Development of a payment channel over the Bitcoin network

Final degree project

David Lozano Jarque <bitcoin@davidlj95.com>

5th July 2017

Outline

1 Introduction
- What is Bitcoin
- How does Bitcoin work?
- The scalability problem

2 Bitcoin & Smart Contracts
- Transactions at low-level detail
- Bitcoin's scripting language
- What is a payment channel?

3 Unidirectional payment channels
- Scheme
- Implementation

4 Bidirectional payment channels
- Scheme
- Implementation
- Problem: channel reseting

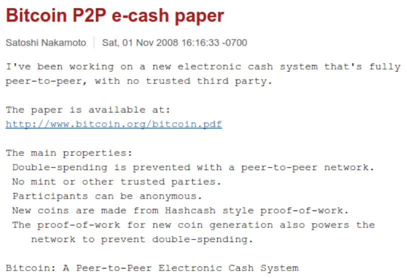
5 The Bitcoin framework

6 Conclusions

Bitcoin's appearance

The creator

Satoshi Nakamoto @ Cryptography (`metzdowd.com`)
November 1st, 2008


Bitcoin P2P e-cash paper

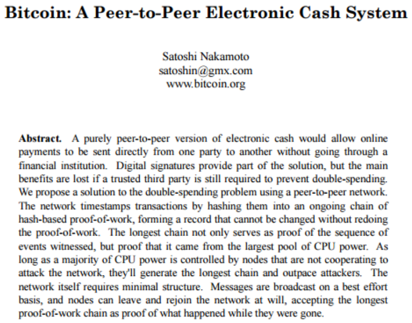
Satoshi Nakamoto | Sat, 01 Nov 2008 16:16:33 -0700

```
I've been working on a new electronic cash system that's fully
peer-to-peer, with no trusted third party.

The paper is available at:
http://www.bitcoin.org/bitcoin.pdf

The main properties:
 Double-spending is prevented with a peer-to-peer network.
 No mint or other trusted parties.
 Participants can be anonymous.
 New coins are made from Hashcash style proof-of-work.
 The proof-of-work for new coin generation also powers the
    network to prevent double-spending.

Bitcoin: A Peer-to-Peer Electronic Cash System
```


Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Features

- Public ledger of transactions

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Features

- Public ledger of transactions
- Public ledger using *blockchain* technology

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Features

- Public ledger of transactions
- Public ledger using *blockchain* technology
- Consensus via *proof-of-work* algorithm

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Features

- Public ledger of transactions
- Public ledger using *blockchain* technology
- Consensus via *proof-of-work* algorithm
- Cryptography-enforced (digital ECDSA signatures & hash functions)

Bitcoin's definition

Definition of Bitcoin

P2P network that allows payments between users without a trusted third party

Features

- Public ledger of transactions
- Public ledger using *blockchain* technology
- Consensus via *proof-of-work* algorithm
- Cryptography-enforced (digital ECDSA signatures & hash functions)
- No trusted 3rd party (Pure P2P)

How do we move currency?

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`
- `inputs`

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`
- `inputs`
- `outputs`

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`
- `inputs`
- `outputs`
- `locktime`

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`
- `inputs`
- `outputs`
- `locktime`

Transactions

What is a Bitcoin transaction?

Message specifying the transfer of currency units (called *bitcoins*)

Transaction fields

A transaction moves currency units given an input to a new output

- `version`
- `inputs`
- `outputs`
- `locktime`

Basic Bitcoin transaction

version	**inputs**	**outputs**	**locktime**

version	***Alice***	***Bob***	**locktime**

Where do we store transactions?

Blocks

What is a Bitcoin block?

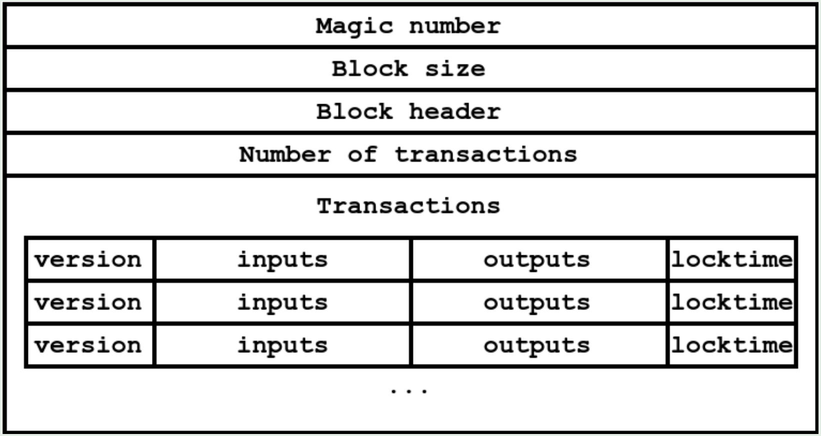
Collection of transactions

Blocks

What is a Bitcoin block?

Collection of transactions

Basic Bitcoin block



Magic number

Block size

Block header

Number of transactions

Transactions

version	inputs	outputs	locktime
version	inputs	outputs	locktime
version	inputs	outputs	locktime

...

Where do we store blocks?

Blockchain

Bitcoin's *blockchain*

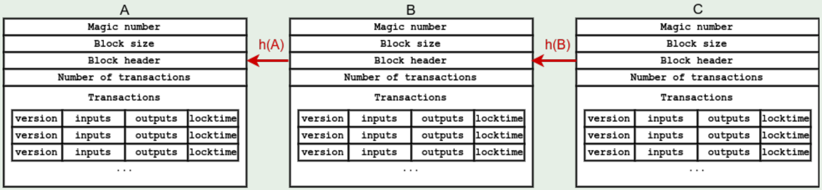
Distributed and replicated database containing a collection of blocks, each one linked to the previous one using **their hashes** forming **a chain**

Blockchain

Bitcoin's *blockchain*

Distributed and replicated database containing a collection of blocks, each one linked to the previous one using **their hashes** forming **a chain**

Basic Bitcoin's *blockchain*



Blockchain

Rewards

Appending a new block to the chain is rewarded with **newly generated currency units** with a *no-input* transaction called a **generation transaction**

Who decides who can create next block?

Consensus

Proof-of-work

Piece of data difficult to generate but easy to verify it meets certain requirements

Consensus

Proof-of-work

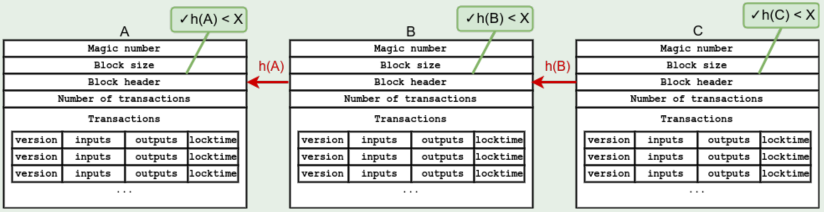
Piece of data difficult to generate but easy to verify it meets certain requirements

Bitcoin's *proof-of-work*

Field in block's header must contain a hash of the block itself whose value **is less than a dynamically adjusted value**

Proof-of-work

Basic Bitcoin's *blockchain* + *proof-of-work*


✓h(A) < X
A
Magic number
Block size
Block header
Number of transactions
Transactions
version inputs outputs locktime
version inputs outputs locktime
version inputs outputs locktime
...
h(A)
✓h(B) < X
B
Magic number
Block size
Block header
Number of transactions
Transactions
version inputs outputs locktime
version inputs outputs locktime
version inputs outputs locktime
...
h(B)
✓h(C) < X
C
Magic number
Block size
Block header
Number of transactions
Transactions
version inputs outputs locktime
version inputs outputs locktime
version inputs outputs locktime
...

How to handle everything?

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)
2. Stores and shares the *blockchain*

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)
2. Stores and shares the *blockchain*
3. Handles keys and creates payment transactions

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)
2. Stores and shares the *blockchain*
3. Handles keys and creates payment transactions

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)
2. Stores and shares the *blockchain*
3. Handles keys and creates payment transactions

*Feature (2) just in **full-nodes**

The Bitcoin client

A Bitcoin client

Software that allows to operate on the Bitcoin network, handling all data structures and network messages

Features

1. Receive and broadcasts messages (transactions, blocks, ...)
2. Stores and shares the *blockchain*
3. Handles keys and creates payment transactions

*Feature (2) just in **full-nodes**

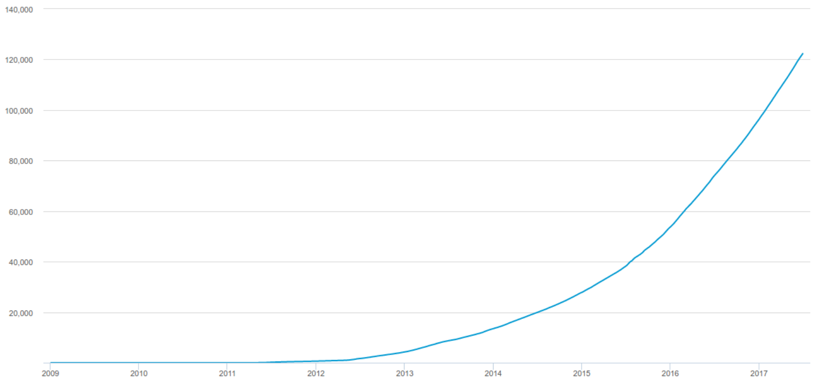
Most used client

Bitcoin Core (`bitcoin.org`) is the most used Bitcoin client (**85%** of nodes in the network)

What is the limit of the technology?

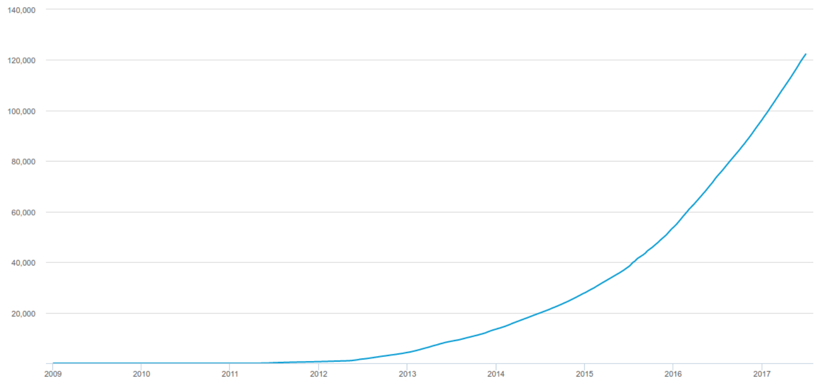
Blockchain size

Blockchain size over time (KBytes/years)



Blockchain size

Blockchain size over time (KBytes/years)



Current blockchain size is approximately **120GB**

Blockchain size

Increasing transaction demand

As Bitcoin becomes more popular, more users arrive therefore more transactions need to be processed

Transaction throughput

Throughput limits

Because of the protocol, blocks must

Transaction throughput

Throughput limits

Because of the protocol, blocks must

1. **Appear every 10 minutes** (approximately) due to *proof-of-work* difficulty adjustment

Transaction throughput

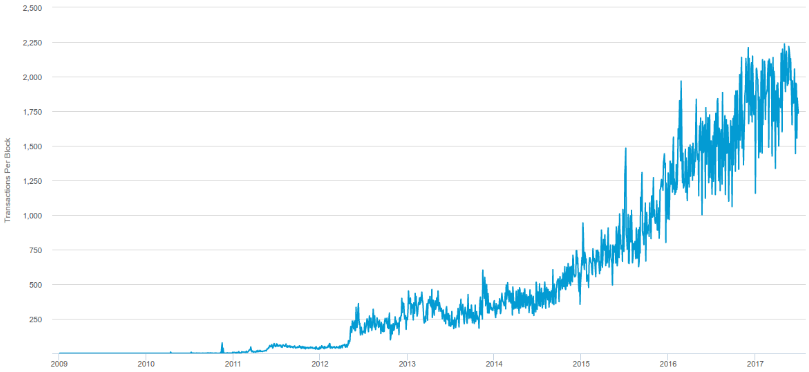
Throughput limits

Because of the protocol, blocks must

1. **Appear every 10 minutes** (approximately) due to *proof-of-work* difficulty adjustment
2. **1MB maximum block size** to control the *blockchain* growth rate

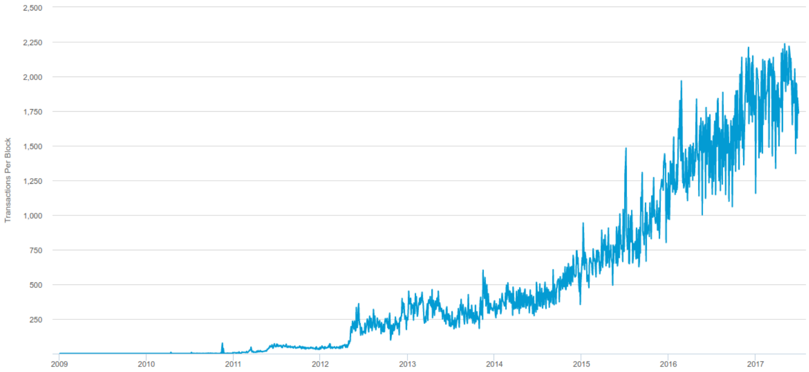
Transaction throughput

Transactions per block over time (tx amount/years)



Transaction throughput

Transactions per block over time (tx amount/years)



Approximately **2.000** transactions per block

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000 \;\; tx}{1 \;\; block} \times$$

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000 \ tx}{1 \ block} \times \frac{1 \ block}{10 \ minutes} \times$$

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000\ tx}{1\ block} \times \frac{1\ block}{10\ minutes} \times \frac{1\ minute}{60\ sec.} \approx$$

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000\ tx}{1\ block} \times \frac{1\ block}{10\ minutes} \times \frac{1\ minute}{60\ sec.} \approx$$

3 transactions per second

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000\ tx}{1\ block} \times \frac{1\ block}{10\ minutes} \times \frac{1\ minute}{60\ sec.} \approx$$

3 transactions per second

VISA's transaction throughput

According to an IBM's studio performed in August of 2010:

Transaction throughput

Bitcoin's transaction throughput

Using previous information:

$$\frac{2.000\ tx}{1\ block} \times \frac{1\ block}{10\ minutes} \times \frac{1\ minute}{60\ sec.} \approx$$

3 transactions per second

VISA's transaction throughput

According to an IBM's studio performed in August of 2010:

24.000 transactions per second

What can we do?

Scalability solutions

Several solutions have been proposed:

Scalability solutions

Several solutions have been proposed:

1. **Increase block size**: Bitcoin Unlimited (1 to 8 MB)

Scalability solutions

Several solutions have been proposed:

1. **Increase block size**: Bitcoin Unlimited (1 to 8 MB)
2. **Reduce transaction size**: `SegWit.co` (do not store transaction signatures, also fixes malleability issues)

Scalability solutions

Several solutions have been proposed:

1. **Increase block size**: Bitcoin Unlimited (1 to 8 MB)
2. **Reduce transaction size**: `SegWit.co` (do not store transaction signatures, also fixes malleability issues)
3. **Decrease the demand of transactions**: Payment channels

Outline

1 Introduction
- What is Bitcoin
- How does Bitcoin work?
- The scalability problem

2 Bitcoin & Smart Contracts
- Transactions at low-level detail
- Bitcoin's scripting language
- What is a payment channel?

3 Unidirectional payment channels
- Scheme
- Implementation

4 Bidirectional payment channels
- Scheme
- Implementation
- Problem: channel reseting

5 The Bitcoin framework

6 Conclusions

Transactions

Transaction fields

Fields of a transaction are:

- `version`

Transactions

Transaction fields

Fields of a transaction are:

- `version`
- `inputs`

Transactions

Transaction fields

Fields of a transaction are:

- `version`
- `inputs`
- `outputs`

Transactions

Transaction fields

Fields of a transaction are:

- `version`
- `inputs`
- `outputs`
- `locktime`

Transactions

Transaction fields

Fields of a transaction are:

- `version`
- `inputs`
- `outputs`
- `locktime`

Basic Bitcoin transaction

`version`	`inputs`	`outputs`	`locktime`

`version`	*Alice*	*Bob*	`locktime`

Transactions

Extra "fields"

All transactions have an id (also called *txId*), that is the double SHA-256 hash of the transaction bytes

How are inputs and outputs specified?

Inputs specification

Input fields

An input consists of the following fields:

Inputs specification

Input fields

An input consists of the following fields:

1. **previousOutput***: An output to be spent (combination of a *txId* and output number)

Inputs specification

Input fields

An input consists of the following fields:

1. **previousOutput***: An output to be spent (combination of a *txId* and output number)
2. **scriptSig**: Script necessary to authorize the output spend

Inputs specification

Input fields

An input consists of the following fields:

1. **previousOutput***: An output to be spent (combination of a *txId* and output number)
2. **scriptSig**: Script necessary to authorize the output spend
3. **sequence**: Number of the transaction in order to enable replacements

Inputs specification

Input fields

An input consists of the following fields:

1. **previousOutput***: An output to be spent (combination of a *txId* and output number)
2. **scriptSig**: Script necessary to authorize the output spend
3. **sequence**: Number of the transaction in order to enable replacements

Inputs specification

Input fields

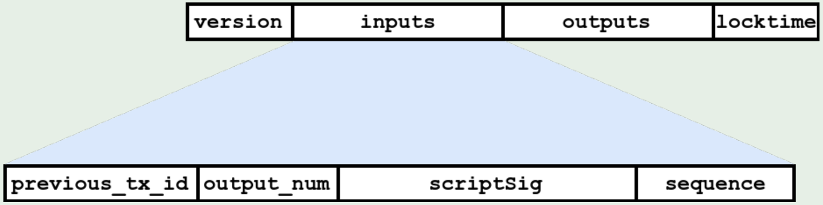
An input consists of the following fields:

1. **previousOutput***: An output to be spent (combination of a *txId* and output number)
2. **scriptSig**: Script necessary to authorize the output spend
3. **sequence**: Number of the transaction in order to enable replacements

* output must not be spent by any other transaction (also called UTXO)

Inputs specification

Basic transaction's input's fields



Outputs specification

Output fields

An output consists of the following fields:

Outputs specification

Output fields

An output consists of the following fields:

1. **value**: number of currency units to be sent to the output

Outputs specification

Output fields

An output consists of the following fields:

1. **value**: number of currency units to be sent to the output
2. **scriptPubKey**: Script specificating the conditions for the output to be spent

Outputs specification

Output fields

An output consists of the following fields:

1. **value**: number of currency units to be sent to the output
2. **scriptPubKey**: Script specificating the conditions for the output to be spent

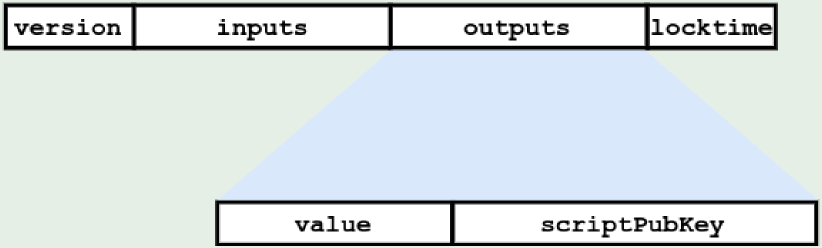
Outputs specification

Output fields

An output consists of the following fields:

1. **value**: number of currency units to be sent to the output
2. **scriptPubKey**: Script specificating the conditions for the output to be spent

Basic transaction's output's fields



How do the scripts work?

Bitcoin's scripting

Bitcoin scripting language

Specificic scripting language for Bitcoin protocol (in transactions)

- Simple

Bitcoin's scripting

Bitcoin scripting language

Specificic scripting language for Bitcoin protocol (in transactions)

- Simple
- Stack-based (processed from left to right)

Bitcoin's scripting

Bitcoin scripting language

Specificic scripting language for Bitcoin protocol (in transactions)

- Simple
- Stack-based (processed from left to right)
- Purposefully not Turing-complete (with no loops)

Bitcoin's scripting

Bitcoin scripting language

Specificic scripting language for Bitcoin protocol (in transactions)

- Simple
- Stack-based (processed from left to right)
- Purposefully not Turing-complete (with no loops)

Bitcoin's scripting

Bitcoin scripting language

Specificic scripting language for Bitcoin protocol (in transactions)

- Simple
- Stack-based (processed from left to right)
- Purposefully not Turing-complete (with no loops)

Technically

Sequentially read 1-byte opcodes that can perform arithmetical operations, store data into the stack, cryptographic operations and some logic and flow control operations

Transactions and scripts

Transactions validity

In order for a transaction to be valid it must:

1. **Valid inputs**: Inputs must refer to existing and non-spent outputs (UTXO)

Transactions and scripts

Transactions validity

In order for a transaction to be valid it must:

1. **Valid inputs**: Inputs must refer to existing and non-spent outputs (UTXO)
2. **Valid amounts**: Outputs' amounts must be less or equal to the inputs amounts

Transactions and scripts

Transactions validity

In order for a transaction to be valid it must:

1. **Valid inputs**: Inputs must refer to existing and non-spent outputs (UTXO)
2. **Valid amounts**: Outputs' amounts must be less or equal to the inputs amounts
3. **Valid scripts**: The input script followed by the output script referred by the input must execute succesfully and leave a non-empty stack

Standard scripts: P2PKH

P2PKH: *pay-to-public-key-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a public key whose hash matches the specified and sign the spending transaction with that public key

Standard scripts: P2PKH

P2PKH: *pay-to-public-key-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a public key whose hash matches the specified and sign the spending transaction with that public key

P2PKH sample

- **scriptPubKey**: `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

Standard scripts: P2PKH

P2PKH: *pay-to-public-key-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a public key whose hash matches the specified and sign the spending transaction with that public key

P2PKH sample

- **scriptSig**: `<signature> <pubKey>`
- **scriptPubKey**: `OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`

Standard scripts: P2SH

P2SH: *pay-to-script-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a **redeem script** that succesfully executes and whose hash matches the specified one

Standard scripts: P2SH

P2SH: *pay-to-script-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a **redeem script** that succesfully executes and whose hash matches the specified one

P2SH sample

- **scriptPubKey**: `OP_HASH160 <redeemScript_hash> OP_EQUAL`

Standard scripts: P2SH

P2SH: *pay-to-script-hash*

The output script (*scriptPubKey*) requires the input script (*scriptSig*) to specify a **redeem script** that succesfully executes and whose hash matches the specified one

P2SH sample

- **scriptSig**: `[<data>] <redeemScript>`
- **scriptPubKey**: `OP_HASH160 <redeemScript_hash> OP_EQUAL`

Smart Contracts

Smart Contracts

Computer protocols intended to facilitate, verify or enforce the negotiation or performance of a contract

Smart Contracts

Smart Contracts

Computer protocols intended to facilitate, verify or enforce the negotiation or performance of a contract

Smart Contracts in Bitcoin

Creation of *redeemScripts* redeemable using P2SH script sets in transactions.

Smart Contracts

Smart Contracts

Computer protocols intended to facilitate, verify or enforce the negotiation or performance of a contract

Smart Contracts in Bitcoin

Creation of *redeemScripts* redeemable using P2SH script sets in transactions.

***redeemScripts* are Bitcoin's smart contracts**

What can we do with Smart Contracts?

What can we do with Smart Contracts?

Payment channels

What is a Payment channel?

Payment channel

Set of techniques designed to allow users to make multiple Bitcoin transactions without commiting all of them to the Bitcoin block chain

What is a Payment channel?

Payment channel

Set of techniques designed to allow users to make multiple Bitcoin transactions without commiting all of them to the Bitcoin block chain

***Off-chain* transactions**

Bitcoin transactions that are not commited to the Bitcoin blockchain but would be valid if they were commited

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

1. **Funding:** Some funds are locked so they can be moved with payments during the channel operation

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

1. **Funding:** Some funds are locked so they can be moved with payments during the channel operation
2. **Payment:** Locked funds are moved to pay to a party of the channel

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

1. **Funding:** Some funds are locked so they can be moved with payments during the channel operation
2. **Payment:** Locked funds are moved to pay to a party of the channel
3. **Closure:** Funds are unlocked and returned to the channel parties with the final balance after all payments

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

1. **Funding:** Some funds are locked so they can be moved with payments during the channel operation
2. **Payment:** Locked funds are moved to pay to a party of the channel
3. **Closure:** Funds are unlocked and returned to the channel parties with the final balance after all payments

Payment Channel basic scheme

Scheme

All payment channels follow a basic scheme:

1. **Funding:** Some funds are locked so they can be moved with payments during the channel operation
2. **Payment:** Locked funds are moved to pay to a party of the channel
3. **Closure:** Funds are unlocked and returned to the channel parties with the final balance after all payments

Which transactions are *off-chain*?

All **payment transactions** are ***off-chain***

Outline

1. Introduction
   - What is Bitcoin
   - How does Bitcoin work?
   - The scalability problem
2. Bitcoin & Smart Contracts
   - Transactions at low-level detail
   - Bitcoin's scripting language
   - What is a payment channel?
3. Unidirectional payment channels
   - Scheme
   - Implementation
4. Bidirectional payment channels
   - Scheme
   - Implementation
   - Problem: channel reseting
5. The Bitcoin framework
6. Conclusions

What does a unidirectional payment channel allows us to do?

Unidirectional payment channel

What allows to do?

Incrementally pay amounts of funds from one party to another

Unidirectional payment channel

What allows to do?

Incrementally pay amounts of funds from one party to another

For instance...

We will create a channel to allow **Alice** pay **Bob** incremental amounts of funds

Locking funds

What do we need to do?

Lock funds into the channel so:

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:
    - Alice must want to pay some amount to Bob (Bob can not pay hisself)

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:
    - Alice must want to pay some amount to Bob (Bob can not pay hisself)
    - Bob must authorize payments in order to check funds are send to him (and not to Alice)

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:
   - Alice must want to pay some amount to Bob (Bob can not pay hisself)
   - Bob must authorize payments in order to check funds are send to him (and not to Alice)
2. **Refunds** must be possible if a party does not cooperate

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:
   - Alice must want to pay some amount to Bob (Bob can not pay hisself)
   - Bob must authorize payments in order to check funds are send to him (and not to Alice)
2. **Refunds** must be possible if a party does not cooperate

Locking funds

What do we need to do?

Lock funds into the channel so:

1. **Both** must authorize a payment:
    - Alice must want to pay some amount to Bob (Bob can not pay hisself)
    - Bob must authorize payments in order to check funds are send to him (and not to Alice)
2. **Refunds** must be possible if a party does not cooperate

How to refund

Lock the funds for an amount of time, so after that time (called the *channel expiry time*) the funds are given back to the funder

Locking funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

Locking funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

1. Create a **funding transaction** and a time-locked **refund transaction**

Locking funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

1. Create a **funding transaction** and a time-locked **refund transaction**
2. Create a ***smart* funding transaction** with the time-lock integrated in the *smart contract*

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

Replace by economical incentive

Bob will **keep the latest payment transaction** and discard previous ones, as **the last will be the one that pays more to him**

Closure

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.

Closure

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if Bob disappears, Alice will **broadcast a refund transaction** to recover the locked funds

Locking the funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

1. Create a **funding transaction** and a time-locked **refund transaction**
2. Create a ***smart* funding transaction** with the time-lock integrated in the *smart contract*

Locking the funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

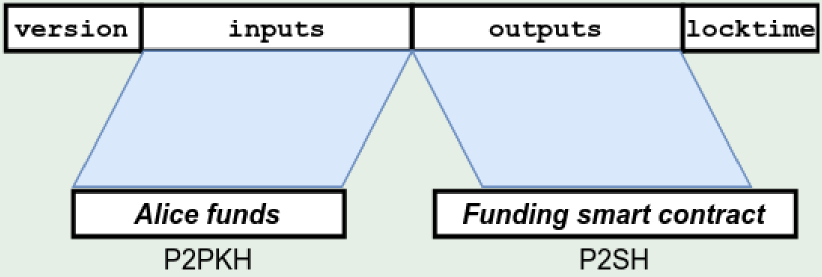
1. Create a **funding transaction** and a time-locked **refund transaction**
2. Create a ***smart* funding transaction** with the time-lock integrated in the *smart contract*

The implementation

With the **BIP-65**, an opcode appeared to create time-locked smart contracts, so we can create a ***smart* funding transaction** with the time lock integrated

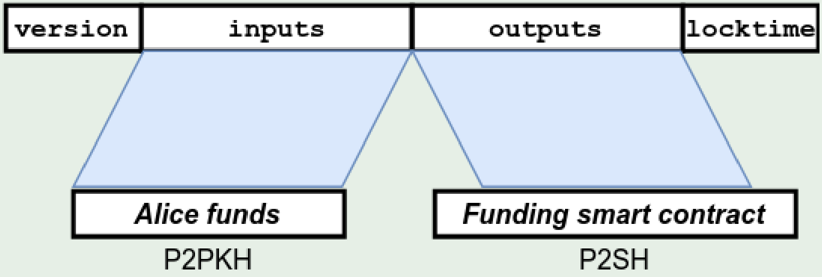
Funding transaction

Funding transaction



Funding transaction

Funding transaction



Funding transaction

Funding smart contract

As we said, we need to design a *redeemScript* in order to create a Bitcoin smart contract:

Funding transaction

Funding smart contract

As we said, we need to design a *redeemScript* in order to create a Bitcoin smart contract:

```
OP_IF <time>
OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG
OP_ELSE
OP_2 <PubKeyAlice_2> <PubKeyBob> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

Funding transaction

Funding smart contract

As we said, we need to design a *redeemScript* in order to create a Bitcoin smart contract:

```
OP_IF <time>
OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG
OP_ELSE
OP_2 <PubKeyAlice_2> <PubKeyBob> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

Technically...

As we are creating a P2SH, then the output script must be:

```
OP_HASH160 <redeemScript_hash> OP_EQUAL
```

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

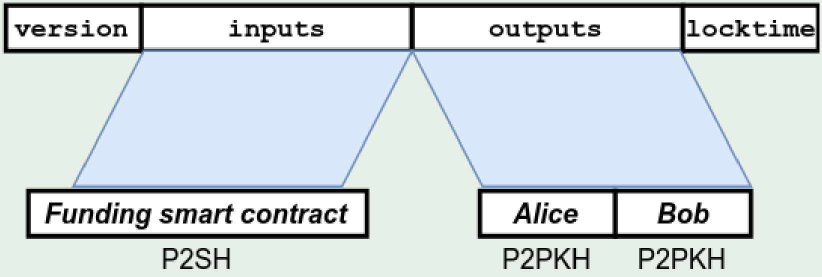
1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

The implementation

Alice creates a transaction that spends the funding transaction, with two outputs: one with some amount for Bob and the rest for herself

Payment transaction

Payment transaction



Payment transaction

Spending funding smart contract

We now need to spend the *redeemScript*

Payment transaction

Spending funding smart contract

We now need to spend the *redeemScript*

```
OP_0 <sig_Alice> <sig_Bob> OP_0
```

Technically...

As we are spending a P2SH, then the input script must be:

```
OP_0 <sig_Alice> <sig_Bob> OP_0 <redeemScript>
```

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if Bob disappears, Alice will **broadcast a refund transaction** to recover the locked funds

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if Bob disappears, Alice will **broadcast a refund transaction** to recover the locked funds

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

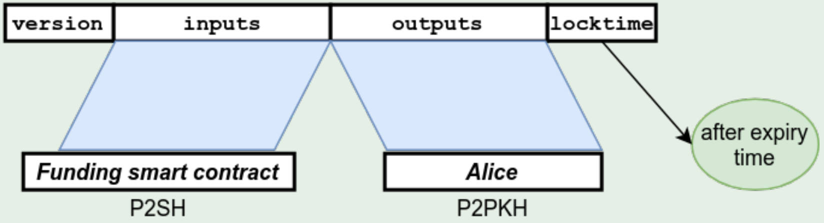
1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if Bob disappears, Alice will **broadcast a refund transaction** to recover the locked funds

Graceful closure

Bob simply broadcasts the latest payment transaction once signed and before channel expiry time

Closure transaction

Closure transaction (refund)



Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

```
<sig_Alice> OP_1
```

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

```
<sig_Alice> OP_1
```

Technically...

As we are spending a P2SH, then the input script must be:

```
<sig_Alice> OP_1 <redeemScript>
```

What if we want Bob to pay Alice too?

Outline

1 Introduction
- What is Bitcoin
- How does Bitcoin work?
- The scalability problem

2 Bitcoin & Smart Contracts
- Transactions at low-level detail
- Bitcoin's scripting language
- What is a payment channel?

3 Unidirectional payment channels
- Scheme
- Implementation

4 Bidirectional payment channels
- Scheme
- Implementation
- Problem: channel reseting

5 The Bitcoin framework

6 Conclusions

Bidirectional payment channel

What allows to do?

Incrementally pay amounts of funds from one party to another **and viceversa**

Bidirectional payment channel

What allows to do?

Incrementally pay amounts of funds from one party to another **and viceversa**

For instance...

We will create a channel to allow **Alice** pay **Bob** incremental amounts of funds **and viceversa**

Bidirectional payment channels' scheme

Source

Obtained from

A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels - Christian Decker & Roger Wattenhofer

Bidirectional payment channels' scheme

Source

Obtained from

> *A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels - Christian Decker & Roger Wattenhofer*

Idea

Use **two unidirectional channels**, one in each way with an **invalidation tree** to perform resets

Locking the funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

1. Create a **funding transaction** and a time-locked **refund transaction**
2. Create a ***smart* funding transaction** with the time-lock integrated in the *smart contract*

Locking the funds

Ways to lock funds

In order to accomplish both properties to lock funds, we can:

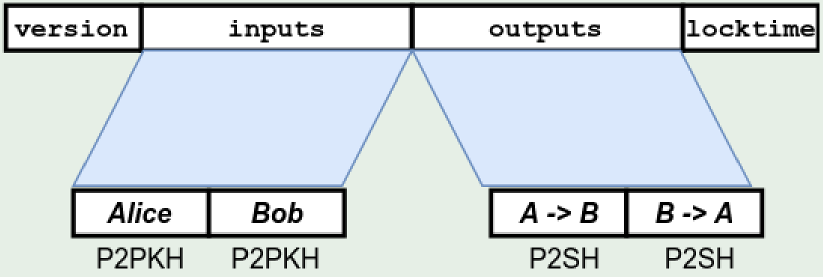
1. Create a **funding transaction** and a time-locked **refund transaction**
2. Create a ***smart* funding transaction** with the time-lock integrated in the *smart contract*

The implementation

We can still use **BIP-65** to create a time-locking smart contract

Funding transaction

Funding transaction



Funding transaction

Funding smart contract

Same as unidirectional channel, but with two outputs

Funding transaction

Funding smart contract

Same as unidirectional channel, but with two outputs

1. **Alice to Bob output**

```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG OP_ELSE OP_2
<PubKeyAlice_2> <PubKeyBob_1> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

Funding transaction

Funding smart contract

Same as unidirectional channel, but with two outputs

1. **Alice to Bob output**

```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG OP_ELSE OP_2
<PubKeyAlice_2> <PubKeyBob_1> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

2. **Bob to Alice output**

```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyBob_2> OP_CHECKSIG OP_ELSE OP_2 <PubKeyAlice_3>
<PubKeyBob_3> OP_2 OP_CHECKMULTISIG OP_ENDIF
```

Funding transaction

Funding smart contract

Same as unidirectional channel, but with two outputs

1. **Alice to Bob output**

```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG OP_ELSE OP_2
<PubKeyAlice_2> <PubKeyBob_1> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```

2. **Bob to Alice output**

```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyBob_2> OP_CHECKSIG OP_ELSE OP_2 <PubKeyAlice_3>
<PubKeyBob_3> OP_2 OP_CHECKMULTISIG OP_ENDIF
```

Funding transaction

Funding smart contract

Same as unidirectional channel, but with two outputs

1. **Alice to Bob output**
```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyAlice_1> OP_CHECKSIG OP_ELSE OP_2
<PubKeyAlice_2> <PubKeyBob_1> OP_2 OP_CHECKMULTISIG
OP_ENDIF
```
2. **Bob to Alice output**
```
OP_IF <time> OP_CHECKLOCKTIMEVERIFY OP_DROP
<PubKeyBob_2> OP_CHECKSIG OP_ELSE OP_2 <PubKeyAlice_3>
<PubKeyBob_3> OP_2 OP_CHECKMULTISIG OP_ENDIF
```

Technically...

As we are creating a P2SH, then the outputs' script must be:

```
OP_HASH160 <redeemScript_hash> OP_EQUAL
```

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

Paying funds

What do we need to do?

In order to create a payment transaction, as both users must authorize payments:

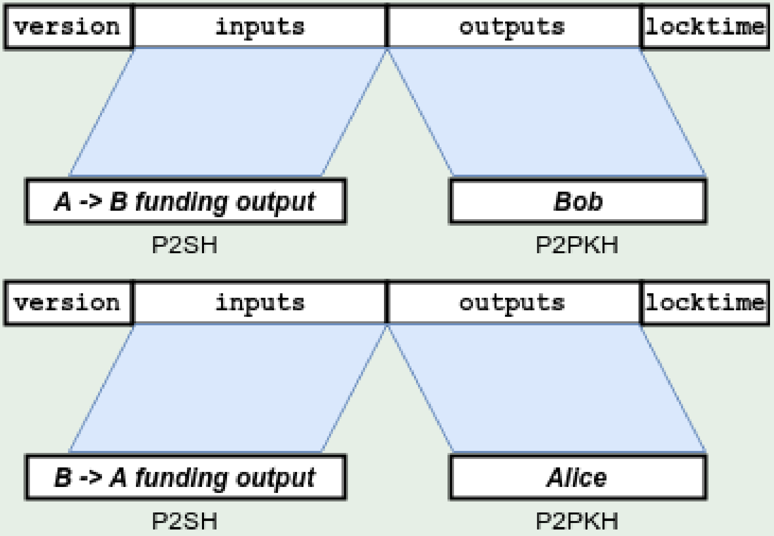
1. **Alice** creates and signs a transaction paying some of the locked funds to **Bob** (and the rest to Alice as return)
2. **Bob** stores the partially signed transaction that pays some amount of money to him
3. If **Alice** wants to pay more, repeats the first step with more funds (spending the same funding transaction)

The implementation

Same of a unidirectional payment channel, but Bob can pay Alice too using his channel

Payment transaction

Payment transaction



Payment transaction

Spending funding smart contract

We now need to spend the *redeemScript*

Payment transaction

Spending funding smart contract

We now need to spend the *redeemScript*

1. **Alice to Bob output**
   `OP_0 <sig_Alice> <sig_Bob> OP_0`

Technically...

As we are spending a P2SH, then the input script must be:

```
OP_0 <sig_Alice> <sig_Bob> OP_0 <redeemScript>
```

Payment transaction

Spending funding smart contract

We now need to spend the *redeemScript*

1. **Alice to Bob output**
   `OP_0 <sig_Alice> <sig_Bob> OP_0`
2. **Bob to Alice output**
   `OP_0 <sig_Alice> <sig_Bob> OP_0`

Technically...

As we are spending a P2SH, then the input script must be:

```
OP_0 <sig_Alice> <sig_Bob> OP_0 <redeemScript>
```

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction of each output is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if any of the parties do not cooperate, they can **broadcast a refund transaction** to recover their locked funds

Closure transaction

What do we need to do?

Two situations can appear when closing the channel:

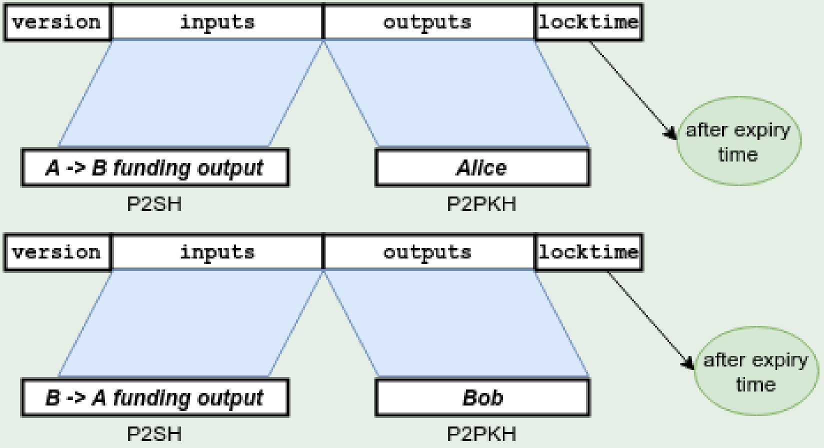
1. **Graceful closure**: the channel has been operated and the expiry time is close, so **latest payment transaction <u>of each output</u> is broadcasted**, spending the funding transaction and closing the channel.
2. **No cooperation**: if any of the parties do not cooperate, they can **broadcast a refund transaction** to recover their locked funds

Graceful closure

Alice and Bob simply broadcast the latest payment transaction once signed and before channel expiry time

Closure transaction

Closure transaction (refund)



Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

1. **Alice to Bob output refund**
   `<sig_Alice> OP_1`

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

1. **Alice to Bob output refund**
   `<sig_Alice> OP_1`
2. **Bob to Alice output refund**
   `<sig_Bob> OP_1`

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

1. **Alice to Bob output refund**
   `<sig_Alice> OP_1`
2. **Bob to Alice output refund**
   `<sig_Bob> OP_1`

Closure transaction

Spending funding smart contract (refund)

We now need to spend the *redeemScript* after the lock time

1. **Alice to Bob output refund**
   `<sig_Alice> OP_1`
2. **Bob to Alice output refund**
   `<sig_Bob> OP_1`

Technically...

As we are spending a P2SH, then the input script must be:

```
<sig_Alice|Bob> OP_1 <redeemScript>
```

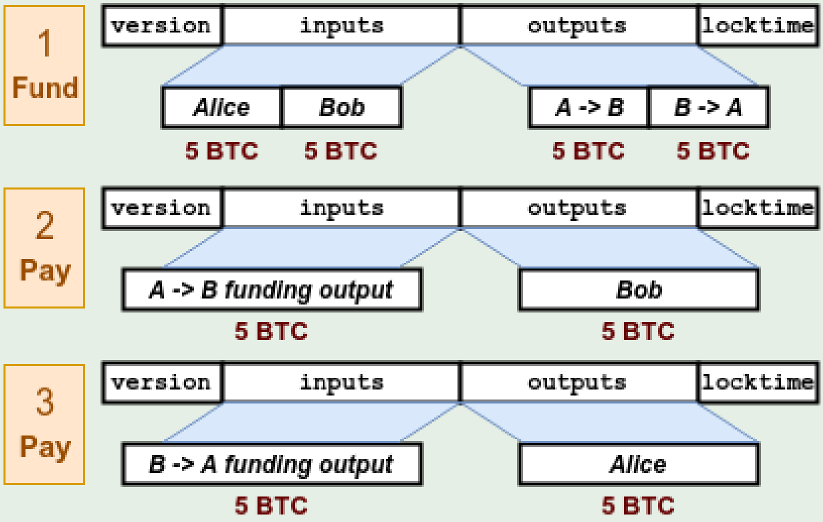
What if one of the payment channels gets exhausted?

What if one of the payment channels gets exhausted?

Channel resetting

Channel resetting

A simple reset example



Channel resetting

Channels are exhausted

Both parties own the same amount of funds as at the beginning of the channel but their respective payment channels have been exhausted. No more incremental payments can be performed

Resetting by invalidation trees

Invalidation tree

Tree of transactions that use the timelock field to invalidate old branches of the tree and be able to create new ones with an updated status of the balances

Resetting by invalidation trees

Invalidation tree

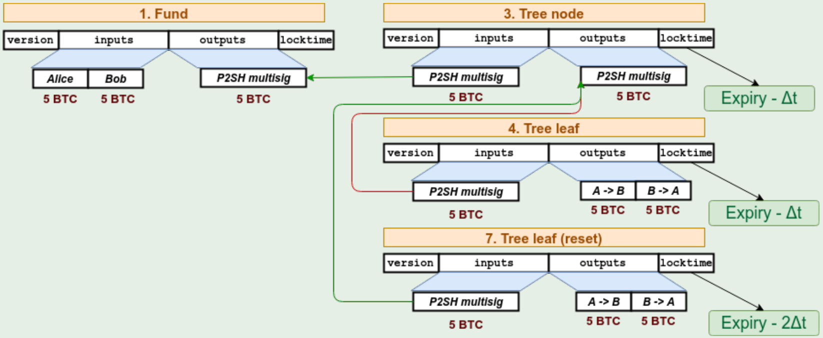
Tree of transactions that use the timelock field to invalidate old branches of the tree and be able to create new ones with an updated status of the balances

Replace by timelock

Create timelocked transactions so that when using timelocks nearer to the present invalidate transactions with later timelocks

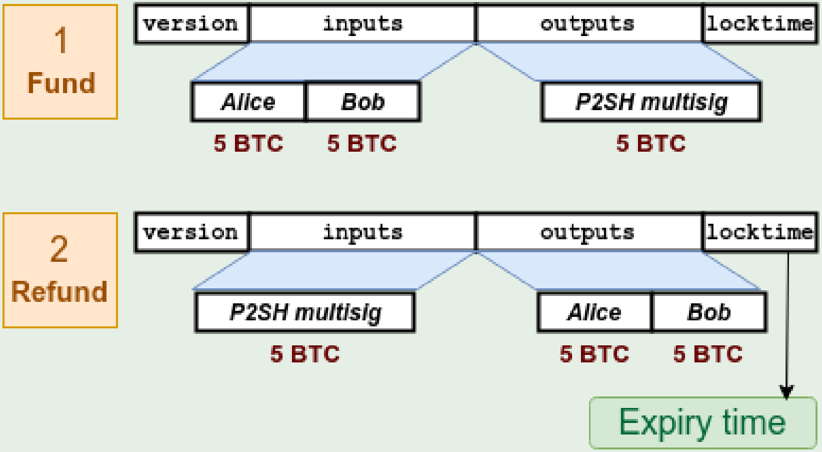
An invalidation tree reset example

Reset by adding a new leaf



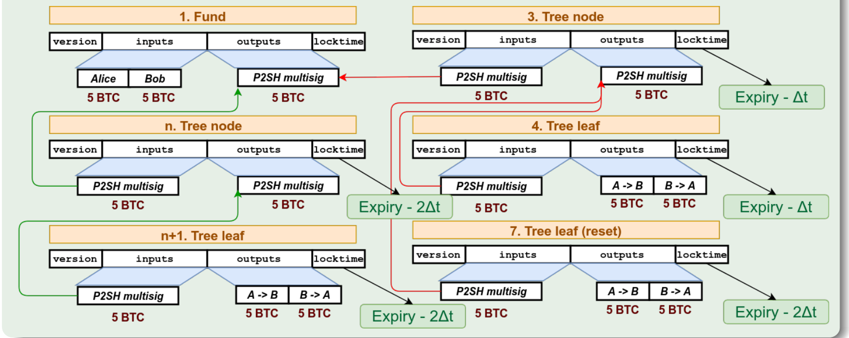
An invalidation tree reset example

Different funding



An invalidation tree reset example

Reset by branching



Differences

Basic duplex channel vs Resetable duplex channel

- **More complex to use BIP-65***: As the tree requires linking P2SH single outputs, using BIP-65 to create a timelock contract is more complex to implement

Differences

Basic duplex channel vs Resetable duplex channel

- **More complex to use BIP-65***: As the tree requires linking P2SH single outputs, using BIP-65 to create a timelock contract is more complex to implement

Differences

Basic duplex channel vs Resetable duplex channel

- **More complex to use BIP-65***: As the tree requires linking P2SH single outputs, using BIP-65 to create a timelock contract is more complex to implement
  *this would require to generate two outputs and inputs in each first tree node with all data required

Differences

Basic duplex channel vs Resetable duplex channel

- **More complex to use BIP-65***: As the tree requires linking P2SH single outputs, using BIP-65 to create a timelock contract is more complex to implement
  *this would require to generate two outputs and inputs in each first tree node with all data required
- **More transactions needed:** in order to create the tree (be careful with signing order of all parties to prevent attacks)

Differences

Basic duplex channel vs Resetable duplex channel

- **More complex to use BIP-65***: As the tree requires linking P2SH single outputs, using BIP-65 to create a timelock contract is more complex to implement
  *this would require to generate two outputs and inputs in each first tree node with all data required
- **More transactions needed:** in order to create the tree (be careful with signing order of all parties to prevent attacks)
- **Reduced expiry time:** each tree branch reduces the channel's effective expiry time

Pros and cons

Pros

- **Simple to create**: no complex transactions needed, unlike the *Lightning Network* smart contracts

Pros and cons

Pros

- **Simple to create**: no complex transactions needed, unlike the *Lightning Network* smart contracts
- **No extra data exchange**: unlike the *Lightning Network*, the protocol does not require to exchange secrets or additional data

Pros and cons

Pros

- **Simple to create**: no complex transactions needed, unlike the *Lightning Network* smart contracts
- **No extra data exchange**: unlike the *Lightning Network*, the protocol does not require to exchange secrets or additional data

Pros and cons

Pros

- **Simple to create**: no complex transactions needed, unlike the *Lightning Network* smart contracts
- **No extra data exchange**: unlike the *Lightning Network*, the protocol does not require to exchange secrets or additional data

Cons

- **Reducing expiry time**: the more resets needed, the more the effective expiry time is reduced (more invalidating branches and leafs)

Pros and cons

Pros

- **Simple to create**: no complex transactions needed, unlike the *Lightning Network* smart contracts
- **No extra data exchange**: unlike the *Lightning Network*, the protocol does not require to exchange secrets or additional data

Cons

- **Reducing expiry time**: the more resets needed, the more the effective expiry time is reduced (more invalidating branches and leafs)
- **Need to store more transactions**: in other solutions for duplex payment channel, like the *Lightning Network*, just the latest payment transaction must be saved, and not an entire tree.

Outline

1 Introduction
- What is Bitcoin
- How does Bitcoin work?
- The scalability problem

2 Bitcoin & Smart Contracts
- Transactions at low-level detail
- Bitcoin's scripting language
- What is a payment channel?

3 Unidirectional payment channels
- Scheme
- Implementation

4 Bidirectional payment channels
- Scheme
- Implementation
- Problem: channel reseting

5 The Bitcoin framework

6 Conclusions

Developing problems

Problems when implementing the channel

- **Lack of documentation**: Bitcoin is missing from good quality, low-level protocol implementation details. Most accurate information is spread around Q&A sites, *Bitcoin Wiki* and *Bitcoin Core's client* C++ code

Developing problems

Problems when implementing the channel

- **Lack of documentation**: Bitcoin is missing from good quality, low-level protocol implementation details. Most accurate information is spread around Q&A sites, *Bitcoin Wiki* and *Bitcoin Core's client* C++ code
- **Lack of low-level, documented libraries**: There are very few libraries that handle the Bitcoin protocol complexities (no library found to create raw transaction signatures with a customized transaction)

Our Bitcoin framework

Solution: our own Bitcoin framework

All what we* learned was implemented in our own Bitcoin framework that has:

- **Designed for ease of use:** Design & Software design patterns

Our Bitcoin framework

Solution: our own Bitcoin framework

All what we* learned was implemented in our own Bitcoin framework that has:

- **Designed for ease of use:** Design & Software design patterns
- **OOP and puzzle-friendliness principles:** Modulable and serializable / deserializable patterns

*developed along Carlos González Cebrecos

Our Bitcoin framework

Solution: our own Bitcoin framework

All what we* learned was implemented in our own Bitcoin framework that has:

- **Designed for ease of use:** Design & Software design patterns
- **OOP and puzzle-friendliness principles:** Modulable and serializable / deserializable patterns
- **Extensive documentation:** Every method is well documented

*developed along Carlos González Cebrecos

Our Bitcoin framework

Solution: our own Bitcoin framework

All what we* learned was implemented in our own Bitcoin framework that has:

- **Designed for ease of use:** Design & Software design patterns
- **OOP and puzzle-friendliness principles:** Modulable and serializable / deserializable patterns
- **Extensive documentation:** Every method is well documented
- **Extensively tested:** All code has been tested with other libraries & *Bitcoin Core* client

*developed along Carlos González Cebrecos

Channel implementation

Fork of the Bitcoin framework

The channel was implemented in a script after forking the framework and can be operated from the CLI passing the required parameters (funds amount, pub/priv keys, previous inputs, ...)

Channel implementation

Fork of the Bitcoin framework

The channel was implemented in a script after forking the framework and can be operated from the CLI passing the required parameters (funds amount, pub/priv keys, previous inputs, ...)

Channel lacks ease of use

Because focused on the **channel protocol's design to enhace security**, no time was missing to automate the operatibility of the channel:

- **Bitcoin Core RPC:** to automate transaction broadcasting, UTXO detection, balance detection, fee calculation, ...

Channel implementation

Fork of the Bitcoin framework

The channel was implemented in a script after forking the framework and can be operated from the CLI passing the required parameters (funds amount, pub/priv keys, previous inputs, ...)

Channel lacks ease of use

Because focused on the **channel protocol's design to enhace security**, no time was missing to automate the operatibility of the channel:

- **Bitcoin Core RPC:** to automate transaction broadcasting, UTXO detection, balance detection, fee calculation, ...
- **Channel state storage:** automatically store in the user's computer the state of the channel

Channel implementation

Fork of the Bitcoin framework

The channel was implemented in a script after forking the framework and can be operated from the CLI passing the required parameters (funds amount, pub/priv keys, previous inputs, ...)

Channel lacks ease of use

Because focused on the **channel protocol's design to enhace security**, no time was missing to automate the operatibility of the channel:

- **Bitcoin Core RPC:** to automate transaction broadcasting, UTXO detection, balance detection, fee calculation, ...
- **Channel state storage:** automatically store in the user's computer the state of the channel
- **Graphical UI:** enable every Bitcoin user enjoy the payment channels' potential

Outline

1 Introduction
- What is Bitcoin
- How does Bitcoin work?
- The scalability problem

2 Bitcoin & Smart Contracts
- Transactions at low-level detail
- Bitcoin's scripting language
- What is a payment channel?

3 Unidirectional payment channels
- Scheme
- Implementation

4 Bidirectional payment channels
- Scheme
- Implementation
- Problem: channel reseting

5 The Bitcoin framework

6 Conclusions

Along this project, I've learned:

- **Low-level understanding of the Bitcoin protocol**: By learning how to implement Smart Contracts on Bitcoin and creating the framework to ease those smart contracts creation

Along this project, I've learned:

- **Low-level understanding of the Bitcoin protocol**: By learning how to implement Smart Contracts on Bitcoin and creating the framework to ease those smart contracts creation
- **Bitcoin lacks of low-level extensive documentation**: Developing in Bitcoin has no formal, low-level detailed guide and most advanced features can just be learned by inspecting the *Bitcoin Core* C++ code

Along this project, I've learned:

- **Low-level understanding of the Bitcoin protocol**: By learning how to implement Smart Contracts on Bitcoin and creating the framework to ease those smart contracts creation
- **Bitcoin lacks of low-level extensive documentation**: Developing in Bitcoin has no formal, low-level detailed guide and most advanced features can just be learned by inspecting the *Bitcoin Core* C++ code
- **Payment Channels are the future of Bitcoin**: Maybe *Lightning Network* has a better structure and protocol implementation, but what is crystal clear is that multi-hop duplex payment channels are the Bitcoin's future after *SegWit.co* activates in the *mainnet* allowing a secure implementation of them.

Thanks for your time and attention

Thanks for your time and attention

Q&A round

For more information

The project work compilation

Documentation:

`https://davidlj95.com/smart-payment-channel`

Code:

`https://github.com/davidlj95/smart-payment-channel`

For more information

The project work compilation

Documentation:

https://davidlj95.com/smart-payment-channel

Code:

https://github.com/davidlj95/smart-payment-channel

The Bitcoin framework

https://github.com/uab-projects/btc-payment-channels

For more information

The project work compilation

Documentation:

`https://davidlj95.com/smart-payment-channel`

Code:

`https://github.com/davidlj95/smart-payment-channel`

The Bitcoin framework

`https://github.com/uab-projects/btc-payment-channels`

Test it!:

```
pip install bitcoin-framework
```