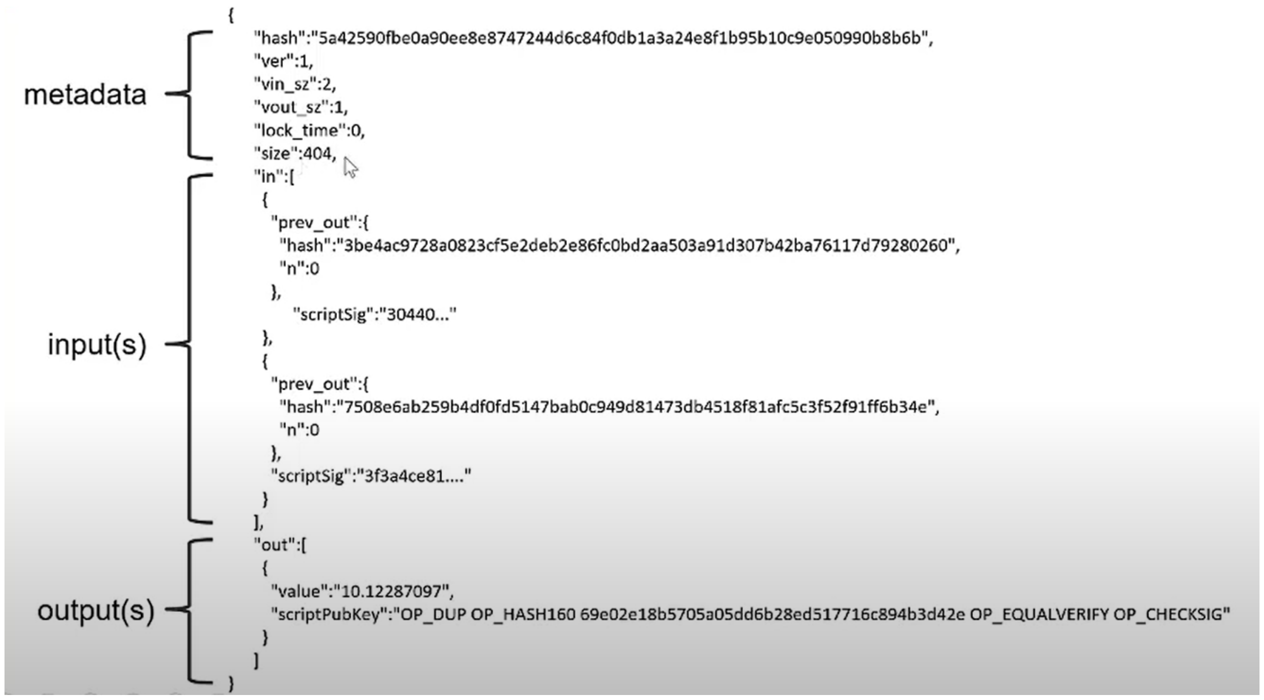
COMP-4541 – HomeWork 2

Farshid Adavi - 4021305965007

Exercise 1



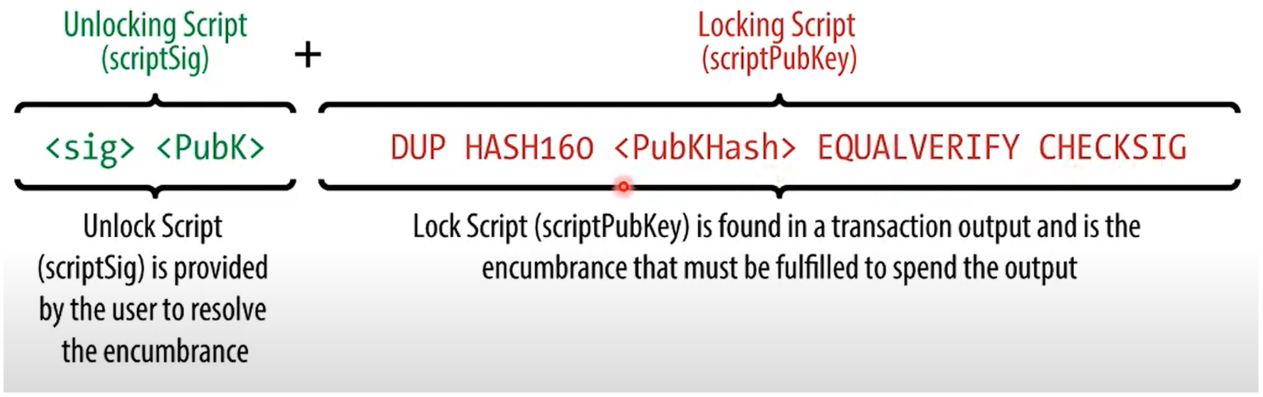
```
{
    "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
    "ver":1,
    "vin_sz":2,
    "vout_sz":1,
    "lock_time":0,
    "size":404,
    "in":[
     {
      "prev_out":{
       "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
       "n":0
      },
         "scriptSig":"30440..."
     },
     {
      "prev_out":{
       "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
       "n":0
      },
      "scriptSig":"3f3a4ce81...."
     }
    ],
    "out":[
     {
      "value":"10.12287097",
      "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
     }
    ]
}
```

The image above shows us a real Bitcoin transaction. Each transaction has three main parts.

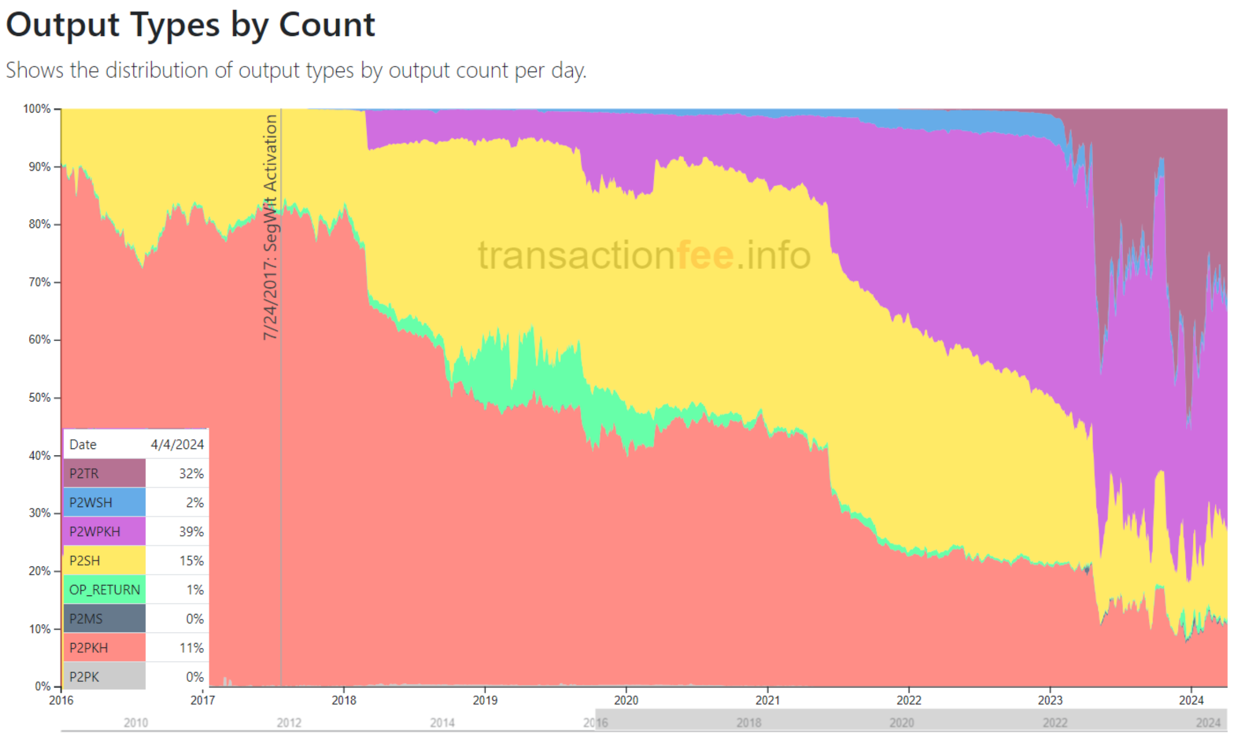
- metadata:
  This part contains information such as the hash of the transaction, which is used later in subsequent transactions to address this transaction, the number of inputs and outputs of this transaction, lock time, which can be set with a time stamp or block number, which means this transaction, until this The block or this time cannot be cashed and the size of the block is reduced.

- input(s)
  Each input contains the address of a previous transaction and the desired output index of that transaction. Also includes signature script.

- output(s)
  Each output contains an amount of Bitcoin and also contains the public key script.

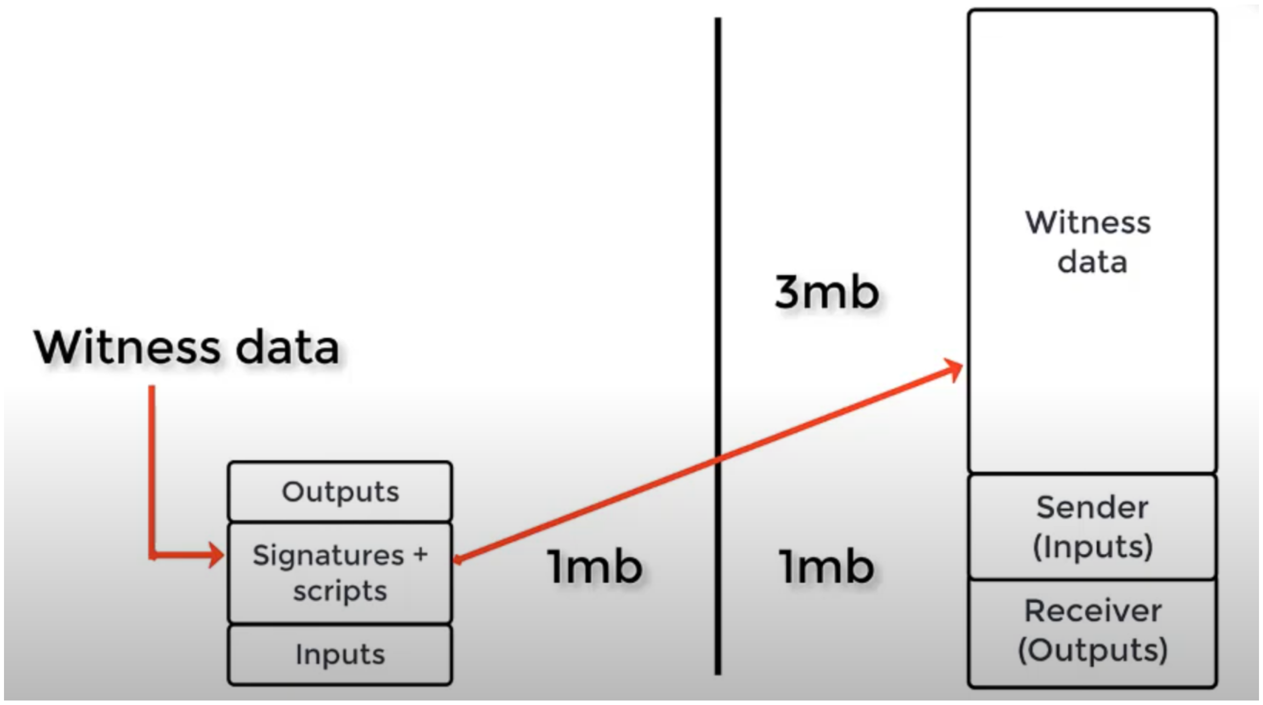

Unlocking Script (scriptSig)
+
Locking Script (scriptPubKey)
<sig> <PubK>
DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
Unlock Script (scriptSig) is provided by the user to resolve the encumbrance
Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

With the scriptsig that is present in the input, we spend an output from the previous transaction, and this is an example for a P2PKH transaction that transferred a bit of bitcoin to the hash of the public key. This is a Bitcoin script that is a stack based language.

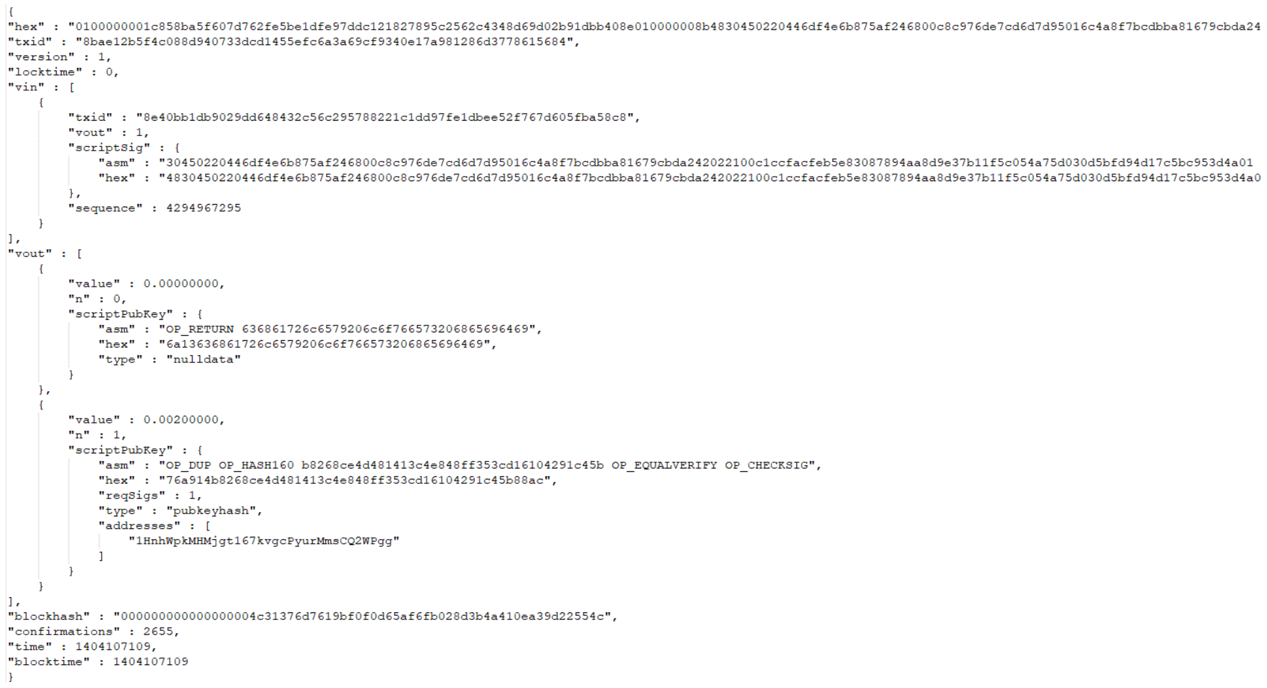


Date	4/4/2024
P2TR	32%
P2WSH	2%
P2WPKH	39%
P2SH	15%
OP_RETURN	1%
P2MS	0%
P2PKH	11%
P2PK	0%

We can see that the transfer of Bitcoin to the public key has reached zero and is logical because our blog space is limited to one megabyte and the transfer to the public key hash occupies less space. Also, in the case of multi-key transfers, we see the replacement of the transfer to hash script. We will also examine other methods such as Segwit and Op-Return.



Segregated Witness (SegWit) refers to a change in Bitcoin's transaction format where the witness information was removed from the input field of the block.
The stated purpose of Segregated Witness is to prevent non-intentional Bitcoin transaction malleability and allow for more transactions to be stored within a block.
SegWit was also intended to solve a blockchain size limitation problem that reduced Bitcoin transaction speed.



```
{
"hex" : "0100000001c858ba5f607d762fe5be1dfe97ddc121827895c2562c4348d69d02b91dbb408e010000008b4830450220446df4e6b875af246800c8c976de7cd6d7d95016c4a8f7bcdbba81679cbda24
"txid" : "8bae12b5f4c088d940733dcd1455efc6a3a69cf9340e17a981286d3778615684",
"version" : 1,
"locktime" : 0,
"vin" : [
    {
        "txid" : "8e40bb1db9029dd648432c56c295788221c1dd97fe1dbee52f767d605fba58c8",
        "vout" : 1,
        "scriptSig" : {
            "asm" : "30450220446df4e6b875af246800c8c976de7cd6d7d95016c4a8f7bcdbba81679cbda242022100c1ccfacfeb5e83087894aa8d9e37b11f5c054a75d030d5bfd94d17c5bc953d4a01
            "hex" : "4830450220446df4e6b875af246800c8c976de7cd6d7d95016c4a8f7bcdbba81679cbda242022100c1ccfacfeb5e83087894aa8d9e37b11f5c054a75d030d5bfd94d17c5bc953d4a0
        },
        "sequence" : 4294967295
    }
],
"vout" : [
    {
        "value" : 0.00000000,
        "n" : 0,
        "scriptPubKey" : {
            "asm" : "OP_RETURN 636861726c6579206c6f766573206865696469",
            "hex" : "6a13636861726c6579206c6f766573206865696469",
            "type" : "nulldata"
        }
    },
    {
        "value" : 0.00200000,
        "n" : 1,
        "scriptPubKey" : {
            "asm" : "OP_DUP OP_HASH160 b8268ce4d481413c4e848ff353cd16104291c45b OP_EQUALVERIFY OP_CHECKSIG",
            "hex" : "76a914b8268ce4d481413c4e848ff353cd16104291c45b88ac",
            "reqSigs" : 1,
            "type" : "pubkeyhash",
            "addresses" : [
                "1HnhWpkMHMjgt167kvgcPyurMmsCQ2WPgg"
            ]
        }
    }
],
"blockhash" : "000000000000000004c31376d7619bf0f0d65af6fb028d3b4a410ea39d22554c",
"confirmations" : 2655,
"time" : 1404107109,
"blocktime" : 1404107109
}
```

Op-Return: OP RETURN is an instruction of the Bitcoin scripting language which allows users to attach metadata to a transaction and therefore save it on the blockchain. There can only exist one OP-RETURN per transaction. The operation stores them in the blockchain, but they are not UTXOs (unspent output from bitcoin transactions). An output using an OP-RETURN is provably unspendable, so we don't need to lock any BTC in this UTXO. However, we still need to pay the miner fees, so we will spend BTC from a previous P2WPKH UTXO, create one nulldata UTXO and an other one for the change, leaving the difference as mining fees.

(1) Alice can sign a message with the private key of an address that holds at least 100 BTC.
The message could include Bob's name, the current timestamp, and a nonce (a number that can only be used once, to prevent replay attacks).
Alice then sends a tiny amount of BTC (100 BTC remain available) to a new address that she also controls and includes the signed message in the OP-RETURN field of the transaction, which can hold arbitrary data.
Bob can then verify:

The message's signature using Alice's public address.
That the address contained at least 100 BTC before the transaction and that the coins have not been spent.
That the message is current and specifically for him, thus not replayable.

(2) Alice can creating a signed message stating her claim of owning 100 BTC. Providing Bob with the transaction IDs (TXIDs) of her previous transactions that aggregate to at least 100 BTC. Including Bob's name, a nonce and current timestamp in the signed message to prevent reuse.

Bob canconfirm that the addresses linked to the provided TXIDs indeed contain the 100 BTC and that the coins have not been spent. He can then verify the signature against the addresses holding the coins using Alice's public key.

(3) Alice can send the 100 BTC to a "burn address." This is address can created by hashing a clear message multiple times until it looks like a Bitcoin address (e.g., starting with "1" as a P2PKH pattern that doesn't correspond to any possible private key). After broadcasting the transaction to this burn address, Alice can show Bob the transaction. Because the private key doesn't exist, the BTC can't be spent, effectively "burning" them.
We know that it is not going to find the equivalent public and private key of this hash.
Collision resistant property tells us that it is not possible to find a public key whose hash is equal to this hash.
And for example, if it was possible, it would be impossible to guess the private key for that public key.

(4) When Alice provides proof to Bob using her public address, Bob can look up all past and future transactions linked to that address on the blockchain. This could expose Alice's financial history and her future transactions.
Alice can transfer exactly 100 BTC to a new address where no other transactions have occurred. She can then sign a message with the private key of this new address. Bob can verify the message and check the balance of this address on the blockchain. Since the address is new, it has no history.
Of course, it is true that this account has no history, but still Bob can easily access future transactions and even see from which account this money came to this address and check the history of that account.
Defense Mechanisms:

Coin Mixing:
Before the proof, Alice could use a coin mixing service to break the link between her past transactions and the 100 BTC.
Confidential Transactions:
If supported by the blockchain, Alice could engage in confidential transactions that obscure the amount transferred.

references links: en.bitcoin.it/wiki/Script
transactionfee.info
bitcoin.stackexchange.com
bitcoindev.network
en.bitcoin.it/wiki/Script

Exercise 2

The scenario described in the question is known as a ”block blocking attack” or ”sabotage attack” in the context of a mining pool. In such an attack, a malicious miner who is part of a mining pool finds a valid Proof of Work (PoW) for a new block but does not share this solution with the pool. This undermines the pool’s total revenue, as it prevents it from receiving the block rewards and transaction fees it could otherwise earn.

- As a miner who controls 1% of the network’s hash power, I join a large mining pool that controls 19% of the network’s hash power. And now it controls 20% of the hash power of the network.

- Then I start normally, and send partial PoW solutions to the pool to build trust.
- When I successfully mine a new block, I discard it instead of sending it to the pool.
  The pool does not receive block rewards and transaction fees, and as a result, its income is reduced.
- I can’t use this block anywhere else because it contains a transaction equal to the block reward and the transaction fee to the public key of the pool, and if I change it, the generated hash will change and it is no longer a valid PoW solution, if the address of the pool account before Finding a valid PoW solution was changing and I could no longer submit partial PoW solutions to the pool. So I have to throw this dear 500K$ block in the trash.
  Of course, I can create a fork with this block I have at a suitable time, for example, when a suitable number of new blocks are added to the blockchain. Although this is ignored, it can make this pool look bad to its members, and also the members notice the malicious people in the pool and leave it. (For example, we are a competitor’s pool and we want to attract them) and also In the future, new people will not join this pool.

- Now the total income of the pool is 19% of the blocks, which gives me one-twentieth of it, which loses exactly the same amount; Normally, I could have an income of 1% of the blocks, but now I have an income of (one twentieth * 0.19) of the blocks and I lost in total, but the damage I did to the pool is 19 times bigger.
  The pool's overall revenue is then reduced because it finds fewer blocks than it should statistically with its hash power.

- Mechanisms Used by Mining Pools:
  Share Submission:
  Mining pools typically require miners to submit shares as proof of work. Shares are partial solutions to the PoW problem that aren't sufficient to solve a block but indicate that the miner is working.
  Payout Schemes:

  (1) Pay-per share (PPS): Allows instant payout solely based on accepted shares contributed by the pool member, who are allowed to withdraw their earnings instantly from the pool's existing balance.

  (2) Proportional (PROP): At the end of a mining round, a reward that is proportional to the number of the member's shares with respect to the total shares in the pool is offered.
  (3) Shared Maximum Pay Per Share (SMPPS): A method similar to PPS but limits the payout to the maximum that the pool has earned.
  (4) Equalized Shared Maximum Pay Per Share (ESMPPS): A method similar to SMPPS, but distributes payments equally among all miners in the bitcoin mining pool.

- Defense against attack:

  (1) Security Measures: Some pools have security systems that analyze the pattern of shares submission to detect anomalies.

  (2) Analysis of Mining Success: Pools may track the expected versus actual blocks found. Significant discrepancies could indicate block withholding.

  (3) Changing Payout Schemes: Pools could change their payout schemes to ones less susceptible to attack, such as PPLNS.

references links:
bitcoin.stackexchange.com
www.sciencedirect.com
www.investopedia.com

Exercise 3

This is a classic scenario for an atomic swap, where two parties want to exchange cryptocurrencies on different blockchains without a trusted third party. Atomic swaps use a type of smart contract known as a Hashed Timelock Contract (HTLC), which ensures that either both parties fulfill the swap, or neither does.

(1) My Trustless Exchange Protocol:

- Let's assume that Bob and Alice have already agreed on things such as the date and amount, etc.
  Bob randomly chooses a large nonce and sends $hash(nonce)$ to Alice.
  Then he send a transaction that can be spent in two ways on the blockchain.

  This transaction type is P2SH, And we deposit bitcoins to the address that is the hash of a redeem script.
  This script can be unlocked in two ways.
  First: there must be Bob's signature and the block number has exceeded a specific number.
  Second: There must be Alice's signature and nonce that Bob created.

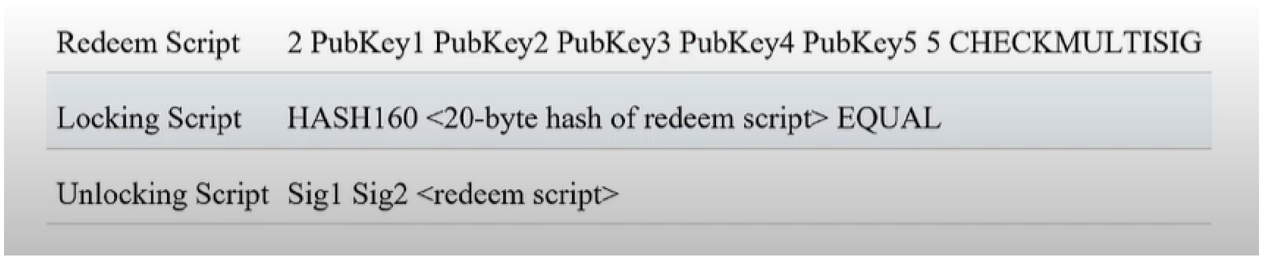
- Let's assume that Bob has set the block number to the current block number plus 16 (average of 4 hours) plus 384 (4 days).
  Allow 4 hours for the time Bob waits for the transaction to be posted on the Bitcoin blockchain.
  After 4 hours, let's assume that the transaction was placed on the network.
  Now Bob sends the transaction to Alice.

- Now Alice puts a smart contract on the Ethereum network and deposits 20 Ethereum to it.
  If Bob sends the desired Nonce for more than three days, he will receive the ethers.
  And Alice now has the desired Nonce and has at least one day to spend the previous transaction on the Bitcoin network.
  And if Bob doesn't send Nonce in three days, Alice will pick up the ethers himself on the 5th day.
  Also, if Alice does not cooperate, Bob will not give Nonce and spend the transaction on the Bitcoin network after four days.

(2) Implementation:

```
Redeem Script: OP_IF
// Part for Alice to redeem the funds
OP_DUP OP_HASH160 <hash(x)> OP_EQUALVERIFY
OP_CHECKSIG
OP_ELSE
// Part for Bob to redeem after time t
< t > OP_CHECKLOCKTIMEVERIFY OP_DROP
OP_CHECKSIG
```

OP_ENDIF
Unlocking Script:
For Alice: <signature_Alice> <public_key_Alice> $< x >$
For Bob: <signature_Bob> <public_key_Bob>



Redeem Script	2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 CHECKMULTISIG
Locking Script	HASH160 <20-byte hash of redeem script> EQUAL
Unlocking Script	Sig1 Sig2 <redeem script>

references links:
What is an atomic swap? let's look at www.bitpanda.com.
www.sciencedirect.com
www.investopedia.com