An Introduction to Bitcoin

Saravanan Vijayakumaran

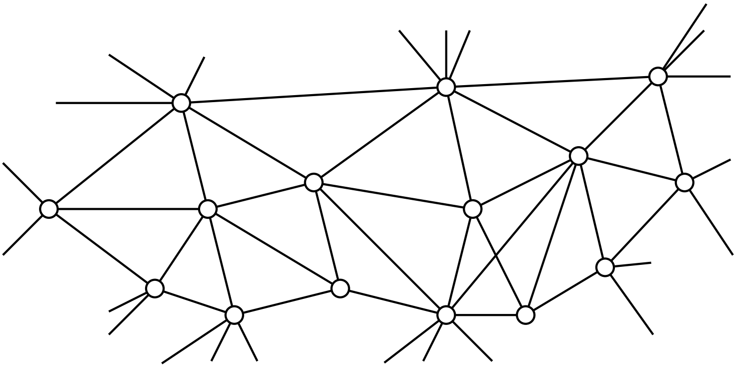
Department of Electrical Engineering
Indian Institute of Technology Bombay
https://www.ee.iitb.ac.in/~sarva

December 19, 2017

Venue: IIT Madras

What is Bitcoin?

- Cryptocurrency
- Open source
- Decentralized network



Decentralization Challenges

- Counterfeiting
- Currency creation rules
- Double spending
  - Alice pays Bob *n* digicoins for pizza
  - Alice uses the **same** *n* digicoins to pay Carol for books
- Centralization solves all three problems

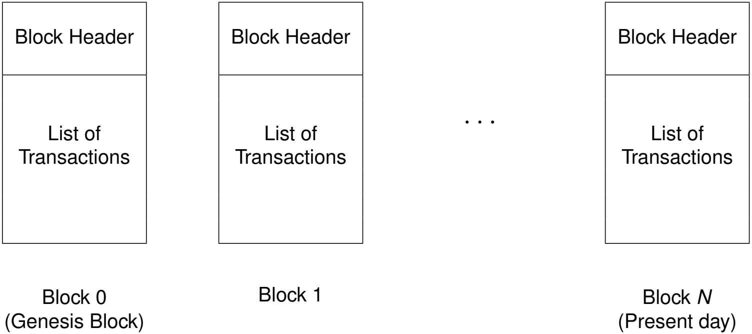
Solution without a central coordinator?

Double Spending

- Familiar to academics
- Submitting same paper to two conferences
- **Possible solution**
  Reviewers google paper contents to find duplicates
- Solution fails if
  - Conferences accepting papers at same time
  - Conference proceedings not published/indexed
- **Better solution**
  A single public database to store all submissions to all conferences

The Blockchain

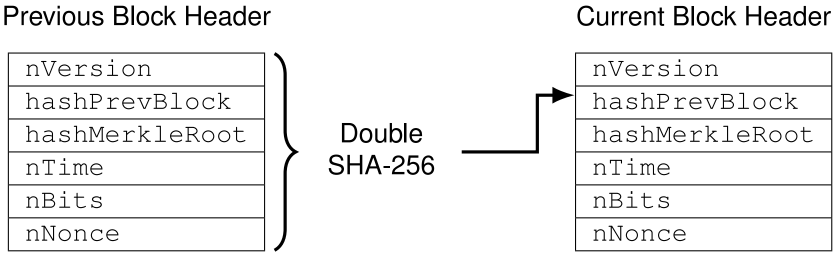
Bitcoin's public database for storing transactions



I see blocks. Where is the "chain"?

Block Header

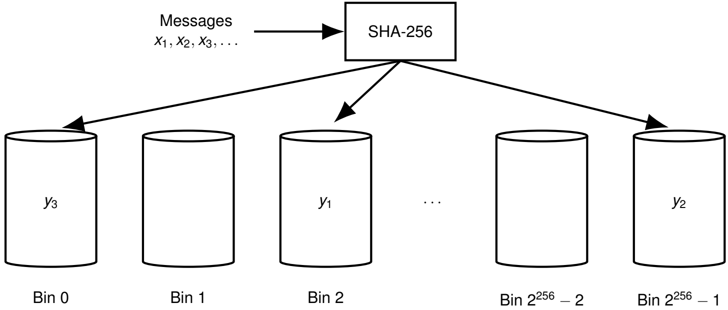
nVersion	4 bytes
hashPrevBlock	32 bytes
hashMerkleRoot	32 bytes
nTime	4 bytes
nBits	4 bytes
nNonce	4 bytes



SHA-256: Cryptographic hash function

Cryptographic Hash Functions

- Input: Variable length bitstrings
- Output: Fixed length bitstrings
- Easy to compute but difficult to invert
  - Given $H(x)$, computationally infeasible to find x
- Collision resistant
  - Computationally infeasible to find $x \neq y$ with $H(x) = H(y)$
- Pseudorandom function



- Demo

Bitcoin Mining

Preventing Spam in Public Databases

- A database you own where anyone in the world can add entries? Your email inbox
- Hashcash was proposed in 1997 to prevent spam
- Protocol
    - Suppose an email client wants to send email to an email server
    - Client and server agree upon a cryptographic hash function H
    - Email server sends the client a challenge string c

$$\begin{tikzcd} \text{Server} \arrow[r, bend left, "c"] & \text{Client} \arrow[l, bend left, "c\|x"] \end{tikzcd}$$

    - Client needs to find a string x such that $H(c\|x)$ begins with k zeros
    - Since H has pseudorandom outputs, probability of success in a single trial is

$$\frac{2^{n-k}}{2^n} = \frac{1}{2^k}$$

- The x corresponding to c is considered **proof-of-work (PoW)**
- PoW is difficult to generate but easy to verify

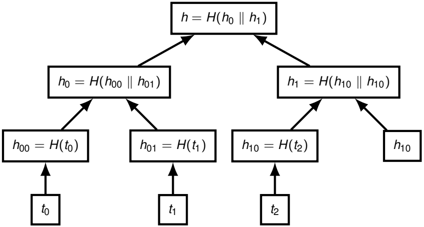
Bitcoin Mining (1/2)

- Process of adding new blocks to the blockchain
- Nodes which want to perform transactions broadcast them
- Miners collect some of these transactions into a candidate block

Block Header
Number of Transactions n
Coinbase Transaction
Regular Transaction 1
Regular Transaction 2
⋮
Regular Transaction $n-1$

`nVersion`
`hashPrevBlock`
`hashMerkleRoot`
`nTime`
`nBits`
`nNonce`

- **`hashPrevBlock`** contains double SHA-256 hash of previous block's header
- **`hashMerkleRoot`** contains root hash of transaction Merkle tree



Bitcoin Mining (2/2)

Block Header
Number of Transactions n
Coinbase Transaction
Regular Transaction 1
Regular Transaction 2
$\vdots$
Regular Transaction $n-1$

nVersion
hashPrevBlock
hashMerkleRoot
nTime
nBits
nNonce

- **nBits** encodes a 256-bit target value T, say

$$T = \mathtt{0x}\,\underbrace{\mathtt{00}\cdots\mathtt{00}}_{\text{16 times}}\underbrace{\mathtt{FFFFF}\cdots\mathtt{FFFFF}}_{\text{48 times}}$$

- Miner who can find **nNonce** such that

$$\text{SHA256}\,(\text{SHA256}\,(\mathbf{nVersion} \parallel \mathbf{HashPrevBlock} \parallel \ldots \parallel \mathbf{nNonce})) \leq T$$

can add a new block

Why is Mining Hard?

Target value T	Fraction of SHA256d outputs $\leq T$
$\mathtt{0x7}\underbrace{\mathtt{FFFF\ FFFF} \cdots \mathtt{FFFF}}_{\text{63 times}}$	$\frac{1}{2}$
$\mathtt{0x0}\underbrace{\mathtt{FFFF\ FFFF} \cdots \mathtt{FFFF}}_{\text{63 times}}$	$\frac{1}{16}$
$\mathtt{0x}\underbrace{\mathtt{00} \cdots \mathtt{00}}_{\text{16 times}}\underbrace{\mathtt{FFFFF} \cdots \mathtt{FFFFF}}_{\text{48 times}}$	$\frac{1}{2^{64}}$

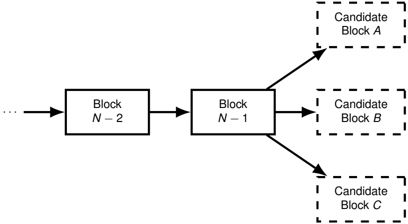
$$\Pr\left[\text{SHA256d output} \leq T\right] \approx \frac{T+1}{2^{256}}$$

Why should anyone mine blocks?

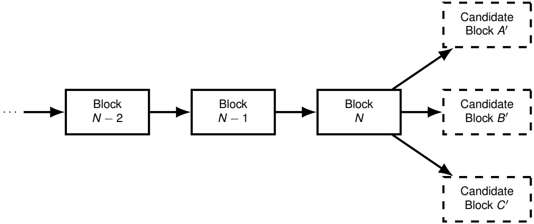
- Successful miner gets rewarded in bitcoins
- Every block contains a **coinbase transaction** which creates 12.5 bitcoins
- Each miner specifies his own address as the destination of the new coins
- Every miner is competing to solve their own PoW puzzle
- Miners also collect the transaction fees in the block

Block Addition Workflow

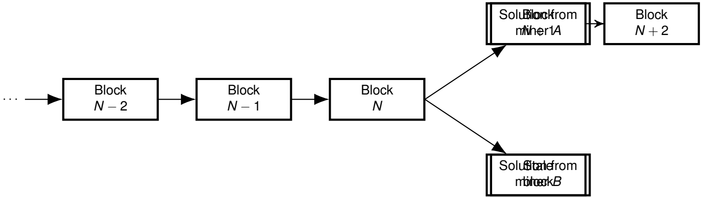
- Nodes broadcast transactions
- Miners accept valid transactions and reject invalid ones (solves double spending)
- Miners try extending the latest block



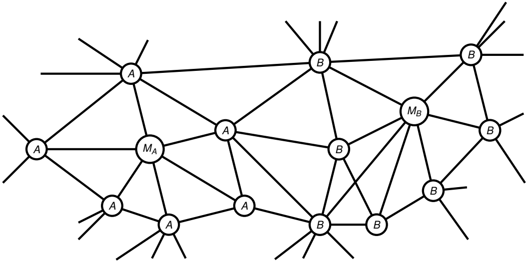
- Miners compete to solve the search puzzle and broadcast solutions
- Unsuccessful miners abandon their current candidate blocks and start work on new ones



What if two miners solve the puzzle at the same time?



- Both miners will broadcast their solution on the network
- Nodes will accept the first solution they hear and reject others



- Nodes always switch to the longest chain they hear
- Eventually the network will converge and achieve consensus

How often are new blocks created?

- Once every 10 minutes

`nVersion`
`hashPrevBlock`
`hashMerkleRoot`
`nTime`
`nBits`
`nNonce`

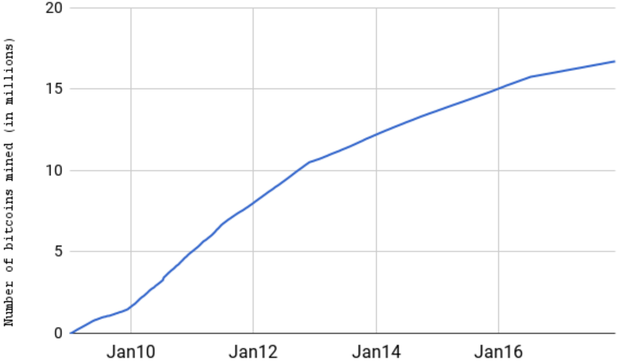
- Every 2016 blocks, the target T is recalculated
- Let t_{sum} = Number of seconds taken to mine last 2016 blocks

$$T_{\text{new}} = \frac{t_{\text{sum}}}{14 \times 24 \times 60 \times 60} \times T$$

- Recall that probability of success in single trial is $\frac{T+1}{2^{256}}$
- If $t_{\text{sum}} = 2016 \times 8 \times 60$, then $T_{\text{new}} = \frac{4}{5} T$
- If $t_{\text{sum}} = 2016 \times 12 \times 60$, then $T_{\text{new}} = \frac{6}{5} T$

Bitcoin Supply

- The block subsidy was initially 50 BTC per block
- Halves every 210,000 blocks $\approx$ 4 years
- Became 25 BTC in Nov 2012 and 12.5 BTC in July 2016
- Total Bitcoin supply is 21 million



- The last bitcoin will be mined in 2140

Bitcoin Payment Workflow

- Merchant shares address out of band (not using Bitcoin P2P)
- Customer broadcasts transaction t which pays the address
- Miners collect broadcasted transactions into a candidate block

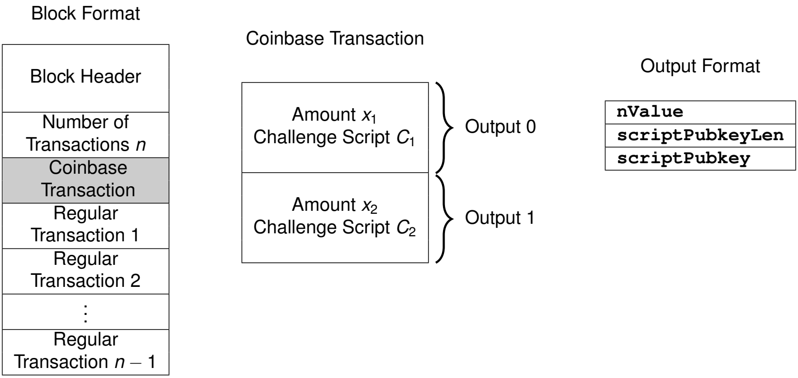
Block Header
Number of Transactions n
Coinbase Transaction
Regular Transaction 1
Regular Transaction 2
$\vdots$
Regular Transaction $n-1$

- One of the candidate blocks containing t is mined
- Merchant waits for confirmations on t before providing goods

Bitcoin Transaction Format

Coinbase Transaction Format

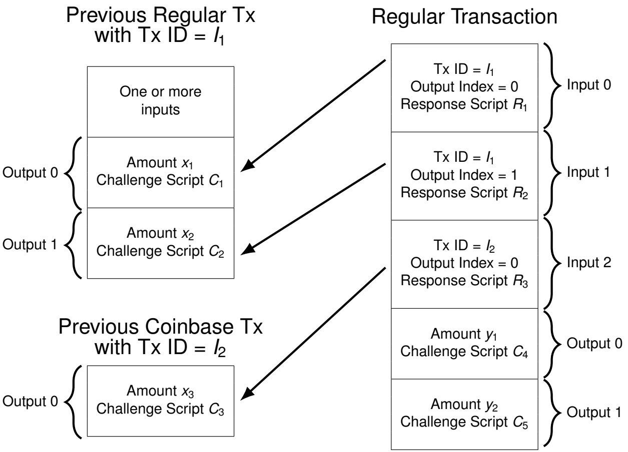
Pre-SegWit



- **`nValue`** contains number of satoshis locked in output
  - 1 Bitcoin = 10^8 satoshis
- **`scriptPubkey`** contains the challenge script
- **`scriptPubkeyLen`** contains byte length of challenge script

Regular Transaction Format

Pre-SegWit



Input Format

`hash`
`n`
`scriptSigLen`
`scriptSig`
`nSequence`

Output Format

`nValue`
`scriptPubkeyLen`
`scriptPubkey`

- **`hash`** and **`n`** identify output being unlocked
- **`scriptSig`** contains the response script

Bitcoin Scripting Language

Script

- Forth-like stack-based language
- One-byte opcodes

Remaining Script	Stack State
`OP_2 OP_3 OP_ADD`	
`OP_3 OP_ADD`	2
`OP_ADD`	3 2
	5

Challenge/Response Script Execution

Remaining Script	Stack State
<Response Script> <Challenge Script>	
<Challenge Script>	x_1 x_2 $\vdots$ x_n
	y_1 y_2 $\vdots$ y_m

Response is valid if top element y_1 evaluates to `True`

Challenge Script Example

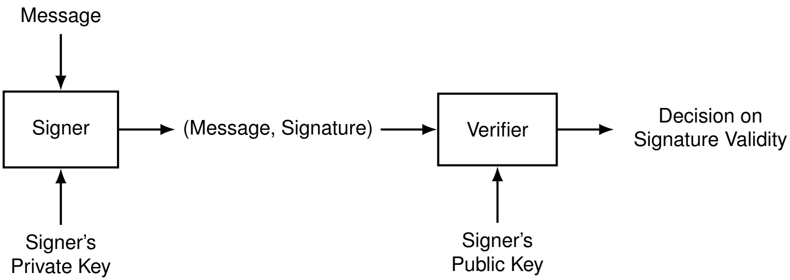
$$\texttt{OP\_HASH256 0x20 } \underbrace{\texttt{<256-bit string>}}_{\texttt{S}} \texttt{ OP\_EQUAL}$$

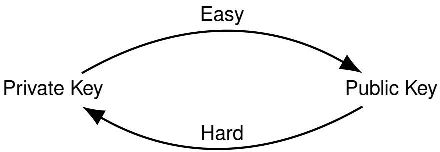
Remaining Script	Stack State
`OP_HASH256 0x20 S OP_EQUAL`	x
`0x20 S OP_EQUAL`	$H(x)$
`OP_EQUAL`	`S` $H(x)$
	0 or 1

Unsafe challenge script! Guess why?

Pay to Public Key

Digital Signatures



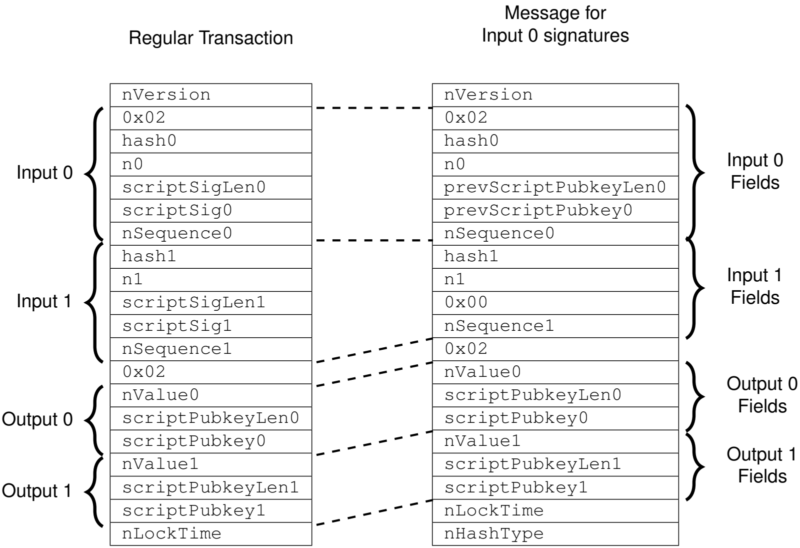


Pay to Public Key

- Challenge script: **0x21 <Public Key> OP_CHECKSIG**
- Response script: **<Signature>**

Remaining Script	Stack State
<Signature> <Public Key> OP_CHECKSIG	
<Public Key> OP_CHECKSIG	**<Signature>**
OP_CHECKSIG	**<Public Key>** **<Signature>**
	True/False

Signatures Protect Transactions



Pay to Public Key Hash

Pay to Public Key Hash Address

- To receive bitcoins, a challenge script needs to be specified
- P2PKH addresses encode P2PKH challenge scripts
- Example: **`1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm`**

Base58 Encoding

1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm

$$\updownarrow$$

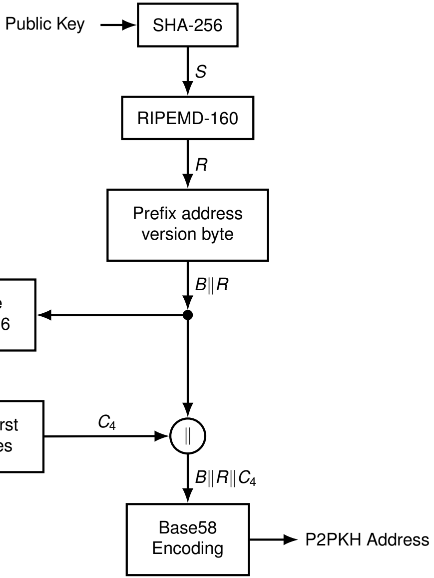
0091B24BF9F5288532960AC687ABB035127B1D28A50074FFE0

- Alphanumeric representation of bytestrings
- From 62 alphanumeric characters 0, O, I, l are excluded

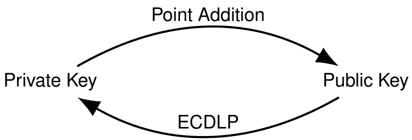
Ch	Int	Ch	Int	Ch	Int	Ch	Int	Ch	Int	Ch	Int	Ch	Int
1	0	A	9	K	18	U	27	d	36	n	45	w	54
2	1	B	10	L	19	V	28	e	37	o	46	x	55
3	2	C	11	M	20	W	29	f	38	p	47	y	56
4	3	D	12	N	21	X	30	g	39	q	48	z	57
5	4	E	13	P	22	Y	31	h	40	r	49		
6	5	F	14	Q	23	Z	32	i	41	s	50		
7	6	G	15	R	24	a	33	j	42	t	51		
8	7	H	16	S	25	b	34	k	43	u	53		
9	8	J	17	T	26	c	35	m	44	v	53		

- Given a bytestring $b_n b_{n-1} \cdots b_0$
  - Encode each leading zero byte as a 1
  - Get integer $N = \sum_{i=0}^{n-m} b_i 256^i$
  - Get $a_k a_{k-1} \cdots a_0$ where $N = \sum_{i=0}^{k} a_i 58^i$
  - Map each integer a_i to a Base58 character

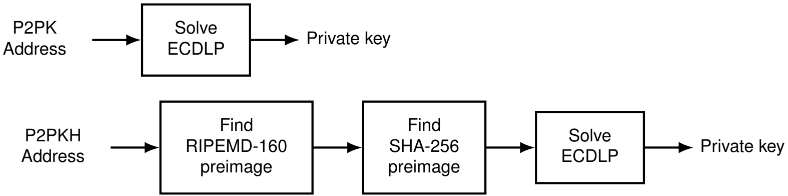
Pay to Public Key Hash Address



Why Hash the Public Key?

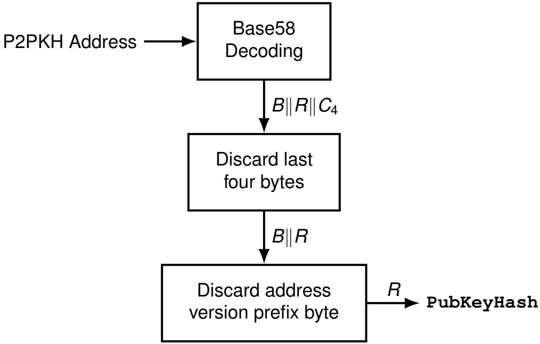


- ECDLP = Elliptic Curve Discrete Logarithm Problem
- ECDLP currently hard but no future guarantees
- Hashing the public key gives extra protection



P2PKH Transaction

- Challenge script
  `OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`



- Response script: **`<Signature> <Public Key>`**

P2PKH Script Execution (1/2)

Remaining Script	Stack State
<Signature> <Public Key> OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	
<Public Key> OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	**<Signature>**
OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	**<Public Key>** **<Signature>**
OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG	**<Public Key>** **<Public Key>** **<Signature>**

P2PKH Script Execution (2/2)

Remaining Script	Stack State

`<PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG`	`<PubKeyHashCalc>`
	`<Public Key>`
	`<Signature>`

`OP_EQUALVERIFY OP_CHECKSIG`	`<PubKeyHash>`
	`<PubKeyHashCalc>`
	`<Public Key>`
	`<Signature>`

`OP_CHECKSIG`	`<Public Key>` `<Signature>`
	`True/False`

Multi-Signature

m-of-*n* Multi-Signature Scripts

- *m*-of-*n* multisig challenge script specifies *n* public keys

  `m <Public Key 1> ··· <Public Key n> n OP_CHECKMULTISIG`

- Response script provides signatures created using **any** *m* out of the *n* private keys

  `OP_0 <Signature 1> ··· <Signature m>`.

- Example: $m = 2$ and $n = 3$
  - Challenge script

    `OP_2 <PubKey1> <PubKey2> <PubKey3> OP_3 OP_CHECKMULTISIG`

  - Response script

    `OP_0 <Sig1> <Sig2>`

2-of-3 Multisig Script Execution

Remaining Script	Stack State
OP_0 <Sig1> <Sig2> OP_2 <PubKey1> <PubKey2> <PubKey3> OP_3 OP_CHECKMULTISIG	
OP_2 <PubKey1> <PubKey2> <PubKey3> OP_3 OP_CHECKMULTISIG	**<Sig2>** **<Sig1>** **<Empty Array>**
OP_CHECKMULTISIG	**3** **<PubKey3>** **<PubKey2>** **<PubKey1>** **2** **<Sig2>** **<Sig1>** **<Empty Array>**
	True/False

Smart Contracts

Smart Contracts

- Computer protocols which help execution/enforcement of regular contracts
- Minimize trust between interacting parties
- Hypothetical example: Automatic fine for noise pollution
  - Campus community hall parties use loudspeakers
  - Party organizers pay bitcoin security deposit
  - If noise rules violated, deposit distributed to nearby residents
- Two actual examples
  - Escrow
  - Micropayments

Smart Contracts

Escrow

Problem Setup

- Alice wants to buy a rare book from Bob
- Alice and Bob live in different cities
- Bob promises to ship the book upon receiving Bitcoin payment
- Alice does not trust Bob
- Alice proposes an escrow contract involving a third party Carol

Escrow Contract

- Alice requests public keys from Bob and Carol
- Alice pays *x* bitcoins to a 2-of-3 multisig output

```
OP_2 <PubKeyA> <PubKeyB> <PubKeyC> OP_3 OP_CHECKMULTISIG
```

- Bob ships book once Alice's transaction is confirmed
- Bitcoins can be spent if **any two of the three** provide signatures
- Any of the following scenarios can occur
  - Alice receives book.
    Alice and Bob sign.
  - Alice receives the book but refuses to sign.
    Bob provides proof of shipment to Carol.
    Bob and Carol sign.
  - Bob does not ship the book to Alice.
    Bob refuses to sign refund transaction.
    Alice and Carol sign.
- Escrow contract fails if Carol colludes with Alice or Bob
- Also proof of shipment is not proof of contents

Smart Contracts
Micropayments

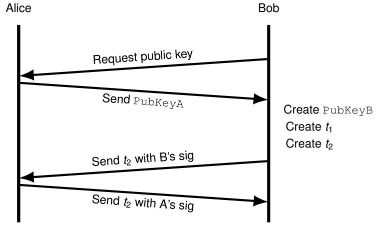
Problem Setup

- Bitcoin transaction fees make small payments expensive
- Micropayments contract can aggregate small payments
- Alice offers proofreading and editing services online
- She accepts bitcoins as payments
- Clients email documents to Alice
- Alice replies with typos and grammatical errors
- Alice charges a fixed amount of bitcoins per edited page
- To avoid clients refusing payment, Alice uses micropayments contract
- Suppose Bob wants a 100 page document edited
- Alice charges 0.0001 BTC per page
- Bob expects to pay a maximum of 0.01 BTC to Alice

Micropayments Contract (1/3)

Creating Refund Transaction

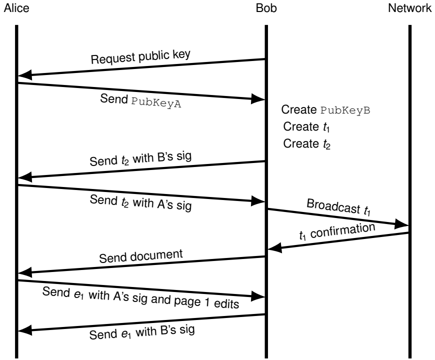
- Bob requests a public key from Alice
- Bob creates a transaction t_1 which transfers 0.01 bitcoins to a 2-of-2 multisig output
- Bob does not broadcast t_1 on the network
- Bob creates a refund transaction t_2 which refunds the 0.01 BTC
- A relative lock time of n days is set on t_2
- Bob includes his signature in t_2 and sends it to Alice
- If Alice refuses to sign, Bob terminates the contract
- If Alice signs t_2 and gives it Bob, he has the refund transaction



Micropayments Contract (2/3)

Getting Paid for First Page Edits

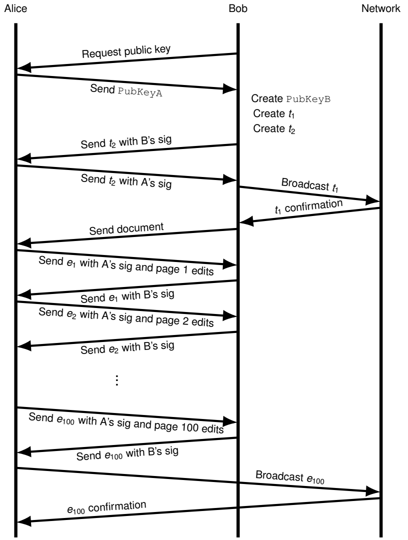
- Bob broadcasts t_1 on the network
- Once t_1 is confirmed, he sends Alice his document
- Alice edits only the first page of the document
- She creates a transaction e_1 which unlocks t_1 and pays her 0.0001 BTC and 0.0099 BTC to Bob
- Alice signs e_1 and sends it to Bob along with the first page edits
    - If Bob refuses to sign e_1, then
        - Alice terminates the contract.
        - Bob broadcasts t_2 after lock time expires
    - If Bob signs e_1 and returns it to Alice, then Alice is guaranteed 0.0001 bitcoins if she broadcasts e_1 before lock time on t_2 expires.



Micropayments Contract (3/3)

Getting Paid for Second Page, Third Page . . .

- Alice edits the second page of the document
- She creates a transaction e_2 which unlocks t_1 and pays her 0.0002 BTC and 0.0098 BTC to Bob
- Alice signs e_2 and sends it to Bob along with the second page edits
  - If Bob refuses to sign e_2, then Alice terminates the contract. Alice broadcasts e_1 and receives 0.0001 BTC.
  - If Bob signs e_2 and returns it to Alice, then Alice is guaranteed 0.0002 bitcoins if she broadcasts e_2 before lock time on t_2 expires.
- Alice continues sending edited pages along with transactions requesting cumulative payments
- She has to finish before the refund transaction lock time expires



Key Takeaways

- Smart contracts reduce the need for trust
- Bitcoin's scripting language enables some smart contracts
- Not powerful enough to express complex contracts

Bitcoin Learning Resources

- Code `https://github.com/bitcoin/bitcoin/`
- Reddit `https://www.reddit.com/r/Bitcoin/`
- Stackoverflow `https://bitcoin.stackexchange.com/`
- Forum `https://bitcointalk.org/`
- IRC `https://en.bitcoin.it/wiki/IRC_channels`
- Books
  - Princeton book `http://bitcoinbook.cs.princeton.edu/`
  - *Mastering Bitcoin*, Andreas Antonopoulos
- Notes
  - `https://www.ee.iitb.ac.in/~sarva/bitcoin.html`

Thanks for your attention

Questions/Comments?

Email: `sarva@ee.iitb.ac.in`
Social: `https://about.me/sv1`