Statechains: Off-chain Transfer of UTXO Ownership

Ruben Somsen

October 3, 2018

Abstract

We describe a protocol for Bitcoin and similar cryptocurrencies that allows changing ownership of Unspent Transaction Outputs (UTXOs) in an off-chain manner by passing on a transitory key – a private key that moves from one owner to the next. All UTXOs are backed by off-chain transactions that the final owner can redeem on-chain. Transfer of ownership is facilitated by an entity that keeps a public record of each transfer. We call this public record a Statechain. The entity cannot cheat unless prior owners of transitory keys are complicit. Fraudulent behavior always produces publicly verifiable evidence, thus exposing the fraudulent activity of the entity and prompting those without compromised transitory keys to safely withdraw on-chain.

1 Statechains in Context

The Statechains protocol provides an alternative method of transferring coins. It only makes use of the blockchain when entering or exiting the system, or in case of a dispute. While not as secure as on-chain transactions, it is thought to be a step up from the multisig security of Federated Sidechains[1]. Compared to the Lightning Network[2], more security assumptions are made, but coins can be transferred directly, without having to find a path on a network of sufficiently funded channels. Statechains also have a unique downside – UTXOs need to be transferred in full and cannot easily be split into smaller amounts.

Statechains utilize Schnorr Signatures[3], Eltoo[4], Adaptor Signatures[5], and (optionally) Graftroot[6]. Schnorr Signatures, more specifically using MuSig[7], enable us to achieve multisig with a single key. Eltoo allows us to create non-expiring channels with multiple participants. Adaptor Signatures are crucial to ensure signatures are swapped atomically, ensuring participants cannot misbehave without evidence of misconduct. Graftroot simplifies the way users can exit the system, and avoids potential issues when a hard fork occurs. We'd like to express our sincere gratitude to the authors of these works. Without it, Statechains would not be possible.

What follows is a short overview of these techniques, as they are a prerequisite to fully understanding Statechains. Please refer to the original work to gain a more complete understanding of each technique.

1.1 Schnorr Signatures

The goal is to prove knowledge of a secret and attach that proof to a message (i.e. a signature). Capitalized letters are curve points. Multiplying by a curve point is irreversible. Private key a has public key $a * G = A$ (where G is a publicly known curve point). To sign a message and prove ownership of a in zero knowledge, we create ephemeral key $r * G = R$. We then create signature $s = r + a * hash(R, message)$. We give R and s to the verifier, who checks that $s * G$ is equal to $R + A * hash(R, message)$. If equal, it proves that the signature came from someone who knows a.

1.2 Adaptor Signatures

The goal of Adaptor Signatures is to create signature A in such a way that it reveals a secret which completes signature B (i.e. both signatures become valid) For notation purposes, we use $\#$ to mark the key as an adaptor secret. For example: $\#\underline{a} * G = \#\underline{A}$, which is a different key than A, but is chosen by the same person. We only use $\#$ once to signify something is an adaptor

secret, e.g. $\#\underline{AB} = \#\underline{A} + \#\underline{B}$. We use s' (notice the apostrophe) to signify an s that is (provably) missing an adaptor secret. In the following example, A and B need to reveal $\#\underline{a}$ and $\#\underline{b}$ before the signatures can be completed:

Adaptor signature A:
$R = R1 + \#\underline{AB}$
$s' = r1 + a * hash(R, message)$

Adaptor signature B:
$R = R2 + \#\underline{AB}$
$s' = r2 + b * hash(R, message)$

Once A and B have received the incomplete signature s', they can reveal their secret to each other. This can either be communicated directly, or via the blockchain by publishing a complete signature ($\#\underline{ab} + s' = s$ and $s - s' = \#\underline{ab}$). Note that the common example is to use a single adaptor secret, but the variation we describe is better suited for situations involving more than two signatures and multiple participants.

1.3 Eltoo

A payment channel in Bitcoin is typically created by locking up coins on-chain between two parties, Alice and Bob. They then proceed to create off-chain transactions, moving the locked up coins between A and B. Every time they update the state, the previous state needs to be made ineffective. Before Eltoo, this was done by revealing a secret that allowed A to take B's money or vice versa if they ever tried to send an old state to the blockchain.

In the case of Statechains, this is not an option. Every time we update the state, a new party gets added into the mix. Imagine there are three parties: A, B, and C. If A cheats, there is no way to decide if B should get the money or C. Eltoo solves this. It doesn't try to punish a user who sends the wrong state, but instead it allows old states to be replaced by newer states. If an old state does get published, one simply overwrites it with the latest state in order to prevent any losses.

For our Statechains implementation, this means that if coins get transferred from B to C, both of them will have a valid on-chain transaction, but C's transaction will always take precedence, and is able to overwrite B's transaction.

2 The Statechain Protocol

A Statechain is a ledger that contains the history of every UTXO that is under its management. This ledger is maintained by the Statechain entity and serves to make them accountable for misbehavior. We will refer to the Statechain entity as key A. Note that while our example shows the entity as having only one key, a real implementation should use a different key for every UTXO. The users of the Statechains will be B, C, D, etc.

When user B wishes to deposit coins into the Statechain, A and B create an Eltoo-style channel. If this was a Lightning-style Eltoo channel, the coins would be locked up between A and B (using MuSig[7]), with a timelock back to B (ensuring B is the actual owner in case of dispute). For our protocol, we do something slightly different. B creates transitory key X – a key that will eventually be passed on to future recipients. The coins will be locked up between A and X, but the timelock will still go back to B.

To transfer the coins from B to C, A and B agree to update the state in such a way that the timelock will now go to C. Knowledge of transitory key X is then transferred to C, enabling her to transfer the coins to the next user in similar fashion.

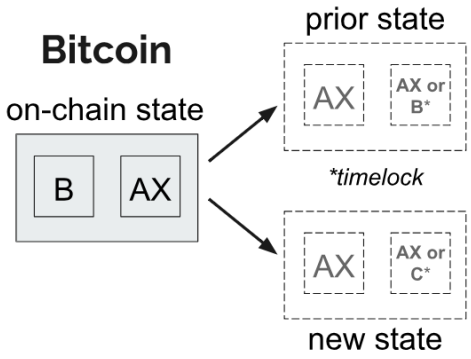

Figure 1: UTXO ownership gets transferred from B (prior state) to C (new state).

While both B and C now know X, entity A promises to only update the state according to the wishes of the last owner (C). If entity A breaks this promise and colludes with a previous owner (B), then the last owner can prove misconduct and ruin the reputation of entity A. The way we make it provable, is by having the owner of a UTXO sign off on the transfer, and publishing it on the Statechain. If A ever facilitates an action that was not signed off on, then this proves misconduct. We'll elaborate on this in later sections.

2.1 Atomic Transfer

When entity A facilitates the transfer from B to C, they need a signature from B, which proves that B specifically requested the state update. They also need a signature from C, where she acknowledges receipt of the state update transaction. All of this needs to happen atomically (either all signatures become valid, or none), this way none of the parties can claim the transaction occurred without their permission. Adaptor Signatures allow us to do just that, and it even enables us to do it with multiple UTXOs.

2.2 Transferring Ownership of a Single UTXO

Prior state: New state:
$AXor(B + timelock) \rightarrow AXor(C + timelock)$

Preparation:
1. A, B, and C create and share adaptor secrets $\#\underline{A} + \#\underline{B} + \#\underline{C} = \#\underline{ABC}$
2. B and C create adaptor signature BC that is missing $\#\underline{ABC}$, and pass it on to A
3. A and X (known by B) create adaptor signature AX that is missing $\#\underline{ABC}$, and pass it on to C

Steps:
1. C reveals $\#\underline{c}$ to B
2. B reveals $\#\underline{b} + \#\underline{c} = \#\underline{bc}$ to A
3. A publishes signature BC to the Statechain, which reveals $\#\underline{abc}$ to C
4. A helps C acquire transitory key X (*) See appendix I

If aborted after step [_]:
1. B can resume the payment at a later time
2. A can resume the payment later, B can cancel this by paying himself
3. The payment is complete, but can only be redeemed on-chain
4. The payment is complete, and can also be sent to others off-chain

2.3 Atomic Swap of Multiple UTXOs

Prior state: New state:
$AXor(B + timelock) \rightarrow AXor(C + timelock)$
$AYor(D + timelock) \rightarrow AYor(E + timelock)$
$AZor(F + timelock) \rightarrow AZor(G + timelock)$

Preparation:
1. A, B, C, D, E, F, G create and share their adaptor secrets, $\#\underline{A} + \#\underline{B} + ... = \#\underline{ABCDEFG}$
2. B, C, D, E, F, G create adaptor signature $BCDEFG$ that is missing $\#\underline{ABCDEFG}$, and pass it on to A
3. A and $X/Y/Z$ (known by $B/D/F$) create adaptor signature $AX/AY/AZ$ that is missing $\#\underline{ABCDEFG}$, and pass it on to $C/E/G$

Steps:
1. $C/E/G$ reveals $\#\underline{c}/\#\underline{e}/\#\underline{g}$ to $B/D/F$
2. $B/D/F$ reveals $\#\underline{bc}/\#\underline{de}/\#\underline{fg}$ to A
3. A publishes signature $BCDEFG$ to the Statechain, which reveals $\#\underline{abcdefg}$ and completes signature $AX/AY/AZ$
4. A helps $C/E/G$ acquire transitory key $X/Y/Z$ (*) See appendix

3 Improving Security

Thus far we have described a model with single entity A. Instead, we can replace A with a federated group of signatories. This can be a threshold (e.g. 8 of 10 plus the transitory key), which matches the security assumptions of federated sidechains.

Another potential security improvement is that the transfer of the transitory key can be handled by a Hardware Security Module (HSM). If the transitory key never leaves the HSM and thus can only act in accordance to the protocol, it becomes impossible for A to collude with transitory key owners. This assumes the HSM is unhackable, which thus far has been proven to be difficult to achieve[8]. We note that the protocol is naturally secure against loss of coins due to HSM failure, because the transitory key is only needed to transfer the coins and the off-chain bitcoin transaction can be stored outside of the HSM.

4 A Public Ledger

The entity that is operating the Statechain is expected to keep a public ledger in which every transfer gets recorded. This acts as evidence against unauthorized withdrawals. Either the unauthorized withdrawal conflicts with the ledger, or the Statechain was forked to match the withdrawal. Fraud can be proven in both cases, provided the users store the evidence that their transaction was at some point included in the Statechain. Unlike conventional blockchains, every UTXO has a history that is independent of the history of other UTXOs, since coins cannot be merged or split into multiple outputs. This allows you to selectively verify and track the history of only the UTXOs you care about.

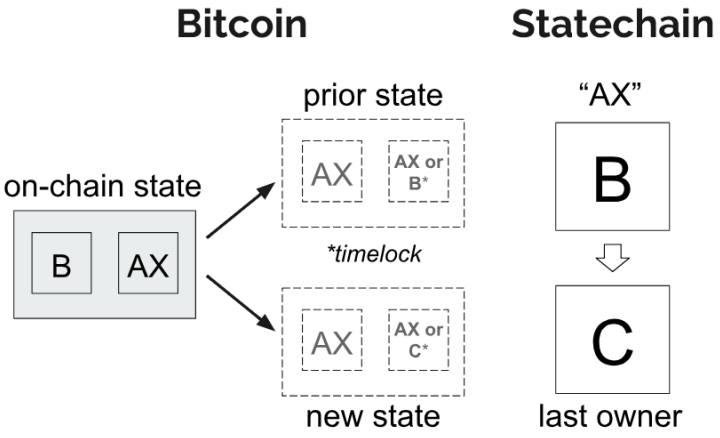

Figure 2: For every off-chain Bitcoin transaction an equivalent Statechain transaction exists.

If the Statechain entity is malicious, at worst they can secretly obtain a bunch of transitory keys and withdraw all of them on-chain simultaneously. After this, the provable misconduct ruins their

reputation, and everybody whose transitory key wasn't compromised can proceed to safely withdraw on-chain.

While the security of the transitory keys lies with the users, and therefore has the potential to leak, it does reduce the liability of the Statechain entity. Even if they were compelled by law to confiscate certain coins, they literally do not have the means to do so.

5 UTXOs as Coins

Since the protocol only allows full UTXOs to be swapped and transferred, the UTXOs can be thought of as fixed-value coins. If you want to pay 0.5 BTC but only have 1 BTC, you will first have to trade your 1 BTC output for two 0.5 BTC outputs with any willing online participant. For smaller amounts this may be infeasible, since on-chain redemption of small amounts may be too costly in terms of network fees. The Statechain should restrict the allowed denominations in order to facilitate trading bigger outputs for smaller ones. Note that the Statechain entity can support multiple cryptocurrencies, meaning the Statechain can enable users to execute swaps between them.

6 Microtransactions Through Payment Channels

In the context of Statechains, microtransactions are amounts that are smaller than the smallest allowed denomination. The system we described thus far, does not deal with microtransactions elegantly. This also means it's hard to charge fees for the service. One solution is to use Lightning channels on top of Statechains.

For example, if user B sends coins to BC, with an on-chain timelocked redemption back to B, we have essentially created a Lightning-compatible payment channel between B and C on the Statechain. The difference compared to a conventional Lightning channel is that the Statechain was off-chain to begin with, so opening and closing payment channels is cheap (if cooperative), allowing channel balances to be easily readjusted. This makes Statechains quite useful as a layer between Bitcoin and the Lightning Network. Also note that updating the Lightning channel and swapping UTXOs can happen atomically.

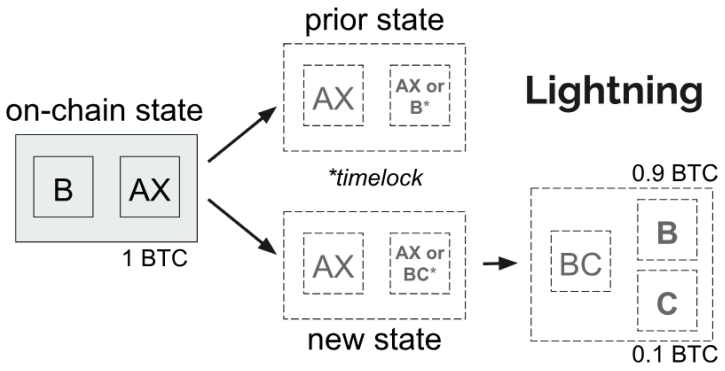

Figure 3: By assigning coins on the Statechain to two owners (BC), we have created a channel.

7 Withdrawing On-Chain

When user C wishes to withdraw from the Statechain, she can collaborate with A to create an on-chain withdrawal transaction. We can utilize Adaptor Signatures here as well in order to swap C's redemption signature with the on-chain transaction. Note that C has to specifically sign the message she wants A to sign, otherwise C can claim A sent the coins somewhere else than the agreed upon destination. Alternatively, assuming the blockchain supports Graftroot, C can swap her redemption signature for a Graftroot signature, allowing her to construct her own on-chain transaction.

One important benefit of using Graftroot is that it gives users a way to withdraw coins on other chains, in the event of a network split due to a hard fork. This also solves the issue for custodians of coins of deciding which side of a hypothetical fork they are going to honor. If the Graftroot script key is equal to the key that signed the withdrawal on the Statechain, then this also proves that A did not illegitimately claim the coins on a fork of Bitcoin.

If previous owner B ever sends their old on-chain transaction to the blockchain, a response is required in order to prevent him from taking the coins. Current owner C should always monitor the blockchain to detect when this happens. In response, she will either have to use her own on-chain transaction, or initiate a withdrawal with entity A.

During a withdrawal, the coins can also be deposited back into the Statechain, causing the UTXO to refresh. In general, it can be beneficial to refresh the UTXO after a certain number of Statechain transactions, since the likelihood of a transitory key leak increases over time.

References

[1] A.Back, M.Corallo, L.Dashjr, M.Friedenbach, G.Maxwell, A.Miller, A.Poelstra, J.Timón, P.Wuille, *Enabling Blockchain Innovations with Pegged Sidechains.* https://blockstream.com/sidechains.pdf, 2014

[2] J.Poon, T.Dryja, *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments.* https://lightning.network/lightning-network-paper.pdf, 2016

[3] J.Lau, G.Maxwell, J.Nick, A.Poelstra, T.Ruffing, R.Russell, A.Towns, P.Wuille, *Schnorr Signatures.* https://github.com/sipa/bips/blob/bip-schnorr/bip-schnorr.mediawiki, 2018

[4] C.Decker, R.Russell, O.Osuntokun, *eltoo: A Simple Layer2 Protocol for Bitcoin.* https://blockstream.com/eltoo.pdf, 2018

[5] Andrew Poelstra, *Scriptless Scripts.* http://diyhpl.us/~bryan/papers2/bitcoin/2017-03-mit-bitcoin-expo-andytoshi-mimblewmble-scriptless-scripts.pdf, 2017

[6] Gregory Maxwell, *Graftroot: Private and efficient surrogate scripts under the taproot assumption.* https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2018-February/015700.html, 2018

[7] G.Maxwell, A.Poelstra, Y.Seurin, P.Wuille, *Simple Schnorr Multi-Signatures with Applications to Bitcoin.* https://eprint.iacr.org/2018/068.pdf, 2018

[8] G.Chen, S.Chen, Y.Xiao, Y.Zhang, Z.Lin, T.H.Lai, *SgxPectre Attacks: Stealing Intel Secrets from SGX Enclaves via Speculative Execution* https://arxiv.org/pdf/1802.09085.pdf, 2018

(*) Appendix: Detailed explanation of transitory key exchange

Prior state: New state:
$AXor(B + timelock) \rightarrow AXor(C + timelock)$

Signature AX is constructed in such a way that A can help C learn transitory private key x.

Adaptor Signature AX:
$R = R1 + R2 + \#\underline{ABC}$
$s1' = r1 + a * hash(R, message)$
$s2' = r2 + x * hash(R, message)$
$s' = s1' + s2'$

Preparation:
1. $r1$ is chosen by A and should never be revealed
2. $r2$ is known to both B and C (if not, C can abort)
3. B gives $s1'$ to A
4. A gives s' and $R1$ to C (R can be derived from $R1$, which also proves $R2$ was used)

The final step of the transfer (after the signature is published on the Statechain):
A reveals $s1'$ to C, allowing her to learn the transitory key: $x = \frac{s' - s1' - r2}{hash(R, message)}$.

B could simply hand the transitory key to C, but this cannot be achieved atomically. Sending it beforehand risks exposing the transitory keys to participants who may abort the protocol early, and sending it afterwards assumes the willing cooperation of B, which at this point is no longer guaranteed. A, on the other hand, will want people to continue using their Statechain.