Evolution of Digital Signatures In Bitcoin - 3as

Nicolas T. Courtois

- **U**niversity **C**ollege **L**ondon, UK

*Introducing Bitcoin

Bitcoin In A Nutshell

- bitocoins are cryptographic tokens
  - stored by people on their PCs or mobile phones
- ownership is achieved through digital signatures:
  - you have a certain cryptographic key, you have the money.
  - publicly verifiable, only one entity can sign

- consensus-driven, a distributed system which has no central authority
  - but I will not claim it is decentralized, this is simply not true!
  - a major innovation is that financial transactions CAN be executed and policed without trusted authorities. Bitcoin is a sort of financial cooperative or a distributed business.
- based on self-interest:
  - a group of some 100 K people called bitcoin miners own the bitcoin “infrastructure” which has costed about 0.5-1 billion dollars (estimation)
  - they make money from newly created bitcoins and fees
  - at the same time they approve and check the transactions.
  - a distributed electronic notary system

Two Key Concepts

- initially money are attributed through **Proof Of Work (POW)** to one public key A
  - to earn bitcoins one has to “work” (hashing) and consume energy (pay for electricity)

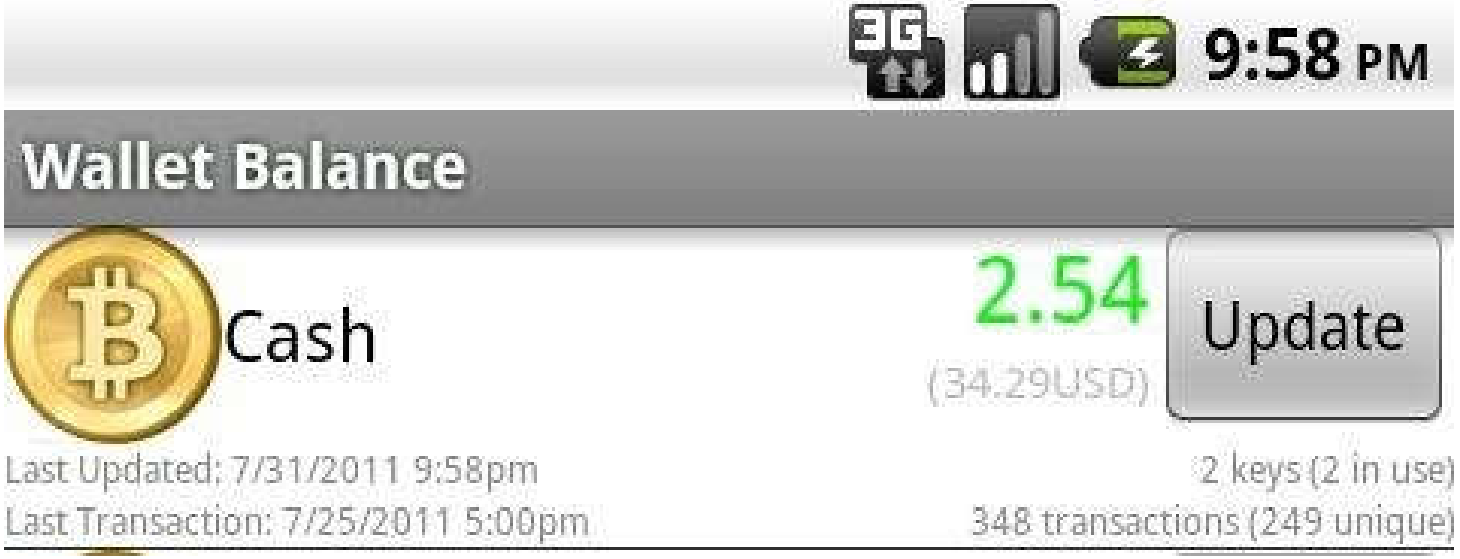
- money transfer from public key A to H/public key B:
  - **like signing a CHEQUE validated by a notary/miner which confirms the signature inside a block**,
  - multiple confirmations: another notary will re-confirm it, then another, etc…
  - we do NOT need to assume that ALL these notaries are honest.
    - at the end it becomes costly to cheat

In Practice


11:00
KEYS
BALANCE
TRANSACTIONS
1Lke9VyQHixh
8zxteh7mENSr
4wsX5N6BeN
249.9 mBTC
~USD 184.43
Receiving 25 mBTC
Send
Receive
1 BTC ~ USD 738.00 (Bitstamp)

Wallets

- Wallet: = def=
  aBitcoin client App



Block Chain

Bitcoin Mining

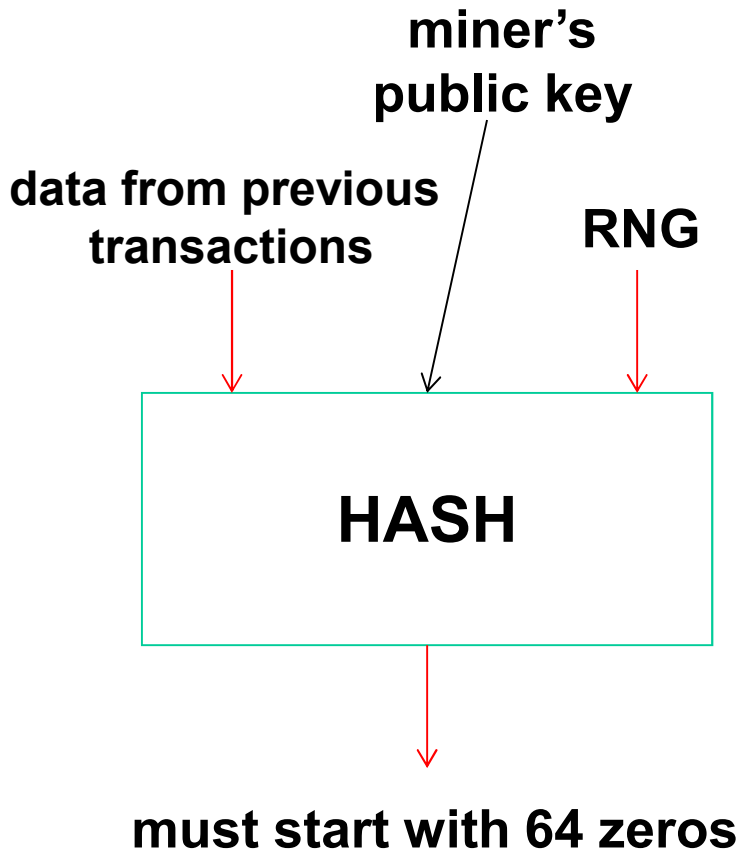
- Minting: creation of new currency.
- Confirmation+re-confirmation of older transactions

Random Oracle – like mechanism

Ownership:

- "policed by majority of miners":
- only the owner can transfer

12.5 BTC produced.



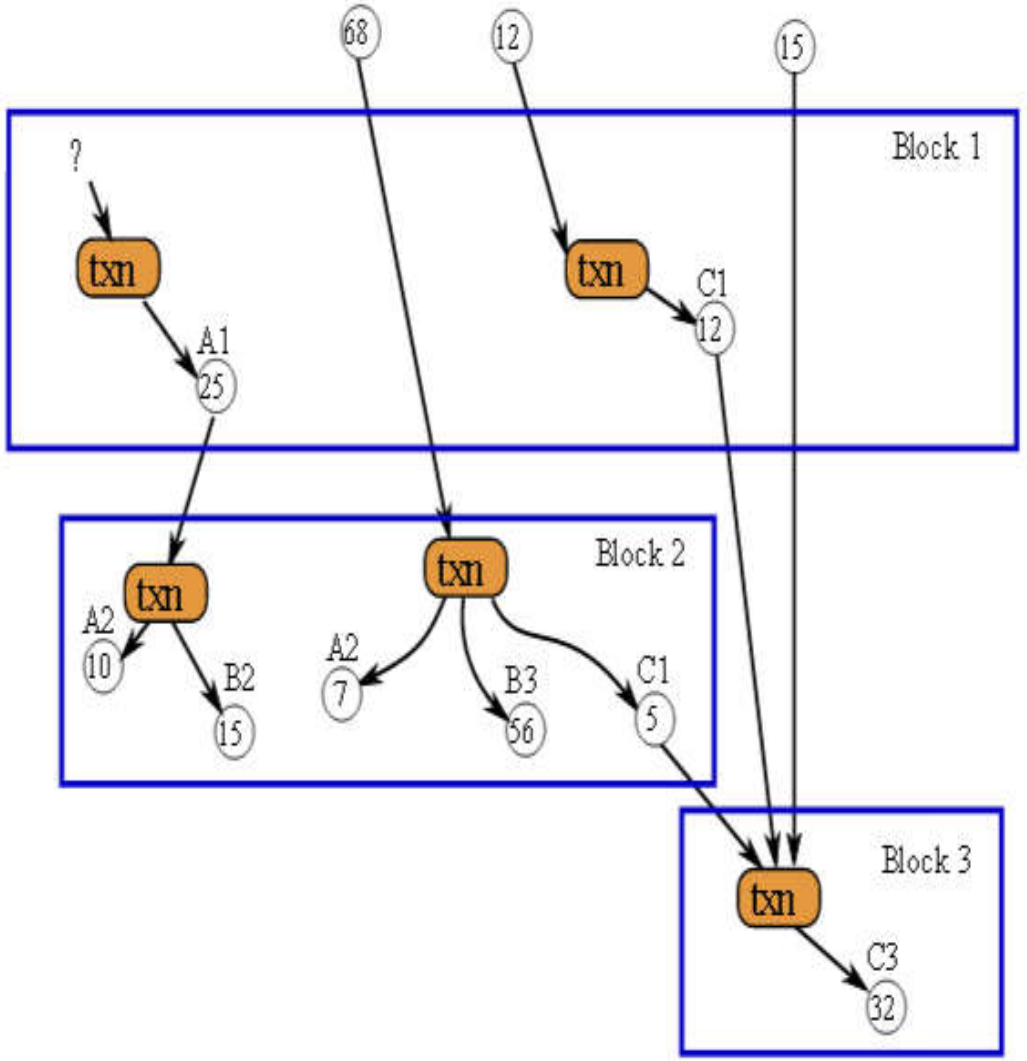
Block Chain

Def:

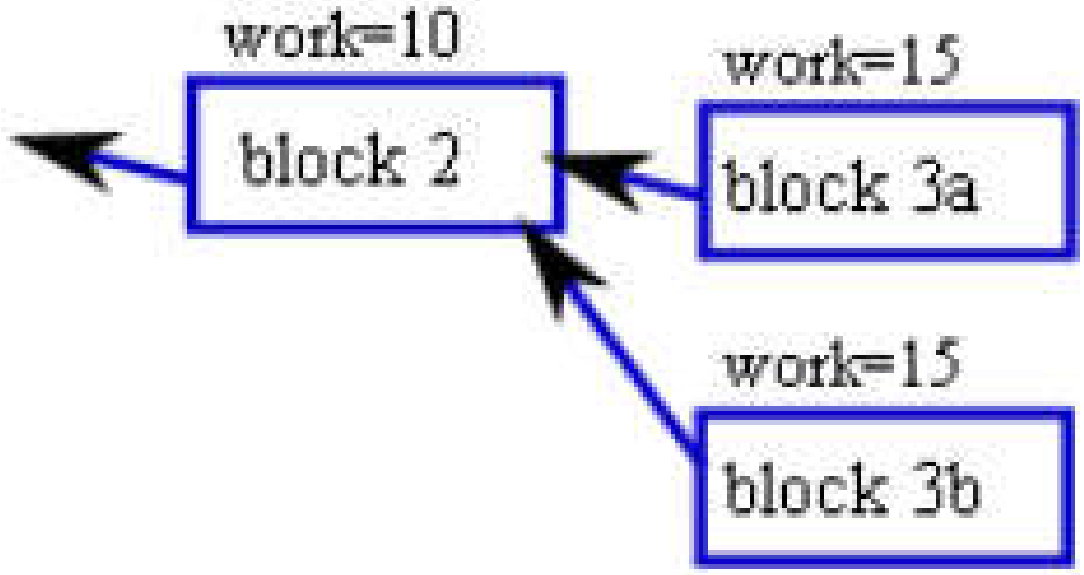
A transaction database shared by everyone.

Also a ledger.

Every transaction since ever is public.

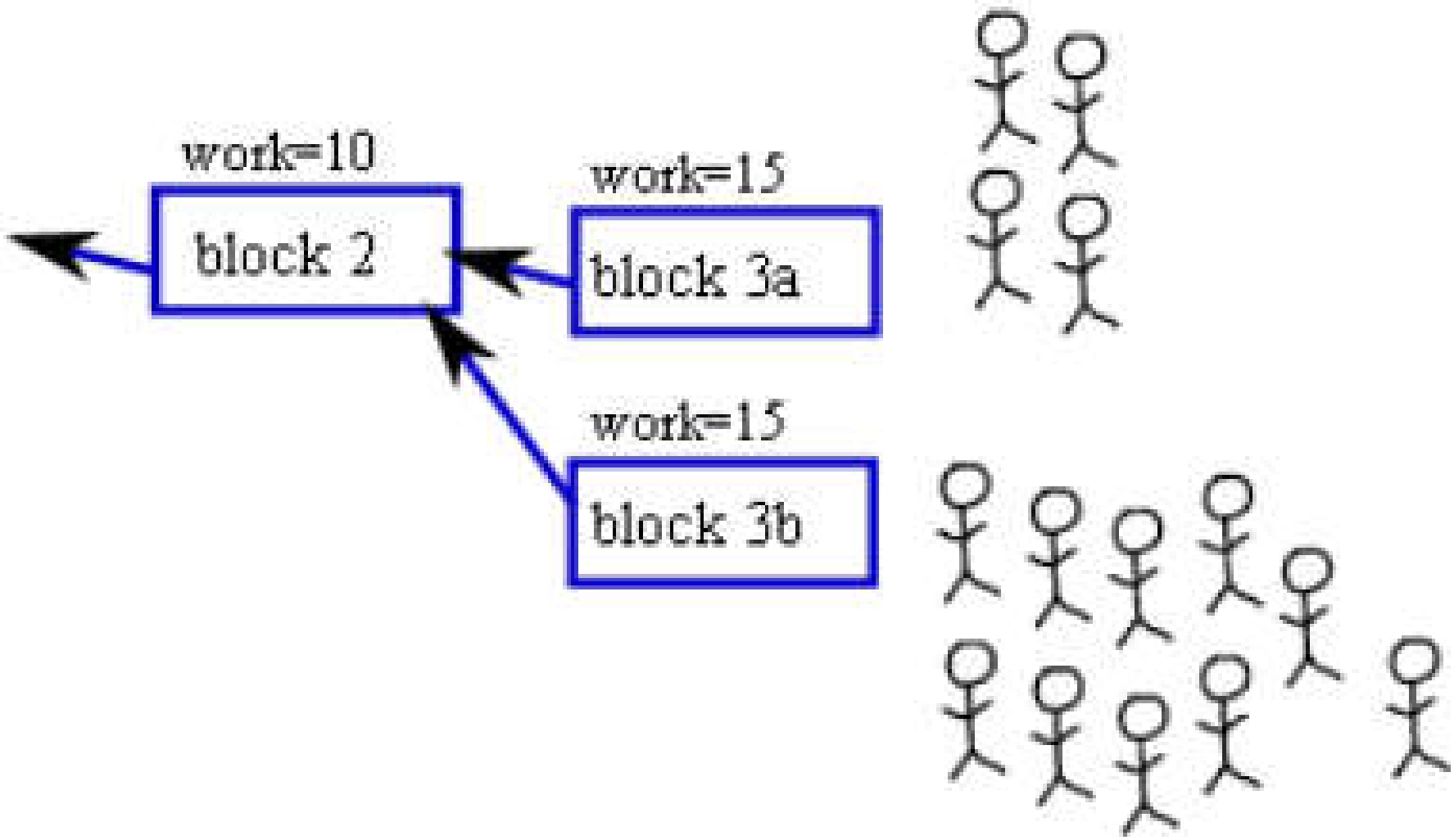


Fork – Hard To Avoid, 1% of the time



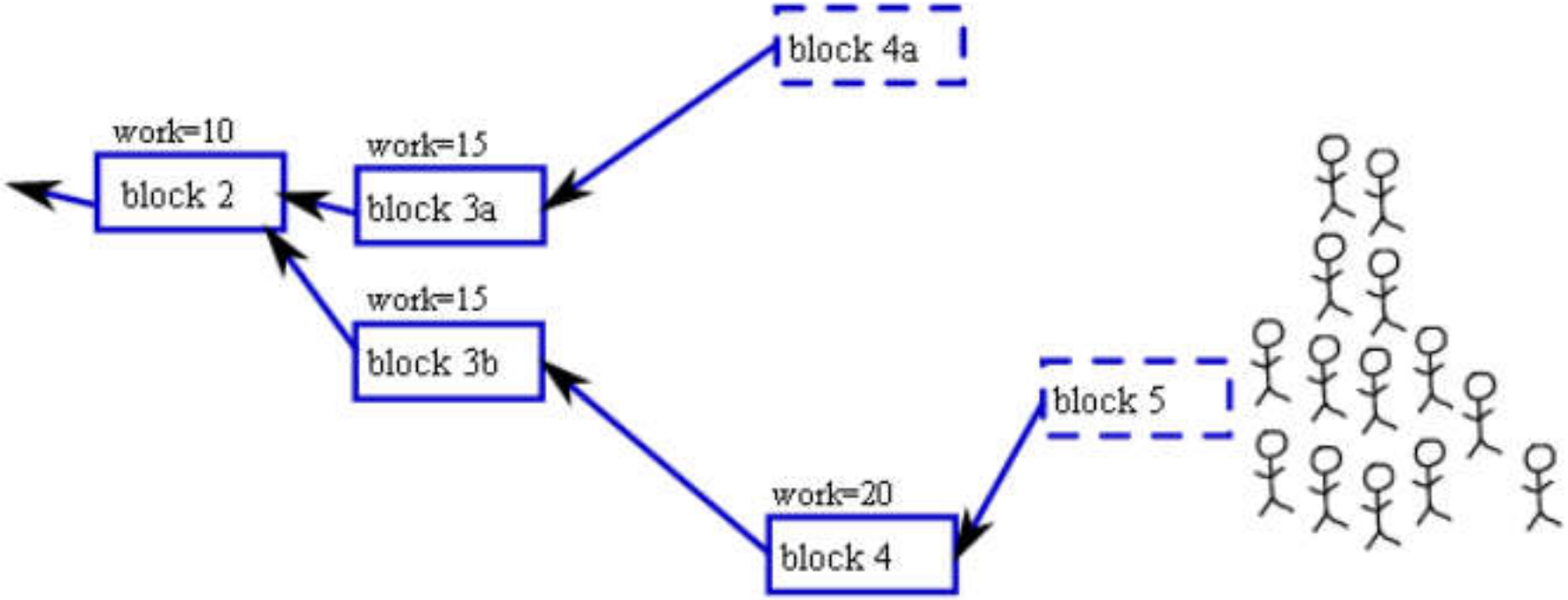
blocks	*wasted*
less than 140,000	0.00%
140,000-149,999	0.21%
150,000-159,999	0.27%
160,000-169,999	1.01%
170,000-179,999	1.77%
180,000-189,999	1.71%
190,000-199,999	1.15%
200,000-209,999	0.88%
210,000-219,999	1.05%
220,000-229,999	1.28%
230,000-239,999	0.78%
240,000-249,999	0.43%
250,000-259,999	0.67%
260,000-now	0.91%

Fork – Miners Mine On Both Branches



Longest Chain Rule – Clear Winner

"1 ASIC 1 vote"



Bitcoin Address

Ledger-Based Currency

A “Bitcoin Address” = a sort of equivalent of a bank account.

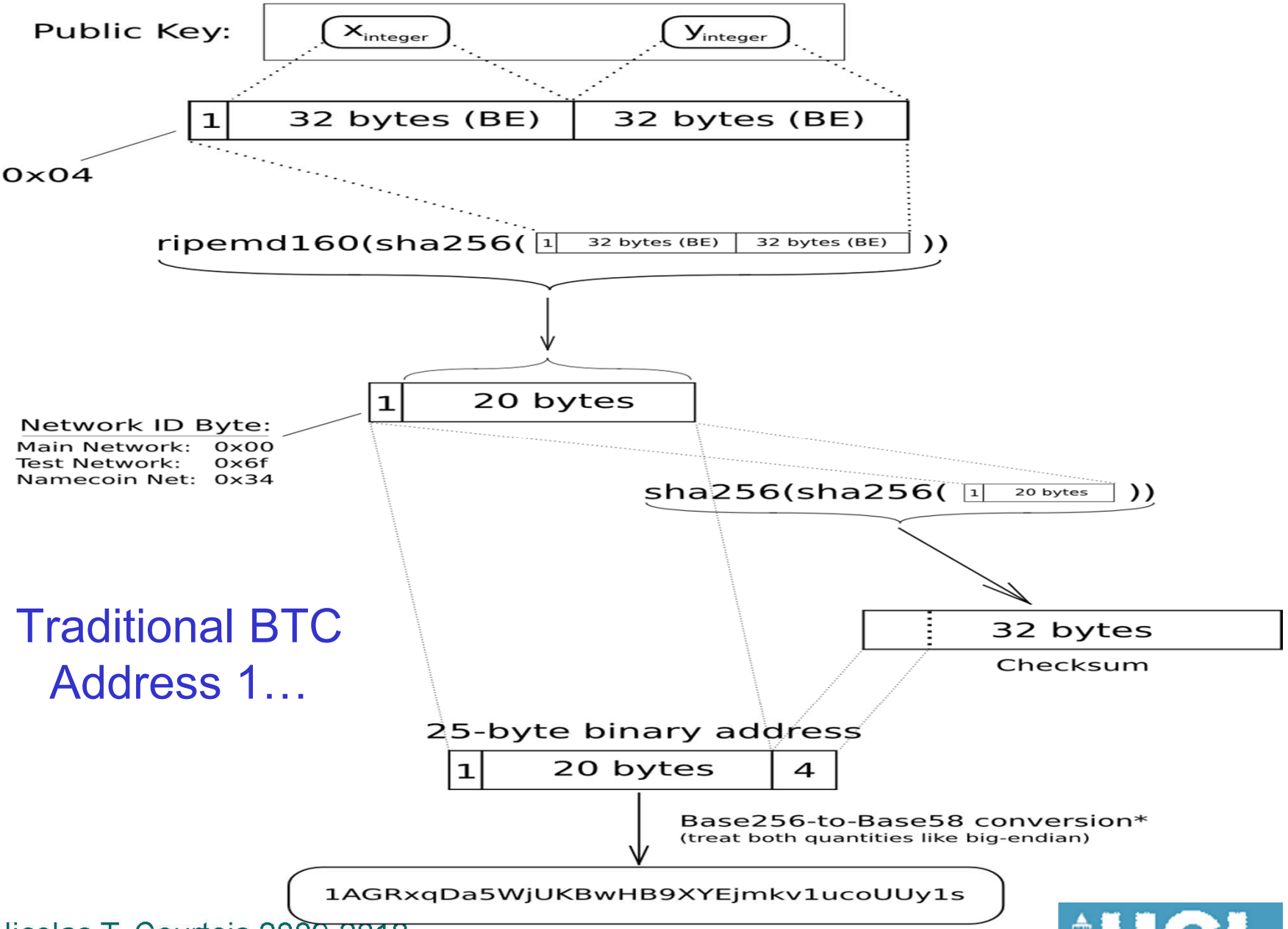
Three traditional formats.

- First format like full Pkey 2*32 byte points, redundant!
  "scriptPubKey":"04a39b9e4fbd213ef24bb9be69de4a118dd0644082e47c01fd9159d38637b83fbcdc115a5d6e970586a012d1cfe3e3a8b1a3d04e763bdc5a071c0e827c0bd834a5 OP_CHECKSIG“
- Hash it on 160 bits, conceals the PK key! (more PQ secure).
  - e.g. 0568015a9facccfd09d70d409b6fc1a5546cecc6
- Recode with checksum on 1+20+4 bytes checksum, 160+32 bits,
  - Base58: 1VayNert3x1KzbpzMGt2qdqrAThiRovi8 27-34 chars

PK itself remains confidential until some part is spent.

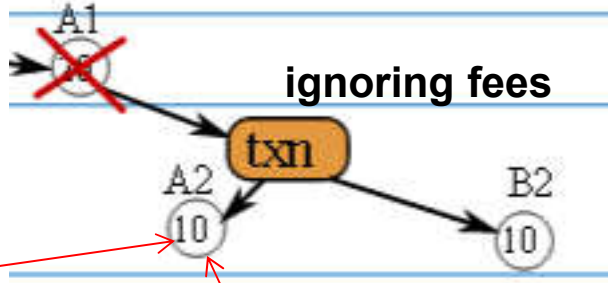
SK = private key is always kept private, allows transfer of funds.

Elliptic-Curve Public Key to BTC Address conversion



Traditional BTC Address 1…

Attributions



DEFINITION

"Attribution to PK" =

act of an owner of

a previous attribution (always destroyed)

which transfers a certain amount to the new PK = A2

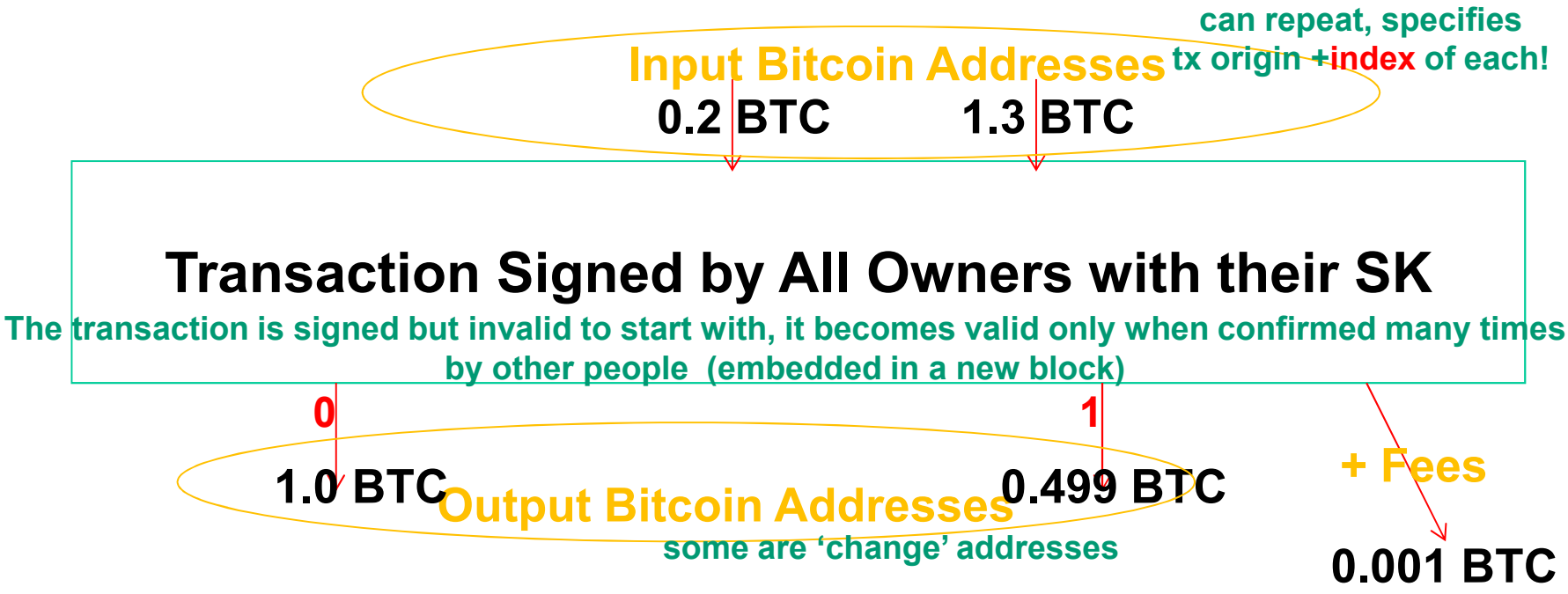
(using a digital signature)

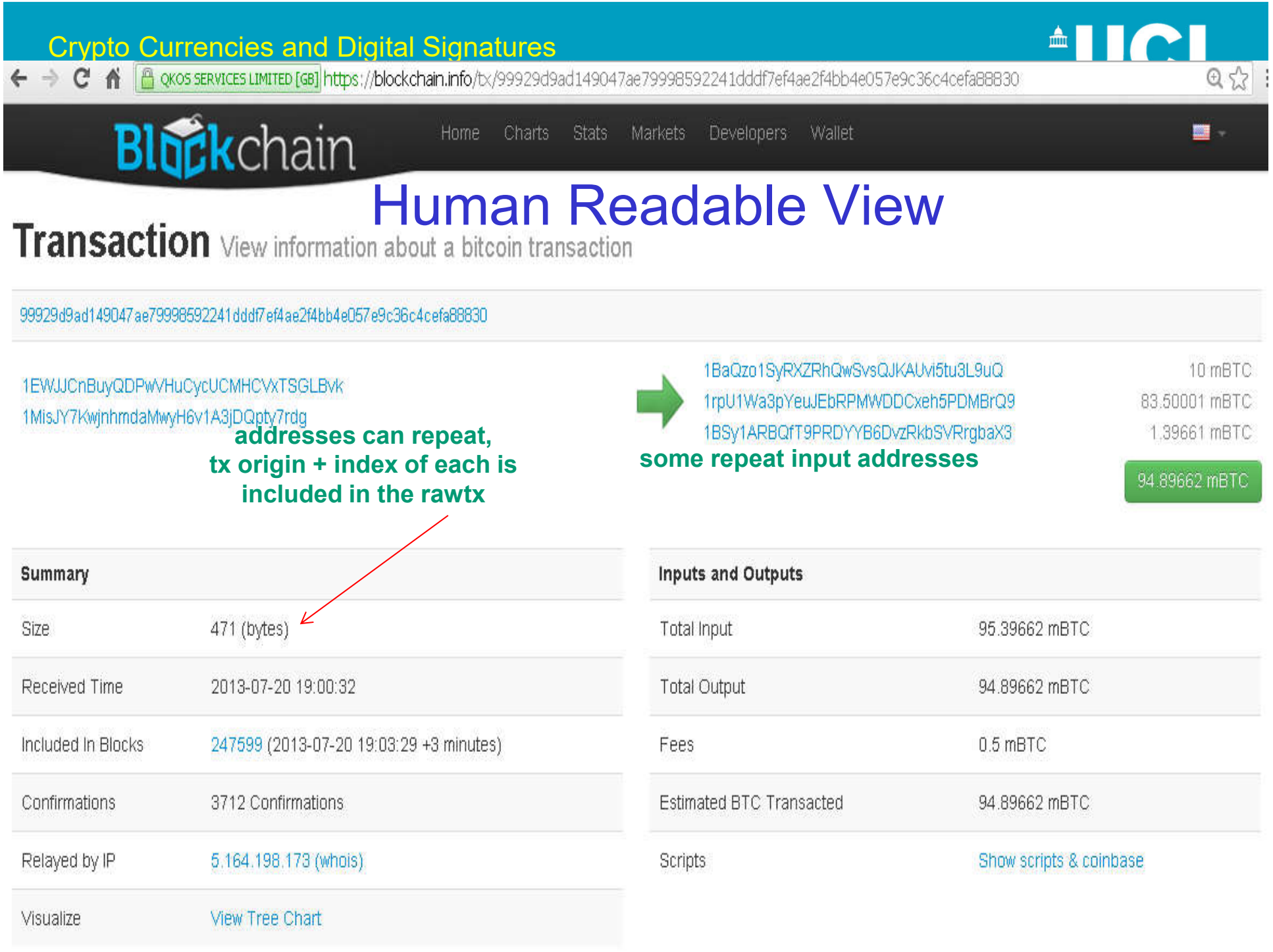
Caveat: Each attribution can be traced back to the initial mining event.

Typical Bitcoin Transfer

multiple inputs and multiple outputs.

- destroys all current attributions,
- requires everybody's signature




QKOS SERVICES LIMITED [GB] https://blockchain.info/tx/99929d9ad149047ae79998592241dddf7ef4ae2f4bb4e057e9c36c4cefa88830
Blockchain
Home Charts Stats Markets Developers Wallet
Human Readable View
Transaction View information about a bitcoin transaction
99929d9ad149047ae79998592241dddf7ef4ae2f4bb4e057e9c36c4cefa88830
1EWJJCnBuyQDPwVHuCycUCMHCVxTSGLBvk
1MisJY7KwjnhmdaMwyH6v1A3jDQpty7rdg
addresses can repeat,
tx origin + index of each is
included in the rawtx
1BaQzo1SyRXZRhQwSvsQJKAUvi5tu3L9uQ 10 mBTC
1rpU1Wa3pYeuJEbRPMWDDCxeh5PDMBrQ9 83.50001 mBTC
1BSy1ARBQfT9PRDYYB6DvzRkbSVRrgbaX3 1.39661 mBTC
some repeat input addresses
94.89662 mBTC
Summary
Size 471 (bytes)
Received Time 2013-07-20 19:00:32
Included In Blocks 247599 (2013-07-20 19:03:29 +3 minutes)
Confirmations 3712 Confirmations
Relayed by IP 5.164.198.173 (whois)
Visualize View Tree Chart
Inputs and Outputs
Total Input 95.39662 mBTC
Total Output 94.89662 mBTC
Fees 0.5 mBTC
Estimated BTC Transacted 94.89662 mBTC
Scripts Show scripts & coinbase

More Details - rawtx


```
{
  "hash":"9837485da283ce8ceb0570e2950bb65ebacef9ebd97f871da268d73ea79292c4",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":257,
  "in":[
    {
      "prev_out":{
        "hash":"ba250a395cf37e2d112859ec1d4379a605a6fd8e96b406c4f6990labc05d5b47",
        "n":1
      },
      "scriptSig":"304402206dcf0ef7ca4bfa573ed8f3dc94dca42f5ea46827e8885056d3dfede88e52d49b022077055f3d3c125cc
    }
  ],
  "out":[
    {
      "value":"5.00000000",
      "scriptPubKey":"OP_DUP OP_HASH160 dcc1120deb91acda0d3e5774a2b8908e3424f532 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"13.07598401",
      "scriptPubKey":"OP_DUP OP_HASH160 88f1271342d5f2202995c6e74ed07b81caec7633 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
},
```


unique ID on 256 bits = the hash of the whole

list of input attributions:
origin tx, index n, ECDSA signature

list of output attributions

amount BTC

0

1

H(recipient PK)

*Remarks:

About XXX million transactions ever made.

to know the balance of one account, we must “in theory” store ALL the transactions which send money for this address and then check ALL transactions made since then to see some of these are not already spent.

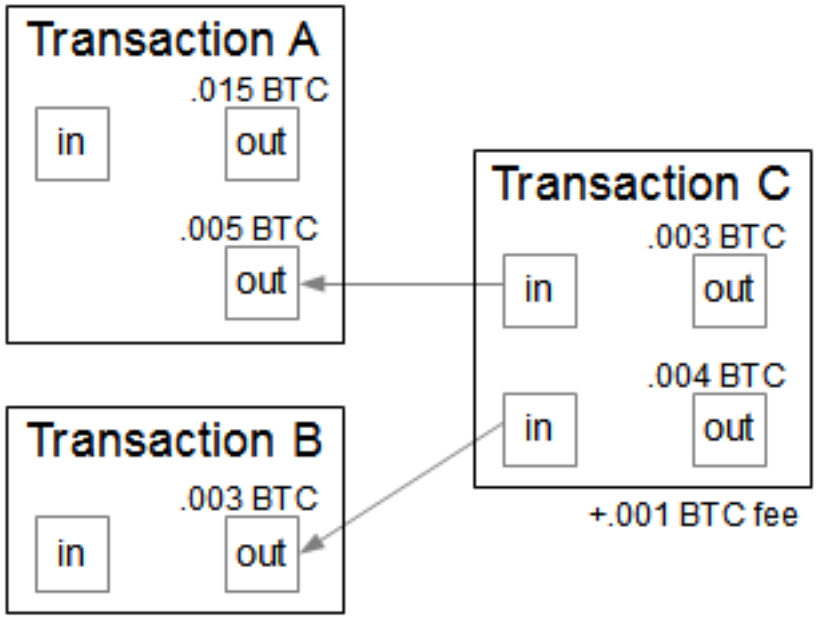
Full bitcoin network nodes stored all transactions ever made and checks their correctness (all the digital signatures).

About 200 Gbytes data, Full download takes weeks.

In practice one could skip checking signatures confirmed by many miners… dangerous though. There is no absolute proof that miners have already checked them (maybe they forgot!).

Transaction Flow

example:



Detailed Encoding of Digital Signatures inside Bitcoin Transaction Scripts

P2PKH=
Pay top PLK Hash

the most common method of spending bitcoins 2009-2017

Bitcoin I/O Scripts


```
{
  "hash":"9837485da283ce8ceb0570e2950bb65ebacef9ebd97f871da268d73ea79292c4",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":257,
  "in":[
    {
      "prev_out":{
        "hash":"ba250a395cf37e2d112859ec1d4379a605a6fd8e96b406c4f69901abc05d5b47",
        "n":1
      },
      "scriptSig":"304402206dcf0ef7ca4bfa573ed8f3dc94dca42f5ea46827e8885056d3dfede88e52d49b022077055f3d3c125cc
    }
  ],
  "out":[
    {
      "value":"5.00000000",
      "scriptPubKey":"OP_DUP OP_HASH160 dcc1120deb91acda0d3e5774a2b8908e3424f532 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"13.07598401",
      "scriptPubKey":"OP_DUP OP_HASH160 88f1271342d5f2202995c6e74ed07b81caec7633 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
},
```


scriptSig: unlocks the previous script in the previous attribution which is spent here

list of output attributions

0

1

H(recipient PK)

script_PK = encodes complex redemption conditions executed to decide when money can be transferred

Typical Example of scriptPubKey

= a script which verifies if one can spend bitcoins

(who and under which conditions)

scriptPubKey

OP_DUP	76
OP_HASH160	a9
PUSHDATA 14	14
public key hash	c8 e9 09 96 c7 c6 08 0e e0 62 84 60 0c 68 4e d9 04 d1 4c 5c
OP_EQUALVERIFY	88
OP_CHECKSIG	ac

len= 25=3+20+2

Bitcoin Signature Scripts =scriptSig


```
{
  "hash":"9837485da283ce8ceb0570e2950bb65ebacef9ebd97f871da268d73ea79292c4",
  "ver":1,
  "vin_sz":1,
  "vout_sz":2,
  "lock_time":0,
  "size":257,
  "in":[
    {
      "prev_out":{
        "hash":"ba250a395cf37e2d112859ec1d4379a605a6fd8e96b406c4f69901abc05d5b47",
        "n":1
      },
      "scriptSig":"304402206dcf0ef7ca4bfa573ed8f3dc94dca42f5ea46827e8885056d3dfede88e52d49b022077055f3d3c125cc
    }
  ],
  "out":[
    {
      "value":"5.00000000",
      "scriptPubKey":"OP_DUP OP_HASH160 dcc1120deb91acda0d3e5774a2b8908e3424f532 OP_EQUALVERIFY OP_CHECKSIG"
    },
    {
      "value":"13.07598401",
      "scriptPubKey":"OP_DUP OP_HASH160 88f1271342d5f2202995c6e74ed07b81caec7633 OP_EQUALVERIFY OP_CHECKSIG"
    }
  ]
},
```


Signature Script: unlocks the previous script in the previous attribution which is spent here

list of output attributions

H(recipient PK)

Typical scriptSig

sign+PKey

len= 1+71+ 1+65 = 138 BUT NOT ALWAYS!

scriptSig

PUSHDATA 47		47
signature (DER)	sequence	30
	length	44
	integer	02
	length	20
	X **r**	2c b2 65 bf 10 70 7b f4 93 46 c3 51 5d d3 d1 6f c4 54 61 8c 58 ec 0a 0f f4 48 a6 76 c5 4f f7 13
	integer	02
	length	20
	Y **s**	6c 66 24 d7 62 a1 fc ef 46 18 28 4e ad 8f 08 67 8a c0 5b 13 c8 42 35 f1 65 4e 6a d1 68 23 3e 82
SIGHASH_ALL		01
PUSHDATA 41		41
public key	type	04
	X	14 e3 01 b2 32 8f 17 44 2c 0b 83 10 d7 87 bf 3d 8a 40 4c fb d0 70 4f 13 5b 6a d4 b2 d3 ee 75 13
	Y	10 f9 81 92 6e 53 a6 e8 c3 9b d7 d3 fe fd 57 6c 54 3c ce 49 3c ba c0 63 88 f2 65 1d 1a ac bf cd

scriptSig1

scriptSig2

*DER Encoding

http://tools.ietf.org/html/rfc3278#section-8.2

One DER-encoded signature is a SEQUENCE(0x30) of 2 large integers:

signature (DER)	sequence	30
	length	44
	integer	02
	length	20
	X	2c b2 65 bf 10 70 7b f4 93 46 c3 51 5d d3 d1 6f c4 54 61 8c 58 ec 0a 0f f4 48 a6 76 c5 4f f7 13
	integer	02
	length	20
	Y	6c 66 24 d7 62 a1 fc ef 46 18 28 4e ad 8f 08 67 8a c0 5b 13 c8 42 35 f1 65 4e 6a d1 68 23 3e 82

Signing/Formatting Problem

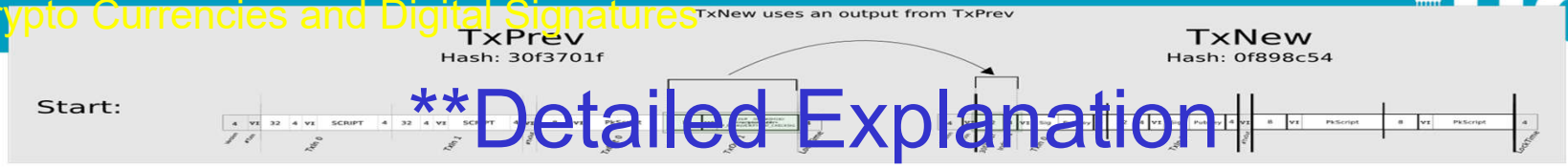
- digital signature can only be verified in spec is exactly followed = exact and correct data formatting etc

- a digital signature cannot sign itself:

  – hash+sign the raw transactions sigs removed/not yet known, 1 script inserted instead

Transaction Verification Steps: OP_CHECKSIG (SIGHASH_ALL only)

**Detailed Explanation



Prepare: Execute TxIn.sigScript to get Sig and Key onto stack, execute TxOut.PkScript up to OP_CHECKSIG

Step 1: Pop public key and signature off the stack: pubKeyStr = stack.pop(), sigStr = stack.pop()

Step 2: From TxPrev.PkScript, create subscript from last OP_CODESEPARATOR to end of script (if no OP_CS, simply copy PkScript)

PrevTx.PkScript: | SCRIPT_PART1 | OP_CODESEPARATOR | SCRIPT_PART2 | OP_CODESEPARATOR | OP_DUP | OP_HASH160 | <recipientAddr> | OP_EQUALVERIFY | OP_CHECKSIG | OP_CODE_SEPARATOR | SCRIPT_PART4 |

Subscript: | OP_DUP | OP_HASH160 | <recipientAddr> | OP_EQUALVERIFY | OP_CHECKSIG | OP_CODE_SEPARATOR | SCRIPT_PART4 |

Step 3: Remove signature from subscript, if present
(Not standard to have a sig in the subscript)

Step 4: Remove OP_CODESEPARATORS from Subscript → | OP_DUP | OP_HASH160 | <recipientAddr> | OP_EQUALVERIFY | OP_CHECKSIG | SCRIPT_PART4 |

Step 5: Extract hashtype from signature:

Before: sigStr = | Signature = 70 bytes (DER) | Hashtype 1 |

After: sigStr = | = 70 bytes (DER) | hashTypeCode = 4 (expand to 4 bytes) LITTLE-ENDIAN !

Cf. https://en.bitcoin.it/w/images/en/7/70/Bitcoin_OpCheckSig_InDetail.png

Step 6: Copy TxNew to TxCopy (to be modified) → TxCopy

Step 7: Set all TxIn scripts in TxCopy to empty strings
Make sure that the VAR_INT's representing script length are reevaluated to a single 0x00 byte for each TxIn → TxCopy

Step 8: Copy Subscript into the TxIn script you are checking:
Make sure VAR_INT preceding SUBSCRIPT is reevaluated to represent the size of SUBSCRIPT → TxCopy

(from Step 4)

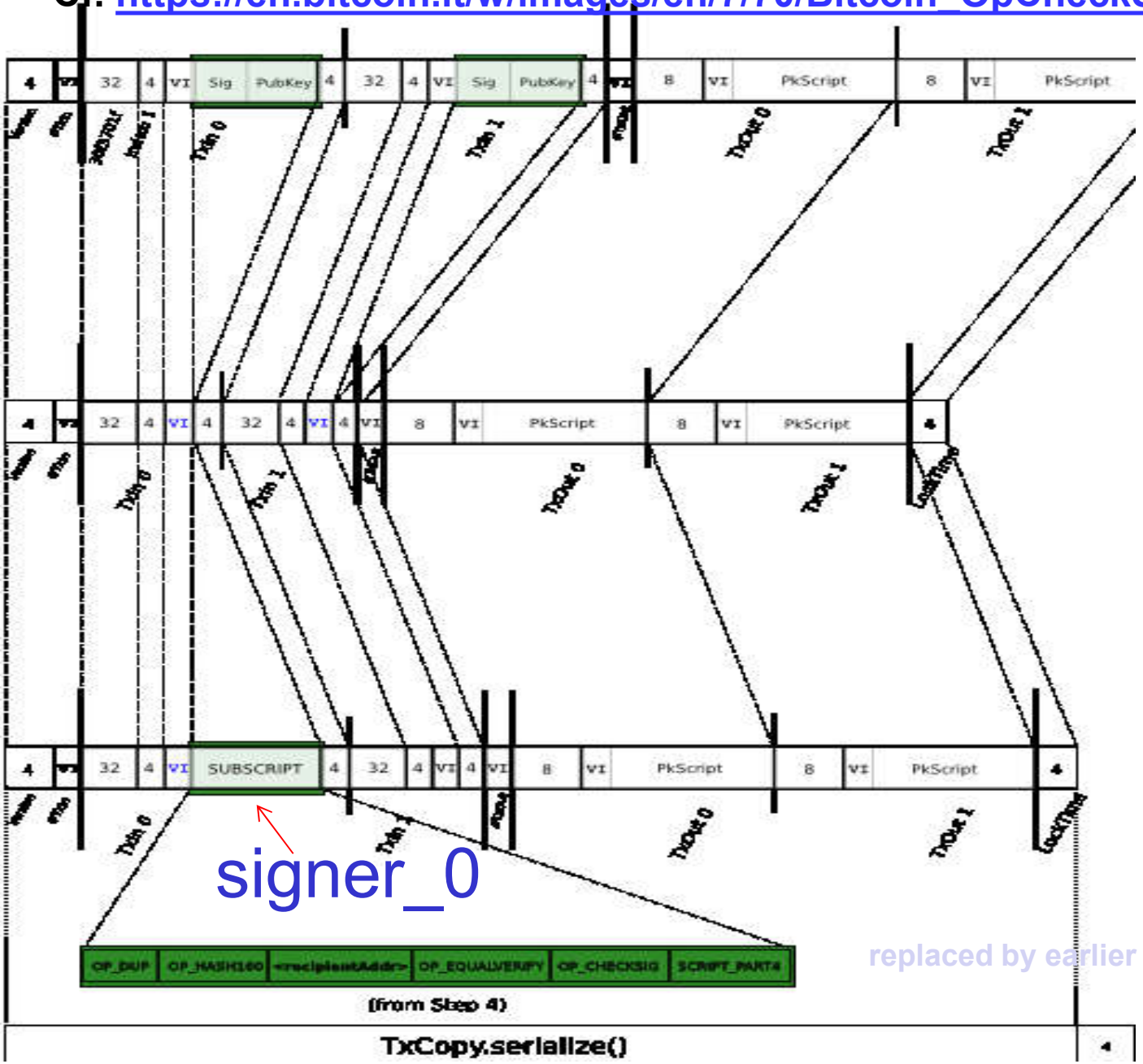
Step 9: Serialize TxCopy, append 4-byte hashTypeCode: → TxCopy.serialize() + hashTypeCode (little-endian) → verifyThisStr

Step 10: Verify signature against string in Step 9, (hashed string needs to be big-endian)

ECDSA_CheckSignature(pubKeyStr , sigStr , $sha256^2$(verifyThisStr))

Repeat all steps for each TxIn object and associated TxOut

Cf. https://en.bitcoin.it/w/images/en/7/70/Bitcoin_OpCheckSig_InDetail.png



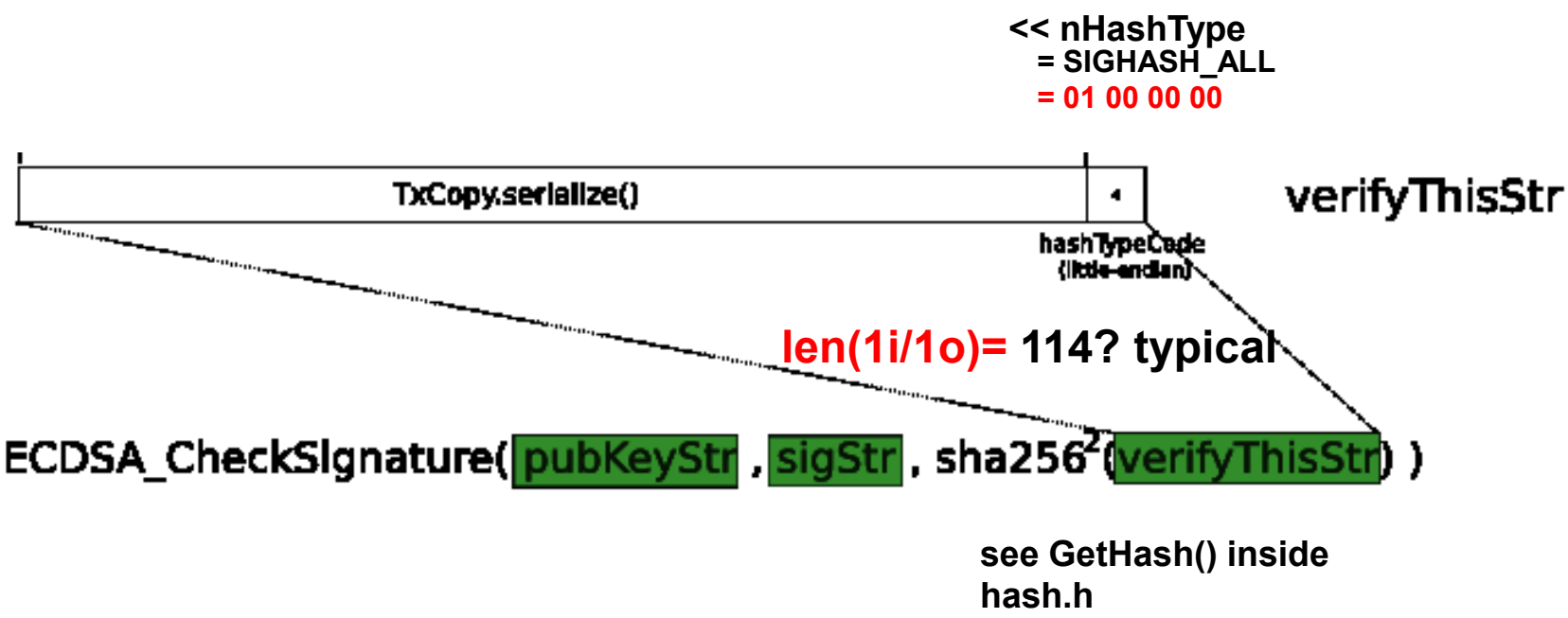
raw tx with several signatures

see SignatureHash() inside script.cpp

to be signed

replaced by earlier script, particularity

**now hash+verif



see GetHash() inside hash.h

SegWit Transactions

New Since 2017

*Ideas:

Current bitcoin blockchain is redundant.
Transactions are long (thousands of bytes).
Cost of storage exorbitant (like 30$/tx).

Questions:

- "Why keep the picture of a cashed check in your records AFTER it has been cashed??"
  - not quite OK. In bitcoin we have crowd-sourced fraud policing. Multiple checks. 100 out of 100 miners can be dishonest?
    - YES, because they maybe all use the same software or go through the same mining pools.

*Waste: 2x => 0x

- Waste: In most cases we store the original transaction twice.
  - Once when it says =>PK3 in scriptPubKey.
  - Second time as spending input maybe not needed. If 1 payment to each PK [most of the time!] it is sufficient to specify PK.
  - There are serious trade-offs speed/storage.

- SegWit: Digital Signature Data will NOT be stored at all!

Antonopoulos writes:
“immediately after
receiving a new transaction and validating witness data, nodes can discard that witness data. “

DANGEROUS…

Witness = Def:

Antonopoulos: https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch07.asciidoc

"In cryptography, the term "witness" is used to describe a solution to a cryptographic puzzle".

In bitcoin,

- a digital signature is *one* type of witness,
  - e.g. scriptSig = D.S. + Public Key
- more generally "witness" == "unlocking script",

  = def = "any solution that can satisfy the conditions imposed on an UTXO and unlock that UTXO for spending."

Separate Witness = Remove Whole scriptSig

sign+PKey

len= 1+71+ 1+65 = 138 BUT NOT ALWAYS!

scriptSig

PUSHDATA 47		47
signature (DER)	sequence	30
	length	44
	integer	02
	length	20
	X **r**	2c b2 65 bf 10 70 7b f4 93 46 c3 51 5d d3 d1 6f c4 54 61 8c 58 ec 0a 0f ~~f4~~ 48 a6 76 c5 4f f7 13
	integer	02
	length	20
	Y **s**	6c 66 24 d7 62 a1 fc ef 46 18 28 4e ad 8f 08 67 8a c0 5b 13 c8 42 35 f1 65 4e 6a d1 68 23 3e 82
SIGHASH_ALL		01
PUSHDATA 41		41
public key	type	04
	X	14 e3 01 b2 32 8f 17 44 2c 0b 83 10 d7 87 bf 3d 8a 40 4c fb d0 70 4f 13 5b 6a d4 b2 d3 ee 75 13
	Y	10 f9 81 92 6e 53 a6 e8 c3 9b d7 d3 fe fd 57 6c 54 3c ce 49 3c ba c0 63 88 f2 65 1d 1a ac bf cd

scriptSig1

scriptSig2

Segregated Witness AD 2017

Old bitcoin nodes can still validate them!

- Appear as transactions
  which ANYONE can spend.
- Deliberate breaking the bitcoin standard.

source Antonopoulos:
https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch07.asciidoc
a solution to a cryptographic puzzle".

Transactions which ANYONE can spend

- Example:

Pay-to-Witness-Public-Key-Hash (P2WPKH) script

- 0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7

Old and new clients interpret them differently.

Transactions which ANYONE can spend

- Example:

Pay-to-Witness-Public-Key-Hash (P2WPKH) script

- 0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7

Old and new clients interpret them differently.

For OLD clients:
anyone can spend!!! No signature, no secrets etc…
An empty signature is accepted!

```
"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
         "scriptSig": "",
```

YES if majority of miners have old software, bitcoins are gone…

Super dangerous… Sth. like 10% of miners can stop them it BUT will lead to tremendous WASTE [orphan blocks rejected later] and MESS [delays]

Transactions which ANYONE can spend

- Example:

P2WPKH

- 0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7

Old and new clients interpret them differently.

For NEW clients: 0=Witness protocol version nb.
+ extra “witness” data are provided to check the signature.

```
"txid": "0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2",
"vout": 0,
         "scriptSig": “”,
]
[...]
“witness”: “<Bob’s witness data>”
[...]
```

4 Types of SegWit

1. Pay-to-Witness-Public-Key-Hash (P2WPKH) script
- 0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7

2. Pay-to-Witness-Script-Hash (P2WSH) script hash-longer(32 bytes)
- 0 a9b7b38d972cabc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89

The difference is that here the witness will contain a more complex spending script [for example for multisig].

4 Types of SegWit

1. Pay-to-Witness-Public-Key-Hash (P2WPKH) script
- 0 ab68025513c3dbd2f7b92a94e0581f5d50f654e7

2. Pay-to-Witness-Script-Hash (P2WSH) script hash-longer(32 bytes)
- 0 a9b7b38d972cabc7961dbfbcb841ad4508d133c47ba87457b4a0e8aae86dbb89

3. Pay-to-Witness-Public-Key-Hash inside Pay-to-Script-Hash: P2SH(P2WPKH)
- HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL

look like any P2SH

4. Pay-to-Witness-Script-Hash inside Pay-to-Script-Hash: P2SH(P2WSH)
- HASH160 3e0547268b3b19288b3adef9719ec8659f4b2b0b EQUAL

3,4 allows transactions created by older software+new address to be processed by new software.

*Key Benefits

Software which sends the money can ignore HOW