**Blockchain Technology**

Lab 02: Understanding Bitcoin’s Scripting Language

* 2TT Team Information
  + 22C11025 – Hoàng Thủy Trúc
  + 22C11046 – Nguyễn Trần Vĩnh Thuyên

Contents

[**Overview 0**](#_heading=h.9bdohr7gu8cy)

[**Project structure 1**](#_heading=h.5wnvwfo4029w)

[**Data structure 1**](#_heading=h.ypofp0ycfeiw)

[1. Bitcoin Address 1](#_heading=h.85l8r9i0hoxc)

[2. Bitcoin Transaction 1](#_heading=h.2ctcs2dfmojr)

[**Task 1: Basic Script Execution 1**](#_heading=h.bn5mse81xgsj)

[**Task 2: Multisignature Transactions 2**](#_heading=h.5ztz6e296dke)

[**Task 3: Analysis and Reflection 3**](#_heading=h.jma36t9tw3g5)

[1. Use cases 4](#_heading=h.umzpz16k5d6s)

[2. Challenges and Lessons 4](#_heading=h.gelwj7br2j61)

[3. Discussion 5](#_heading=h.zdpoovc8d1ud)

[**References 5**](#_heading=h.4p6j2ifl50d8)

# Overview

The lab demonstrates how the Bitcoin network operates by using the Bitcoin Script. The lab has implemented three features in the Bitcoin network:

* Create a Bitcoin address.
* Create a Bitcoin transaction with a basic Bitcoin script.
* Create a Bitcoin transaction with a multi-signature script.

# Project structure

The project is written in Python on the Google Colaboratory service. Its structure only has 1 file Lab02.ipynb which contains code to execute Bitcoin’s features.

Link: Lab02.ipynb

# Data structure

## Bitcoin Address

A Bitcoin address is derived from a public key, which is in turn derived from a private key.

* Private key: is created from a random 32-byte sequence.
* Public key: is derived from the private key.

## Bitcoin Transaction

A Bitcoin transaction has the following structure:

* TxInput: contains the previous unspent output transaction identifier and provides funds for a new transaction.
* TxOutput: specifies the desired destination address and amounts of the transferred funds.
* Script: contains a set of opcodes that perform various operations to determine if a transaction is valid.

# Task 1: Basic Script Execution

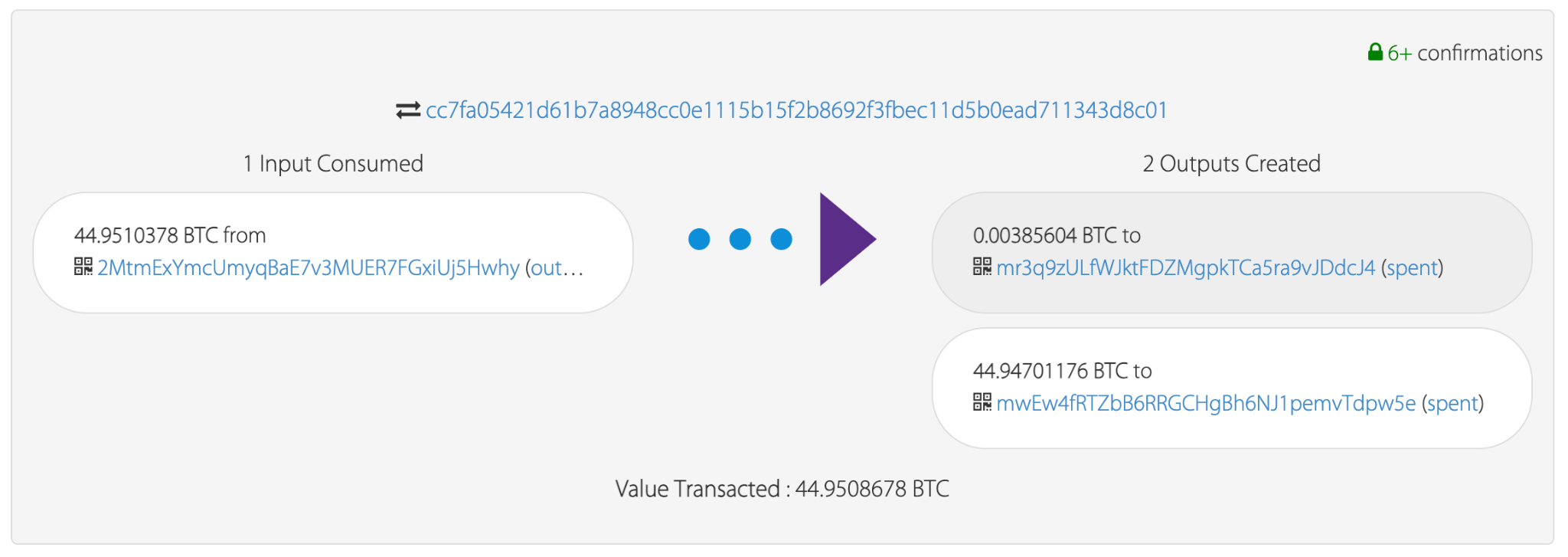
This task is to create a Bitcoin script that locks funds to a P2PKH Bitcoin address and spends the locked funds.

The system will execute three steps:

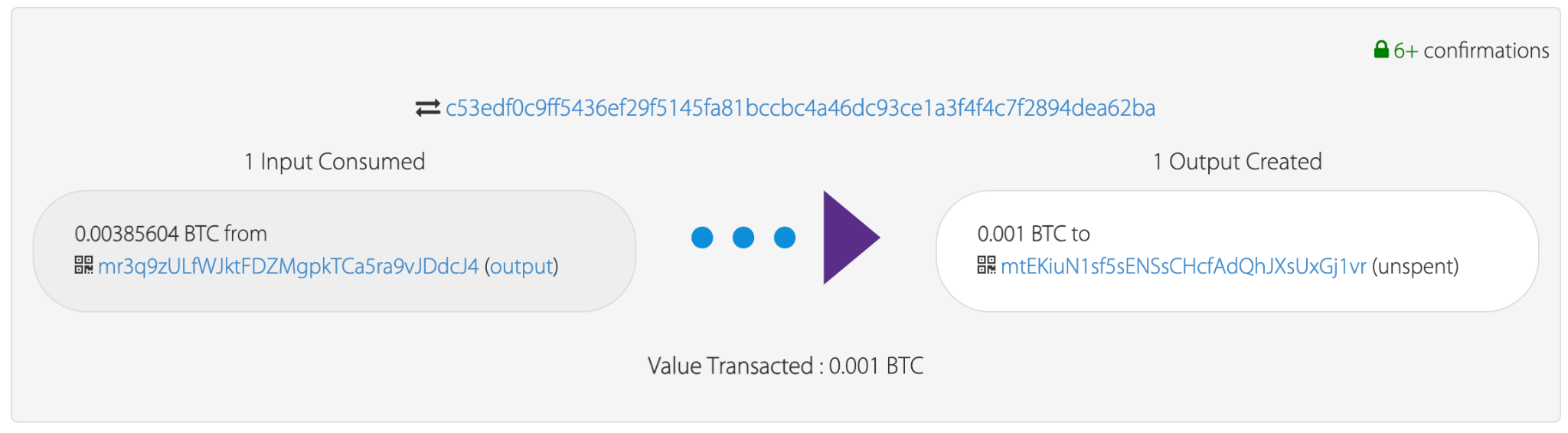
* Step 1: Create 2 Bitcoin testnet addresses that correspond to a sender and a receiver in a transaction.

| Sender: |
| --- |
| Receiver: |

* Step 2: Use a Bitcoin testnet faucet to send testnet BTC to the sender’s address.



* Step 3: Create a new transaction to spend the previous locked transaction above.



# Task 2: Multisignature Transactions

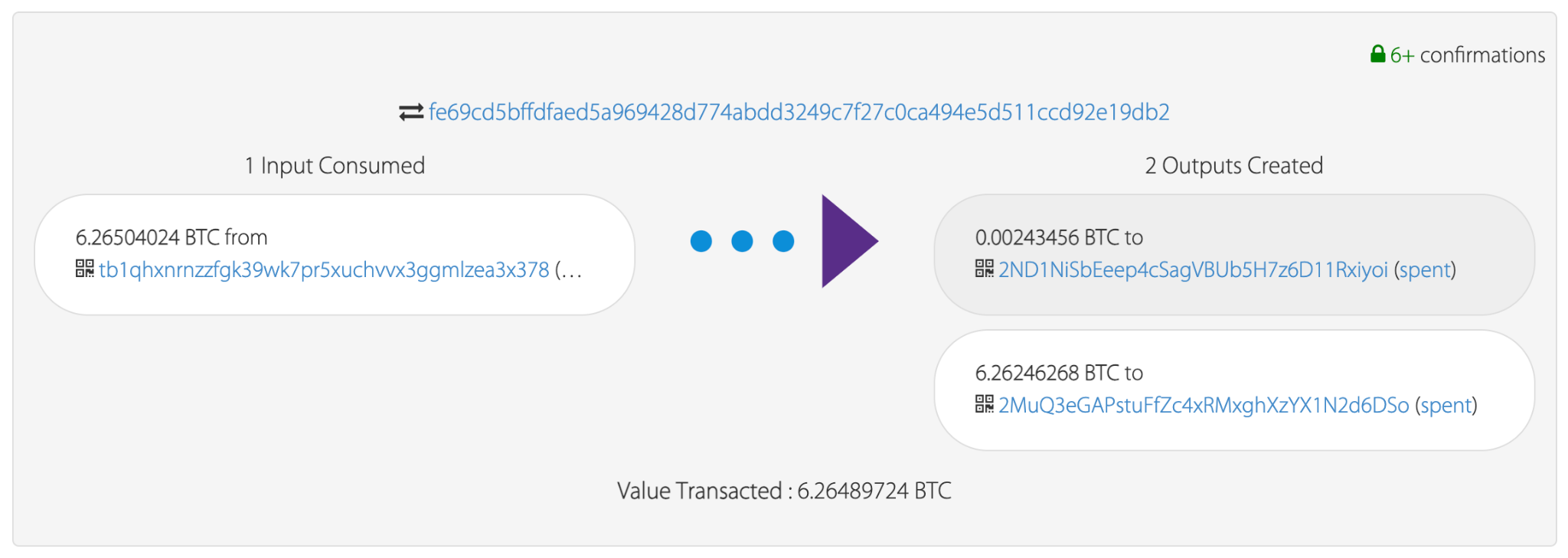
This task is to create a 2-of-2 multisig script that locks funds to a P2SH Bitcoin address and spends the locked funds.

The system will execute three steps:

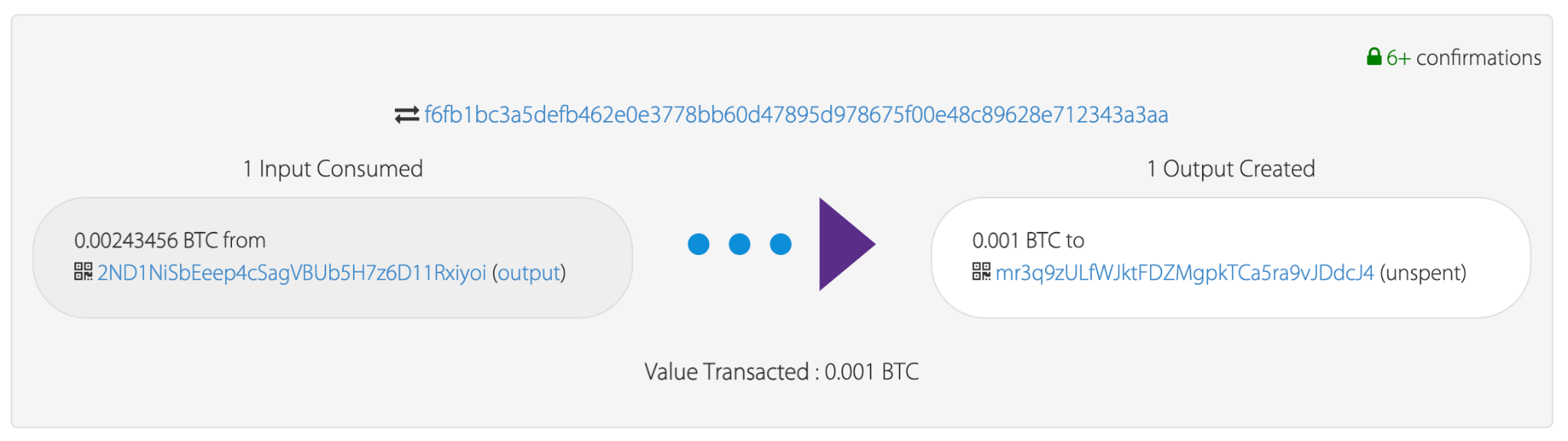
* Step 1: Create a P2SH Bitcoin testnet address as the sender to send funds to the sender’s address in task 1.

| Sender: |
| --- |
| Receiver: |

* Step 2: Use a Bitcoin testnet faucet to send testnet BTC to the sender’s address.



* Step 3: Create a new transaction to spend the previous locked transaction above.



# Task 3: Analysis and Reflection

In this task, we will discuss some use cases of the scripts we have created, experiences gained, advantages, and limitations of Bitcoin Script.

## Use cases

Let’s talk about the two scripts we have created:

| **“OP\_DUP”, “OP\_HASH160”, to\_addr.to\_hash160(), “OP\_EQUAlVERIFY”, “OP\_CHECKSIG”** |
| --- |

This script is a locking funds script in the output transaction. The recipient’s address is hashed and then it’s embedded in the locking script. To spend funds from this output, **the spender must provide a signature and a public key that, when hashed, matches the hash in the locking script.**

This context can be used in some use cases:

* **Crypto wallet**: provides Bitcoin transfer service between the sender and the recipient.
* **Online payment**: allows the seller and the buyer to exchange goods directly without the third party as the intermediate.

| **“OP\_2”, public\_key1.to\_hex(), public\_key2.to\_hex(), “OP\_2”, “OP\_CHECKMULTISIG”** |
| --- |

This script implements a 2-of-2 multisignature requirement. **To spend funds locked with this script, the spender needs to provide signatures corresponding to both public\_key1 and public\_key2.** The "OP\_CHECKMULTISIG" operation checks whether at least 2 of the provided signatures are valid for the corresponding public keys, and if so, the script is considered satisfied.

This context can be used in some use cases:

* **Smart contract**: allows the creation of transactions where funds are released only if certain conditions are met.
* **Escrow services**: allows the creation of transactions where a third party holds funds until certain conditions are met.

## Challenges and Lessons

* There are many libraries for learning how the Bitcoin network operates, so we have tried all of them to create executable Bitcoin scripts.
* The multisign script has thrown some validation errors, so we have searched for the Bitcoin script blogs to learn how it works.
* Some Bitcoin testnet faucet services don’t work in Vietnam, such as Bitpay. Each request to get coins has some limits (1 request per 12 hours, 1 request per IP).
* In a transaction, if we don’t spend all the funds, the remaining funds will be transferred to the miner by default.

## Discussion

**The advantages of the Bitcoin script:**

* **Security & Privacy**: Each transaction is only spent when the spender provides a signature that matches the public key embedded in that transaction. This ensures a high level of confidence in the transaction details, especially when compared to centralized intermediate systems.
* **Flexibility in signature requirement:** The script allows customization of the required number of signatures, providing flexibility in setting up multisignature conditions with different signing requirements.

**The limitations of the Bitcoin script:**

* **Limited data storage**: Bitcoin Script has limited data storage capabilities. The amount of data that can be stored in a Bitcoin transaction is constrained, which makes it unsuitable for certain applications that require extensive data storage.
* **Limited operator**: Bitcoin Script does not support looping or iteration structures. This limitation makes it challenging to create more complex and dynamic smart contracts that require repetitive execution of code.

# References

* Bitcoin library: <https://github.com/karask/python-bitcoin-utils/tree/master>
* Bitcoin testnet faucet: <https://coinfaucet.eu/en/btc-testnet/>
* Bitcoin transactions tracker: ​​<https://live.blockcypher.com/btc-testnet/>