Bitcoin Script 2.0 and Strengthened Payment Channels

Johnson Lau, Bitcoin protocol developer
Olaoluwa Osuntokun, Co-founder Lightning Labs

Presented at Scaling Bitcoin 2017
Stanford, CA
November 4th 2017

A brief history of Bitcoin script evolution

Emergency bug fix (2009-2010)

- Skip signature check with OP_RETURN and malformed scriptSig
- Accidental consensus fork: OP_VER and OP_VERIF
- Potential DoS: CAT, SUBSTR, LEFT, RIGHT, INVERT, AND, OR, XOR, 2MUL, 2DIV, MUL, DIV, MOD, LSHIFT, RSHIFT

Fixed-size address for arbitrarily complex scripts (2012)

- Pay-to-script ➔ Pay-to-hash-of-script
- BIP16 Pay-to-script-hash

A brief history of Bitcoin script evolution

Strict DER signature format (BIP66, 2015)

- Consensus bug due to inconsistencies in signature handling in OpenSSL

Lock-time and Relative Lock-time (2015-2016)

- OP_CHECKLOCKTIMEVERIFY (BIP65)
- OP_CHECKSEQUENCEVERIFY (BIP112)
- Priority resolution in smart contracts

Malleability fix (2016-2017)

- BIP141: Segregated witness

Shortcomings - Lack of upgrade mechanism

- Original solutions including OP_VER, OP_VERIF and OP_RETURN led to critical consensus failure and were disabled
- OP_NOP1 to OP_NOP10 allowed new “pass-or-fail” type operations, but not any stack-manipulating operations (push, move, remove)
- Not possible to redefine existing operations
- “Witness version” in Segregated Witness (BIP141) allows introduction of new script system without modifying existing script functions

Shortcomings - Lack of string and bitwise operations

- Most string and bitwise operations were disabled in a rush in 2010:
  - OP_CAT, OP_SUBSTR, OP_LEFT, OP_RIGHT, OP_INVERT, OP_AND, OP_OR, OP_XOR
- Unable to combine strings or examine part of a string
- Potential use:
  - Tree signatures with OP_CAT: O(logN) script size for very complicated multi-sig
  - Deterministic random number generation with OP_XOR: combining secret values from different parties
  - Weak hash with OP_LEFT: to save witness space when 160-bit is not necessary
- Safely re-enabled in the Elements Project

Shortcomings - Limited numeric operations

- Disabled in 2010: OP_MUL, OP_2MUL, OP_DIV, OP_2DIV, OP_MOD, OP_RSHIFT, OP_LSHIFT
- Range of value is limited and confused
  - CScriptNum are processed as int64 internally
  - Input: Up to 32-bit signed
  - Output: Potentially up to 64-bit signed
- Input size cannot cover the maximum amount of bitcoin supply
  - $21{,}000{,}000 * 10^8 = 2^{50.899}$
  - Needs at least 51-bit unsigned or 52-bit signed
- Proposal
  - Expand the valid input range to 56-bit signed (7-byte)
  - Limit the maximum output size to 7-byte
  - Safely re-enable operations within the limited input and output range

Shortcomings - Cannot commit to additional scripts

- Functional (non-push) script operations in scriptSig has no practical use
  - Malleable by third parties, as not covered by the signature operations in scriptPubKey
  - For example, any <sig> <pubkey> OP_CHECKSIG pattern in scriptSig could be simply replaced by a OP_1 or OP_0
- Potential use:
  - Delegation: inclusion of additional scripts without spending and re-creating UTXO. For example “my son may spend this UTXO later, if it is not spent by me within 1 year”
  - Replay protection: with OP_PUSHBLOCKHASH (push the hash of a block of specified height to stack), it makes sure a transaction is valid only in a specified blockchain fork
- Proposal: OP_CHECKSIG needs the ability to sign additional scripts which will be executed

Shortcomings - Limited access to tx components

- OP_CHECK(MULTI)SIG(VERIFY) are the only operations that could examine different components in a transaction, in 6 very restricted SIGHASH combinations:
  - (SIGHASH_ALL or SIGHASH_SINGLE or SIGHASH_NONE) ± SIGHASH_ANYONECANPAY
- Advantage of SIGHASH design
  - Very compact: 1-byte to indicate which components to sign
- Disadvantage of SIGHASH design
  - Very inflexible: meaning of SIGHASH flags are set in stone once deployed
  - Complicated and error-prone design, e.g. $O(N^2)$ bug and SIGHASH_SINGLE bug
- Proposal: SIGHASHV2 with 0 to 2 bytes, covering transaction nVersion, nLockTime, inputs (value, hash, nSequence), outputs (script, value), fees, additional scripts. All components are individually optional.

Shortcomings - Limited access to tx components

- Another proposal: OP_PUSHTXDATA - push the value of different components of a transaction to the stack
- Advantage over SIGHASH
  - Easier to implement and review
  - More than “equal to”, e.g. “value of output X must be at least Y BTC”, “version must not be Z” (with 7-byte numeric comparison)
  - Combination of different components, e.g. “fees must be at least X satoshi per weight unit” (with OP_MUL or OP_DIV)
  - Very flexible, e.g. “sign only inputs 1, 3, 5 and outputs 2, 4, 6 and ignore the rest”
  - Covenant: predefining the output script, e.g. “to spend this UTXO, script of the output X must be in some restricted form and the value must be at least Y.” (with OP_CAT or OP_SUBSTR)
- Disadvantage over SIGHASH
  - Use more witness space
  - Money may be lost with poorly designed covenant (true for any poorly designed smart contracts)
    - Anyone-can-spend
    - No-one-can-spend
    - Locking money in an endless loop

Other useful new functions

Merkalized Abstract Syntax Tree (MAST)

- Expose only executed branch, and keep the rest hidden as hash
- O(logN) space efficiency instead of O(N)
- Allow very big scripts with many branches that are not possible today
- Better privacy as unused scripts are hidden

Public Key Aggregation

- n-of-n multi-sig becomes single-sig
- Increased privacy, less space

Other useful new functions

OP_CHECKSIGFROMSTACK (OP_CSFS)

- 3 arguments: public key, 32-byte message, signature
- Implemented in the Elements Project
- Potential use:
  - New commitment invalidation scheme
  - Signature for another Bitcoin UTXO
  - Signature for non-Bitcoin message, e.g. cross-chain swap

OP_ECADD, OP_ECMUL

- Performing elliptic curve point addition and multiplication
- Potential use:
  - More private replacement for HTLCs

Related Work-in-progress

Johnson Lau: Merkalized script (BIP114 and more https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-September/014963.html)

Mark Friedenbach: Merkle branch verification & tail-call execution semantics (https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-September/014932.html)

Luke Dashjr: version-1 witness program (https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-October/015141.html)

Russell O'Connor: Simplicity (https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2017-October/015217.html)

Questions related to script design philosophy

- Static analyzability of script
- Turing completeness and recursion
- Limiting validation resources (sigop)
- Best way for further upgradability

Case Study: Re-Designing Payment Channels

- W/ new **Script extensions**, can improve channels over **multiple dimensions:**
  - **Reduce** amount **of client side storage**:
    - Historical chan state: `C + O(log k)` ➡ SOON `O(1)`
      - C = set of keys for script template
      - K = height of revocation tree
    - HTLC storage for latest chan state: `O(N)` ➡ SOON `O(1)`
      - N = num active HTLCs (need sig for each)
  - **Reduce** amount of **WatchTower Storage:**
    - O(M) + O(N) + O(log k) ➡ SOON O(1)
      - M = num HTLC's **ever,** N = num states
  - **Allow for** trap door anyone-can-revoke outputs:
    - **Special clause** in WatchTower contract to ensure **inevitable enforcement**
  - Channel open + cooperative close **indistinguishable from regular payments**
    - (can actually be done **today**)
  - **Indistinguishable** payment **identifiers** for multi-hop payments

Review of Commitment Invalidation

- Critical **safety** mechanism of BDP (BiDi Payment Channels):
  - We ensure both parties are incentivized to **only** broadcast the **latest** state
  - Otherwise, their **entire balance** within channel is **slashed!**
- History of prior **commitment invalidation** mechanisms:
  - **Decrementing** sequence locks (utilizes **BIP 68**)
    - **How**: use relative time-locks s.t latest state can go in **before** prior states
    - **Drawback:** limits number of possible **updates**
  - Commitment **invalidation tree** (used in **Duplex Payment Channels** (cdecker))
    - **How:** structure commitments in **tree** s.t **parent must be broadcast before leaf**
      - Roots have **decrementing** time lock w/“kick-off” allows for indefinite lifetime
    - **Drawback:** at **cost** of **increased on-chain foot print**
  - Commitment **Revocations** (hash or key based, current channel design)
    - **How:** must reveal secret of **prior state** when accepting new state
    - **Drawback:** MUST **critically store** O(log N) of remote party, more complex key derivation

What if I told you....we **don't need revocations!**

- Enter **OP_CHECKSIGFROMSTACK**
  - Review: allows checking signatures on **arbitrary messages**
  - Use: contracts can **enforce structure** on signed messages
- Invalidation via **signed sequence commitments**
  - Invalidation clause is now:
    - Present: **(sig, n, r),** s.t **verify(sig, key, c) && open(c) == (n', r) && n' > n**
    - **"I know of an opening to a signed commitment (by broadcaster) of a newer seqno"**
  - **R** is random value to ensure commitments are **hiding**
    - **Avoids revealing # of updates** in case of unilateral broadcast
    - **Re-use** sequence+locktime **obfuscation mask (BOLT #3)**
- Maintains **same** channel commitment **state machine (BOLT #2)**
  - **Simplifies key derivation** in current channels
- Reduces storage for both parties to **O(1)** (sig + commitment opening)
  - Has implications for the **WatchTower**

Review of WatchTower State Outsourcing

- LN assumes **decentralized** mining, **on-chain liveness**
  - On-chain censorship major issue
  - CSV value **T** acts as **time-based security parameter**
    - **Configurable on a channel to channel basis**
- If **unable** to be **eternally vigilant**, can outsource to **WatchTower**
  - Under current design:
    - For commitments:
      - Send initial base points (needed to construct **witness script template**)
      - For **each** state send a **new signature** for **justice transaction**
    - For HTLCs
      - Encrypt opaque blob with **txid[:16]**
  - Various **compensation/authentication** mechanisms possible
    - **ZKP's** for authentication
    - Pay-per-state, only provide bonus upon action, subscription, etc

Delegated Trapdoor Channel Outsourcing

- Using **commitment seqno** based revocation:
  - Due to **seqno** invalidation requirements only **latest commitment required!**
    - Each new sign commitment seqno **replaces** a lower seqno
    - Able to **skip** sending states as **no strict ordering requirement**
- **Delegated** Outsourcing:
  - With above still need to send **sig** for **each state**
    - Invalidation achieved, but **need to bind to a pukey** to ensure security
  - Solution:
    - Using covenants and **OP_CHECKSIGFROMSTACK** we'll "**bless**" a pubkey
    - **Blessed pubkeys** can present final signature to satisfy invalidation
    - Use covenants to **restrict** structure of spending transaction
      - Use to require they take a % as fee, pay to my key, etc, etc
    - Can use MAST to bless a **set** of pubkeys
- Free for all trapdoor: given public seqno commitment, **let anyone spend after delay**

Eliminate Historical Second-Level HTLC Storage

- In current commitment design (BOLT#3) **CSV+CLTV decoupled** in HTLC's:
  - **Prior issue** where if **CSV is large**, **CLTV in total hop must be >>**
  - Solved by making **HTLC claiming** a **2-stage state machine**
    - **Off-chain** multi-sig **covenants**
    - **Attest** (broadcast) -> **Delay** (csv) -> **Claim** (sweep)
  - Cons:
    - Requires **distinct** transaction for **each HTLC**
    - Must store **signature** for **each HTLC**
    - New state updates require **signing+verifying** N sigs (for each HTLC)
      - (**post-schnorr** can be **batched** tho)
- Solution:
  - Use **actual covenants** in HTLC outputs!
  - **Eliminates** sig+verify w/ commitment creation
  - **Eliminates** sig **storage** of current state
  - Add **independant script** for HTLC revocation clause (reuse commitment invalidation technique!)

Modifications for Increased Privacy

- Channels currently **identifiable on-chain**:
  - **2-of-2** multi-sig outputs **stick out** amongst other traffic
  - Candidate for miners to **censor**, outlawing contracts (**censorshipResistance--**)
- Multi-Sig -> Single-Sig (via multi-signatures):
  - Disguise channel openings are **regular transactions**
  - Use 2-party signing to generate signature for joint public key:
    - ECDSA: https://eprint.iacr.org/2017/552.pdf
      - Uses paillier, zero knowledge proofs of correctness
    - Schnorr: https://cseweb.ucsd.edu/~mihir/papers/multisignatures-ccs.pdf
      - Multi-signatures w/ built-in de-linearization
- **Replace** HTLC's using **EC operations** (like Sphinx's one little-trick):
  - Sphinx payload = (Q, P, r) s.t (Q = P + r*G)
    - Send P on outgoing HTLC
    - On settle, learn p, calc: q = p + r)
    - Use q to settle incoming HTLC