Cryptocurrencies (WS2021/22) Prof. Matteo Maffei

Project 3 – Bitcoin Applications

Deadline: February 27, 2022 at 23:59 TU Wien

Group 04

Our group consists of the following members:

Tobias Eidelpes, Mehmet Ege Demirsoy, Nejra Komic
01527193, 01641187, 11719704

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. wait for some time. Only after some time, Bob can claim his balance. 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.

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: punishing 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.**

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 way 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.**

```
1  OP_DUP OP_HASH160
2  OP_PUSHBYTES_20 <dave_pk_hash>
3  OP_EQUALVERIFY OP_CHECKSIGVERIFY
4  OP_IF
5    OP_PUSHBYTES_3 fbd42f OP_CLTV
6    OP_DROP OP_DUP OP_HASH160
7    OP_PUSHBYTES_20 <dave_pk_hash>
8  OP_ELSE
9    OP_SHA256
10   OP_PUSHBYTES_32 <hash_lock>
11   OP_EQUAL OP_2DUP OP_HASH160
12   OP_PUSHBYTES_20 <irrelevant_hash>
13   OP_2ROT OP_DUP OP_DUP
14 OP_ENDIF
15 OP_EQUALVERIFY OP_CHECKSIGVERIFY
16 OP_2DROP OP_DROP OP_NOT
```

The script provided to us contains a bug where the `OP_EQUAL` opcode is used but the return value is never checked (line 11). Unlocking the script before the locktime has expired is thus possible without knowing the preimage of the hash in line 10. The value on the stack is hashed with `SHA256` and compared with the hash lock. Execution of the script continues regardless of the outcome of this comparison. If the `OP_EQUAL` opcode is replaced with `OP_EQUALVERIFY`, the script will halt if the comparison fails, restoring intended behavior.

The following unlocking script allows successful spending of the output:

```
<sig_dave>
<id_dave.pk.to_hex()>
OP_0
OP_0
<sig_dave>
<id_dave.pk.to_hex()>
```

Work distribution