The BitCoin Scripting Language

交易实例

Transaction View information about a bitcoin transaction

921af728159e3019c18bbe0de9c70aa563ad27f3f562294d993a208d4fcfdd24

1MaBFqBEfcQyXPv3fm5WAW9aQuJuKHaA3A (0.76469684 BTC - Output)

19z8LJkNXLrTv2QK5jgTncJCGUEEfpQvSr - (Unspent) 0.22684 BTC
1LvGTpdyeVLcLCDK2m9f7Pbh7zwhs7NYhX - (Spent) 0.53756644 BTC

23 Confirmations | 0.76440644 BTC

Summary	
Size	226 (bytes)
Weight	904
Received Time	2018-07-06 03:08:26
Included In Blocks	530657 (2018-07-06 03:12:07 + 4 minutes)
Confirmations	23 Confirmations
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.76469684 BTC
Total Output	0.76440644 BTC
Fees	0.0002904 BTC
Fee per byte	128.496 sat/B
Fee per weight unit	32.124 sat/WU
Estimated BTC Transacted	0.22684 BTC
Scripts	Hide scripts & coinbase

Input Scripts

ScriptSig: PUSHDATA(72)
[30450221009284 96fb0d2a25e4e7c99b9c60d4d0d12fcf8974a0faffcb30119b0d385872a30220253d3d0c507e5e44e123bc28b795ab4a38bf3b205455403e77aa72d58d9e17
PUSHDATA(33)[022ef8d3a6dd8a7039e513acc8ecf9b094ed7e85439824a1d11920f85927cd0018]

Output Scripts

DUP HASH160 PUSHDATA(20)[628ed6567c0b9056067309f07bbea2992ecad743] EQUALVERIFY CHECKSIG

DUP HASH160 PUSHDATA(20)[da7d57dfd02c6f5a9c649e891b5ac199ad012cd2] EQUALVERIFY CHECKSIG

交易结构


```
"result": {
    "txid": "921a…dd24",
    "hash": "921a…dd24",
    "version": 1,
    "size": 226,
    "locktime": 0,
    "vin": […],
    "vout": […],
    "blockhash": "00000000000000002c510d…5c0b",
    "confirmations": 23,
    "time": 1530846727,
    "blocktime": 1530846727
}
```

交易的输入

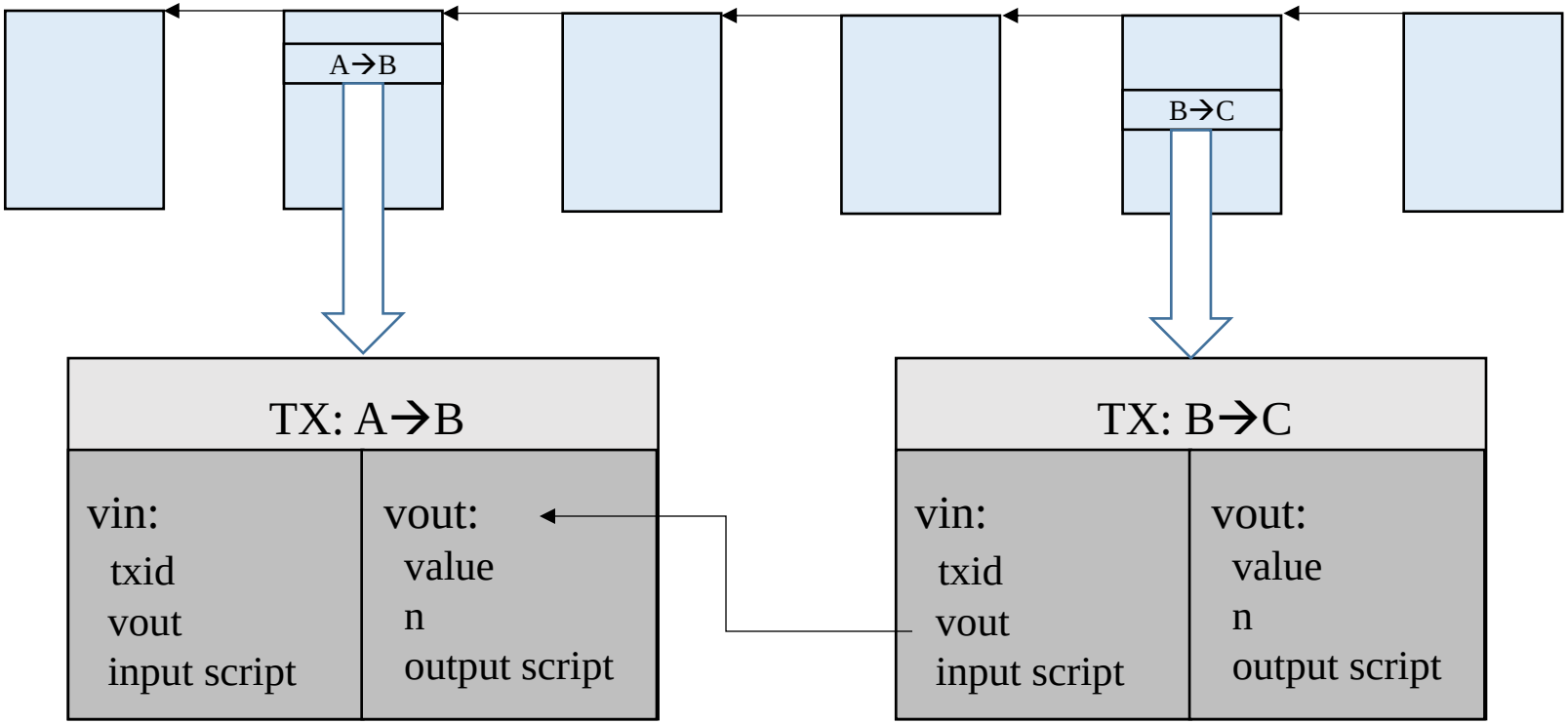
```
"vin": [{
    "txid": "c0cb…c57b",
    "vout": 0,
    "scriptSig": {
        "asm": "3045...0018",
        "hex": "4830...0018"
    },
}],
```

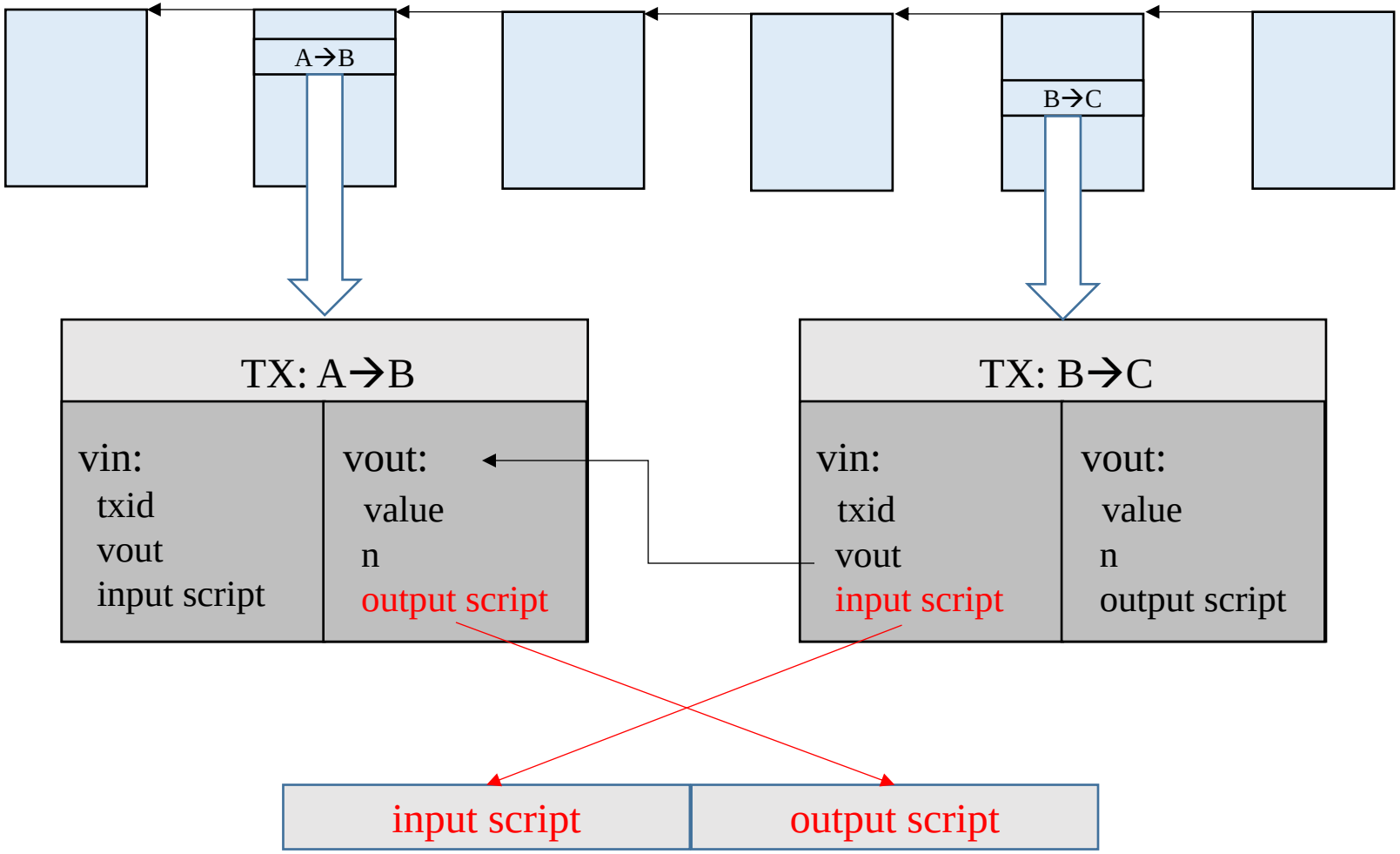
这里的“scriptSig”之后将以input script指代

交易的输出


```
"vout": [{
    "value": 0.22684000,
    "n": 0,
    "scriptPubKey": {
        "asm": "DUP HASH160 628e…d743 EQUALVERIFY CHECKSIG",
        "hex": "76a9…88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [ "19z8LJkNXLrTv2QK5jgTncJCGUEEfpQvSr"]
    }
 },{
    "value": 0.53756644,
    "n": 1,
    "scriptPubKey": {
        "asm": "DUP HASH160 da7d…2cd2 EQUALVERIFY CHECKSIG",
        "hex": "76a9…88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": ["1LvGTpdyeVLcLCDK2m9f7Pbh7zwhs7NYhX"]
    }
}],
```


这里的“scriptPubKey”之后将以output script指代


A→B
B→C
TX: A→B
vin:
txid
vout
input script
vout:
value
n
output script
TX: B→C
vin:
txid
vout
input script
vout:
value
n
output script



拼接成一段完整的在栈上运行的脚本

P2PK (Pay to Public Key)

input script:

```
PUSHDATA(Sig)
```

output script:

```
PUSHDATA(PubKey)
CHECKSIG
```

脚本执行

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

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

CHECKSIG

Sig

脚本执行

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

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

CHECKSIG

TRUE

实例

交易ea44e97271691990157559d0bdd9959e02790c34db6c006d779e82fa5aee708e的第一个输入：

Input Scripts

ScriptSig: PUSHDATA(71)
[30440220576497b7e6f9b553c0aba0d8929432550e092db9c130aae37b84b545e7f4a36c022066cb982ed80608372c139d7bb9af335423d5280350fe3e06bd510e695480914f01]

交易f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16的第一个输出：

Output Scripts

PUSHDATA(65)[04ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1baded5c72a704f7e6cd84c]
CHECKSIG

P2PKH (Pay to Public Key Hash)

input script:

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

output script:

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

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG

Sig

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

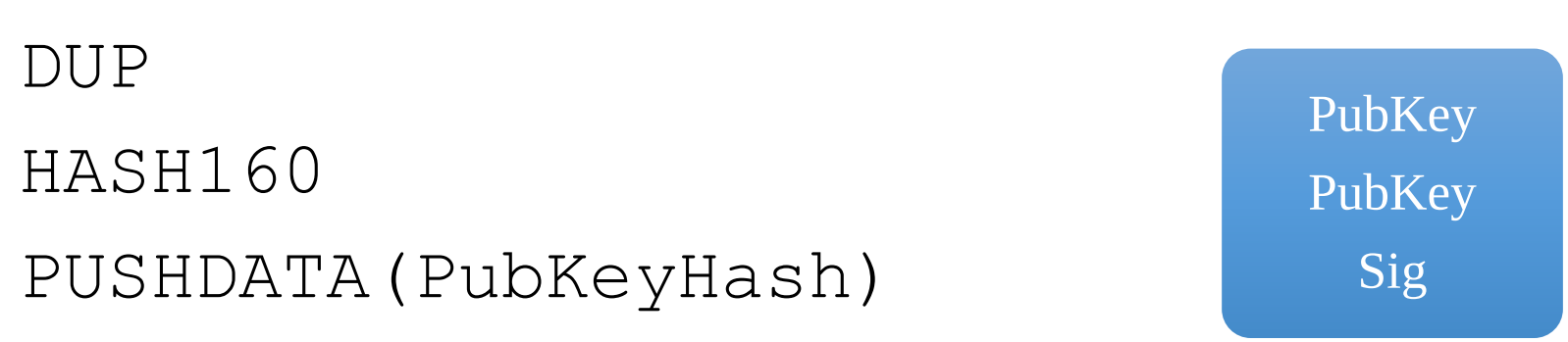
DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG



脚本执行

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

PubKeyHash
PubKey
Sig

脚本执行

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

PubKeyHash
PubKeyHash
PubKey
Sig

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG

脚本执行

PUSHDATA(Sig)

PUSHDATA(PubKey)

DUP

HASH160

PUSHDATA(PubKeyHash)

EQUALVERIFY

CHECKSIG

TRUE

实例

交易921af728159e3019c18bbe0de9c70aa563ad27f3f562294d993a208d4fcfdd24的第一个输入：

Input Scripts

ScriptSig: PUSHDATA(72)
[3045022100928496fb0d2a25e4e7c99b9c60d4d0d12fcf8974a0faffcb30119b0d385872a30220253d3d0c507e5e44e123bc28b795ab4a38bf3b205455403e77aa72d58d9e17
PUSHDATA(33)[022ef8d3a6dd8a7039e513acc8ecf9b094ed7e85439824a1d11920f85927cd0018]

交易c0cb92ca8e41070233bf965d808b0fc4bac144dab05690b17823fac3e184c57b的第一个输出：

Output Scripts

DUP HASH160 PUSHDATA(20)[e1a8cdae6411b17ee1d4cecfe47bafce37e14d14] EQUALVERIFY CHECKSIG

P2SH (Pay to Script Hash)

采用BIP16的方案

input script:

…

```
PUSHDATA(Sig)
```

…

```
PUSHDATA(serialized redeemScript)
```

output script:

```
HASH160
PUSHDATA(redeemScriptHash)
EQUAL
```

进一步说明

input script要给出一些签名（数目不定）及一段序列化的redeemScript

验证时分两步：

- 第一步验证这段序列化的redeemScript是否与output script中的哈希值匹配？
- 第二步反序列化并执行redeemScript，配合前边的签名是否可以执行通过？

redeemScript可以设计成多种形式，比如前面介绍的P2PK或者P2PKH形式，以及后面要介绍的多重签名形式

用P2SH实现P2PK

redeemScript：

```
PUSHDATA(PubKey)
CHECKSIG
```

input script:

```
PUSHDATA(Sig)
PUSHDATA(serialized redeemScript)
```

output script:

```
HASH160
PUSHDATA(redeemScriptHash)
EQUAL
```

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

RSH

Sig

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(Sig)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

Sig

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

```
PUSHDATA(PubKey)
CHECKSIG
```

Sig

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

- PUSHDATA(PubKey)
  CHECKSIG

PubKey
Sig

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

P2SH+P2PK的执行

PUSHDATA(PubKey)

CHECKSIG

TRUE

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

多重签名

最早的多重签名，目前已经不推荐使用

input script:

x

```
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
...
PUSHDATA(Sig_M)
```

outputScript：

```
M
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
...
PUSHDATA(pubkey_N)
N
CHECKMULTISIG
```

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

FALSE

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

Sig_1
FALSE

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

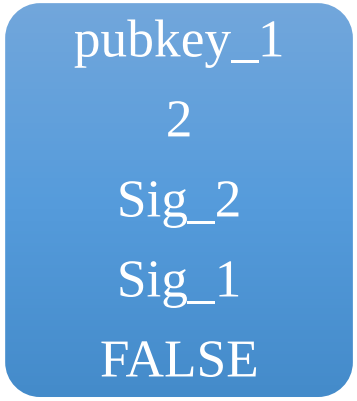
Sig_2
Sig_1
FALSE

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

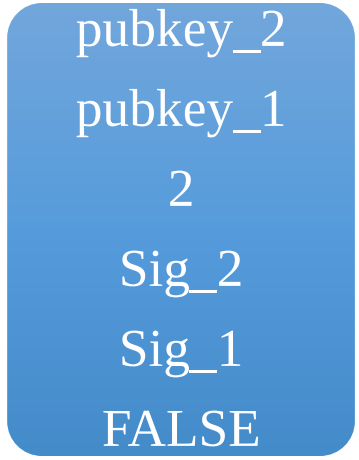
脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



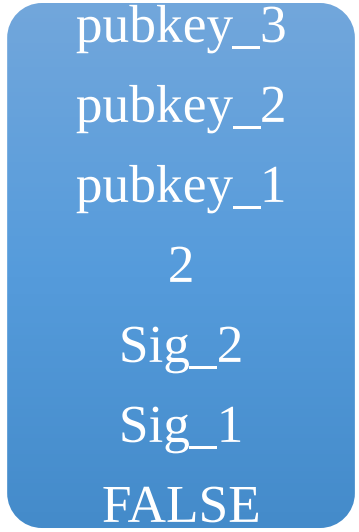
脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



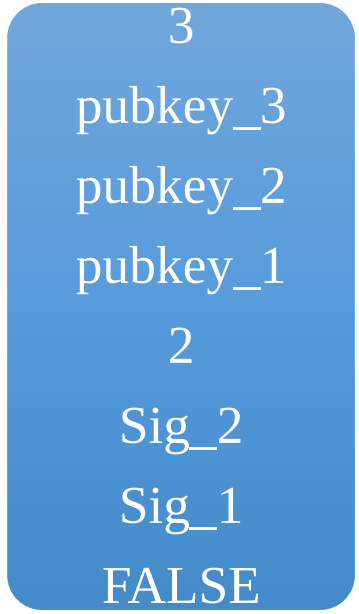
脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



脚本执行

FALSE

PUSHDATA(Sig_1)

PUSHDATA(Sig_2)

2

PUSHDATA(pubkey_1)

PUSHDATA(pubkey_2)

PUSHDATA(pubkey_3)

3

CHECKMULTISIG

TRUE

用P2SH实现多重签名

input script:

```
x
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
...
PUSHDATA(Sig_M)
PUSHDATA(serialized RedeemScript)
```

redeemScript:

```
M
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
...
PUSHDATA(pubkey_N)
N
CHECKMULTISIG
```

output script:

```
HASH160
PUSHDATA(RedeemScriptHash)
EQUAL
```

脚本执行

FALSE

PUSHDATA(Sig_1)

PUSHDATA(Sig_2)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL

FALSE

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

FALSE

PUSHDATA(Sig_1)

PUSHDATA(Sig_2)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

Sig_1

FALSE

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL
```

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL
```

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL
```

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

```
FALSE
PUSHDATA(Sig_1)
PUSHDATA(Sig_2)
PUSHDATA(seriRS)
HASH160
PUSHDATA(RSH)
EQUAL
```

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

FALSE

PUSHDATA(Sig_1)

PUSHDATA(Sig_2)

PUSHDATA(seriRS)

HASH160

PUSHDATA(RSH)

EQUAL

seriRS: serialized RedeemScript
RSH: RedeemScriptHash

脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

Sig_2

Sig_1

FALSE

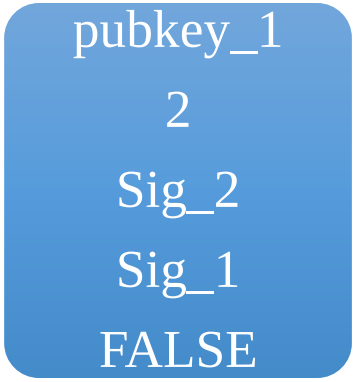
脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

2
Sig_2
Sig_1
FALSE

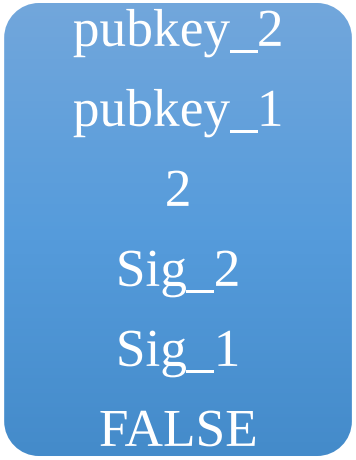
脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



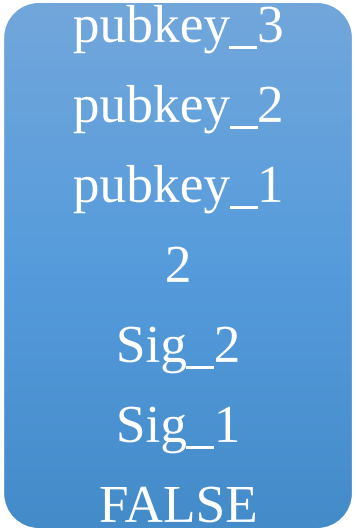
脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



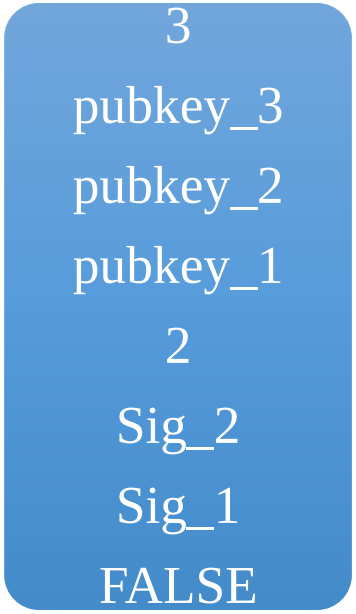
脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```



脚本执行

```
2
PUSHDATA(pubkey_1)
PUSHDATA(pubkey_2)
PUSHDATA(pubkey_3)
3
CHECKMULTISIG
```

TRUE

实例

交易bc26380619a36e0ecbb5bae4eebf78d8fdef24ba5ed5fd040e7bff37311e180d的第一个输入：

Input Scripts

ScriptSig: 0[] PUSHDATA(72)[3045022100f98068a026e2fc75cfeffe84bbac4223ed172df42bca01fd748a14bd960b1695022062c61a7f4f2a63a65d96b0feaf2a048bc2ca93e5[304402201ce986e3fd780f4fe81f40ceb271a8ff34c3845e385b8424f8b20d1b91f1282102205dc71831baf5606f59d06b1d115bda3ec28817cdb4bf9df06643d550c30ef19301] PUSHDATA1[5221027ca87e1aa2595ec7771afee8fdc6efdbc301b8370c4386731b4bd82247dc74a321022cc9874ba092095dda47a4e4edb1781c43c35b3ec0429ac005df37

push的最后一个数据是序列化的脚本，反序列化后得到：

```
2 027c…74a3 022c…c94b 0357…ce3a 3 CHECKMULTISIG
```

交易0ac29fc675909eb565a0984fe13a47dae16ca53fb477b9e03446c898b925ab6b的第二个输出：

Output Scripts

HASH160 PUSHDATA(20)[80cff499983050ec4268d749a1f898bec53e9fc2] EQUAL

Proof of Burn

output script:

```
RETURN
…[zero or more ops or text]
```

包含了这样的output script的output被称为Provably Unspendable/Prunable Outputs。

假如有一个交易的input指向这个output，不论input里的input script如何设计，执行到`RETURN`这个命令之后都会直接返回`false`，`RETURN`后面的其他指令也就不会执行了，所以这个output无法再被花出去，对应的UTXO也就可以被剪枝了，无需保存。

实例

Transaction View information about a bitcoin transaction

090a9343da1158dce3634076f3df0ff868ea777c97b3b2cf6f3d85ca7ea1f7b2

No Inputs (Newly Generated Coins)	18cBEMRxXHq... (ViaBTC Bitcoin Mining Pool) - (Spent) Unable to decode output address - (Unspent)	12.65566873 BTC 0 BTC
		12.65566873 BTC

Summary	
Size	257 (bytes)
Weight	920
Received Time	2018-07-05 12:34:39
Reward From Block	530572
Scripts	Hide scripts & coinbase
Visualize	View Tree Chart

CoinBase

038c1808162f5669614254432f4d696e65642062792077676c312f2cfabe6d6d26a96a921da6cbb032a60989ecfa36e83b52565e31744b7137eaf6e6191dc153040000000000000010ba10e
(decoded) □□□□□/ViaBTC/Mined by wgl1/,��mm&�j�□�.2 ���6;RV^1tKq7���□□S□□□□□j□□□□□xB

Output Scripts

DUP HASH160 PUSHDATA(20)[536ffa992491508dca0354e52f32a3a7a679a53a] EQUALVERIFY CHECKSIG

RETURN PUSHDATA(36)[aa21a9ed29817b27b64c8e79e0f0a7214fa67135872e2b8a5b5b1a6923f83a37c23bfc96]
(decoded) !)'L y!O q5.+[[i#:7;

实例

Transaction View information about a bitcoin transaction

1a2e22a717d626fc5db363582007c46924ae6b28319f07cb1b907776bd8293fc

1MQaYLejR39TvN9PTxpAQcLBxFUqNHXx3M (0.05 BTC - Output) → Unable to decode output address - (Unspent) 0 BTC

0 BTC

Summary	
Size	188 (bytes)
Weight	752
Received Time	2013-03-29 04:32:21
Included In Blocks	228596 (2013-03-29 14:18:58 + 587 minutes)
Confirmations	303536 Confirmations
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.05 BTC
Total Output	0 BTC
Fees	0.05 BTC
Fee per byte	26,595.745 sat/B
Fee per weight unit	6,648.936 sat/WU
Estimated BTC Transacted	0 BTC
Scripts	Hide scripts & coinbase

Input Scripts

ScriptSig: PUSHDATA(71)
[3044022055bcb36c829a614451787fe8c9bfb3798b683809b65b92037a015eccb5ff659702202461d2c708a4fd57c839e43634e8c02935d7b7d1db5b978432b0674c44645ec
PUSHDATA(33)[032c1ea520c25c4e66831cd395a3cd26f0e0a1472a3103fc8a4a63ef10e92d123c]

Output Scripts

RETURN PUSHDATA(20)[215477656e747920627974652064696765573742e]
(decoded) !Twenty byte digest.