Cryptocurrencies (WS2023/24) Dr. Zeta Avarikioti, Prof. Matteo Maffei

Project 6a – Bitcoin Applications

Deadline: 16th January, 2024 at 23:59 TU Wien

Project overview

The aim of this project is to write and interact with applications/smart contracts built on Bitcoin, using Bitcoin script. You will interact with the Bitcoin testnet, an alternative Bitcoin blockchain, where the coins have no value, used mainly for testing. In TUWEL, you can find your configuration file that contains some private keys, pre-images and some unspent transaction outputs (UTXOs). Additionally, there is a solution skeleton that you can use to solve the tasks below. In this configuration file, there are the following data:

- ⟨ALICE_SK⟩: secret key for Alice (Exercise A)
- ⟨BOB_SK⟩: secret key for Bob (Exercise A)
- ⟨CAROL_SK⟩: secret key for Carol (Exercise B)
- ⟨DAVE_SK⟩: secret key for Dave (Exercise C)
- ⟨PUNISH_SECRET⟩: pre-image of a SHA256 hash used for punishing the old payment channel state in Exercise B
- ⟨UNKNOWN_HASH⟩: SHA256 hash of an unkown pre-image (Exercise C)
- ⟨TX_IN_EX1_HASH⟩: The transaction holding one UTXO, which is for your Exercise A
- ⟨TX_IN_EX2_HASH⟩: The transaction holding one UTXO, which is for your Exercise B
- ⟨TX_IN_EX3_HASH⟩: The transaction holding one UTXO, which is for your Exercise C

Every student has received their own set of these values in a feedback file in TUWEL ("Parameters for Task 6a (Payment Channels)") and is supposed to solve the exercises by themselves. Additionally, you are required to write a short report that contains your script and a short explanation why it solves the challenge. This report should be a simple markdown or text file. Upload it as a .md or .txt file on TUWEL.

On TUWEL, we provided you with a Python solution skeleton using the bitcoin-utils library including interaction with the testnet. You are free to use any other programming language/library, but note that if you do, we do not offer support. You can investigate `scripts.py` to view the scripts of each of the given exercise you need to spend along with how scripts are constructed using bitcoin-utils in Python. Be sure to check `https://en.bitcoin.it/wiki/Script` for the exact behavior of the op-codes. In your solution you need to construct a scriptSig that is able to spend the scriptPubKey of the given UTXO (see `https://en.bitcoin.it/wiki/Transaction#Verification`). If you use

the solution skeleton, the main code you want to modify and execute is in `students.py`, the other files are mainly helpers.

To investigate the transactions and track your progress, we recommend that you use a blockchain explorer for the testnet, for example you could use `https://live.blockcypher.com/btc-testnet/` or `https://blockstream.info/testnet/`, which allows you to view details of transactions, scripts, addresses, etc., and also lets you post raw transactions to the testnet manually, should you wish. Each of your UTXOs holds 5000 satoshis, which should be more than enough to pay for fees if you use the values indicated in the exercises. Finally, we consider an exercise solved if you:

1. have spent the UTXO corresponding to that exercise,
2. have given this money to the addresses as indicated in the exercise description and
3. included the script you developed and explained it in your report.

Again, you can check the first two conditions yourself on the testnet explorer.

Exercise A: Payment Channel: honest closure

Your task: You have been provided with a secret key for both Alice ⟨ALICE_SK⟩ and Bob ⟨BOB_SK⟩, as well as an on-chain funding transaction of the channel ⟨TX_IN_EX1_HASH⟩. This funding transaction locks some money in a Multisig address of Alice and Bob. They have used this channel for a while now and wish to close it. In the final state of the channel, Alice has a balance of 1000 satoshis and Bob has 2000 satoshis. Since they both cooperatively wish to close the channel, they create a new transaction that does not have any timelock or revocation mechanism, but simply takes the Multisig from the funding as input and creates two outputs, one for Alice and one for Bob. You need to create the transaction of the final state of Bob, sign it and post it on the Bitcoin testnet. **This exercise is considered solved if your UTXO for exercise A is spent in a transaction with two outputs, Alice owning 1000 and Bob 2000 satoshis, both as P2PKH (see `https://learnmeabitcoin.com/technical/p2pkh`).** In other words, this means that, e.g., for Alice, 1000 satoshis need to be locked in the following script: `OP_DUP OP_HASH160 <Alice_pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`. You can use the attribute `Id.p2pkh` of the helper class `Id` we provide.

Hint: The opcode `OP_CHECKMULTISIG` actually has a bug where it pops one extra element of the stack. Take this into account when spending from the Multisig.

This exercise is worth 2 points.

Bonus question: As described, this final state differs from a regular state, as it has no revocation mechanism, timelock and is not duplicated. Why do we need these things in a regular state and how does the duplication work?

Exercise B: Payment Channel: punish misbehavior

Your task: Carol ⟨CAROL_SK⟩ and Mallory have opened a Lightning-style payment channel. You have been provided with a secret key for Carol, as well as an on-chain commitment transaction ⟨TX_IN_EX2_HASH⟩ of the channel between Carol and Mallory. However, Mallory tried to cheat

and posted a commitment transaction on-chain representing an old state where he has a lot more money than in the most recent state. Since old states are revoked, you know the revocation secret ⟨PUNISH_SECRET⟩. You need to create a punishment transaction taking the balance that belongs to Mallory (which is the output with index 0 of the commitment transaction), giving it to Carol and post it on the testnet. **This exercise is considered solved if your UTXO for exercise B is spent in a transaction with one output, which gives Carol 2000 satoshis as P2PKH. This exercise is worth 3 points.**

Bonus question: To have multi-hop payments in a Payment Channel Network (PCN), payment channels can be used to route HTLC-based (Hash Time-Lock Contract) payments. These HTLCs are additional outputs in a channel, that need to be punished. Following the example above, assume that Mallory has posted an old state that holds one (or more) HTLCs. Now Carol needs to punish the output holding Mallory's balance plus each output holding an HTLC. How can she make this punishment more efficient? Is there a way to decrease the amount of *things* you need to put on-chain?

Exercise C: Exploit faulty script

Dave ⟨DAVE_SK⟩ and Mallory have locked some funds in an output, that looks similar to a commitment transaction ⟨TX_IN_EX3_HASH⟩. However, we placed some "bugs" in this script. This allows you to spend the money locked in this contract to Dave's address even though the timelock is far in the future. **This exercise is considered solved if your UTXO for exercise C is spent in a transaction with one output, which gives Dave 4000 satoshis as P2PKH. This exercise is worth 5 points.**