Blockchain and Digital Currencies

Lecture 3

PHBS 2024 M3

Agenda

- Review of digital signatures
- Bitcoin transactions and transcripts

Digital Signatures

Q: What do we want from signatures?

Only you can sign, but anyone can verify.

Signature is tied to a particular document,
i.e., cannot be cut-and-pasted to another document.

Slides borrowed from Bitcoin and Cryptocurrency Technologies https://bitcoinbook.cs.princeton.edu/

Digital Signature Scheme

Digital Signature Scheme consists of 3 algorithms:

- (sk,pk) := generateKeys(keysize) generates a key pair
  - *sk* is secret key, used to sign messages
  - *pk* is public verification key, given to anybody
- sig := sign(sk, msg) outputs signature for *msg* with key *sk*.
- verify(pk,msg,sig) returns true if signature is valid and false otherwise.

Requirements for Digital Signature Scheme

Valid signatures must verify!

verify(pk, msg, sign(sk, msg)) == true

Signatures must be unforgeable!

An adversary who

- knows *pk*
- has seen signatures on messages of her choice

cannot produce a verifiable signature on a new message.

Digital Signatures in Practice

Key generation algorithms must be randomized.

.. need good source of randomness

Sign and verify are expensive operations for large messages.

Fix: use *H(msg)* rather than *msg*.

Check this out:

Signing a hash pointer "covers" the whole data structure!

Cryptography Roadmap

	Symmetric-key	Asymmetric-key
Confidentiality	• One-time pads • Block ciphers with chaining modes (e.g. AES-CBC) • Stream ciphers	• RSA encryption • ElGamal encryption
Integrity, Authentication	• MACs (e.g. HMAC)	• Digital signatures (e.g. RSA signatures)

- Hash functions
- Pseudorandom number generators
- Public key exchange (e.g. Diffie-Hellman)

- Key management (certificates)
- Password management

RSA Encryption: Definition

- KeyGen():
  - Randomly pick two large primes, p and q
    - Done by picking random numbers and then using a test to see if the number is (probably) prime
  - Compute $N = pq$
    - N is usually between 2048 bits and 4096 bits long
  - Choose e
    - Requirement: e is relatively prime to $z = (p - 1)(q - 1)$ e,z互质
    - Requirement: $2 < e < z$
  - Compute $d = e^{-1} \bmod z$ (**actually e*d mod z = 1**)
    - Algorithm: Extended Euclid's algorithm (CS 70, but out of scope)
  - **Public key**: N and e
  - **Private key:** d

RSA Signatures: Definition

- KeyGen():
  - Same as RSA encryption:
    - **Public key**: N and e
    - **Private key:** d
- Sign(d, M):
  - Compute $H(M)^d \bmod N$
- Verify(e, N, M, sig)
  - Verify that $H(M) \equiv sig^e \bmod N$

☑ Maybe it is time to review the simple example given in the information security class?

Signatures, Public Keys, and Identities

If you see a signature *sig* such that

verify(pk, msg, sig)==true,

think of it as

pk says, "*[msg]*".

Why?

Because to "speak for" *pk*, you must know the matching secret key *sk*.

How to Create a new Identity

Create a new, random key-pair *(sk, pk)*

- *pk* is the public "name" you can use
  [usually better to use *Hash(pk)*]
- *sk* lets you "speak for" the identity

You control the identity,
because only you know *sk*.

If *pk* "looks random", nobody needs to know who you are.

Decentralized Identity Management

By creating a key-pair,
anybody can make a new identity at any time.

Make as many as you want!

No central point of coordination.

These identities are called addresses in Bitcoin.

Identities and Privacy

Addresses are not directly connected to real-world identity.

But observer can link together an address' activity over time, and make inferences about real identity.

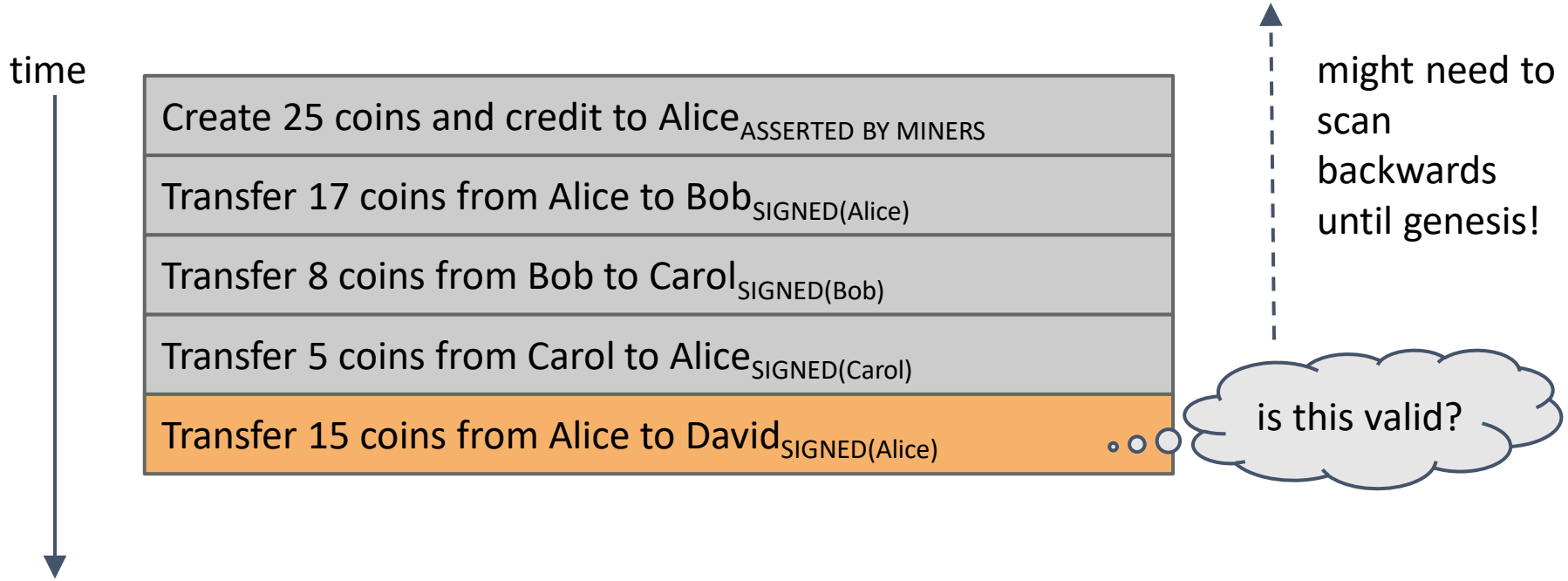
We will talk about privacy later

Bitcoin Transaction and Scripts

Very detailed analysis, requires basic programming knowledge.

In particular, how stack works.

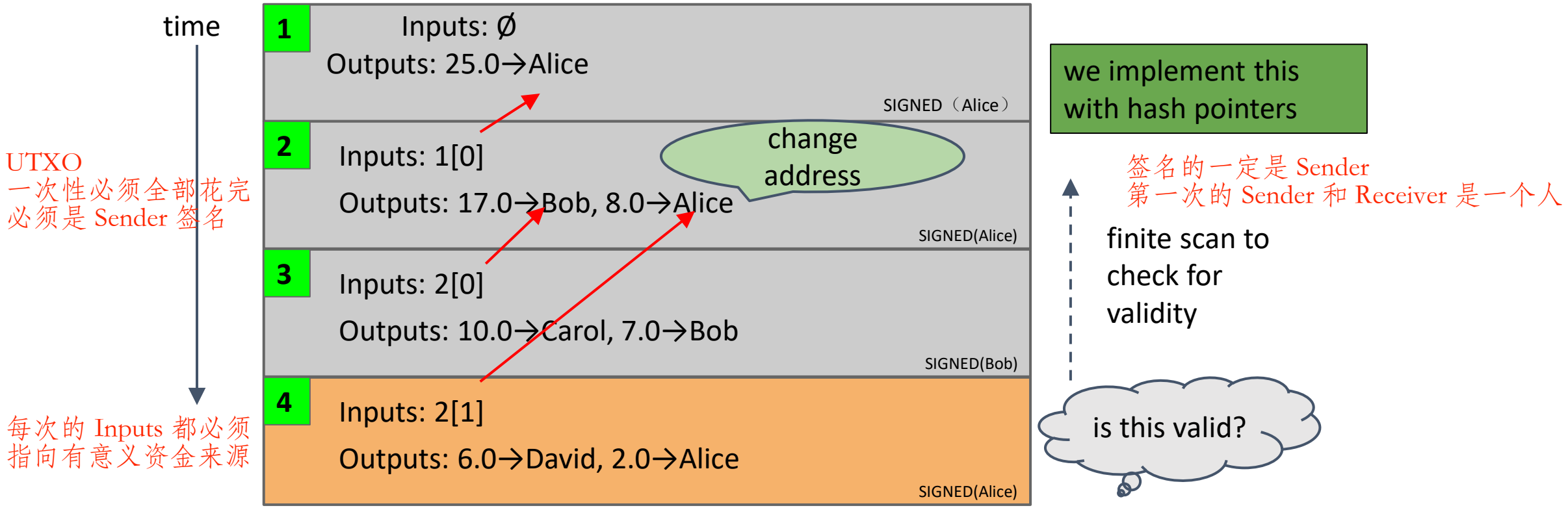
An account-based Ledger (not Bitcoin)



Why do we need a signature for each transaction?

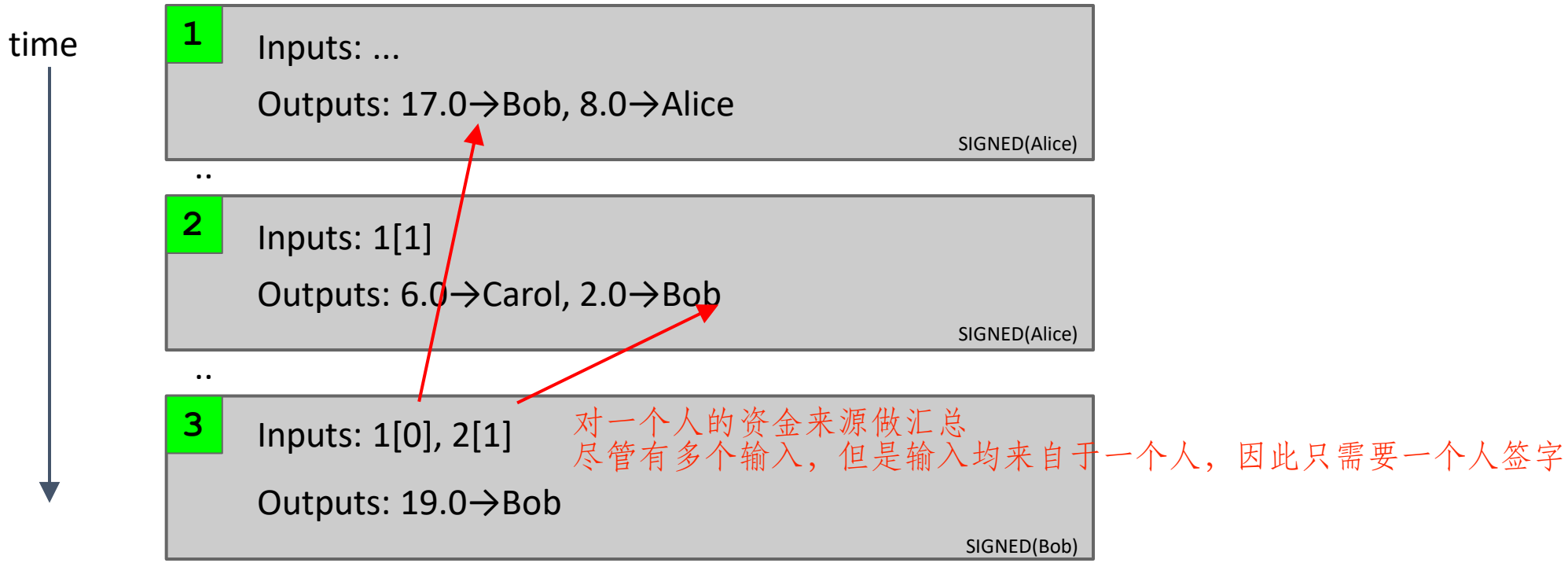
SIMPLIFICATION: only one transaction per block

A transaction-based Ledger (Bitcoin) 必考点



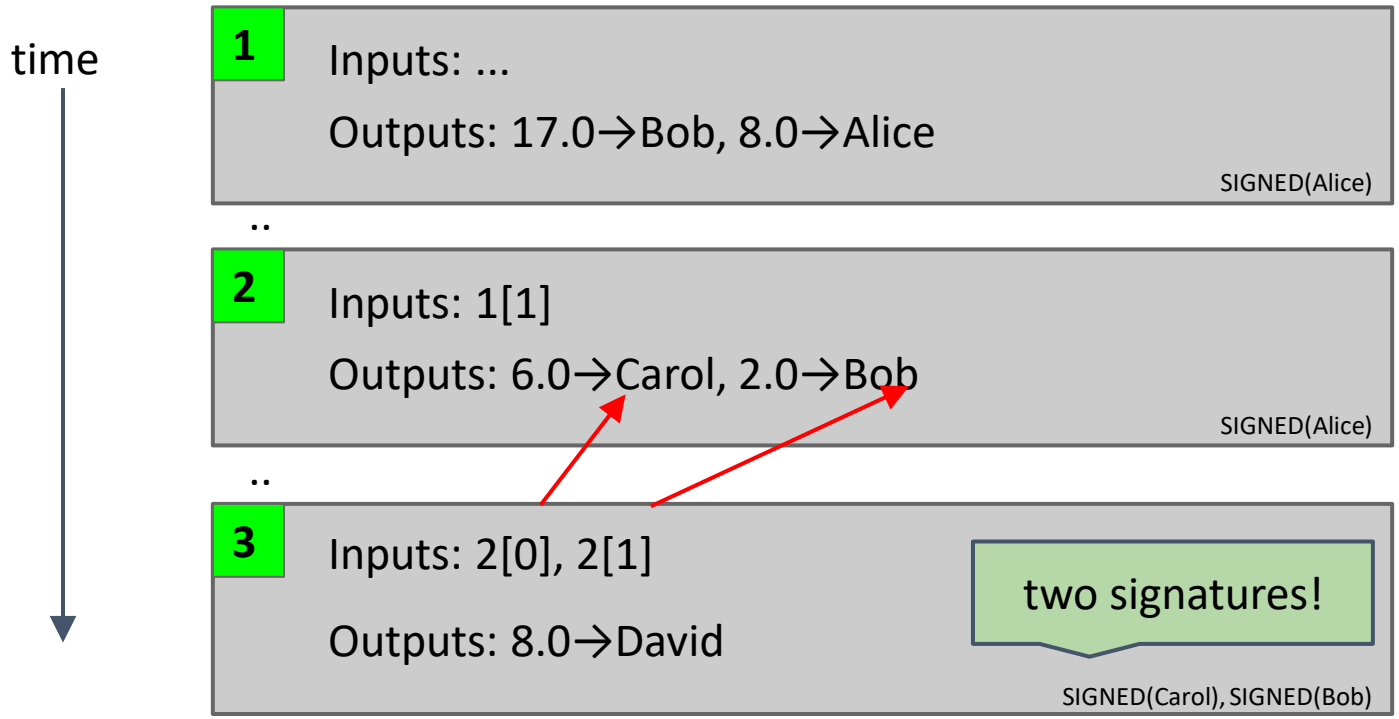
SIMPLIFICATION: only one transaction per block

Merging Value



SIMPLIFICATION: only one transaction per block

Joint Payments



资金的输入方必须签名

SIMPLIFICATION: only one transaction per block

The Real Deal: a Bitcoin Transaction

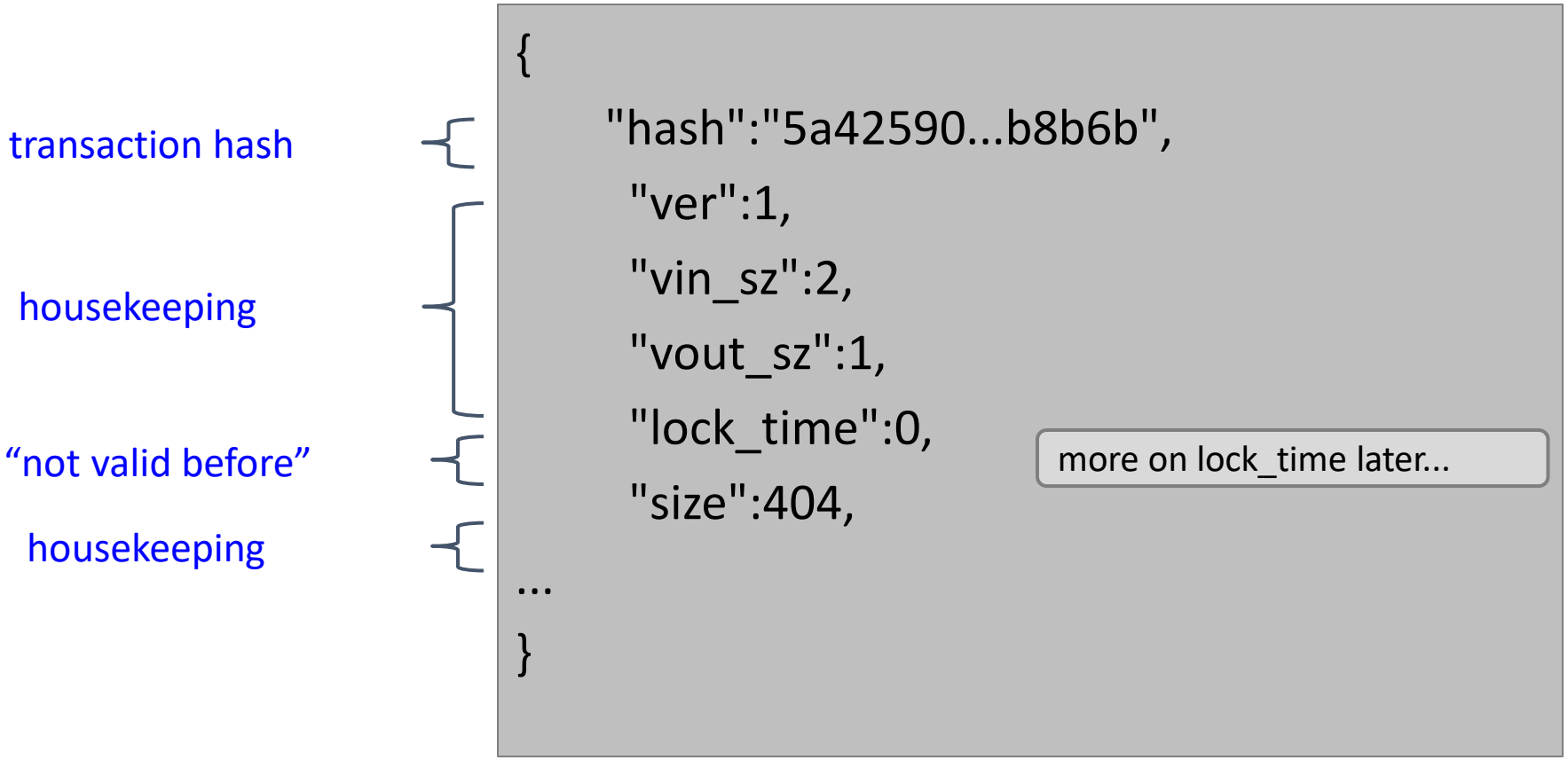
metadata

input(s)

output(s)


```
{
    "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
     "ver":1,
     "vin_sz":2,
     "vout_sz":1,
     "lock_time":0,
     "size":404,
     "in":[
      {
       "prev_out":{
        "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
        "n":0
       },
       "scriptSig":"30440..."
      },
      {
       "prev_out":{
        "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
        "n":0
       },
       "scriptSig":"3f3a4ce81...."
      }
     ],
     "out":[
      {
       "value":"10.12287097",
       "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e
OP_EQUALVERIFY OP_CHECKSIG"
      }
     ]
}
```

The Real Deal: Transaction Metadata



The Real Deal: Transaction Inputs

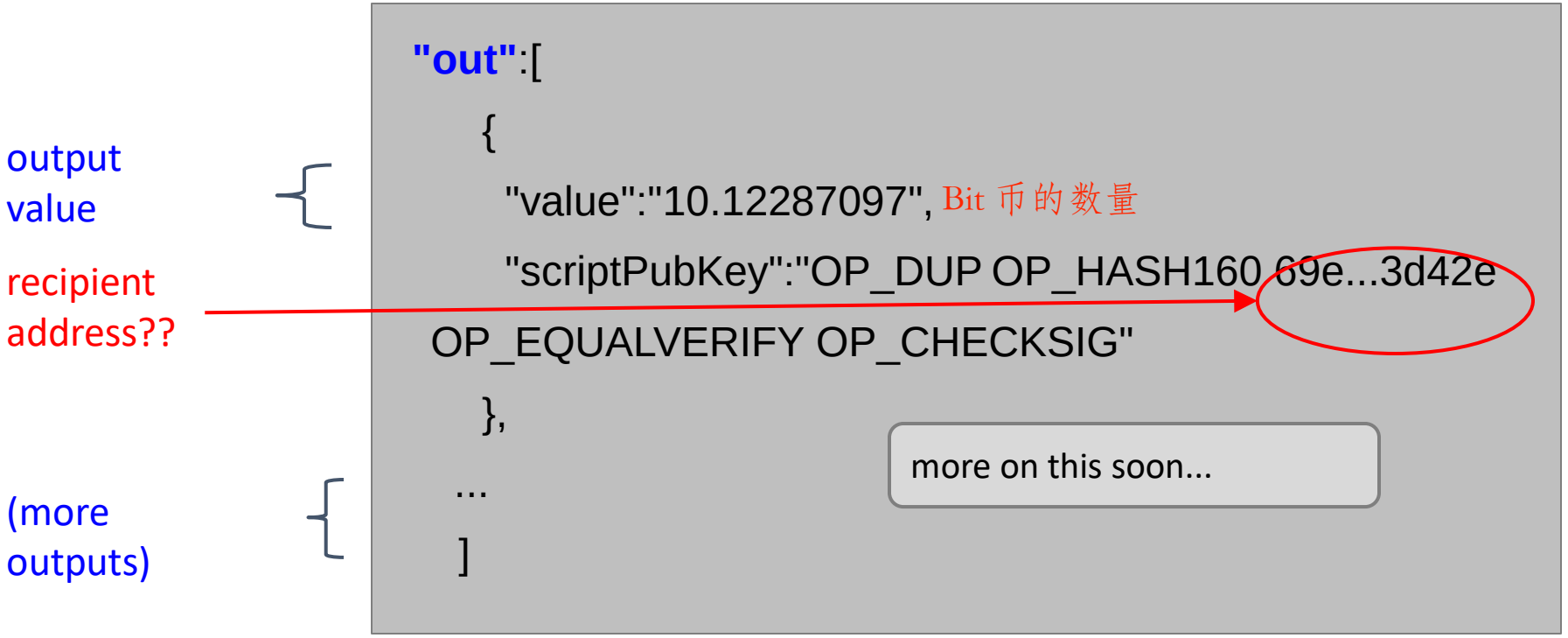
previous transaction

signature

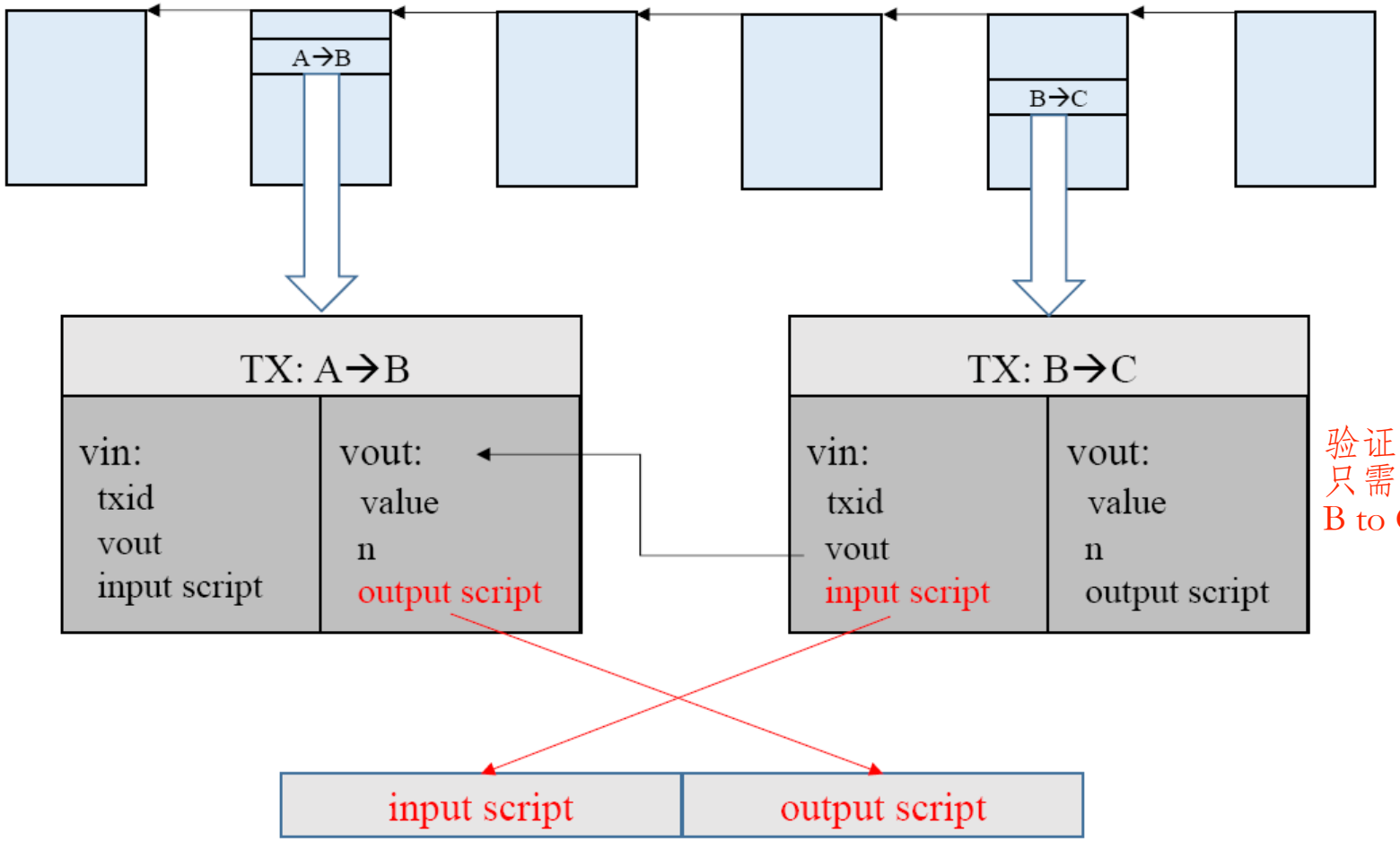
(more inputs)

```
"in":[
      {
        "prev_out":{
          "hash":"3be4...80260",
          "n":0
        },
        "scriptSig":"30440....3f3a4ce81"
      },
     ...
     ],
```

The Real Deal: Transaction Outputs



From Professor Zhen Xiao



验证 B to C 的时候，
只需要看之前到 B 的 out
B to C 的 out 根本不重要

拼接成一段完整的在栈上运行的脚本

Two Commonly Used Scripts (Lock-Key Pairs)

P2PK (Pay to Public Key)

input script:

```
PUSHDATA(Sig)
```

output script:

```
PUSHDATA(PubKey)
CHECKSIG
```

P2PKH (Pay to Public Key Hash)

input script:

```
PUSHDATA(Sig)
PUSHDATA(PubKey)
```

如果在 output 的时候只提供了 PubKeyHash
那就需要在 input 的时候提供 PubKey

output script:

```
DUP
HASH160
PUSHDATA(PubKeyHash)
EQUALVERIFY
CHECKSIG
```

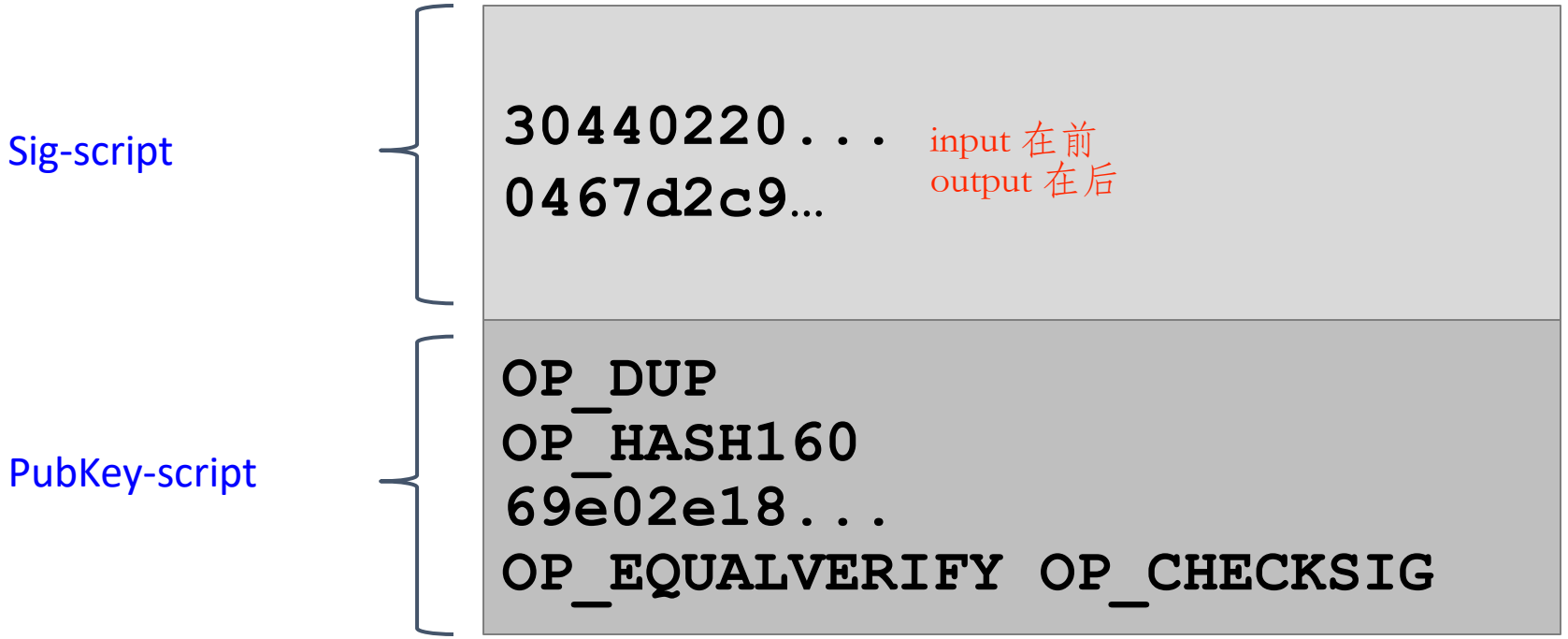
https://developer.bitcoin.org/devguide/transactions.html

Outputs are really *Scripts*

```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

Inputs are *also* Scripts

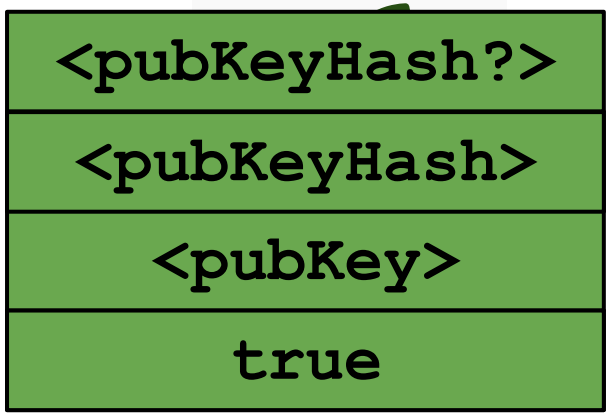
NOTE: The Sig-script and PubKey-script are from **different** transactions!



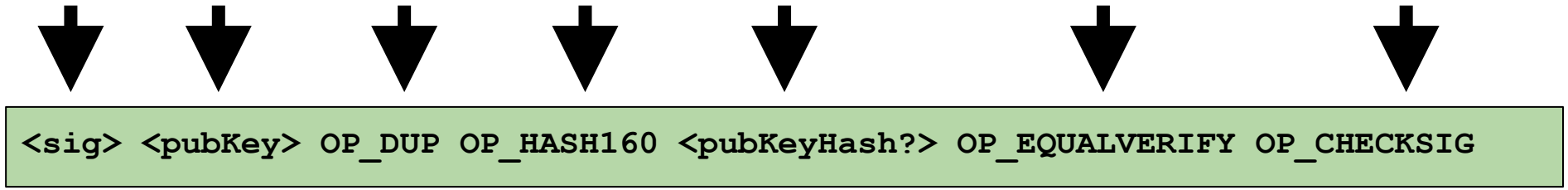
TO VERIFY: Concatenated script must execute completely with no errors

Bitcoin Script Execution Example

```
30440220...
0467d2c9…
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```



得到 pubKey 并和得到的做比较，
如果符合进一步比较（RSA 非对称加密）



Why Scripts?!

☑ What is this about? What is the usage?

Redeem previous transaction by signing with correct key

"This can be redeemed by a signature from the owner of address *X*"

Recall: address *X* is hash of public key

What is public key associated with *X*?!

"This can be redeemed by a public key that hashes to *X*, along with a signature from the owner of that public key"

Bitcoin Scripting Language ("Script")

Design "goals":

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Stack-based
- No looping
- Support for cryptography
- Limits on time/memory
- Not Turing complete!

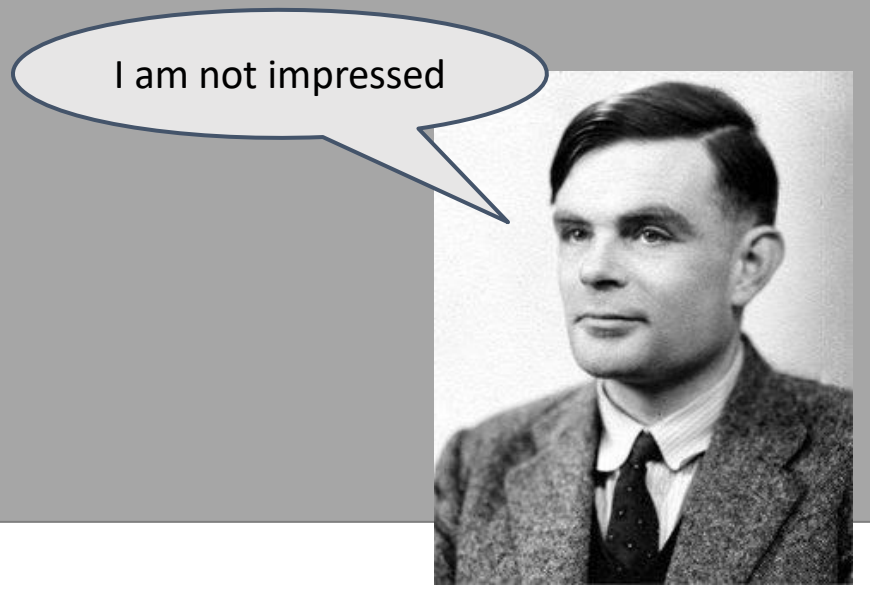


image via Jessie St. Amand

Bitcoin Script Instructions

256 opcodes total (15 disabled, 75 reserved)

- Arithmetic
- If/then
- Logic/data handling
- Crypto!

OP_DUP	Duplicates the top item on the stack
OP_HASH160	Hashes twice: first using SHA-256 and then RIPEMD-160
OP_EQUALVERIFY	Returns true if the inputs are equal. Returns false and marks the transaction as invalid if they are unequal
OP_CHECKSIG	Checks that the input signature is a valid signature using the input public key for the hash of the current transaction
OP_CHECKMULTISIG	Checks that the *k* signatures on the transaction are valid signatures from *k* of the specified public keys.

OP_CHECKMULTISIG

Built-in support for joint signatures

Specify *n* public keys

Specify *t*

Verification requires *t* signatures

Incidentally: There is a bug in the **multisig** implementation.
Extra data value popped from the stack and ignored 多个输入需要每个都看

Scripts in Practice (as of 2015)

Theory: Scripts let us specify arbitrary conditions that must be satisfied to spend coins.

Q: Is any of this used in practice?

- 99.9% are simple signature checks
- ~0.01% are **`MULTISIG`**
- ~0.01% are Pay-to-Script-Hash
- Remainder are errors, proof-of-burn

More on this soon

Most nodes whitelist known scripts

Proof-of-Burn

this script can never be redeemed ☹

```
OP_RETURN
<arbitrary data>
```

Uses for Proof-of-Burn:

- Destroy coins and transfer them to alternative currency
- Add arbitrary data to block chain (to create more hash values during mining)