SECURE THE BAG

Jeremy Rubin
Why are we here?
What is Scaling?

Increasing Transaction Throughput
Scaling = Tradeoffs

- Decentralization
- Redundancy
- Privacy
- Censorship Resistance
- Layerization Complexity
- Latency
- Cost
- Peak/Trough Provisioning
- Reliability
- Interactivity
- Bandwidth Requirements
- Storage Requirements
- Fairness
- "Scanability"
- Homogeneity of use
- Collateralization
- Smart Contract Complexity
- Quantum Resistance
- Reorg Safety
- Orphan Rates
- Etc...
Acceptable tradeoffs?
Block Size Increases

**PRO**
- Conceptually Simple

**CON**
- Reliability/DoS
- Centralization
- Hard-Fork
- Storage Requirements
- Bandwidth
- Orphan Rate Increase
Lightning Network

**PRO**
- Low Latency
- Privacy
- Low On-Chain Usage

**CON**
- Contract Complexity
- No Settlement Finality
- Collateralization
- Interactivity
- Intermediation/Middle Men
- No Reorg Safety
<table>
<thead>
<tr>
<th>PRO</th>
<th>CON</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conceptually simple</td>
<td>Real time Bandwidth reduction</td>
</tr>
<tr>
<td>Low Latency</td>
<td>N.A.</td>
</tr>
<tr>
<td>Privacy</td>
<td>Soft Fork</td>
</tr>
<tr>
<td>Immediate Settlement Finality</td>
<td>Reorg Safety</td>
</tr>
<tr>
<td>No over-collateralization</td>
<td>etc...</td>
</tr>
<tr>
<td>Low contract complexity</td>
<td></td>
</tr>
<tr>
<td>Non Interactive</td>
<td></td>
</tr>
</tbody>
</table>
Intuition Building
Intuition Building:
Committed UTXOs / "Certified Cheques"

Normal

Committed
Intuition Building: Batched Payments

Payment

Batch Payment

Diagram showing the relationship between individual payments and batched payments.
Intuition Building: Two Phase Payments

Batch Payment

Pay Phase 1:
\textit{Spend}(3); \textit{Create}(12);

2-Phase Payment

Spend Phase 1:
\textit{Spend}(3); \textit{Create}(1);

Receive Phase 2:
\textit{Spend}(1); \textit{Create}(12);
Intuition Building: Multi Phase Payments

2-Phase Payment

Spend Phase 1: Spend(3); Create(1);
Receive Phase 2: Spend(1); Create(12);

Chained Payment

Spend Phase 1: Spend(3); Create(1);
Receive Phase 2: Spend(1); Create(6);
Receive Phase 3: Spend(1); Create(7);
Intuition Building: Tree Payments

Chained Payment

Tree Payment
Intuition Building: Receiving Tree Payments

Spending Tree Payment

Receiving Tree Payment
What's the magic?
Four Options

- 👎 "Covenants" (OP_COV) 👎
- 👎 Pubkey Recovery (CHECK.SIGFROM.STACK, ANY.PREVOUT/NO.INPUT) 👎
- 👎 Presigned Transactions 👎

- 🔄 NEW 👍 OP_SECURE.THE.BAG 👍 NEW
Alternatives? 👎👎👎

- **OP_COV**
  - Too Powerful → Too Much Technical Risk
  - Covenant "viruses"
  - Complex implementation rules
  - Specific outputs

- **Presigned TxMultisig**
  - Interactivity OR Trusted Third Party
  - Fancy ECDSA OR Schnorr protocols (fairness impossibility problems)
  - Can't prove receiving guarantee to third party
  - Key Deletion "Toxic waste"

- **Pubkey Recovery (CHECKSIGFROMSTACK, ANYPREVOUT/NOINPUT)**
  - Possible recursion with OP_ECTWEAK
  - Abstraction violation "Keys should be Keys, Signatures, Signatures"
  - Incompatible with message digest including pubkey; related key attacks
OP_SECURETHEBAG

- Multibyte OpCode: `OP_SECURETHEBAG 0x20 <arg>`
- \(\text{STB}(tx) = H(\text{tag} \mid \mid \text{ver} \mid \mid \text{locktime} \mid \mid H(\text{outs}) \mid \mid H(\text{seqs}) \mid \mid \# \text{inps} \mid \mid \text{scriptSigs})\)
- \(\text{STB}(tx)\) commits info which mutates TXID except input COOutpoints
- OP_STB verifies \(\text{STB}(tx)\) matches what can be computed from tx
- Multibyte Op structure ensures the desired TX is known at spend time
  - Disallows all recursive covenants
  - Future safe w.r.t. Above: There is no set of pure extensions* to script E such that enabling E and OP_SECURETHEBAG as proposed enables recursive covenants, but E alone does not enable recursive covenants?
- Multiple inputs allowed
  - Generally not safe to use \#inps > 1! -- "half spend problem"
- Deployment: inside of Tapscript or standalone
Implementation Progress

- Draft BIP
- Proof of Concept Code for Opcode Available
- Experimental core wallet support in progress
- Minor BIP options in flux (pushless multibyte opcode v.s. taint tracking v.s. …)
- Deployment Strategy T.B.D.
Impact
WARNING

Simulated Results May Not Match Reality

This message brought to you by respect for the scientific process; results reproducible from https://github.com/JeremyRubin/bips/blob/op-secure-the-bag/bip-secure-the-bag/simulation.py
Transaction Compression Performance with 50% Adoption During 2 Spikes

- Unconfirmed Mempool without Congestion Control
- Unconfirmed Mempool with Congestion Control
- Transactions Per Second
- Confirmed Congestion Control Pending
- Maximum Average Transactions Per Second

transactions per second

pending and unconfirmed transactions

block days
Transaction Compression Performance with 50% Adoption During 2 Spikes

- Unconfirmed Mempool without Congestion Control
- Unconfirmed Mempool with Congestion Control
- Transactions Per Second
- Confirmed Congestion Control Pending
- Maximum Average Transactions Per Second

Transactions per Second

Pending and Unconfirmed Transactions

Block Days
More Time; Less Adoption

Transaction Compression Performance with 5% Adoption During Spike

- Mempool without Congestion Control
- Mempool with Congestion Control (1x Rate)
- Mempool with Congestion Control (2x Rate)
- Congestion Control Pending (1x Rate)
- Transactions Per Second (1x Rate)
- Transactions Per Second (2x Rate)
- Maximum Average Transactions Per Block

Confirmed or Pending Transactions

Unconfirmed Transactions

10 x
More Time; Less Adoption
Summary: OP_STB is a Txn Bypass Capacitor

- Smooths out the Backlog
- Soaks up excess txs, releases them later
- **Private benefit large even with small adoption**
- Private use benefits entire public (mempool decongestion)
- Healthier backlog of low-priority transactions
- Reorg Safety:
What's the catch?
First: Multi-Radix Congestion Controlled Transactions

IF STB A
ELSE STB B
Probably True Claim; Fancy Way of Saying No-Cons

Given:

- O(1) overhead amortized per input & O(n) overall, where w/o STB cost is O(n) also
- Multi-radix setups (OP_IF, OP_MBV, or Taproot) (Huffman Encoded)
  - Simple radix-2 and radix-N expansion IF P(radix-2 used) = O(1/n) is E[O(c)] overhead
- Ability to defer and wait for ‘asymptotically cheaper’ blockspace (fees discounted O(1/n))
- Smaller Size/verification of interior node txns compared to normal txns (no signatures)
- Prunability of interior nodes (recomputable from leafs)
- Optimal Tree Structure (leafs at different depths)
- Subtree application of the above principles

The overhead of OP_STB is E[O(c)], where the actual overhead c is a small constant.
Quickfire:
Advanced Topics in Secure The Bag
Inter Business Traffic

OP_STB withdraw from Exchange A can be immediately credited to Exchange B

Funds are effectively in "cold storage"

Businesses can manage their liquidity

Let users receive goods/trade once confirmed.
N participants; $O(N \log N)$ channels
Setup: $O(1)$
Closing 1 Channel: $O(\log(N \log N)) = O(\log(N) + \log \log N) = O(\log(N))$
Closing all of a User's Channels: $O(N \log N / N) = O(\log N)$
Closing Channels Amortized Per Channel: $O(N \log N / N \log N) = O(1)$

N participants; $O(N^2)$ channels
Setup: $O(1)$
Closing 1 Channel: $O(\log(N^2 )) = O(\log(N))$
Closing all of a User's Channels: $O(N^2 / N) = O(N)$
Closing Channels Amortized Per Channel: $O(N^2 / N^2) = O(1)$
Smart Contracts
OP_STB unroll looped programs into finite steps

Original Program Intent

while (true):
    if (sign key A):
        wait(100 blocks)
    else if (sign key B & C):
        return ALLOW_SPEND
    return ALLOW_SPEND

Pick large RUN_LIMIT
Pick acceptable default action

for int i = 0; i < RUN_LIMIT; ++i:
    if (sign key A):
        wait(100 blocks)
    else if (sign key B & C):
        return ALLOW_SPEND
    wait(sign key B & C)
Smart Vaults: Using Control Programs

**TXN A:**
- +1 Hour
- 2 Inputs
- Create UTXO 1

**TXN B:**
- +1 Hour
- 2 Inputs
- Create UTXO 2

**TXN C:**
- +1 Hour
- 2 Inputs
- Create UTXO 3

**Control UTXO 0:** STB(TXN A)
- Cold UTXO C0
- Cold UTXO C1
- Cold UTXO C2

**Control UTXO 1:** STB(TXN B)
- Hot UTXO H0
- Hot UTXO H1
- Hot UTXO H2

**Control UTXO 2:** STB(TXN C)
- sighash none

**TXN W:**
- +1 Hour
- Spend Arbitrary

**TXN F:**
- Create UTXO D0

**Deep Cold UTXO D0**
Non Interactive Channels (works w/ Ball Lightning)

- **Arbitrary UTXO**
  - **TXN A:** Create UTXO 1
  - Main Channel UTXO: 2 of 2 Cooperative or STB (TXN B)
  - **TXN B:** Create UTXO 2
  - Uncooperative Close Initiated UTXO: 2 of 2 Cooperative or STB (TXN C)
  - **TXN C:** +2 Weeks Create UTXO 3
  - UTXO 3: Close State 0

- **UTXO i+1:** Close State i+1
  - **TXN Coop[i+1]:** Create UTXO i+1 (Half Signed)
  - **TXN Uncoop[i+1]:** Create UTXO i+1 (Half Signed)
Coordination Free Decentralized Mining Pool Payouts
Summary: OP_STB "סבבה"
Deployment
Do we **need** this feature? Yes
How Urgently?

Later

Fees are low right now.

Other exciting changes on the way.

Limited engineering resources.

NOW

Why wait for the sickness?

Changes are slow, better to push when not suffering.

Exchanges spend millions per year on BTC fees; invest more eng time in reducing fee burden.

Healthy backlog of low priority important as halving approaches.
Options

**Tapscript Extension**

**Pro**
- Merkle Branch Lookups
- Easier to change opcode semantics

**Con**
- Delay
- Can't use with legacy scripts

**Standalone OP NOP Upgrade**

**Pro**
- Available broadly
- Don't need to wait for Taproot

**Con**
- Can't use Tapscript OP_SUCCESS
- Less "forced" Taproot privacy benefit
- Messier OpCode semantics
FIN

How to Get Involved:

Review the BIP.

Sponsor me: I'm a starving independent researcher.

Work on the implementation.

Work on integrating OP_STB in your products.

Chime in on the mailing lists.

Follow @JeremyRubin / Tweet your support!