Chia Script

Script kinda sucks.

Script is weird

Like -0

Yes. Negative Zero.

No. It’s not a good idea.

OP_BOOLAND // OP_BOOLOR

Pop 2 stack items

Check if they are null strings

Output a boolean

??????

Script is hard to upgrade

New opcodes pretend to be OP_NOPs

```
1517946706
OP_CHECKLOCKTIMEVERIFY
OP_DROP
```

Script is messy

OP_IF is not a conditional

OP_PICK mutates the stack before reading from it

OP_ENDIF is just terrible

Simplicity will fix everything

In 2025

How do we fix it?

Chia Script

Chia Script is clear

Chia Script is clear

- No more negative zero
- OP_BOOLAND ands bools, not strings
- No integer overflows

Chia Script is versatile

Chia Script is versatile

- Improved OP_IF semantics

- Better stack management

- Upgraded signature scheme

Chia Script is upgradable

Chia Script is upgradable

- Abort + Succeed

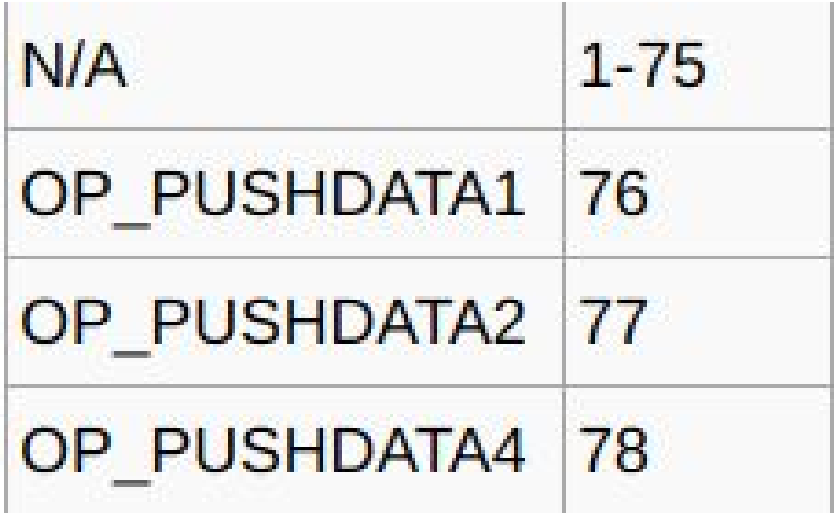
- OP_ABORTSUCCEED

- Reclaim most opcodes

Where do we start?

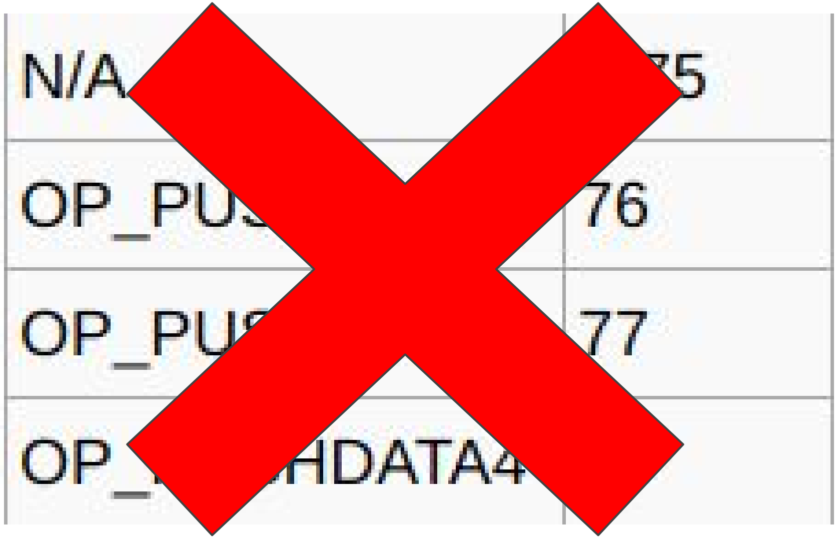
Delete. Everything.

Data pushes?



N/A	1-75
OP_PUSHDATA1	76
OP_PUSHDATA2	77
OP_PUSHDATA4	78

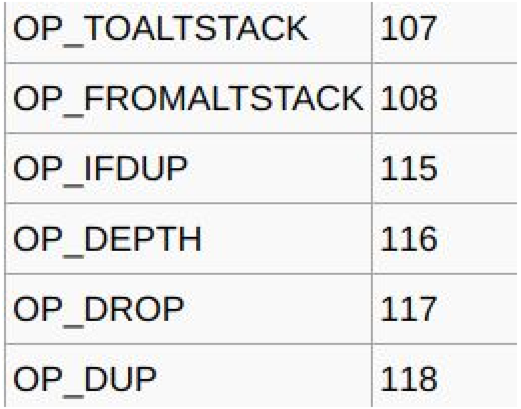
Data pushes?



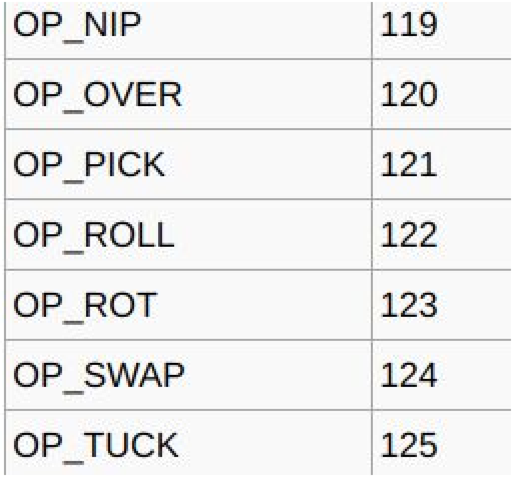
N/A	75
OP_PU	76
OP_PUS	77
OP_ HDATA4	

Chia Script has OP_PUSHDATA.
And nothing else.

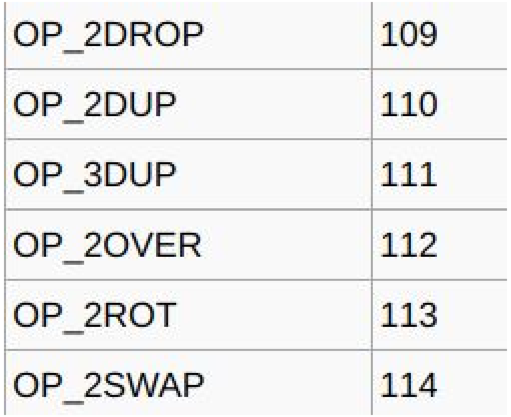
Stack Management?



OP_TOALTSTACK	107
OP_FROMALTSTACK	108
OP_IFDUP	115
OP_DEPTH	116
OP_DROP	117
OP_DUP	118

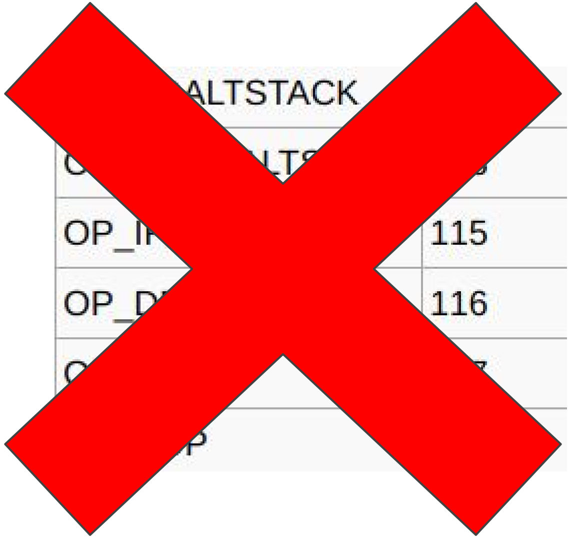


OP_NIP	119
OP_OVER	120
OP_PICK	121
OP_ROLL	122
OP_ROT	123
OP_SWAP	124
OP_TUCK	125

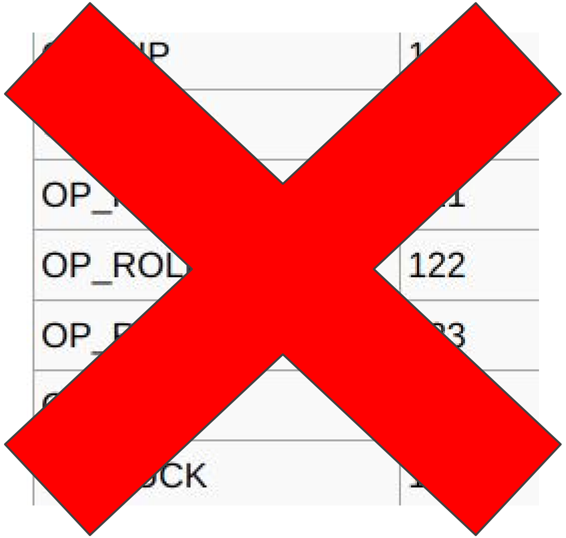


OP_2DROP	109
OP_2DUP	110
OP_3DUP	111
OP_2OVER	112
OP_2ROT	113
OP_2SWAP	114

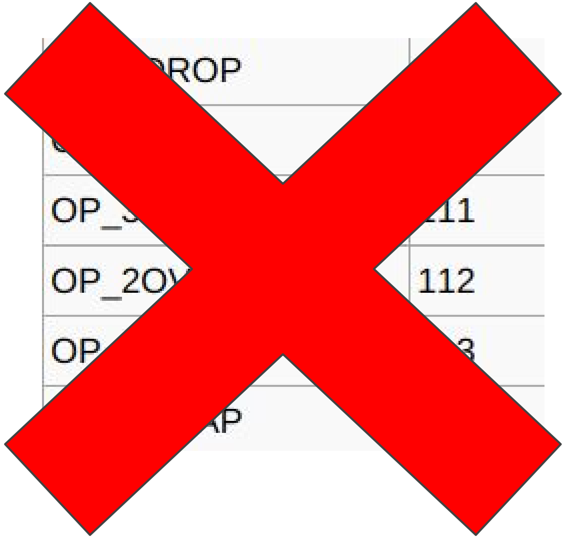
Stack Management?



ALTSTACK	
C LTS	
OP_I	115
OP_D	116
C	7
P	



IP	1
OP_	1
OP_ROL	122
OP_	3
C	
CK	1

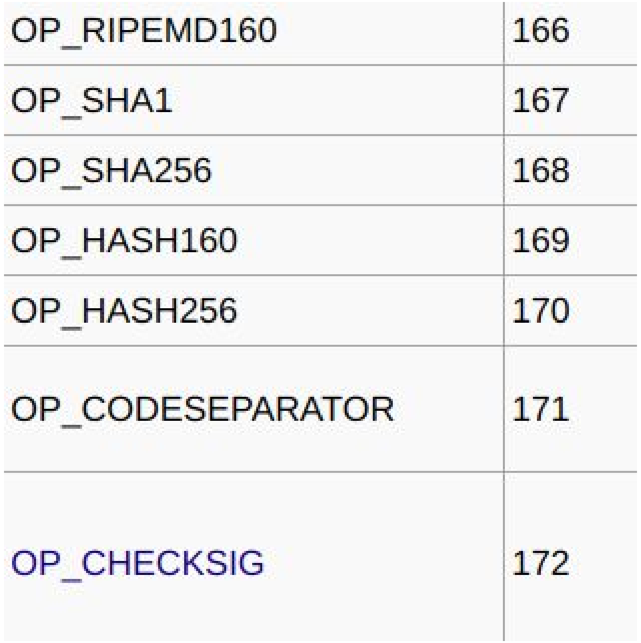


ROP	
C	
OP_	11
OP_2O	112
OP	3
AP	

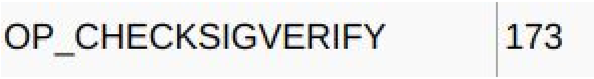
You can keep OP_DEPTH
That’s fine

OP_DROP is fine too.

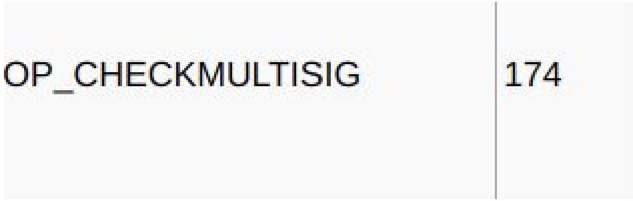
Crypto?



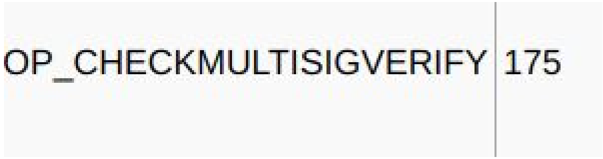
OP_RIPEMD160	166
OP_SHA1	167
OP_SHA256	168
OP_HASH160	169
OP_HASH256	170
OP_CODESEPARATOR	171
OP_CHECKSIG	172



OP_CHECKSIGVERIFY	173

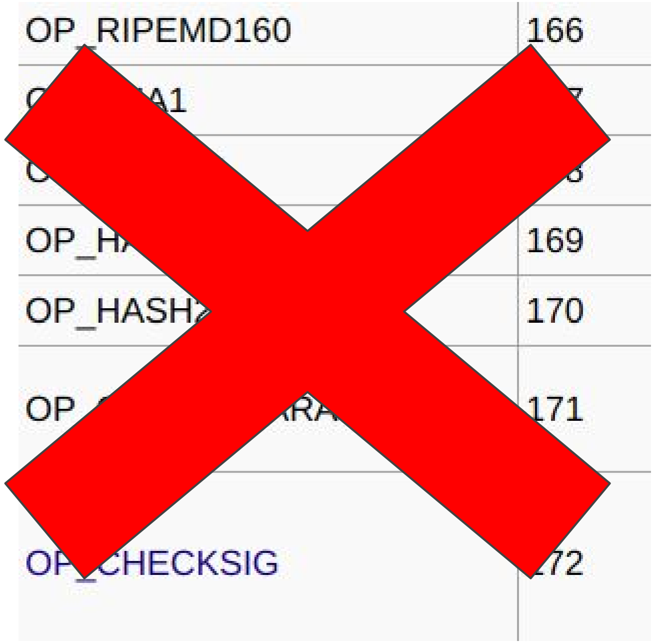


OP_CHECKMULTISIG	174

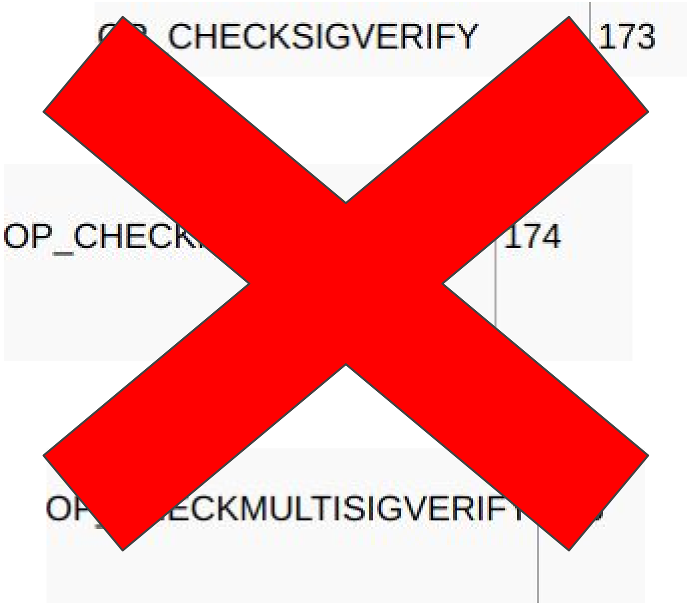


OP_CHECKMULTISIGVERIFY	175

Crypto?



OP_RIPEMD160	166
O A1	7
O	8
OP_H	169
OP_HASH2	170
OP_ RA	171
OP_CHECKSIG	172



O _CHECKSIGVERIFY	173
OP_CHECK	174
OP ECKMULTISIGVERIFY	

You get OP_SHA256.

Double-sha is pointless, and RMD160 is old

But what about signatures?

We’ll get to signatures in a second

Next up!

Make everything better

MAST first

MAST is a no brainer

Then we improve all the opcodes

Abort Success
OP_ABORTSUCCESS

Abort Success // OP_ABORTSUCCESS

Does what it says on the tin.

Stops script evaluation, and returns success

This is the **DEFAULT BEHAVIOR** for unknown opcodes

OP_CLTVDROP
OP_CSVDROP

OP_CLTVDROP // OP_CSVDROP

Behaves as CLTV and CSV

Pops a stack item

Fixes that annoying OP_NOP thing

OP_PULL
OP_DEREF

OP_PULL // OP_DEREF

New ways to read the stack

Copy stack item at specified index

Pull from top, or deref from the bottom

Kill OP_PICK

OP_PULL // OP_DEREF

STACK

0x01

<48 byte pubkey>

0xFF

SCRIPT

OP_PULL

0x02

OP_DEREF

0x01

OP_PULL // OP_DEREF

STACK	SCRIPT
<48 byte pubkey>	OP_DEREF
0x01	0x01
<48 byte pubkey>	
0xFF	

OP_PULL // OP_DEREF

STACK	SCRIPT
0xFF	
<48 byte pubkey>	
0x01	
<48 byte pubkey>	
0xFF	

OP_IFJUMP
OP_IFNJUMP
OP_JUMP

OP_IFJUMP // OP_IFNJUMP // OP_JUMP

New flow controls

Jump forward fixed number of bytes

Replace OP_ELSE and OP_ENDIF

OP_IFJUMP // OP_IFNJUMP // OP_JUMP

```
---
OP_IFJUMP 0x33

<pubkey 1 0x30 bytes> OP_BLSAGGREGATE

OP_JUMP 0x31

<pubkey 2 0x30 bytes> OP_BLSAGGREGATE
```

What’s OP_BLSAGGREGATE?

OP_BLSAGGREGATE
OP_BLSAGGREGATEFROMSTACK

OP_BLSAGGREGATE

Reads a pubkey from the stack

Computes and caches the mapping

Adds it to the aggregation verification queue

OP_BLSAGGREGATEFROMSTACK

Reads a pubkey and message from the stack

Computes and caches the mapping

Adds it to the aggregation verification queue

This may not make sense to you yet

Don’t worry about it :)

So now what?

Bitcoin -> Chia transpiler

Transpiler Example

Old and Busted

```
OP_IF

<pubkey> OP_CHECKSIGVERIFY

OP_ELSE

<pubkeys> OP_CHECKMULTISIGVERIFY

OP_ENDIF
```

New Hotness

```
OP_IFNJUMP 0x33

<pubkey> OP_BLSAGGREGATE

OP_JUMP 0x31

<msig pubkey> OP_BLSAGGREGATE
```

MAST tooling

MAST Tooling Ideas

Merklize a script

Select portions to execute

Verify the merkle proof

Execute the script

Do it all from command line with debugging

Alright

Back to BLS

BLS is a signature scheme

Computational Diffie-Hellman Problem

In a cyclic group

if you know (g^x, g^y)

compute g^{xy}.

Decisional Diffie-Hellman Problem

In a cyclic group

if you know (g^x, h^y)

determine if x == y.

Co-decisional Diffie-Hellman Problem

In two cyclic groups

if you know (a, a^x, b, b^y)

where a is in G_1 and b is in G_2,

determine if x == y.

These problems can be easy or hard

ECDSA relies on CDH being hard

BLS has CDH and DDH as hard, but co-DDH easy

Bilinear Mapping

$$a \in G_1;\ b \in G_2$$

$$e(a^x, b^y) == e(a, b^y)^x$$

$$== e(a^x, b)^y$$

$$== e(a, b)^{xy}$$

This lets us make a really nice sig scheme

BLS Signature Scheme

- Keygen (priv) $x \in \mathbf{Z}_p$
- Keygen (pub) $v := g_2^x \in G_2$
- Hashing $h := H(M) \in G_1$
- Signing $\sigma := h^x \in G_1$
- Verification $e(\sigma, g_2) =? e(h, v) \in G_T$

Verification

$$e(\sigma, g_2) =? e(h, v)$$

$$\sigma := h^x$$

$$v := g_2{}^x$$

$$e(h^x, g_2) =? e(h, g_2{}^x)$$

Aggregate Signatures

- Keygen (priv) $x_i \in \mathbf{Z}_p$
- Keygen (pub) $v_i := g_2^{xi} \quad \in G_2$
- Hashing $h_i := H(M_i) \quad \in G_1$
- Signing $\sigma_i := h^{xi} \quad \in G_1$
- Verification $e(\Pi\sigma_i, g_2) =? \Pi(e(h_i, v_i))$

Aggregate Verification

$$e(\Pi\sigma_i, g_2) =? \Pi(e(h_i, v_i))$$

$$e(\sigma_1\sigma_2, g_2) =? e(h_1, v_1) * e(h_2, v_2)$$

$$e(h_1^{x1}h_2^{x2}, g_2) =? e(h_1, g_2^{x1}) * e(h_2, g_2^{x2})$$

Problems

More assumptions, fewer implementations

Pubkeys are 96 bytes

Pubkeys are revealed on-chain

Solution: Reverse BLS

BLS Signature Scheme

- Keygen (priv) $x \in \mathbf{Z}_p$
- Keygen (pub) $v := g_1^x \in G_1$
- Hashing $h := H(M) \in G_2$
- Signing $\sigma := h^x \in G_2$
- Verification $e(g_1, \sigma) =? e(v, h)$

Verification

$$e(g_{1,}\ \sigma)\ {=}?\ e(v,\ h)$$

$$\sigma\ := h^{x}$$

$$v\ := g_{1}{}^{x}$$

$$e(g_{1},\ h^{x})\ {=}?\ e(g_{1}{}^{x},\ h)$$

Now pubkeys are 48 bytes

And we need just one signature per tx

Space savings start at 2 inputs

Questions?