Miniscript

Un pas vers de vrais smart contract Bitcoin ?

Sosthène

www.sosthene.net @Sosthene___ contact@sosthene.net

23AA B267 CA6A B731 DCD1 4254 014A 7F46 6A0E EE28

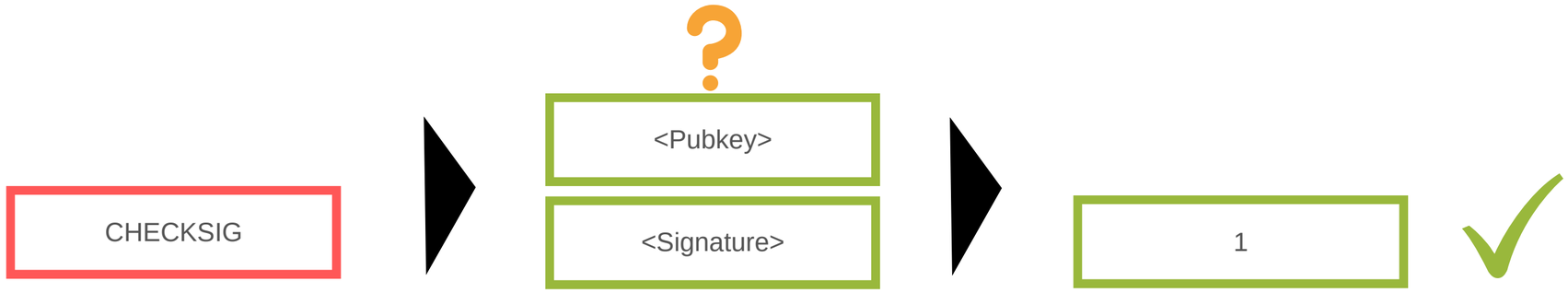
Neuchâtel Bitcoin Meetup 16/10/2019

*Miniscript is, in a theoretical sense, more limiting than script, but it can do everything that people **actually** use script for.*

Peter Wuille

Qu'est-ce que Script ?

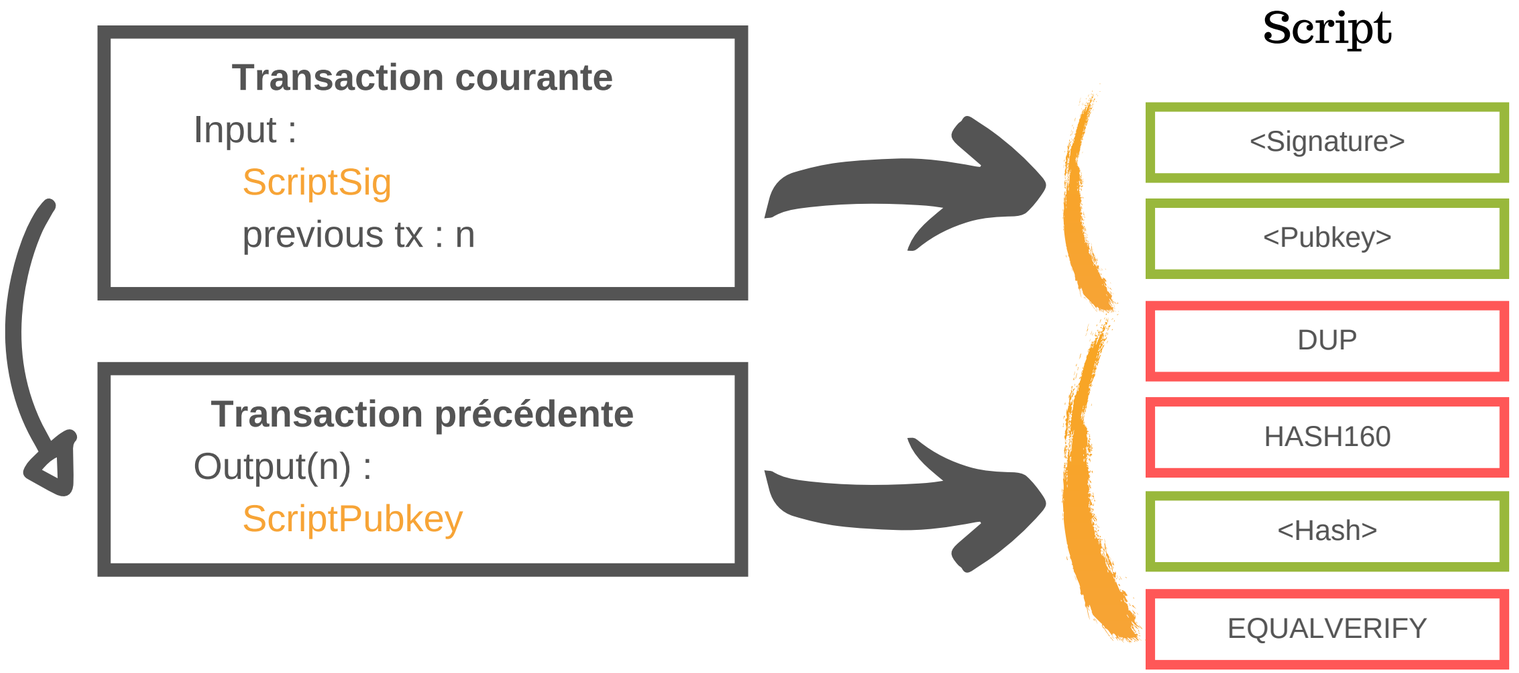
- Un langage *stack-based* ("push"/"pop") inspiré de FORTH
- Inventé par Satoshi, il a peu évolué jusqu'à aujourd'hui
- Script permet de définir les conditions auxquelles un UTXO peut être dépensé, et de tester si un input remplit effectivement ces conditions (="smart-contract")
- Il comprend des data (*elements*) et des opérations (*op_codes*)
- Si une fois toutes les opérations effectuées, l'élément sur le haut de la stack est différent de 0 : le script est valide



Comment construire un Script ?

Un script est constitué de deux parties :

- ScriptSig dans l'input de la transaction courante
- ScriptPubkey dans l'output (UTXO) d'une transaction antérieure



Les opcodes

Il existe plus d'une centaine d'opcodes :

- pour contrôler le flux du Script : IF/NOTIF, ELSE, VERIFY
- Pour interagir avec les éléments de la pile : TOALTSTACK, SWAP, IFDUP
- pour effectuer des opérations cryptographiques : RIPEMD160, SHA256, CHECKSIG
- pour effectuer des opérations arithmétique sur les data, opérateurs bitwise etc
- Beaucoup d'opcodes ont été désactivés par soucis de sécurité, mais même ainsi l'écrasante majorité ne sont virtuellement jamais utilisés
- 99,99999% des transactions utilisent des templates : P2PK, P2PKH, P2SH, P2WKH...

Les défis de Script

- Analysabilité : les scripts sont difficiles à lire et à analyser
  - sécurité : est-ce que seule une dépense autorisée est possible ?
  - optimisation : probabilité de réalisation d'une branche ? Fees ?
- Usabilité : il est risqué de concevoir un script en-dehors des templates couramment utilisés. De nombreux opcodes ne sont jamais utilisés en pratique car inutiles/dangereux
- Compatibilité : faire interagir différents services sur les mêmes scripts n'est généralement pas possible ou du moins très difficile
  - Non-composabilité : il est difficile de composer un script à partir de plusieurs scripts existants
- **Script rend difficile la réalisation de conditions de dépenses au-dessus d'un certain niveau de complexité**

Enter Miniscript

- miniscript est un subset des opérations possibles dans Script, qui comprend les opérations sur les clés, les timelocks et les hashes
- un langage "haut niveau" permettant de rédiger et de représenter les scripts d'une façon humainement lisible
- un interpréteur permettant de composer des scripts à partir de ce langage
- miniscript ne concerne que la partie scriptpubkey, il n'évalue pas les scripts lors de la dépense. Il s'agit d'un langage combinatoire pour concevoir des conditions de dépenses d'UTXO
- Workflow :
  - Rédiger une policy
  - Compiler un miniscript
  - Encoder un script

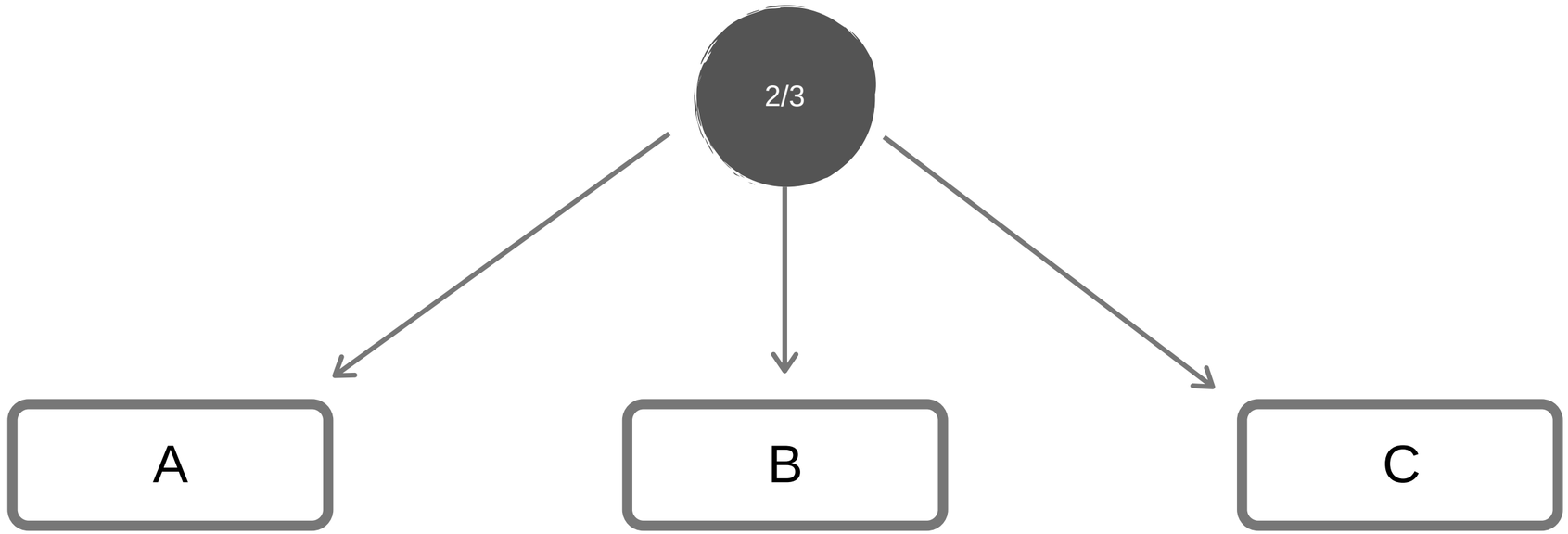
  - (Représenter une policy)
  - Analyser un miniscript
  - Décoder un script

Exigences

- Compatibilité avec les règles du consensus et de "standardness" (=node policy)
- "Soundness" : il est impossible de produire un witness valide sans satisfaire les conditions exprimées dans la policy
- Non-malléabilité (si certaines conditions sont remplies, par exemple l'attaquant ne possède pas de clés privées)

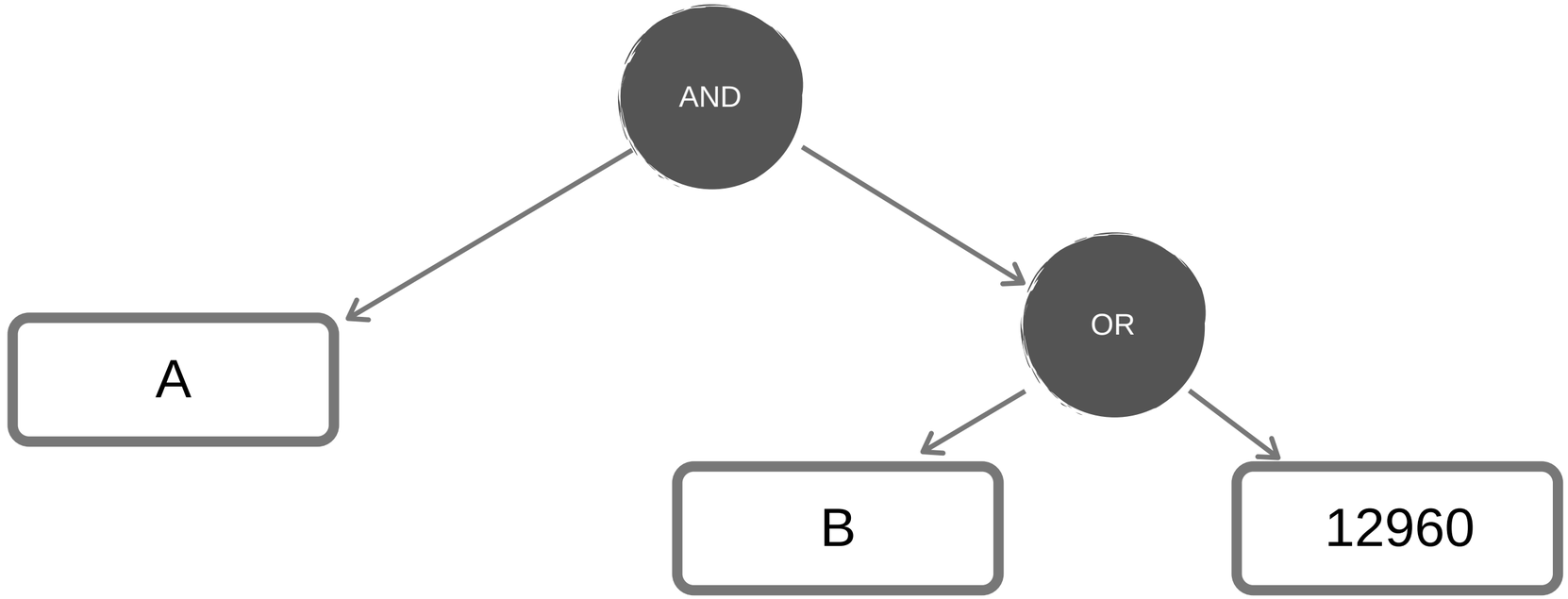
Exemple de policy - Escrow

- Policy : thresh(2,pk(A),pk(B),pk(C))
- Miniscript : thresh_m(2,A,B,C)
- Script : 2 <A> <B> <C> 3 OP_CHECKMULTISIG



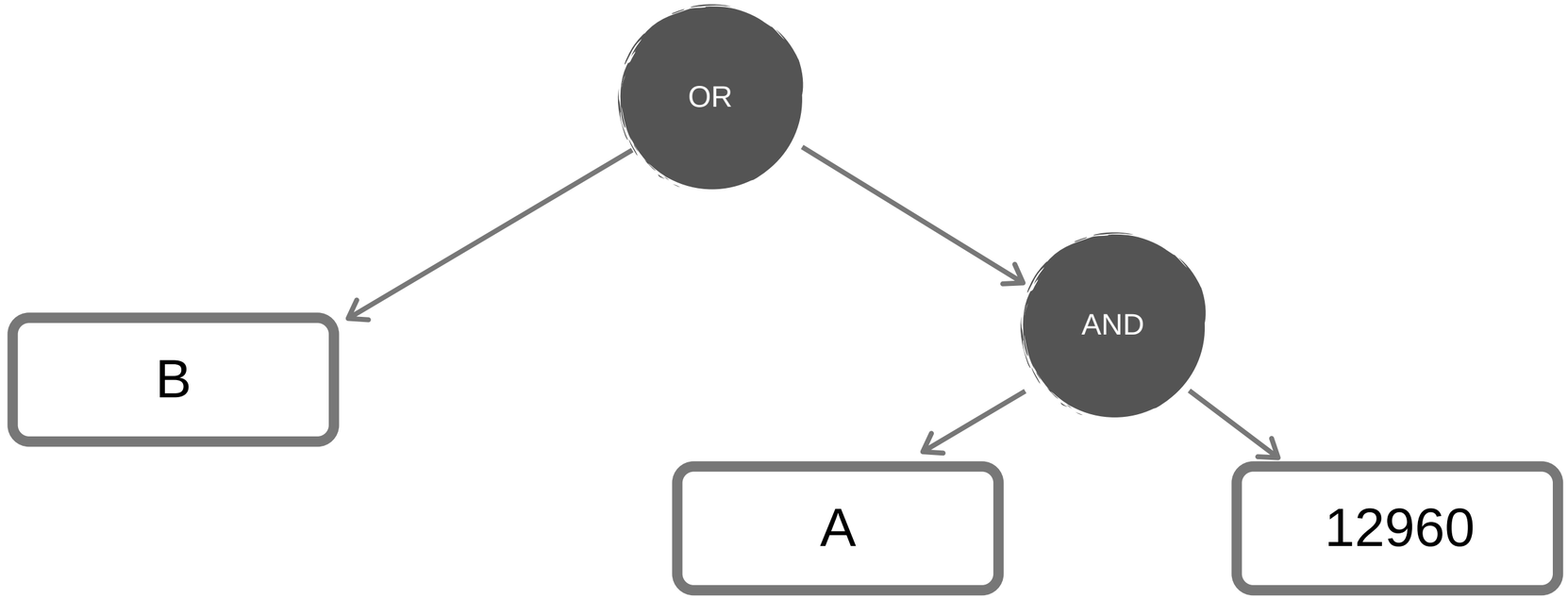
Exemple de policy - 2-of-2 avec Timelock

- Policy : and(pk(A),or(99@pk(B),older(12960)))
- Miniscript : and_v(vc:pk(A),or_d(c:pk(B),older(12960)))
- Script : <A> OP_CHECKSIGVERIFY <B> OP_CHECKSIG OP_IFDUP OP_NOTIF <a032> OP_CHECKSEQUENCEVERIFY OP_ENDIF



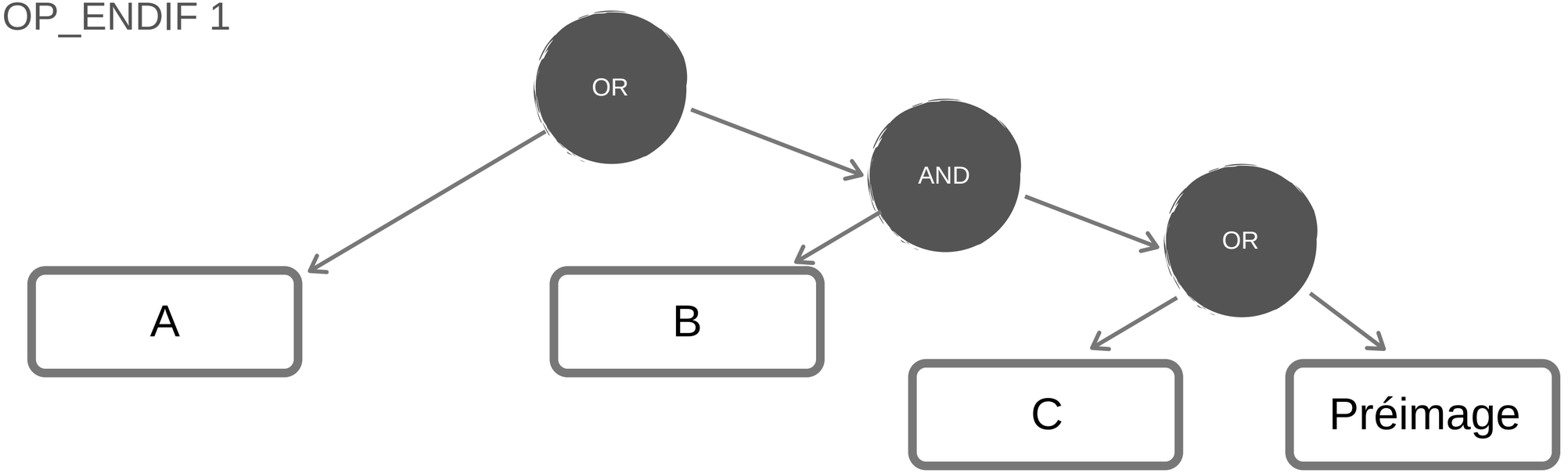
Exemple de policy - une autre clé dépense après 90 jours

- Policy : or(and(pk(A),older(12960)),99@pk(B))
- Miniscript : or_d(c:pk(B),and_v(vc:pk_h(A),older(12960)))
- Script : <B> OP_CHECKSIG OP_IFDUP OP_NOTIF OP_DUP OP_HASH160 <HASH160(A)> OP_EQUALVERIFY OP_CHECKSIGVERIFY <a032> OP_CHECKSEQUENCEVERIFY OP_ENDIF



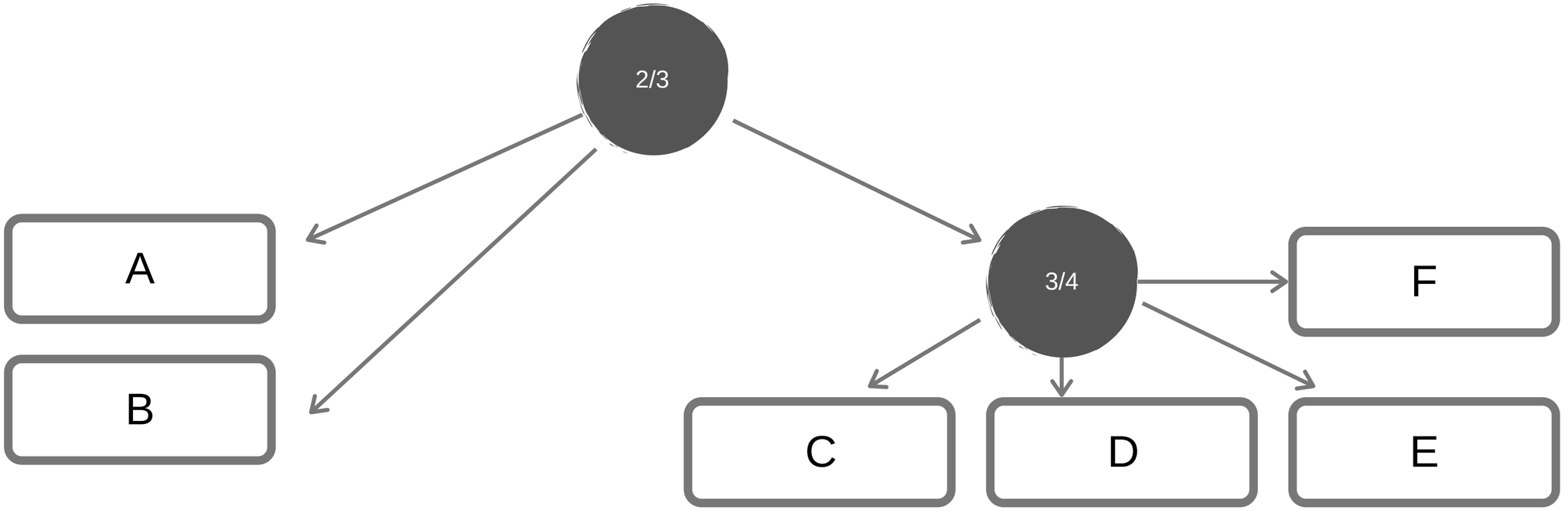
Exemple de policy - HTLC

- Policy : or(pk(A),and(pk(B),or(pk(C),hash160(H))))
- Miniscript : t:or_c(c:pk(A),and_v(vc:pk(B),or_c(c:pk(C),v:hash160(H))))
- Script : <A> OP_CHECKSIG OP_NOTIF <B> OP_CHECKSIGVERIFY <C> OP_CHECKSIG OP_NOTIF OP_SIZE <20> OP_EQUALVERIFY OP_HASH160 <h> OP_EQUALVERIFY OP_ENDIF OP_ENDIF 1



Remplacer une clé par une autre expression

- Policy : thresh(2,pk(A),pk(B),thresh(3,pk(C),pk(D),pk(E),pk(F)))
- Miniscript : thresh(2,thresh_m(3,C,D,E,F),sc:pk(A),sc:pk(B))
- Script : 3 <C> <D> <E> <F> 4 OP_CHECKMULTISIG OP_SWAP <A> OP_CHECKSIG OP_ADD OP_SWAP <B> OP_CHECKSIG OP_ADD 2 OP_EQUAL



2 implémentations

https://github.com/sipa/miniscript

https://github.com/apoelstra/rust-miniscript

Travaux liés

- Output descriptors : un langage conçu par Wuille et Poelstra pour décrire les outputs des transactions Bitcoin
  sh(multi(2,022f01e5e15cca351daff3843fb70f3c2f0a1bdd05e5af888a67784ef3e10a2a01,03acd484e2f0c7f65309ad178a9f559abde09796974c57e714c35f110dfc27ccbe))
- Ivy : un autre langage bas-niveau pour programmer des scripts bitcoin. Il n'est pas aussi facile à analyser que Miniscript
- Simplicity : autre langage bas-niveau pour programmer et surtout obtenir des preuves formelles de validité d'un script

Travaux liés

- PSBT (partially signed bitcoin transaction) : format standard pour représenter des transactions à des états intermédiaires. Il permet à différents services ou wallets d'interagir pour updater et signer une transaction.

Tant qu'un script n'utilise que des opérations comprises dans le cadre de Miniscript, un script peut être échangé et updaté par différents services en compilant et en parsant les scripts bitcoins obtenus

Cas d'usage

- Lightning : signer les updates des transactions lightning dans Bitcoin Core, ou avec un HW wallet
- Faciliter la maintenance des wallets : même si les scripts évoluent, tant qu'un wallet comprend miniscript, il a toutes les informations pour produire un witness qui permet de déverrouiller un script
- Compatibilité dans le futur : garantit qu'un script produit aujourd'hui avec une solution donnée pourra toujours être dépensée à l'avenir
- Fédération (Liquid) : permet de mettre à jour et d'auditer les scripts quand les participants de la fédération évoluent

Sources

- Bitcoin Optech Newsletter
- Peter Wuille à Stanford (01/2019)
- Article d'Andrew Poelstra
- Présentation de BitHyves
- Démo en js + doc par Sipa

Question ?