Bitcoin Smart Contracts

Saravanan Vijayakumaran
sarva@ee.iitb.ac.in

Department of Electrical Engineering
Indian Institute of Technology Bombay

August 3, 2018

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
  - IITB hillside 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

Escrow Contract

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

Micropayments

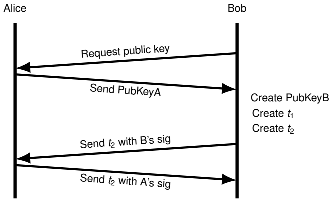
Problem Setup

- 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

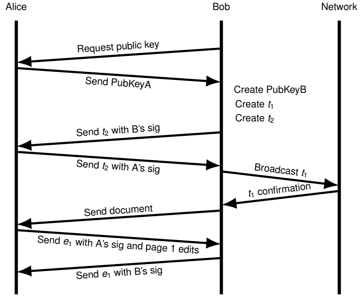
- Bob requests a public key from Alice
- Bob creates a transaction t_1 which transfers 0.01 bitcoins to a 2-of-2 multisig output
- Bob does not broadcast t_1 on the network
- Bob creates a refund transaction t_2 which refunds the 0.01 BTC
- A relative lock time of n days is set on t_2
- Bob includes 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

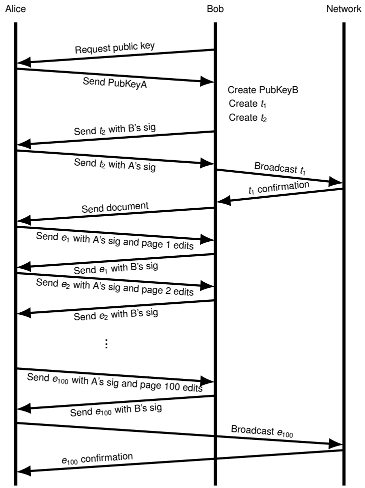
- Bob broadcasts t_1 on the network
- Once t_1 is confirmed, he sends Alice his document
- Alice edits only the first page of the document
- She creates a transaction e_1 which unlocks t_1 and pays her 0.0001 BTC and 0.0099 BTC to Bob
- Alice signs e_1 and sends it to Bob along with the first page edits
  - 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



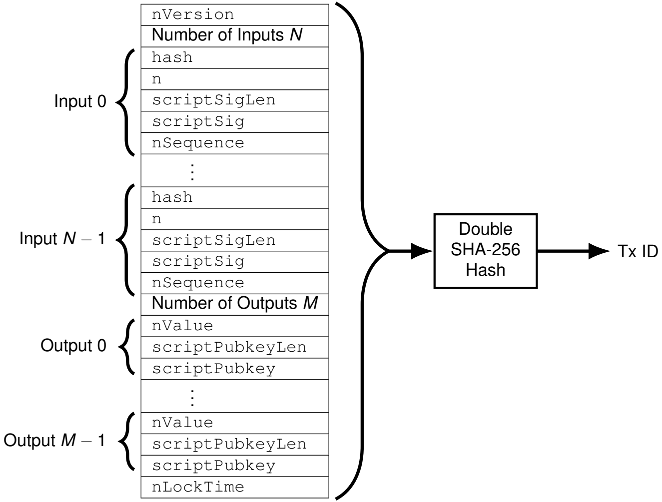
Key Takeaways

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

SegWit for Safer Contracts

Transaction ID

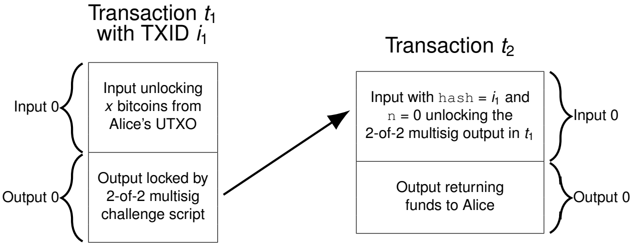
Regular Transaction

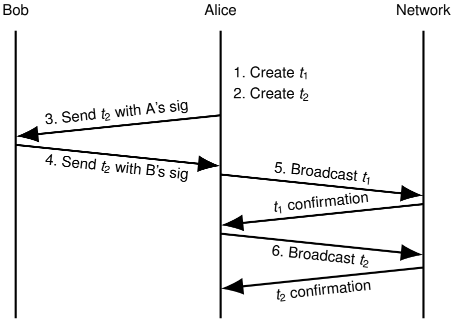


Refund Protocol

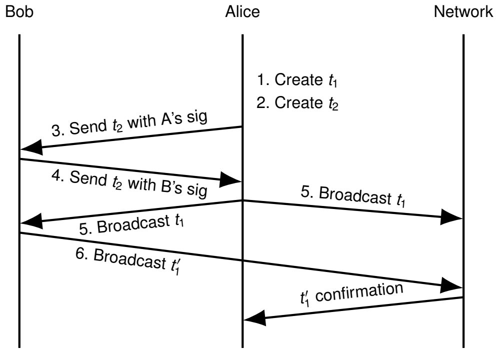
- Alice wants to teach Bob about transactions
- Bob does not own any bitcoins
- Alice decides to transfer some bitcoins to Bob
- Alice does not trust Bob
- She wants to ensure refund

Refund Protocol





Exploiting Transaction Malleability



- If (r, s) is a valid ECDSA signature, so is $(r, n - s)$
- The t_1' transaction cannot be spent by t_2
- SegWit = Segregated Witness
- Solves problems arising from transaction malleability

SegWit Standard Scripts

- Pay to Witness Public Key Hash (P2WPKH)
- Pay to Witness Script Hash (P2WSH)
- Both can be embedded in a P2SH template
  - P2SH-P2WPKH
  - P2SH-P2WSH

Pay to Witness Public Key Hash

```
scriptPubkey:   OP_0 0x14 <PubKeyHash>,
   scriptSig:   (empty),
scriptWitness:  <Signature> <Public Key>.
```

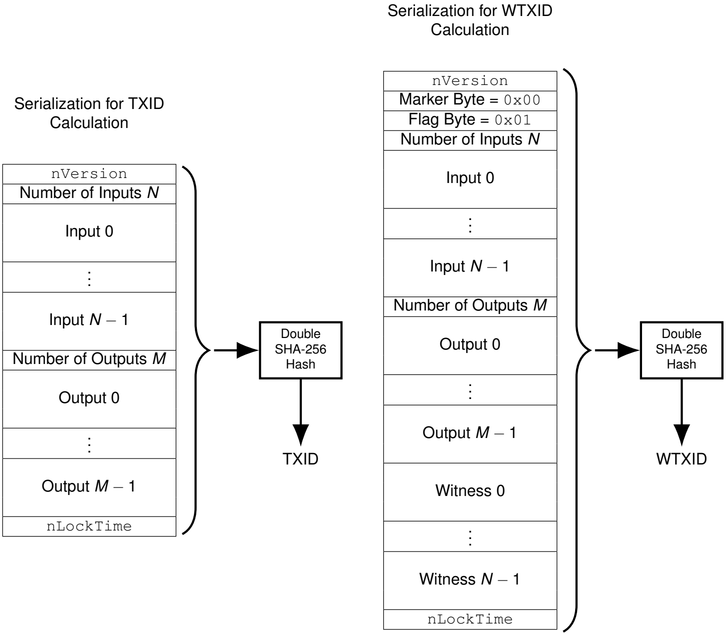
- Challenge script is 22 bytes long
- First byte indicates script version number
- `scriptSig` does not contain signatures
- `scriptWitness` is sent only to SegWit-capable nodes

P2WPKH Execution by pre-SegWit Nodes

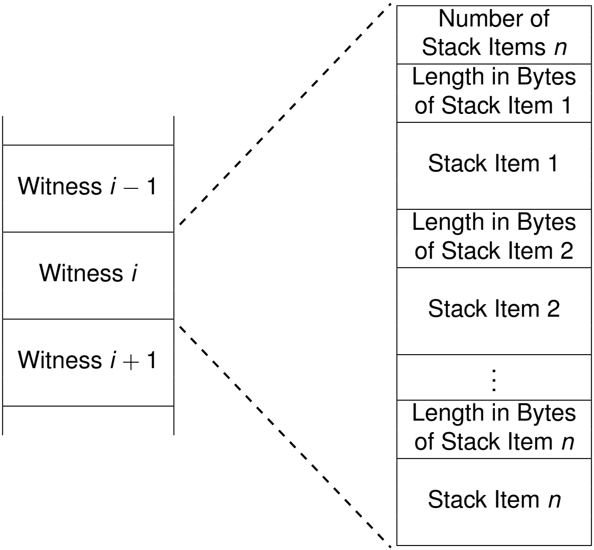
Remaining Script	Stack State
<Empty Response Script> OP_0 0x14 <PubKeyHash>	
OP_0 0x14 <PubKeyHash>	
0x14 <PubKeyHash>	**<Empty Array>**
	<PubKeyHash> **<Empty Array>**

- P2WPKH outputs look like **anyone-can-spend outputs** to pre-SegWit nodes
- SegWit-capable nodes take the response from scriptWitness
- If majority of hashpower follows SegWit rules, output is not anyone-can-spend

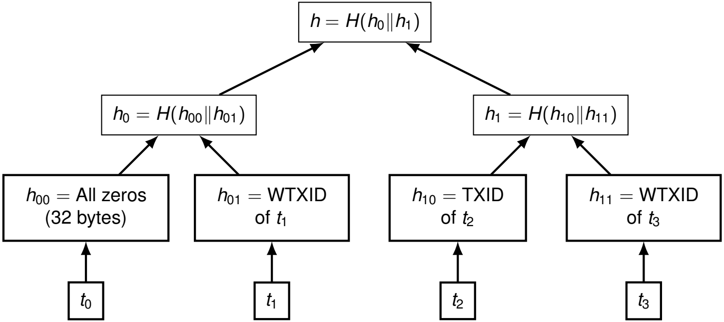
TXID and WTXID Calculations



Witness Format



WTXID Merkle Tree



- WTXID of coinbase transaction is fixed to all zeros
- For non-SegWit transactions, WTXID = TXID
- Witness root hash is stored in a coinbase null data output

$$\texttt{OP\_RETURN 0x24 } \underbrace{\texttt{0xAA21A9ED}}_{\substack{\text{Commitment}\\\text{header}}} \; \underbrace{H(\text{Witness root hash} \| \text{Witness reserved value})}_{\text{Commitment hash (32 bytes)}}$$

SegWit Coinbase Transaction

Coinbase Transaction Format

`nVersion`
Number of Inputs = 1
Dummy Input
Number of Outputs = 2
Output 0 (P2PKH output)
Output 1 (Null data output)
Witness 0
`nLockTime`

Witness Commitment Output

`nValue` = `0`
`scriptPubkeyLen` = `0x26`
`scriptPubkey` = `OP_RETURN 0x24 0xAA21A9ED` `<32-byte Commitment Hash>`

Witness Structure Storing Reserved Value

Number of Stack Items = `0x01`
Length in Bytes of Stack Item 1 = `0x20`
Stack Item 1 = 32-byte witness reserved value

SegWit Block Size Increase

- Before SegWit, the maximum block size was 1 MB = 10^6 bytes
- Let *base size* be block size using pre-SegWit serialization of transactions
- Let *total size* be block size using pre-SegWit serialization of transactions
- Define the *block weight* as follows

$$\text{Block Weight} = 3 \times \text{Base Size} + \text{Total Size}.$$

- SegWit imposes the restriction

$$\text{Block Weight} \leq 4 \text{ MB} = 4 \times 10^6 \text{ bytes}$$

- As total size = base size + witness size, we get

$$\text{Base Size} + \frac{\text{Witness Size}}{4} \leq 1 \text{ MB}.$$

- Base size cannot exceed 1 MB irrespective of witness size

Key Takeaways

- SegWit introduced to solve the transaction malleability issue
- Backward compatibility requirement resulted in complicated design
- Successful deployment required the majority of the hashpower to follow SegWit rules

References

- Chapters 5, 6 of *An Introduction to Bitcoin*, S. Vijayakumaran, www.ee.iitb.ac.in/~sarva/bitcoin.html

Bitcoin Learning Resources

- Code `https://github.com/bitcoin/bitcoin/`
- Reddit `https://www.reddit.com/r/Bitcoin/`
- 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`