943).

b) Berechnung zur Signierung: $n = p_2 * q_2 = 97 * 29 = 2813$ Wir berechnen ausserdem: $\phi(n) = (p_2 - 1) * (q_2 - 1) = 96 * 28 = 2688$. Als teilerfremde Zahl verwenden wir diesmal $e_2 = 5$. So kommen wir auf ein $d_2 = 1613$. Betrachten wir nun die Verschlüsselung und die Signatur:

$$c = m^{e_1} \mod n_1 = 3169^7 \mod 1403 = 558$$

$$m = c^{d_1} \mod n_1 = 558^{943} \mod 1403 = 363$$

$$sig = m^{d_2} \mod n_2 = 3169^{1613} \mod 2813 = 677$$

$$m = sig^{e_2} \mod n_2 = 677^5 \mod 2813 = 356$$

Aufgabe 3.3

a) Ja
b) Ja
c) Ja
d) Ja
e) Ja
f) Nein

Aufgabe 3.4

a) Gültig. Alice 7, Bob 12, Carol 6.
b) Ungültig. Bei Tx3 hat Txin gerade einmal 12,5 coins, es sollen aber 14,5 rausgehen (Txout).
c) Gültig. Alice 6, Bob 9, Carol 7. Man könnte meinen es wäre ungültig, da die Transaktionen Tx1 und Tx2 jeweils nicht den vollen Betrag ausnutzen von dem ursprüunglichen Txin, man kann aber davon ausgehen, dass es sich um die restlichen Coins um transaktionskosten gehandelt haben muss.
d) Ungültig. Bei Tx2 Signiert Bob die Transaktion, obwohl Txin: 1[1] zu Alice gehört.

Aufgabe 3.5

a) 595.19 EH/s (SHA-256)
b) 612668; Quelle: https://jochen-hoenicke.de/queue/#BTC,all,weight
c) Verschiedene Nodes haben verschiedene Sichtweisen auf die Chain und somit auch einen unterschiedlichen Blick auf die Anzahl unbestätigter Transaktionen.
d)
a) 57043
b) Sender: 1XPTgDRhN8RFnzniWCddobD9iKZatrvH4 Empfänger: 17SkEw2md5avVNyYgj6RiXuQKNwkXa
c) 10.000,99 BTC | 100,00 USD -> Davon 0,99 BTC Miner FEE
d) Diese Transaktion ist bekannt als die erste real-life Transaktion, wo zwei Pizzas in BTC bezahlt wurden.

Aufgabe 3.6

locktime: Feld in der Transaktion. Bestimmt frühsten Zeitpunkt oder Block, ab dem Transaktion gültig ist. Die Transaktion kann nicht in einen Block aufgenommen werden, bevor der Zeitpunkt oder Block erreicht ist. Dies kann verwendet werden, um Zahlungen zu verzögern oder um sicherzustellen, dass eine Transaktion nur unter bestimmten zeitlichen Bedingungen verarbeitet wird.
CHECKLOCKTIMEVERIFY: Wird in einem Bitcoin-Transaktionsskript verwendet. Ermöglich dem Skript, die Gültigkeit einer Transaktion basierend auf Zeit- oder Blockparameter zu überprüfen. Es muss demnach nicht die gesamte Transaktion bis zu einem Zeitpunkt oder Block gesperrt werden.
Beispiele: Man könnte sicherstellen, dass eine Zahlung erst nach einem bestimmten Datum oder Block erfolgt (bspw. Daueraufträge od. Vertragszahlungen). <- locktime
CHECKLOCKTIMEVERIFY ermöglicht die Erstellung von Bedingungen, die erfüllt sein müssen, bevor Gelder freigegeben werden (Smart Contracts).