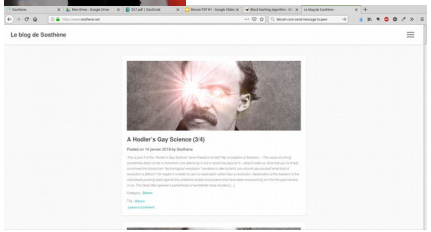
Bitcoin P2P

Schnorr, Taproot

Objectifs

1. Faire le point sur quelques développements majeurs en cours sur Bitcoin
2. Comprendre les trade-offs
3. Que penser du soft fork à venir ?

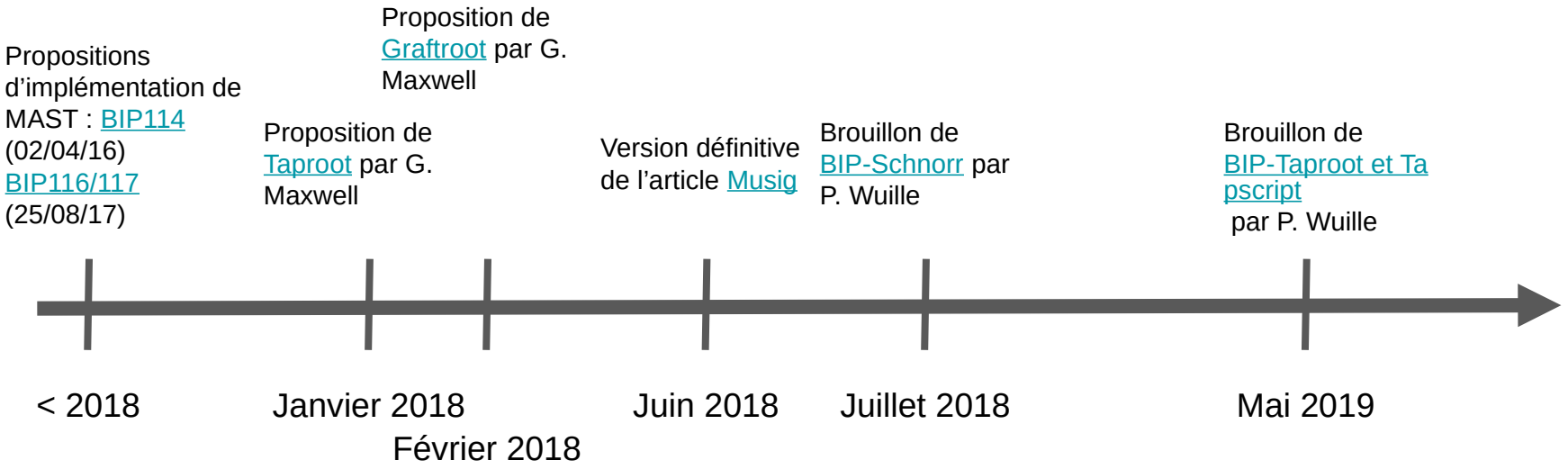
www.sosthene.net
@Sosthene@bitcoinhackers.org



Schnorr ?

- Schnorr : algorithme de signature (en remplacement d’ECDSA)
- Musig : protocole de multisignature tirant parti de Schnorr
- MAST : “Merklized Abstract Syntax Tree”, une idée qui est discuté dans Bitcoin depuis au moins 2013
- Taproot : implémentation de MAST
- Graftroot : proposition qui permet d’obtenir les mêmes résultats que MAST par des moyens différents (délégation)

Timeline


Propositions d'implémentation de MAST : BIP114 (02/04/16) BIP116/117 (25/08/17)
Proposition de Graftroot par G. Maxwell
Proposition de Taproot par G. Maxwell
Version définitive de l'article Musig
Brouillon de BIP-Schnorr par P. Wuille
Brouillon de BIP-Taproot et Tapscript par P. Wuille
< 2018
Janvier 2018
Février 2018
Juin 2018
Juillet 2018
Mai 2019

Schnorr

Schnorr

- Algorithme de signature décrit par Claus Schnorr en 1989
- Utilise les mêmes bases mathématiques qu'ECDSA (courbes elliptiques)
- Il est très simple, sûr, fournit des signatures relativement compactes (64 bytes), et a d'autres propriétés intéressantes :
  - les signatures sont non malléables
  - les signatures peuvent être agrégées
- Protégée par un brevet jusqu'en février 2008, au moment où Satoshi écrivait Bitcoin il n'existait donc pas de librairies standards incluant les signatures de Schnorr, c'est pourquoi il préféra utiliser ECDSA en dépit de son infériorité sur quasiment tous les plans

ECDSA vs Schnorr

- Compatibilité des clés privées et publiques avec l’existant ECDSA
- Sécurité prouvable : je peux prouver que j’ai la clé privée correspondant à une clé publique sans dévoiler d’informations (0 knowledge proof)
- Linéarité: la somme de *n* clés peut valider la somme de *n* signatures produites par ces mêmes clés
- Non-malléabilité: il n’est pas possible de produire une signature valide sans accès à la clé privée (c’est un problème d’ECDSA qui a été résolu par SegWit)

ECDSA vs Schnorr

Schnorr simplement en remplacement d’ECDSA n’est pas très intéressant :

- Schnorr est un peu plus rapide à vérifier qu’ECDSA
- On peut gagner 6 bytes en se débarrassant du format DER des signatures ECDSA

En revanche, Schnorr est la condition nécessaire pour :

- Optimiser les multisigs (réduction de 30 à 75% de la taille sur la blockchain)
- Améliorer la fungibilité (scripts et multisig indiscernables d’une signature simple)
- Implémenter des scripts plus complexes (via Tap/Graftroot)
- Réaliser des scriptless scripts

Algorithme

Source: https://youtu.be/j9Wvz7zI_Ac

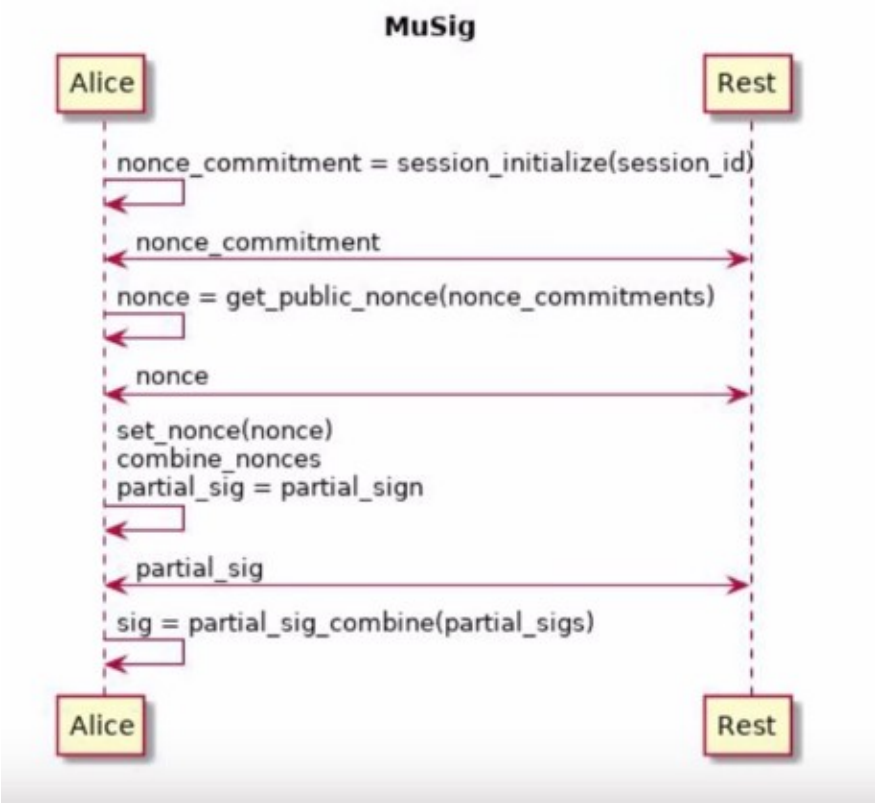
- Génération: P = xG (je multiplie ma clée privée par le générateur *G*)
- Signature:
  - R = kG (je crée un nouveau couple de clés à usage unique, *nonce*)
  - e = H(P,R,m) (je hash ma clé publique, mon nonce et le message)
  - s = k + ex (j'additionne mon nonce "privé" et le produit du hash et de ma clé privée)
  - signature = (s, R)
- Vérification: assert sG == R + eP (= kG + exG)
- *k* doit être aléatoire sans quoi cela met en danger la clé *x*.
- On peut mettre en gage (*commitment*) une donnée *c* directement dans l'algorithme :
  - $R_0 = kG$
  - $R = R_0 + H(R_0 \,||\, c)G$
  - $s = (k + H(R_0 \,||\, x)) + ex$

Multisignature Schnorr

https://youtu.be/j9Wvz7zI_Ac?t=1007

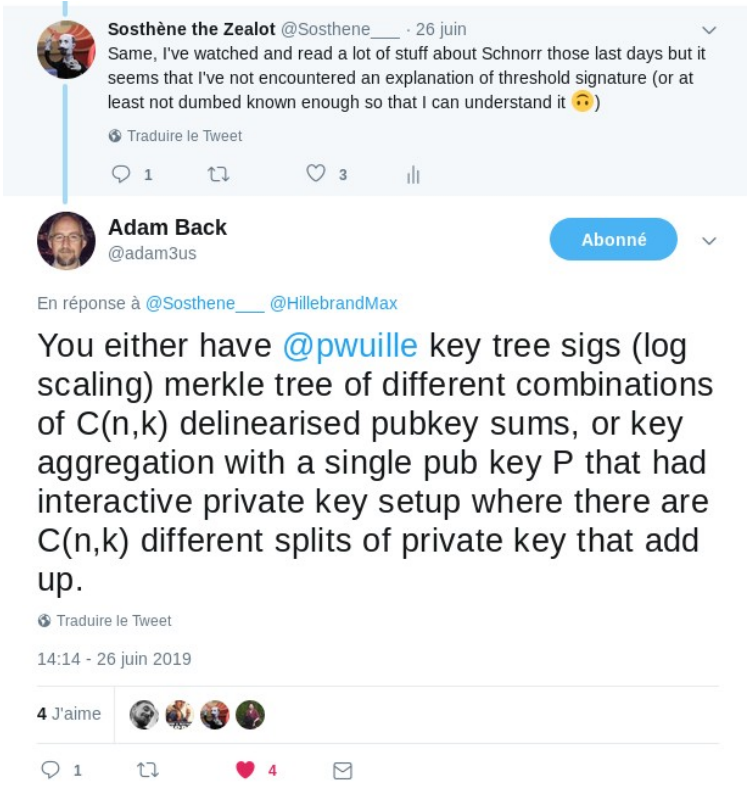
- $P = \Sigma P_i$
- $s = \Sigma k_i + \Sigma e x_i$
- Problème: l'un des signataires peut donner une clé publique qui annule les autres, si bien qu'à la fin il pourra dépenser l'output seul
- Solution proposée dans le cadre de Musig : chaque signataire ajoute un coefficient μ à sa clé publique :
  - $\mu_i = H(H(P_1 \,||\, P_2 \,||\, \dots \,||\, P_n) \,||\, i)$
  - $P_i = \mu_i x_i G$
- Problème : la création de la clé publique et la signature nécessitent une interaction entre les signataires
- Autre difficulté: les multisig avec Schnorr sont *unaccountable*, i.e. impossible de savoir quel subset de clés a produit une signature valide

Protocole signature Musig



https://youtu.be/DKOG0BQMmmg?t=22870

Threshold signatures



Sosthène the Zealot @Sosthene___ · 26 juin

Same, I've watched and read a lot of stuff about Schnorr those last days but it seems that I've not encountered an explanation of threshold signature (or at least not dumbed known enough so that I can understand it 🙃)

Traduire le Tweet

1 3

Adam Back
@adam3us

Abonné

En réponse à @Sosthene___ @HillebrandMax

You either have @pwuille key tree sigs (log scaling) merkle tree of different combinations of C(n,k) delinearised pubkey sums, or key aggregation with a single pub key P that had interactive private key setup where there are C(n,k) different splits of private key that add up.

Traduire le Tweet

14:14 - 26 juin 2019

4 J'aime

1 4

Trade-off entre les différents modes de multisig

source : https://youtu.be/fDJRy6K_3yo?t=1156

Construct	Fungiblity / Fees	Interactive key setup	Interactive signing	Account ability
Musig k-of-n threshold sigs	Great	Yes	Yes	No
Musig k-of-n keytree	Good	No	Yes	Internal
Musig n-of-n	Great	No	Yes	Internal
Traditional	Poor	No	No	Public

Adaptor signature

https://joinmarket.me/blog/blog/flipping-the-scriptless-script-on-schnorr/

- Puisqu'on peut additionner les clés et les signatures, on peut aussi les soustraires et produire des signatures valides !
- Alice va générer 2 signatures, la "vraie" signature et une "adaptor" en utilisant 2 nonces (R, T) au lieu de R seul
- Elle peut ensuite partager l'adaptor s', R et T à Bob. s' = s - t
- Sans s, Bob ne peut pas calculer t qui lui manque pour produire une signature valide
- Si Alice signe une transaction avec s, Bob peut simplement la récupérer dans la blockchain et calculer t, et produire une signature valide !

Cross-input signature aggregation

- En théorie, il est possible d’agréger toutes les pubkey et toutes les signatures fournies dans une transaction
- Cela serait très intéressant pour toutes les constructions qui reposent sur un nombre d’inputs important (CoinJoin)
- cela pose des problèmes d’implémentation en pratique, et risque de ne pas être disponible avant un moment

Taproot

Rappel: les Scripts Bitcoin

- Bitcoin a un langage de smart contracts, simplement appelé “Scritpt”
- C’est un langage volontairement limité :
  - 2 types d’instructions : “elements” (= données) et “operations” (= transformation/évaluation des données)
  - Pas de boucles (!= “Turing Completness”), car les scripts sont supposés être effectués par tous les noeuds et il faut donc être certain que tout script puisse être évalué en un nombre limité d’opérations. Par ailleurs l’existence de boucles rendrait les scripts encore plus difficile à analyser qu’ils ne le sont aujourd’hui
- 3 contraintes majeures:
  - Confidentialité
  - Compacité
  - Efficacité computationnelle

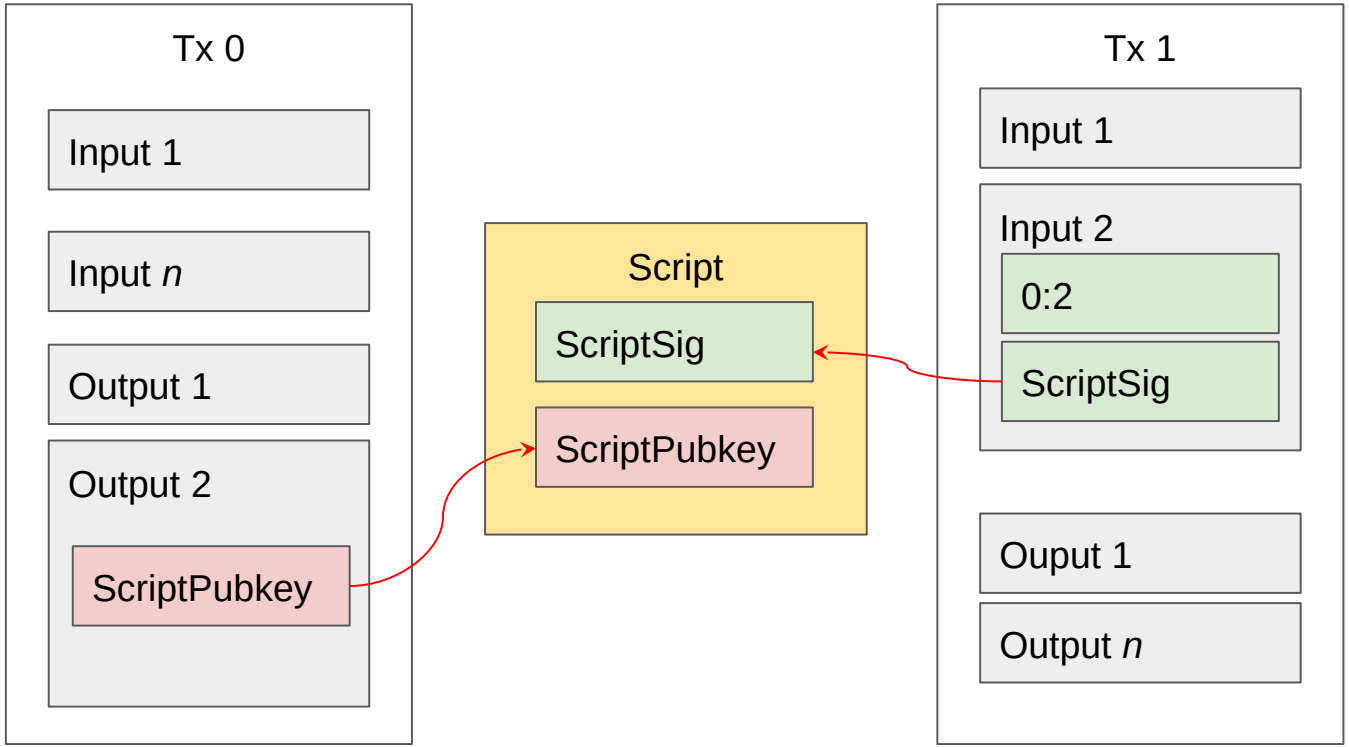
Quelques remarques sur les scripts

- Il est important de penser Script comme un langage de vérification, et pas d’exécution
- Le but est que chacun puisse vérifier qu’une transaction est valide (i.e. le payeur remplit les conditions pour dépenser l’output).
- Idéalement, on ne s’intéresse qu’à la preuve et à sa validité, pas au contenu.
- Les 3 contraintes (confidentialité, compacité, efficacité) sont corrélées : en général, améliorer les scripts de Bitcoin sur un aspect a des bénéfices pour les 2 autres.
- Les scripts Bitcoin sont difficile à analyser, en théorie on peut produire n’importe quelle condition arbitraire, en pratique tout le monde utilise une poignée de templates éprouvés

Rappel: Signatures et Scripts

- ScriptPubkey = partie de chaque output d’une transaction, contient un script déterminant les conditions auxquelles cet outputs peut être dépensé (à l’origine, une clé publique)
- ScriptSig = partie de l’input, contient les données nécessaires pour déverrouiller l’output (principalement, une signature)
- Au moment de la validation de la transaction, un Script est généré avec le ScriptPubkey et le ScriptSig
- Les données sont poussées dans le “Stack” (là où est évalué le Script), puis traitées au moyen d’instructions contenues dans le Script
- Quand le Script est vide, si le Stack != 0, le Script est valide

Transactions et outpoint

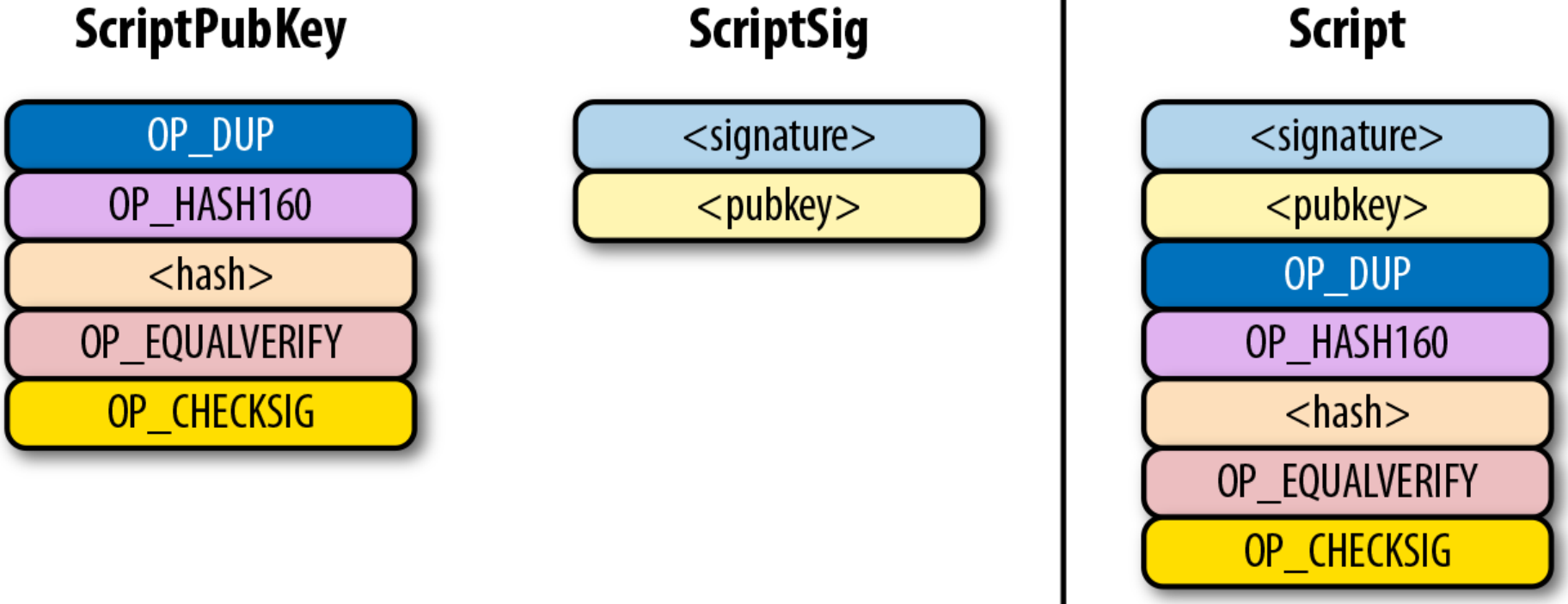


Rappel: les principaux modèles de scripts

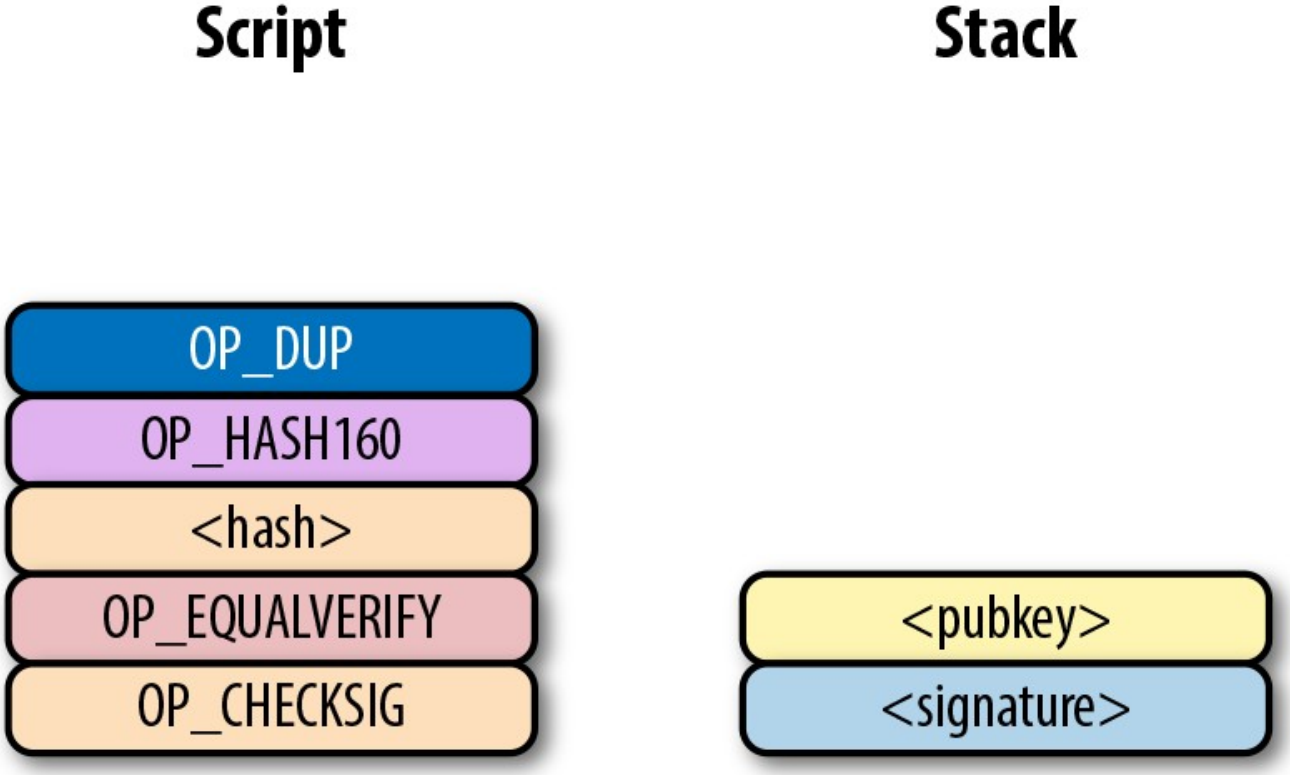
- Pay to pubkey (P2PK)
  - ScriptPubkey: <pubkey> CHECKSIG
  - ScriptSig: <sig>
- Pay to pubkey hash (P2PKH)
  - ScriptPubkey: DUP HASH160 <pubkeyhash> EQUALVERIFY CHECKSIG
  - ScriptSig: <sig> <pubkey>
- Pay to script hash (P2SH) Multisig
  - ScriptPubkey: HASH160 <hash> EQUAL
  - ScriptSig: [<3> <pubkey1> <pubkey2> <pubkey3> <2> CHECKMULTISIG] <sig1> <sig2>
  - P2SH externalise le stockage de la condition à laquelle l'output peut être dépensé hors de la blockchain

Validation: ScriptPubkey et ScriptSig

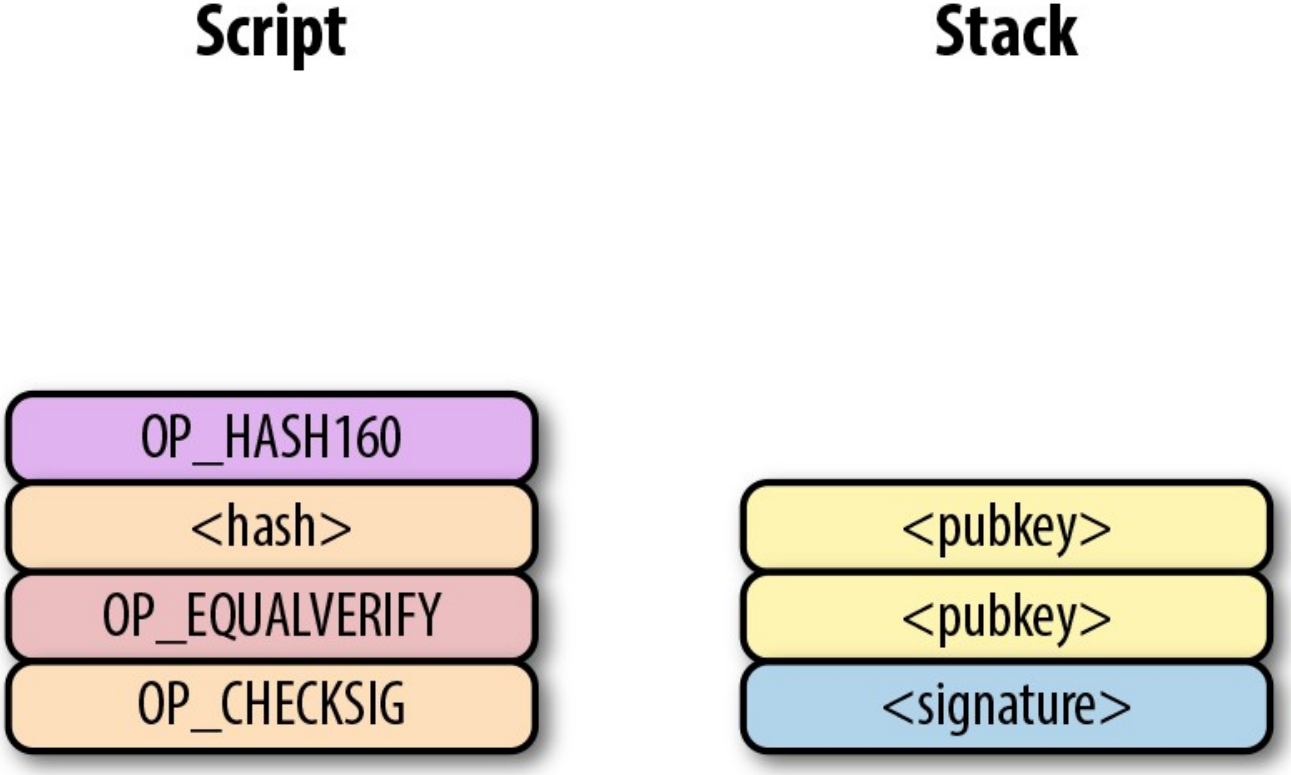
source: https://github.com/jimmysong/programmingbitcoin/blob/master/ch06.asciidoc



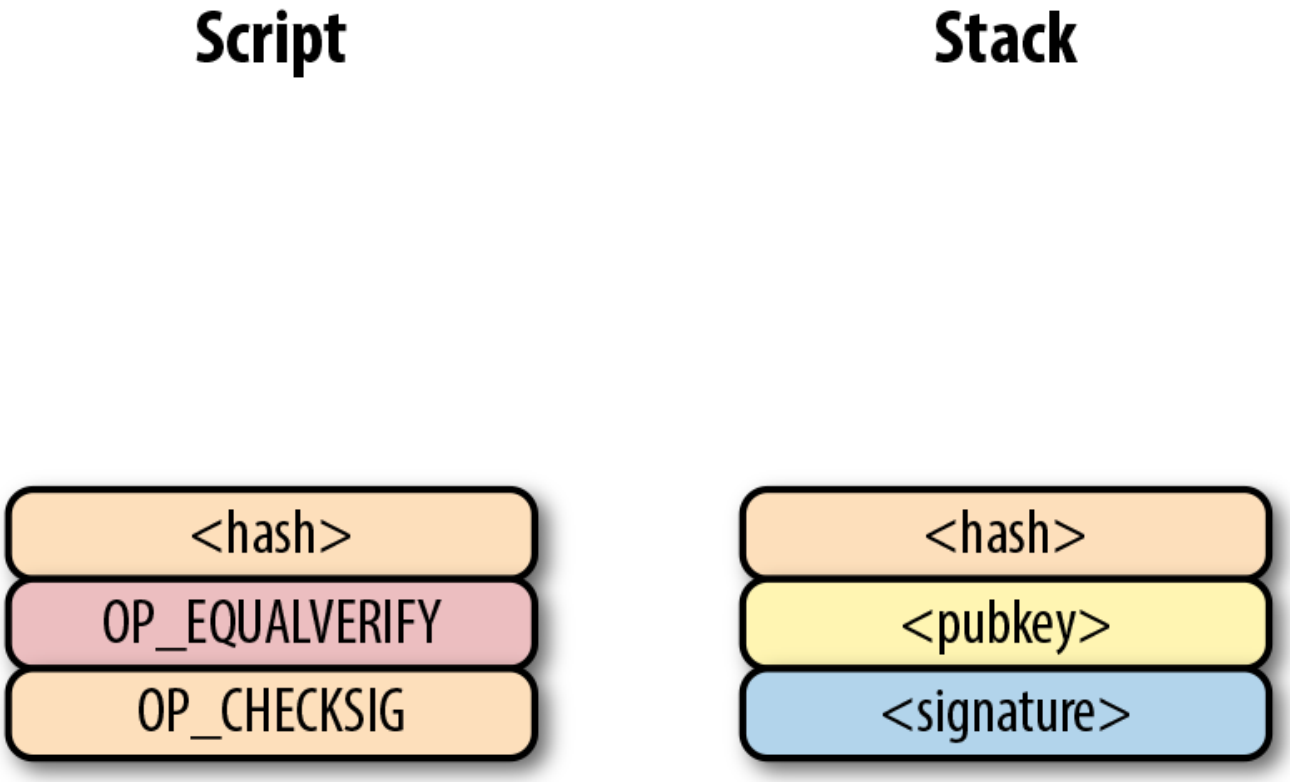
Les éléments sont poussés dans le Stack



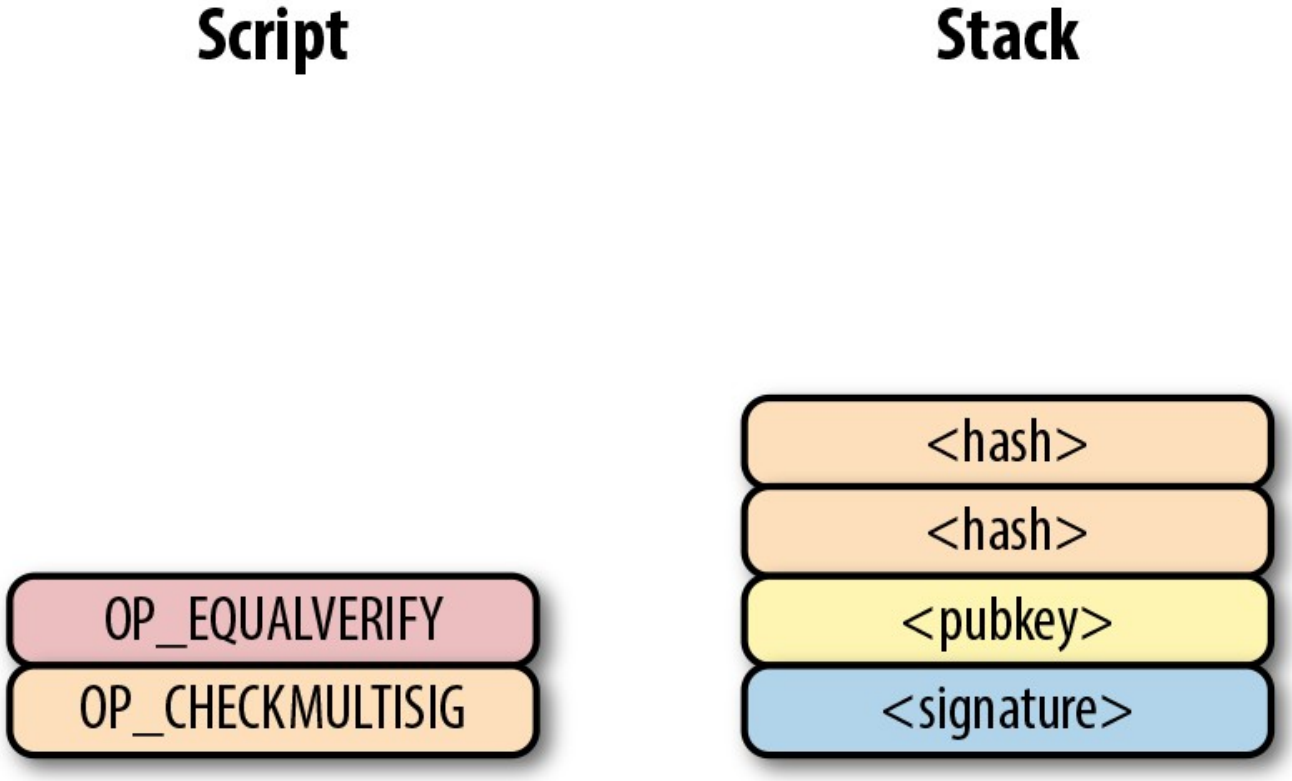
OP_DUP duplique le 1er élément



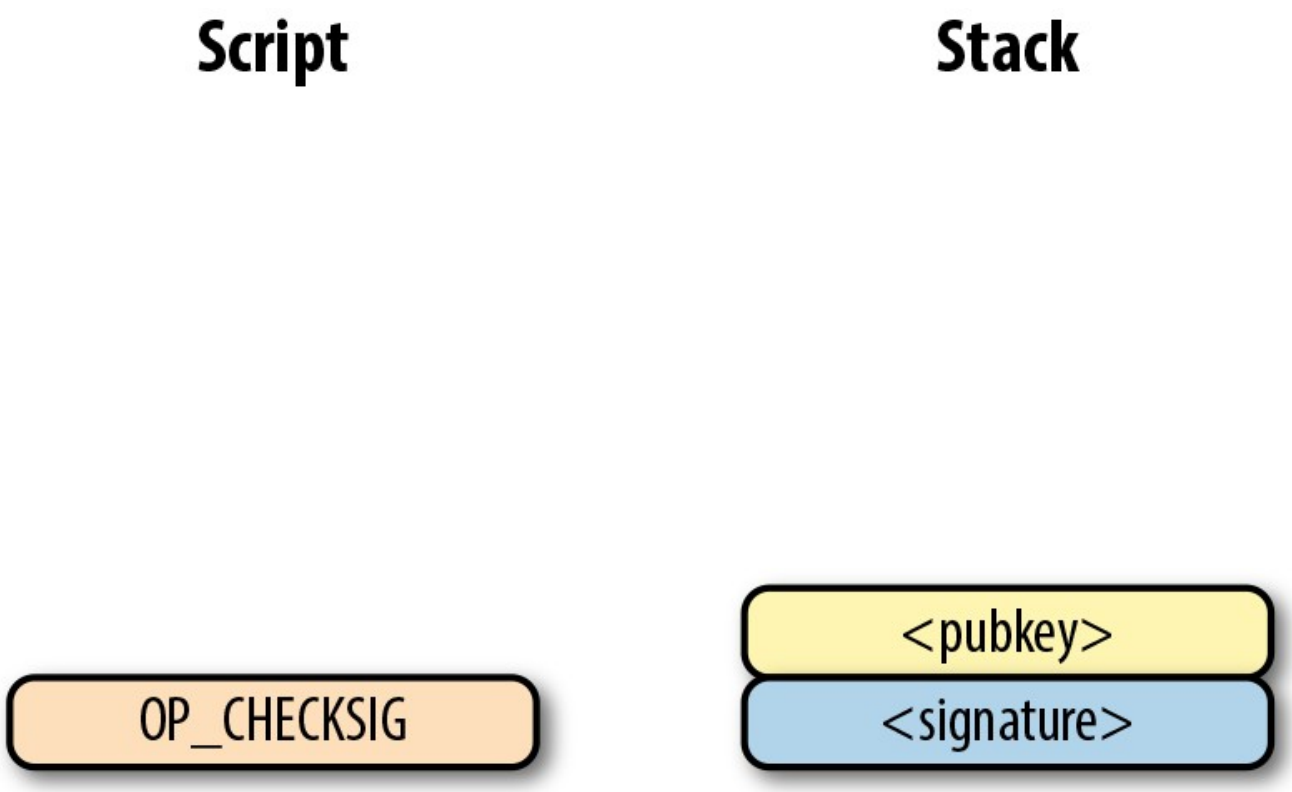
OP_HASH160 hache la pubkey



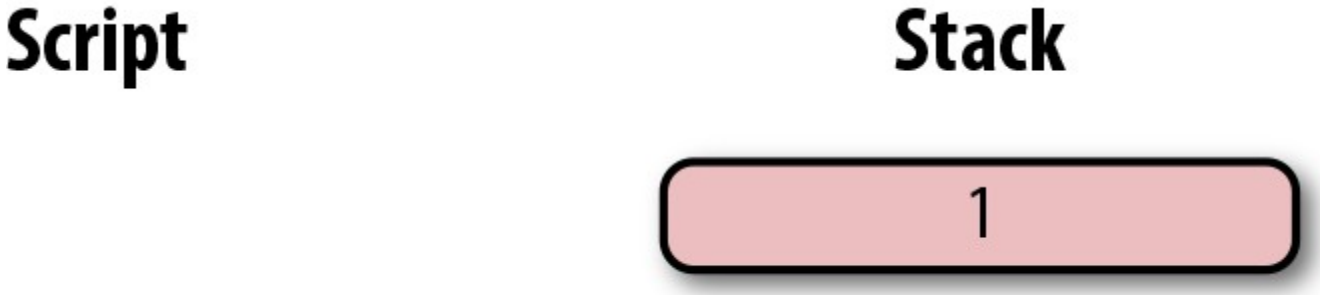
Le prochain élément est poussé dans le stack



OP_EQUALVERIFY vérifie que les 2 éléments sont égaux

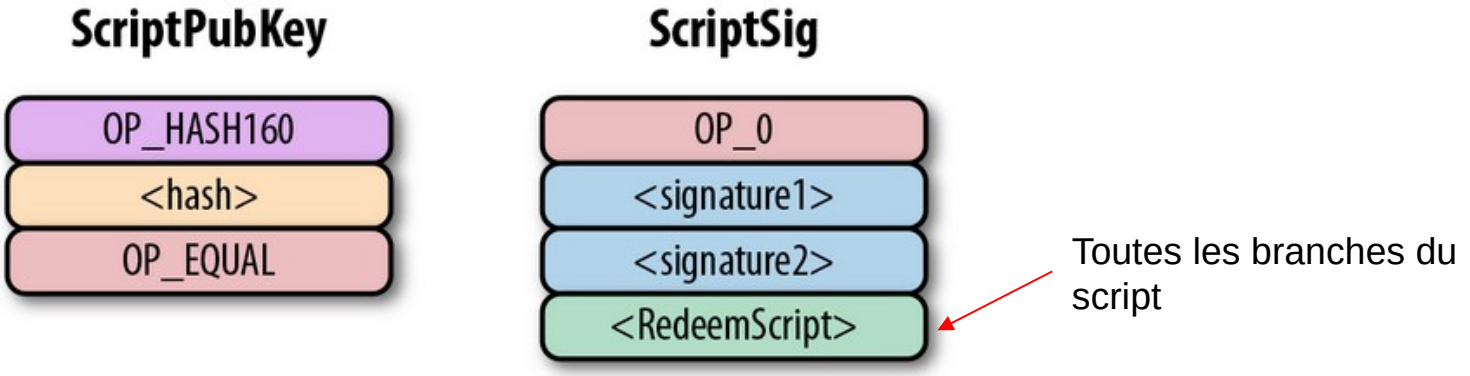


Si la signature est correcte, il ne reste que “1” dans le stack



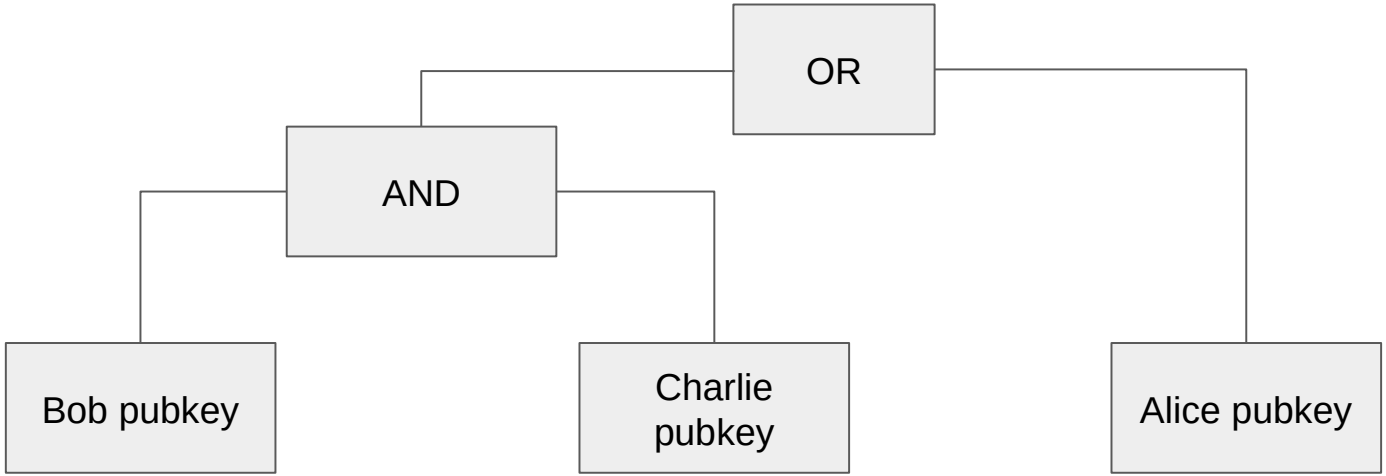
Problèmes des Scripts

- Fongibilité : les Scripts ont des patterns très facilement identifiables
- Confidentialité : des informations sensibles deviennent publiques au moment où un output est dépensé
- Coût : l'espace de la blockchain est cher, et les scripts "complexes" ne sont pas du tout économes

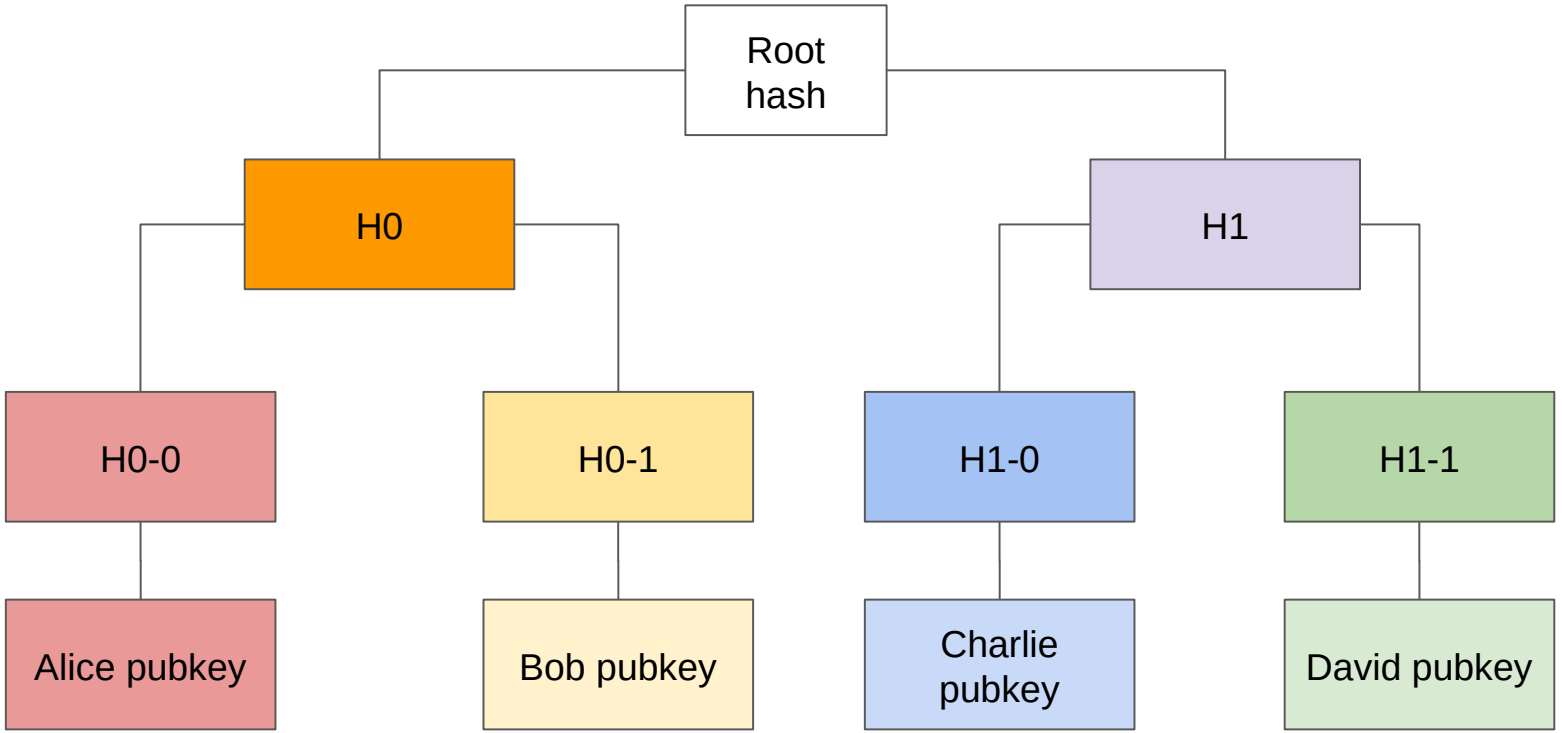


MAST (“Merklized Abstract Syntax Tree”)

- Un arbre de la syntaxe abstraite est une représentation d’un programme informatique sous forme d’arbre

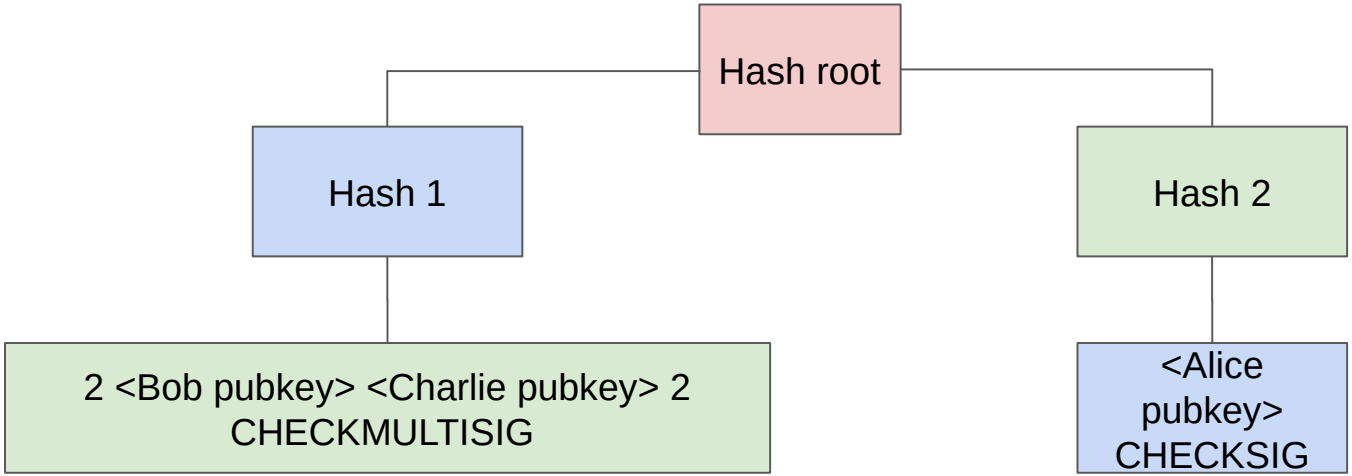


MAST (“Merklized Abstract Syntax Tree”)



MAST (“Merklized Abstract Syntax Tree”)

- Dans l’output = hash root
- Dans l’input = script + preuve de sa présence dans l’arbre

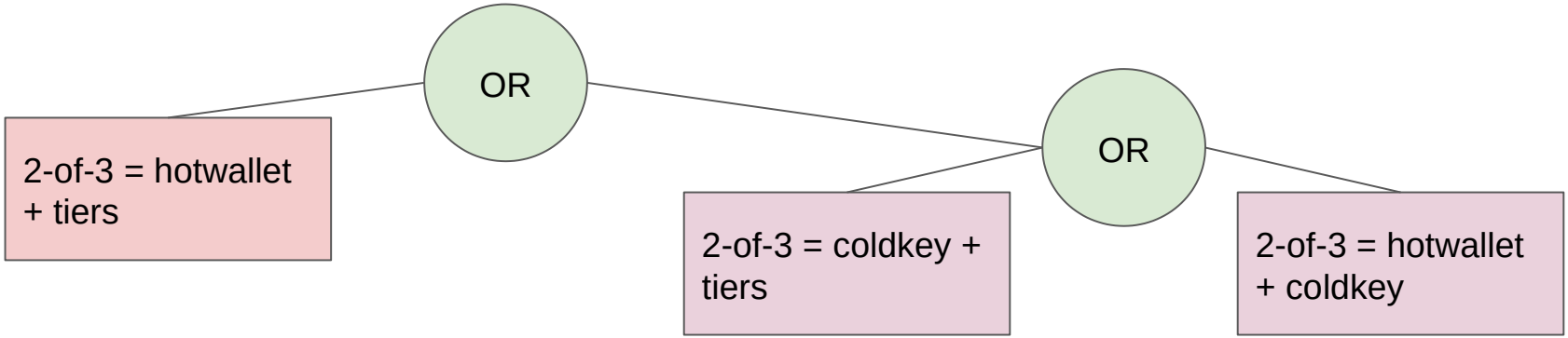


Pourquoi MAST ?

P2SH	MAST
Toutes les branches du script sont inscrites dans la blockchain	Seule la branche qui est utilisée est inscrite
Les branches non réalisées sont visibles publiquement	Seule la branche réalisée peut être vue
La complexité des scripts est limitée par la taille de la transaction	Il n’y pratiquement plus de limites à la complexité d’un script

Dissymétrie dans les scripts

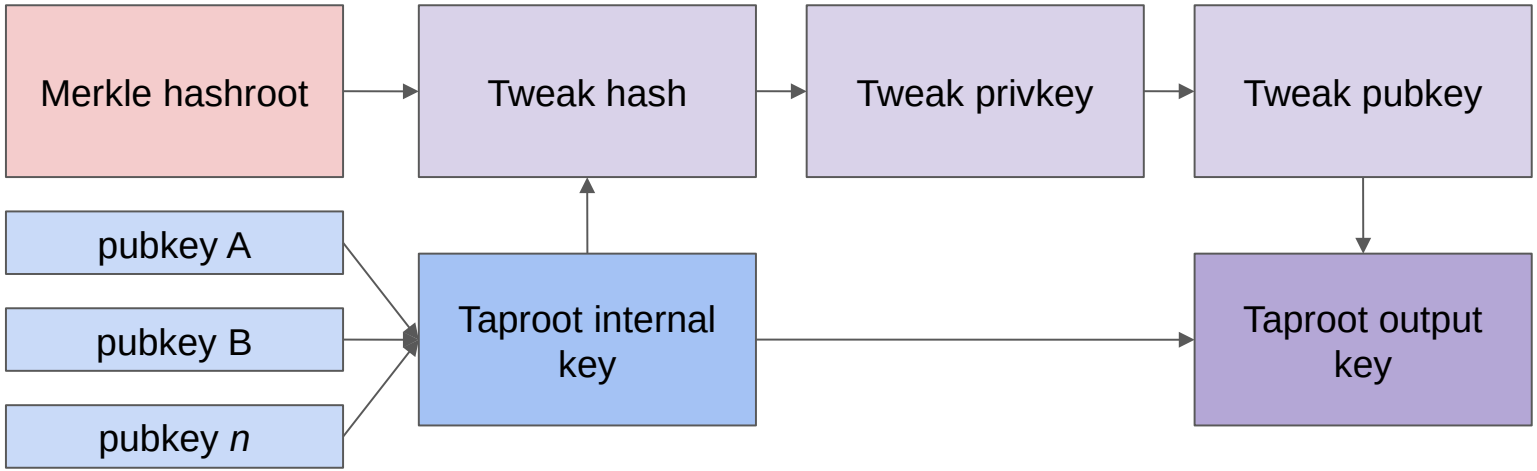
- Hypothèse :
  - un cas “simple” où tous / un subset des participants sont d’accord pour réaliser une transaction
  - un ou plusieurs “fallback” qui ne servent qu’en cas de conflit ou d’incapacitation d’un ou plusieurs signataire
- Exemple :



Composition

- Pay to Contract (P2C)
  - Construction utilisée pour la sidechain Liquid
  - Un “contrat” (= script, ici *S*) est mis en gage dans une clé publique qui verrouille l’output :
    - P’ = P + H(P || S)G
- Schnorr
  - En vertu de la linéarité des signatures de Schnorr, cette construction peut être réalisée en utilisant une clé publique agrégée entre les participants
    - Alice et Bob agrègent leur clés respectives A et B, le résultat est C
    - P = C + H(C || S)G
  - La nouvelle clé publique *P* peut-être agrégée à celle des participants *C*

Taproot



- Key spend : tous les signataires signent l'output key
- Script spend : un signataire peut fournir l'internal key, un script (et la preuve de Merkel), et valider qu'on obtient bien à nouveau l'output key. Cela déclenche la lecture du script correspondant, que le signataire doit évidemment satisfaire

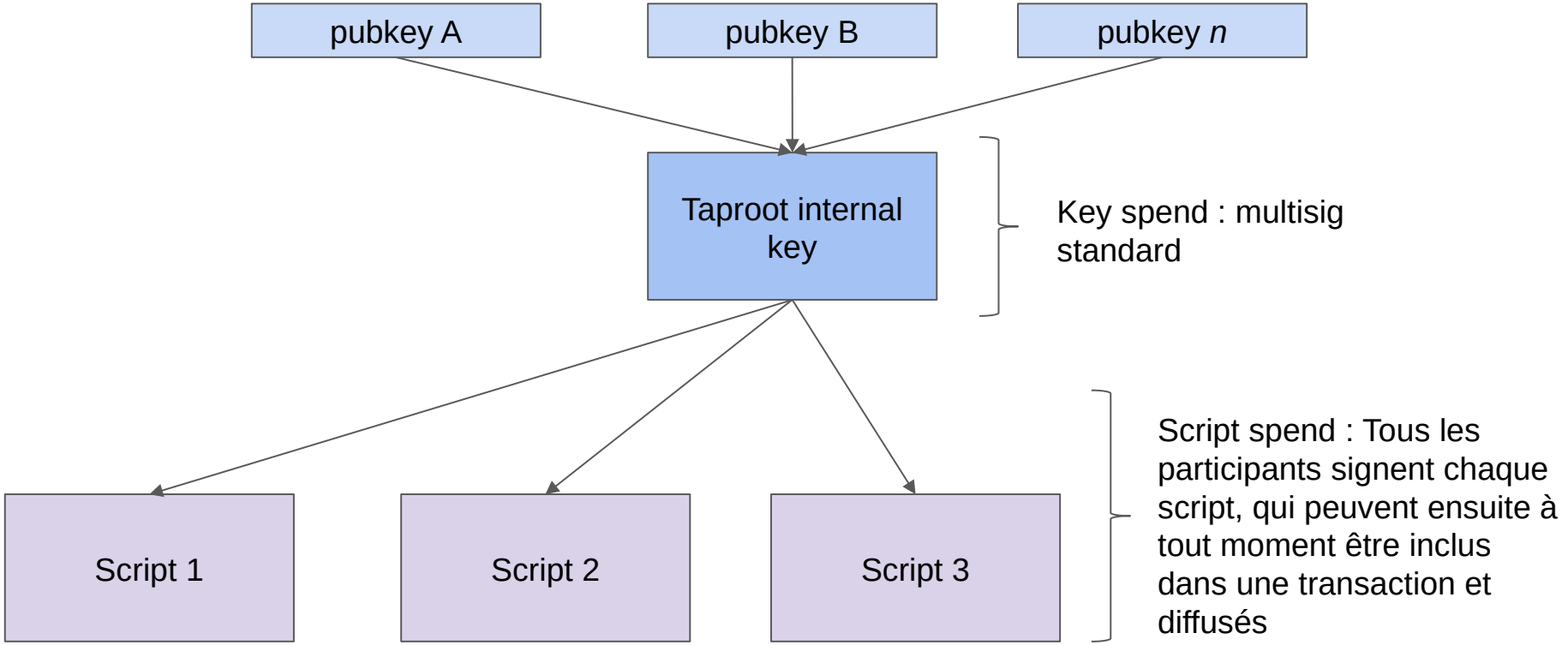
Avantages

- Dans le cas d’un key spend, le résultat sur la blockchain est indistinguable d’une “simple” signature de Schnorr
- S’il est fait appel à un script, le nombre et le contenu des autres scripts restent invisibles
- Cela permet de réintroduire une forme d’auditabilité : on peut créer autant de branches qu’il y a de configurations de signataires possibles, et ainsi selon la branche utilisée on sait avec certitude qui a signé

Objections ?

- M. Friedenbach pense que Taproot n’est pas assez souple, et qu’il vaudrait mieux faire évoluer les scripts en ajoutant des fonctionnalités génériques (autrement dit il est en faveur de rendre Script plus expressif, quitte à perdre en analysibilité et robustesse)
- Quid de l’implémentation dans les services existants ?
- Modalités de déploiement encore floues, les outputs taproot seront “anyone can spend” pour les clients non compatibles

Graftroot


pubkey A
pubkey B
pubkey *n*
Taproot internal key
Key spend : multisig standard
Script 1
Script 2
Script 3
Script spend : Tous les participants signent chaque script, qui peuvent ensuite à tout moment être inclus dans une transaction et diffusés

Avantages et inconvénients

- Avantages :
  - Plus simple, toute la complexité est reportée à l’extérieur de la blockchain
  - Mêmes avantages que Taproot
- Inconvénients:
  - Délégation de signature : si le dépositaire d’une signature la perd et que le délégataire n’est pas en mesure de la refaire, le script est perdu

Conclusion : le déploiement ?

- Encore en phase de feedback, les propositions peuvent encore évoluer
- Il est préférable de déployer le plus de ces nouvelles fonctionnalités possible en une seule fois, sans quoi les gains sont peu importants
- Déploiement par soft fork :
  - l'expérience de Segwit nous apprend que déployer de tels upgrades est long et difficile, même si le changement est ici *a priori* moins polémique que Segwit
  - certaines fonctionnalités (notamment les cross-input aggregations) sont en suspens pour l'instant car trop compliqué à implémenter dans le contexte d'un soft fork
  - La vérification par batch des signatures dans un bloc ne marche que si toutes les transactions sont Schnorr
- Même après le déploiement sur le réseau, quel impact sur les services existants ?

Références

- Résumé du BIP-taproot dans la newsletter Bitcoin Optech
- Présentation de Schnorr et Taproot par P. Wuille, juillet 2018
- Musig par A. Poelstra, mars 2019
- Présentation par S. Lee (Bitcoin Optech), mai 2019
- Présentation des signatures de Schnorr par P. Wuille, janvier 2018
- Bip-Schnorr par P. Wuille
- Bip-Taproot par P. Wuille
- Proposition de Taproot par G. Maxwell, janvier 2018
- Proposition de Graftroot par G. Maxwell, février 2018
- Présentation de MAST par D. Harding, septembre 2017
- Musig par G. Maxwell, A. Poelstra, Y. Seurin et P. Wuille, mai 2018
- Le site de Y. Seurin
- Présentation des adaptor signatures par Waxwing