

# Miniscript

**An introduction to BIP 379**

**<https://github.com/qlrd/miniscript-presentation>**

qlrd

# Miniscript

## **Definition 1:** BIP 379.

(...) a language for writing (a subset of) **Bitcoin Scripts** in a structured way, enabling analysis, composition, generic signing and more. [1]

# Back to the basics

# Bitcoin script

## **Definition 2: .**

(...) an unusual stack-based language with many edge cases designed for implementing spending conditions consisting of various combinations of signatures, hash locks, and time locks. [1]

## Bitcoin script

Common transactions from [2] and [3]

Comment	Unlock	Lock
P2PK	<sig> <pk>	OP_CHECKSIG
P2PKH	<sig> <pk>	OP_DUP OP_HASH160 <pkh> OP_EQUALVERIFY OP_CHECKSIG
Multisig 2-of-3	OP_0 <sigA> <sigB>	2 <pkA> <pkB> <pkC> 3 OP_CHECKMULTISIG

## Bitcoin script

Freezing funds until a time in the future from [2]

Unlock	Lock
<sig> <pk>	<expiry time> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP OP_HASH160 <pkh> OP_EQUALVERIFY OP_CHECKSIG

## Bitcoin script

Timelock variable multisignature from [3]: 2-of-3 multisig; after 30 days 1-of-3 with a lawyers's signature; after 90 days the lawyer's signature.

Unlock	Lock
<pre>OP_0 &lt;sigA&gt; &lt;sigB&gt; OP_TRUE       OP_TRUE</pre>	<pre>OP_IF OP_IF 2 OP_ELSE &lt;30 days&gt; OP_CHECKSEQUENCEVERIFY       OP_DROP &lt;sigD&gt; OP_CHECKSIGVERIFY 1 OP_ENDIF       &lt;sigA&gt; &lt;sigB&gt; &lt;sigC&gt; 3 OP_CHECKMULTISIG OP_ELSE &lt;90 days&gt; OP_CHECKSEQUENCEVERIFY       OP_DROP &lt;sigD&gt; OP_CHECKSIG       OP_ENDIF</pre>

# The issue [1]



Given a combination of spending conditions, it is challenging to:

- find the most economical script to implement it;
- implement a composition of their spending conditions;
- find out what spending conditions it permits.

...

# The motivation

**Miniscript** has a structure that allows composition: a representation for **scripts** that makes these type of operations possible.

## Miniscript [4]

Policy for a singlesig

Miniscript	Script
<code>pk(&lt;key_1&gt;)</code>	<code>&lt;key_1&gt; OP_CHECKSIG</code>

## Miniscript [4]

Miniscript for One of two keys (equally likely)

Miniscript	Script
<pre>or_b(   pk(key_1),   s:pk(key_2) )</pre>	<pre>&lt;key_1&gt; OP_CHECKSIG OP_SWAP &lt;key_2&gt; OP_CHECKSIG OP_BOOLOR</pre>

## Miniscript [4]

Miniscript for One of two keys (one likely, one unlikely)

Miniscript	Script
<pre>or_d(   pk(key_1),   pkh(key_2) )</pre>	<pre>&lt;key_1&gt; OP_CHECKSIG OP_IFDUP OP_NOTIF OP_DUP OP_HASH160 &lt;HASH160(key_2)&gt; OP_EQUALVERIFY OP_CHECKSIG OP_ENDIF</pre>

## Miniscript [4]

Miniscript for 3-of-3 that turns into a 2-of-3 after 90 days

Miniscript	Script
<pre>thresh(   3,   pk(key_1),   s:pk(key_2),   s:pk(key_3),   sln:older(12960) )</pre>	<pre>&lt;key_1&gt; OP_CHECKSIG OP_SWAP &lt;key_2&gt; OP_CHECKSIG OP_ADD OP_SWAP &lt;key_3&gt; OP_CHECKSIG OP_ADD OP_SWAP OP_IF 0 OP_ELSE &lt;a032&gt; OP_CHECKSEQUENCEVERIFY OP_0NOTEQUAL OP_ENDIF OP_ADD 3 OP_EQUAL</pre>

## Miniscript [4]

Miniscript for Lightning: BOLT #3 to\_local.

Miniscript	Script
<pre>andor(   pk(key_local),   older(1008),   pk(key_revocation) )</pre>	<pre>&lt;key_local&gt; OP_CHECKSIG OP_NOTIF &lt;key_revocation&gt; OP_CHECKSIG OP_ELSE &lt;f003&gt; OP_CHECKSEQUENCEVERIFY OP_ENDIF</pre>



# Specification [1]

# Specification

Miniscript analyzes scripts to determine properties.

# Specification

**Not expected** to be used with:

- BIP 16 (p2sh);

**Expected** to be used within:

- BIP 382: wsh descriptor;
- BIP 386: tr descriptor.

And together with:

- BIP 380: Key expressions:

[<fingerprint>/<purpose>/<cointype>/<index>]

# Specification

From a user's perspective, Miniscript is not a separate language, but rather a significant expansion of the descriptor language. [1]

# Specification

Liana's simple inheritance wallet [5].

```
wsh(  
  or_d(  
    pk([07fd816d/48'/1'/0'/2']tpub...wd5/<0;1>/*),  
    and_v(  
      v:pkh([da855a1f/48'/1'/0'/2']tpub...Hg5/<0;1>/*),  
      older(36)  
    )  
  )  
)#lz4jfr7g
```

## Specification

Liana's simple inheritance wallet TR [6]. First key expression is a NUMS (“nothing-up-my-sleeves”) point [7].

```
tr(  
  [07fd816d/48'/1'/0'/2']tpub...mwd5/<0;1>/*,  
  and_v(  
    v:pk([da855a1f/48'/1'/0'/2']tpub...Hg5/<0;1>/*),  
    older(36)  
  )  
)#506utvsp
```

## Specification

Liana's variable multisig [8].

```
wsh(  
  or_d(  
    multi(2,  
      [07fd816d/48'/1'/0'/2']tpub...wd5/<0;1>*,  
      [da855a1f/48'/1'/0'/2']tpub...Hg5/<0;1>*  
    ),  
    and_v(  
      v:thresh(2,  
        pkh([07fd816d/48'/1'/0'/2']tpub...mwd5/<2;3>*),  
        a:pkh([da855a1f/48'/1'/0'/2']tpub...Hg5/<2;3>*),  
        a:pkh([cdef7cd9/48'/1'/0'/2']tpub...Ak2/<0;1>*)  
      ),  
      older(36)  
    )  
  ) )#wa74c6se
```

## Specification

Liana's variable multisig TR [9]. First key expression is a NUMS (“nothing-up-my-sleeves”) point [7].

```
tr(tpub...pMN/<0;1>/*, {  
  and_v(  
    v:multi_a(2,  
      [07fd816d/48'/1'/0'/2']tpub...mwd5/<2;3>/*,  
      [da855a1f/48'/1'/0'/2']tpub...DHg5/<2;3>/*,  
      [cdef7cd9/48'/1'/0'/2']tpub...SAk2/<0;1>/*  
    ),  
    older(36)  
  ),  
  multi_a(2,  
    [07fd816d/48'/1'/0'/2']tpub...mwd5/<0;1>/*,  
    [da855a1f/48'/1'/0'/2']tpub...DHg5/<0;1>/*  
  )  
})#tvh3u2lu
```



# Specification

- **Translation** table;
- **type** system;
- condition **satisfaction** system;

# Translation

## Definition 3: .

**Miniscript** consists of a set of **script** fragments which are designed to be safely and correctly composable (...) targeted by spending policy compilers

# Translation

## Normal fragments

`fragment(arg1)`

`fragment(arg1,arg2,...)`

## Translation

Wrappers: fragments that do not change the semantics of their subexpressions, separated by a colon and each one is applied to the next fragment

Fragments	Interpretation
<code>x:fragment(arg)</code>	<code>x -&gt; fragment</code>
<code>xy:fragment(arg)</code>	<code>x -&gt; y -&gt; fragment</code>
<code>xyz:fragment(arg)</code>	<code>x -&gt; y -&gt; z -&gt; fragment</code>

# Translation

Simple validation semantics

Miniscript	Script
0	0
1	1

# Translation

Check key semantics

Miniscript	Script
<code>pk_k(key)</code>	<code>&lt;key&gt;</code>
<code>pk_h(key)</code>	<code>DUP HASH160 &lt;HASH160(key)&gt; EQUALVERIFY</code>

## Translation

Wrapped check key semantics

Miniscript	Script
$\text{pk}(\text{key}) = \text{c:pk}_k(\text{key})$	<code>&lt;key&gt; CHECKSIG</code>
$\text{pkh}(\text{key}) = \text{c:pk}_h(\text{key})$	<code>DUP HASH160 &lt;HASH160(key)&gt; EQUALVERIFY CHECKSIG</code>

# Translation

## Time semantics

Miniscript	Script
older(n)	<n> CHECKSEQUENCEVERIFY
after(n)	<n> CHECKLOCKTIMEVERIFY



# Translation

## Hash semantics

Miniscript	Script
sha256(h)	SIZE <20> EQUALVERIFY SHA256 <h> EQUAL
hash256(h)	SIZE <20> EQUALVERIFY HASH256 <h> EQUAL
ripemd160(h)	SIZE <20> EQUALVERIFY RIPEMD160 <h> EQUAL
hash160(h)	SIZE <20> EQUALVERIFY HASH160 <h> EQUAL

# Translation

## Boolean semantics

Miniscript	Script
andor(X,Y,Z)	[X] NOTIF [Z] ELSE [Y] ENDIF
and_v(X,Y)	[X] [Y]
and_b(X,Y)	[X] [Y] BOOLAND
and_n(X,Y) = andor(X,Y,0)	[X] NOTIF 0 ELSE [Y] ENDIF
or_b(X,Z)	[X] [Z] BOOLOR
or_c(X,Z)	[X] NOTIF [Z] ENDIF
or_d(X,Z)	[X] IFDUP NOTIF [Z] ENDIF
or_i(X,Z)	IF [X] ELSE [Z] ENDIF

# Translation

## Multisig semantics

Only	Miniscript	Script
	<code>thresh(k,X_1,...,X_n)</code>	<code>[X_1] [X_2] ADD ... [X_n] ADD ... &lt;k&gt; EQUAL</code>
<code>p2wsh</code>	<code>multi(m,key_1,...,key_n)</code>	<code>&lt;k&gt; &lt;key_1&gt; ... &lt;key_n&gt; &lt;n&gt; CHECKMULTISIG</code>
<code>tapscript</code>	<code>multi_a(k,key_1,...,key_n)</code>	<code>&lt;key_1&gt; CHECKSIG &lt;key_2&gt; CHECKSIGADD ... &lt;key_n&gt; CHECKSIGADD &lt;k&gt; NUMEQUAL</code>

# Translation

## Wrappers semantics

Miniscript	Script
a:X	TOALTSTACK [X] FROMALTSTACK
s:X	SWAP [X]
c:X	[X] CHECKSIG
t:X = and_v(X,1)	[X] 1
d:X	DUP IF [X] ENDIF
v:X	[X] VERIFY (or VERIFY version of last opcode in [X])
j:X	SIZE 0NOTEQUAL IF [X] ENDIF
n:X	[X] 0NOTEQUAL
l:X = or_i(0,X)	IF 0 ELSE [X] ENDIF
u:X = or_i(X,0)	IF [X] ELSE 0 ENDIF

# Type system

# Type system

Not every Miniscript expression can be composed with every other.

# Type system

BIP379 define a correctness type system for Miniscript to model properties and its requirements:

- Correctness;
- timelock mixing;
- malleability.

# Type system (correctness)

- Basic types
  - B: Base;
  - V: Verify;
  - K: Key;
  - W: Wrapped;
- Type modifiers
  - z: zero-arg;
  - o: one-arg;
  - n: non-zero;
  - d: dissatisfiable;
  - u: unit.



## Type system (correctness)

Keys semantics.

Miniscript	Requires	Type	Properties
pk_k(key)		K	o; n; d; u
pk_h(key)		K	n; d; u

# Type system (correctness)

Time semantics.

Miniscript	Requires	Type	Properties
older(n), after(n)	$1 \leq n < 2^{31}$	B	z

## Type system (correctness)

Hash semantics.

Miniscript	Requires	Type	Properties
sha256(h)		B	o; n; d; u
ripemd160(h)		B	o; n; d; u
hash256(h)		B	o; n; d; u
hash160(h)		B	o; n; d; u

## Type system (correctness)

Boolean semantics.

Miniscript	Requires	Type	Properties
<code>andor(X,Y,Z)</code>	X is Bdu; Y and Z are both B, K, or V	same as Y/Z	$z = zXzYzZ;$ $o = zXoYoZ$ or $oXzYzZ;$ $u = uYuZ;$ $d = dZ$
<code>and_v(X,Y)</code>	X is V; Y is B, K, or V	same as Y	$z = zXzY;$ $o = zXoY$ or $zYoX;$ $n = nX$ or $zXnY;$ $u = uY$

## Type system (correctness)

Multisig semantics.

Miniscript	Requires	Type	Properties
<code>thresh(   k,   X1,   ...,   Xn )</code>	$1 \leq k \leq n$ ; X1 is Bdu; others are Wdu	B	z=all are z; o=all are z except one is o; d; u

# Type system (timelock mixing)

Four timelock types:

- absolute or relative time-based;
- absolute or relative height-based;

## Type system (timelock mixing)

must not be mixed in an incompatible way:

## Type system (timelock mixing)

It is not valid to mix height-based **and** time-based timelocks in:

- and fragment combinations; and
- thresh fragment combinations where  $k \geq 2$ ,

For all other combinators, it is legal to mix timelock types.



## Type system (malleability)

Ability for a third party to modify an existing satisfaction into another valid satisfaction.

## Type system (malleability)

**Third party:** someone who does not hold a participating private key

## Type system (malleability)

To analyze the malleability guarantees of a script we define three additional type properties:

- s: signed;
- f: forced;
- e: expressive.

# Satisfaction

# Satisfaction

The Miniscript-compliant data (e.g., signatures, preimages) required to authorize a Bitcoin script's execution by meeting its spending conditions.

## Satisfaction

Examples for key semantics. See more at [BIP 379's satisfaction section](#)

Miniscript	Dissatisfaction	Satisfaction
<code>pk_k(key)</code>	<code>0</code>	<code>&lt;sig&gt;</code>
<code>pk_h(key)</code>	<code>0</code>	<code>&lt;sig&gt; &lt;pubKey&gt;</code>

## Satisfaction

Examples for key semantics. See more at [BIP 379's satisfaction section](#)

Miniscript	Dissatisfaction	Satisfaction
sha256(h)	any 32-byte vector except the preimage	preimage
hash160(h)	any 32-byte vector except the preimage	preimage

## Satisfaction

Examples for multisig semantics. See more at [BIP 379's satisfaction section](#)

Miniscript	Dissatisfaction	Satisfaction
<code>multi(   k   key_1,   ...,   key_n )</code>	<code>0 0 ... 0</code>	<code>0 &lt;sig1&gt; &lt;sig2&gt; ... &lt;sigN&gt;</code>



# Implementations

- Peter Wuile's reference implementation
- C++:
  - Bitcoin-core
- Rust:
  - rust-miniscript
  - Liana
- Go:
  - Tutorial: Understanding Bitcoin Miniscript - Part III
- Python:
  - Embit's miniscript.py
  - Krux (branch p2wsh\_miniscript)
  - Krux (branch tr\_miniscript)

**Thanks!**

# Bibliography

- [1] Bitcoin Improvement Proposals, “BIP 379: Miniscript Policy.” [Online]. Available: <https://github.com/bitcoin/bips/blob/master/bip-0379.md>
- [2] Bitcoin FAQ, “Script.” [Online]. Available: <https://en.bitcoin.it/wiki/Script>
- [3] A. M. Antonopoulos and D. A. Harding, “Mastering Bitcoin: Programming the Open Blockchain (Third Edition).” [Online]. Available: <https://github.com/bitcoinbook/bitcoinbook>
- [4] P. Wuille, “Miniscript: A New Language for Bitcoin Scripts.” [Online]. Available: <https://bitcoin.sipa.be/miniscript/>
- [5] jdldcl, “Bitcoin Core Watch-Only: Liana Simple-Inheritance WSH.” [Online]. Available: <https://gist.github.com/jdldcl/b0dea22a8a6caf0fd7c40b244357d8d2>

- [6] jdlcdl, “Bitcoin Core Watch-Only: Liana Simple-Inheritance TR.” [Online]. Available: <https://gist.github.com/jdlcdl/b17c6b551839adb5b7b7d4ef9574e48e>
- [7] jaonoctus, “NUMS secp256k1 .” [Online]. Available: <https://nums-secp256k1.jaonoctus.dev/>
- [8] jdlcdl, “Bitcoin Core Watch-Only Liana Expanding-Multi WSH.” [Online]. Available: <https://gist.github.com/jdlcdl/d83a83ec7d47d98888a8647b636d567d>
- [9] jdlcdl, “Bitcoin Core Watch-Only Liana Expanding-Multi TR.” [Online]. Available: <https://gist.github.com/jdlcdl/c38e1b80cd814e48e1d158a98cf704f6>