

Bitcoin Transactions

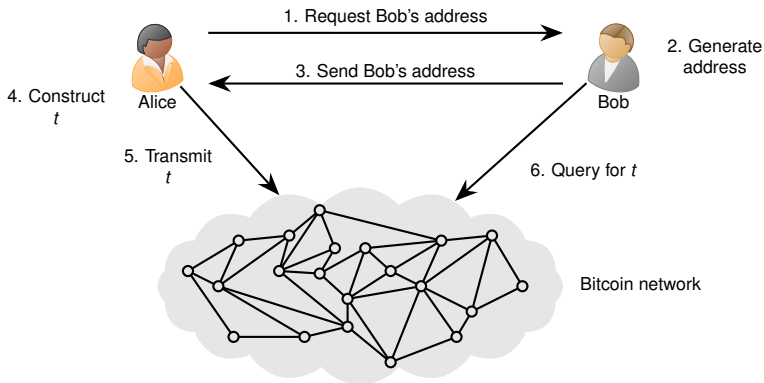
Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

August 3, 2018

Bitcoin Transactions

Bitcoin Payment Workflow



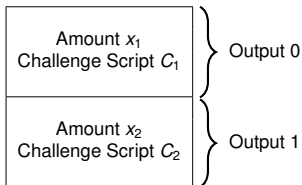
- Merchant Bob shares address out of band (not using Bitcoin P2P)
- Customer Alice broadcasts transaction t which pays the address
- Miners collect broadcasted transactions into a candidate block
- One of the candidate blocks containing t is mined
- Merchant waits for confirmations on t before providing goods

Coinbase Transaction Format

Block Format

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

Coinbase Transaction

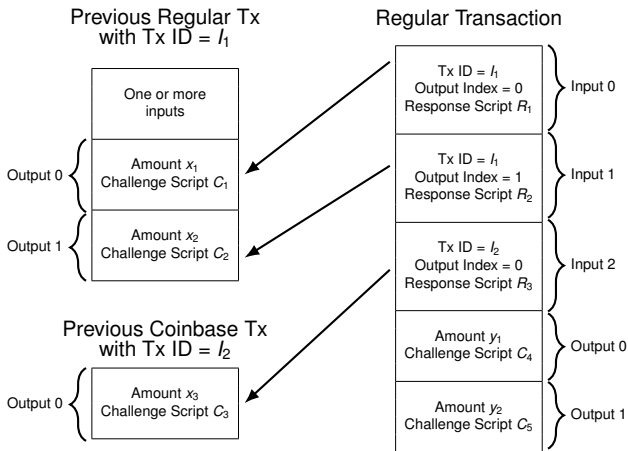


Output Format

nValue
scriptPubkeyLen
scriptPubkey

- 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



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



OP_ADD



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

OP_HASH256 0x20 $\underbrace{\text{<256-bit string>}}_S$ **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

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

Remaining Script

Stack State

<Signature> <Public Key> OP_CHECKSIG

--

<Public Key> OP_CHECKSIG

<Signature>

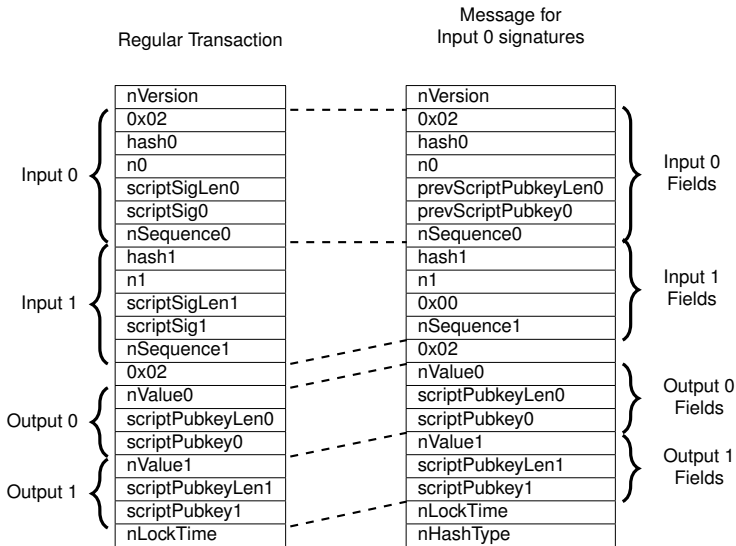
OP_CHECKSIG

<Public Key>
<Signature>

True/False

--

Signatures Protect Transactions



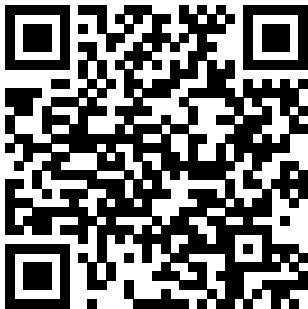
Key Takeaways

- Coinbase transactions have no inputs; outputs have challenge scripts
- Regular transaction inputs unlock previous outputs; outputs again have challenge scripts
- Scripts are expressed in a stack-based language
- Signatures prevent tampering of unconfirmed transactions

Bitcoin Addresses

Bitcoin Addresses

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



- Bitcoin payment workflow (recap)
 - Merchant shares address out of band (not using Bitcoin P2P network)
 - Customer transmits transaction which pays the address
 - Merchant waits for transaction confirmations before providing goods/service

Base58 Encoding

1EHNa6Q4Jz2uvNExL497mE43ikXhwF6kZm



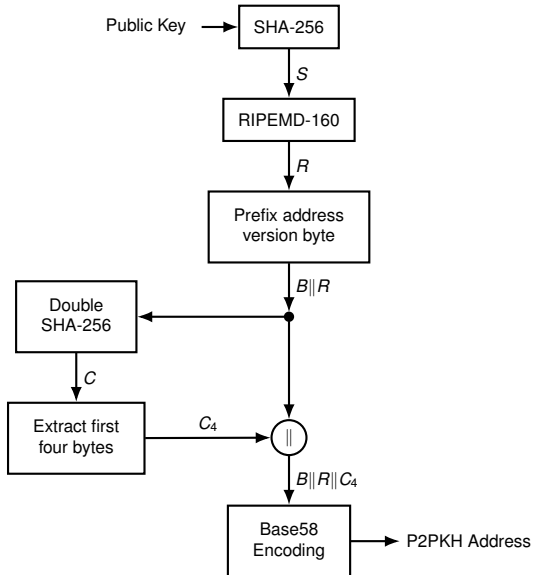
0091B24BF9F5288532960AC687ABB035127B1D28A50074FFE0

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

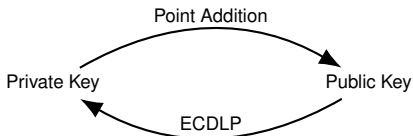
Ch	Int	Ch	Int	Ch	Int	Ch	Int	Ch	Int	Ch	Int
1	0	A	9	K	18	U	27	d	36	n	45
2	1	B	10	L	19	V	28	e	37	o	46
3	2	C	11	M	20	W	29	f	38	p	47
4	3	D	12	N	21	X	30	g	39	q	48
5	4	E	13	P	22	Y	31	h	40	r	49
6	5	F	14	Q	23	Z	32	i	41	s	50
7	6	G	15	R	24	a	33	j	42	t	51
8	7	H	16	S	25	b	34	k	43	u	52
9	8	J	17	T	26	c	35	m	44	v	53

- 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

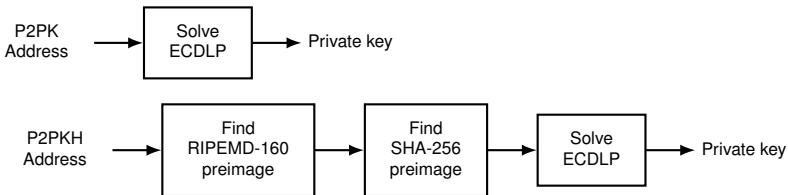
Pay to Public Key Hash Address



Why Hash the Public Key?



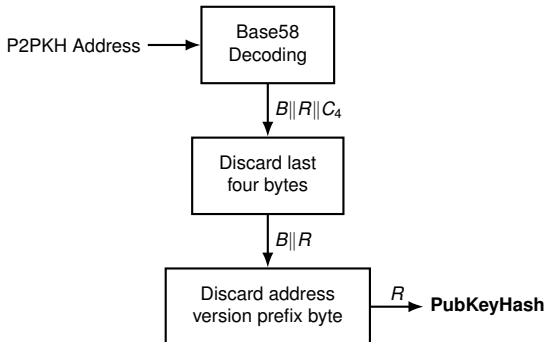
- ECDLP = Elliptic Curve Discrete Logarithm Problem
- ECDLP currently hard but no future guarantees
- Hashing the public key gives extra protection



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

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

--

Pay to Script Hash Script

- Specify arbitrary scripts as payment destinations
- Challenge script

OP_HASH160 <RedeemScriptHash> OP_EQUAL

- Response script

<Response To Redeem Script> <Redeem Script>

- Example

- 1-of-2 Multisig Challenge Script

OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG

- 1-of-2 Multisig Response Script

OP_0 <Sig1> or OP_0 <Sig2>

- P2SH Multisig challenge script

OP_HASH160 <RedeemScriptHash> OP_EQUAL

- P2SH Multisig response script

<u>OP_0 <Sig1></u>	<u>OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG</u>
Response to Redeem Script	Redeem Script

P2SH Multisig Script Execution (1/2)

Remaining Script

Stack State

OP_0 <Sig1>
<OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG>
OP_HASH160 <RedeemScriptHash> OP_EQUAL

--

<OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG>
OP_HASH160 <RedeemScriptHash> OP_EQUAL

<Sig1>
<Empty Array>

OP_HASH160 <RedeemScriptHash> OP_EQUAL

OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG
<Sig1>
<Empty Array>

<RedeemScriptHash> OP_EQUAL

<RedeemScriptHashCalc>
<Sig1>
<Empty Array>

OP_EQUAL

<RedeemScriptHash>
<RedeemScriptHashCalc>
<Sig1>
<Empty Array>

P2SH Multisig Script Execution (2/2)

Remaining Script

Stack State

OP_1 <PubKey1> <PubKey2> OP_2 OP_CHECKMULTISIG

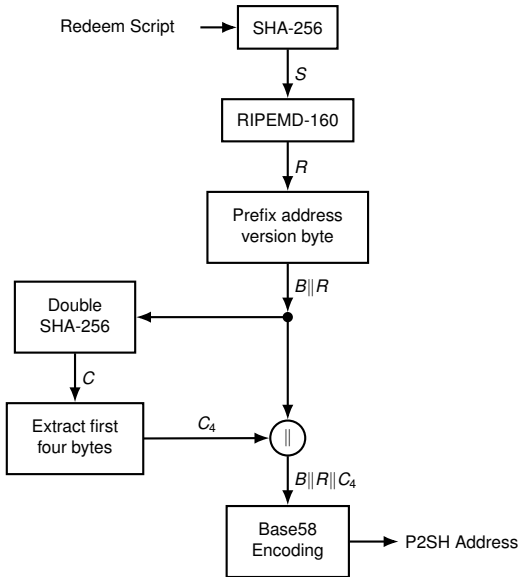
<Sig1>
<Empty Array>

OP_CHECKMULTISIG

2
<PubKey2>
<PubKey1>
1
<Sig1>
<Empty Array>

True/False

Pay to Script Hash Address



Key Takeaways

- Bitcoin addresses are shared over the Internet
- Transactions paying these addresses are broadcast on the Bitcoin network
- P2PKH addresses are obtained by hashing public keys
- Signatures created using private keys unlock P2PKH outputs
- P2SH addresses are obtained by hashing scripts
- Unlocking P2SH outputs requires both redeem script and valid response to it

References

- Chapter 5 of *An Introduction to Bitcoin*, S. Vijayakumaran,
www.ee.iitb.ac.in/~sarva/bitcoin.html