Bitcoin Clique: Channel-free Off-chain Payments using Two-Shot Adaptor Signatures

Siavash Riahi[1] and Orfeas Stefanos Thyfronitis Litos[2]

[1] TU Darmstadt
[2] Imperial College London
o.thyfronitis-litos@imperial.ac.uk

Abstract. Blockchains suffer from scalability limitations, both in terms of latency and throughput. Various approaches to alleviate this have been proposed, most prominent of which are payment and state channels, sidechains, commit-chains, rollups, and sharding. This work puts forth a novel commit-chain protocol, Bitcoin Clique. It is the first trustless commit-chain that is compatible with all major blockchains, including (an upcoming version of) Bitcoin.
Clique enables a pool of users to pay each other off-chain, i.e., without interacting with the blockchain, thus sidestepping its bottlenecks. A user can directly send its coins to any other user in the Clique: In contrast to payment channels, its funds are not tied to a specific counterparty, avoiding the need for multi-hop payments. An untrusted operator facilitates payments by verifiably recording them.
Furthermore, we define and construct a novel primitive, Two-Shot Adaptor Signatures, which is needed for Bitcoin Clique while being of independent interest. This primitive extends the functionality of normal Adaptor Signatures by allowing the extraction of the witness only after *two* signatures are published on the blockchain.

1 Introduction

Blockchain technologies have gained increasing popularity in the past decade as they provide a robust, secure, and decentralized infrastructure that allows parties to make monetary transactions, as well as execute applications. The main ingredient used in virtually all blockchains are consensus protocols, which guarantee that all honest parties have received and agree on the latest state of the system. Unfortunately, because of their distributed nature, public blockchains do not scale well in terms of throughput and latency [1]. For example, Bitcoin needs at least 1h to finalize a new transaction [2] and can process around 7 transactions per second, in contrast to centralized, trusted payment processors that achieve instant finality and can process tens of thousands of transactions per second.

To tackle this issue, *off-chain* protocols were introduced. An off-chain protocol allows parties to make transactions without involving the blockchain and only come on-chain in case of disputes, vastly increasing throughput. The first type of widely deployed off-chain protocols is *payment channels* [3,4,5,6,7]. Two parties open a channel with a single on-chain transaction, locking their funds into a "joint account". They can then pay each other many times entirely off-chain, via a fast two-party protocol. An honest party can always unilaterally retrieve its rightful funds on-chain, thus it does not need to trust its counterparty.

Nevertheless, locking coins for exclusive use with a single counterparty is a severe limitation. *Payment Channel Networks* (PCNs) [6,7] mitigate this by enabling *atomic multi-hop payments*. A routing algorithm specifies a path of channels between the payer and the payee, then each intermediary receives funds in one channel and atomically sends the same amount (minus a fee) to the other.

In order for a channel to serve as an intermediate hop, it needs to have enough balance on one of the two sides of the channel. Unfortunately, intermediary channels are often used excessively in one direction, leading to *channel imbalance*. *Payment Channel Hubs* (PCHs) [8,9,10] were introduced to mitigate this. A PCH is a PCN node that offers liquidity and reliability in exchange for higher fees.

To deliver on these guarantees, the PCH must have the capacity to handle a scenario in which all parties simultaneously pay all their coins to the same party. This needs a large amount of locked funds: Consider a PCH with n clients, each of which owns c coins in its channel with the hub. The latter must have $(n-1)c$ coins in its channel with each client P in order to support everyone else each giving c coins to P, for a grand total of $n(n-1)c$ coins locked by the hub. Due to these scalability issues, practical hubs restrict the allowed payments and charge the users high fees to compensate for the opportunity cost of their locked funds.

To tackle this limitation of PCHs, an alternative off-chain approach that foregoes channels completely was introduced: *plasma* or *commit-chain* protocols [11]. Here a separate log of transactions between participating users is maintained by an *untrusted* operator that periodically commits the latest system state on-chain efficiently. Due to this need for on-chain commitments, contrary to PCNs, commit-chains do not achieve *instant finality*. Still, they greatly reduce the required operator *collateral* while maintaining high throughput and low fees. In most such protocols the operator either needs no collateral at all or has to lock nc coins, a linear improvement compared to PCHs. A popular subcategory of commit-chains are *rollups* [12,13]. They store all transaction data on-chain, but carry out the associated computation off-chain.

To date, all commit-chain protocols need the Turing-complete capabilities of, e.g., Ethereum [14] to validate exit requests and disputes. In this work we present *Bitcoin Clique*, the first commit-chain protocol suitable for blockchains with a limited scripting language such as Bitcoin [2]. Clique enables its users to pay each other off-chain without having to lock coins with a specific counterparty, therefore completely avoiding the issues that PCNs face. A payment only needs the active participation of the payer, the payee and an untrusted *operator*. To achieve this we leverage `OP_CHECKTEMPLATEVERIFY` (`OP_CTV`) [15], an opcode that is a prime candidate for inclusion in the next Bitcoin soft fork, as well as a novel primitive of independent interest, *Two-Shot Adaptor Signatures*, which extends Adaptor Signatures [16] and is for the first time formally defined and provided with an efficient and provably secure construction in this work. At a high level, the latter enables the atomic exchange of a signature for a secret that satisfies a specific relation. This is useful for a range of applications [17,18,19,20]. Extending this primitive, we build Two-Shot Adaptor Signatures that disclose the secret upon the publication of *two* adapted signatures instead of just one.

As we formally prove, Bitcoin Clique achieves security and scalability, needing only three off-chain messages per payment and a single on-chain transaction of minimal size at fixed intervals. Building on top of Bitcoin brings commit-chains to blockchains with constrained scripting capabilities, providing Bitcoin users more versatility of off-chain solutions and expanding the use cases of the cryptocurrency. Furthermore, it informs designers of future blockchains that pursue minimal on-chain scripting capabilities without compromising on the achievable off-chain functionality.

Similarly to other commit-chains and optimistic rollups [13], our solution only finalizes payments upon an on-chain commitment. We find this to be an acceptable tradeoff in exchange for drastically higher throughput than on-chain payments, as well as more flexibility and less collateral than payment channels.

1.1 Our Contributions

Two-Shot Adaptor Signatures. We formally define and construct a novel cryptographic primitive, *Two-Shot Adaptor Signatures* (2-AS), which builds on Adaptor Signatures (AS) [16]. The security of our construction is fully proven. Intuitively, 2-AS ensures that a party that publishes a signature on each of *two* predefined messages atomically leaks a secret, whereas if only one of the two signatures is published, then the secret remains hidden. 2-AS are both useful for Bitcoin Clique and of independent interest. The construction works by combining two AS instances and the 2-AS secret is the *sum* of the two AS secrets.

Bitcoin Clique. We provide the first commit-chain that is compatible with Bitcoin and other UTXO-based blockchains, enabling trustless off-chain payments between commit-chain users with superior throughput and lower fees than on-chain transactions, while avoiding the shortcomings of payment channels. We use of two special tools to design our protocol: Firstly, we employ the to-be-added `OP_CTV` opcode, which enables securely updating the state of Clique with the active participation of just a single party, the *operator*. Secondly, we leverage our novel primitive *Two-Shot Adaptor Signatures* at the heart of our construction, which underpins a punishing mechanism against users that try to maliciously obtain twice their rightful coins upon exiting. Relevant security properties are defined and formally proven.

1.2 Related Work

Off-chain Channels. There has been extensive work on off-chain channels. The first line of works focused on off-chain payments over blockchains with a limited scripting language such as Bitcoin [4,5,6,21,22,23,24]. In [25] the Lightning Network (LN) [6] is formally proven secure in the UC framework [26]. State channels generalize payment channels by allowing parties to execute off-chain any application that is supported by the underlying blockchain, not just payments. 2-party state channels over Bitcoin are constructed in [16]. Most state channels constructions (e.g., [27,28,29,30]) function over Ethereum.

Commit-chains. The original concept of a commit-chain was introduced by Plasma [11]. Many different plasma protocol variants such as MVP [31], Cash [32] Debit [33] and Snapp [34] were introduced thereafter. These have been mostly discussed at `https://ethresear.ch` without formal treatment.

Formal treatment of commit-chain/Plasma solutions was first presesnted by NOCUST and NOCUST-ZKP [35]. Their solution requires the underlying blockchain to support Turing-complete smart contracts. Another technique [36] achieves better efficiency in comparsion to preexisting solutions but relies on Trusted Execution Environments (which our work does not require). Liquid [37] is a centralized commit-chain that functions on top of Bitcoin: users need to trust a supermajority of a fixed federation of servers. Compared to channels, commit-chains avoid imbalance issues, payment routing, complex channel management and unsustainable collateral in exchange for instant finality.

Fast Finality Techniques. Snappy [38] and LDSP [39] speed up transactions and are optimized for a small set of merchants that receive payments from a large set of customers. A subset of the merchants (a.k.a *statekeepers*) guarantee fast payment finality using the customer's collateral, before the transaction becomes finalized on-chain. They only allow for unidirectional payments and put all transactions on-chain. We compare LN-based PCHs, NOCUST, Snappy, and the current work in Table 1. There, for Snappy it is epoch = latency period [38].

Rollups. Finally, a solution similar to commit-chains is called *rollups.* This approach aims at performing expensive computation (i.e., executing smart contracts) off-chain, while committing all (unprocessed) data to the blockchain, effectively using the latter as a *data availability layer* while the rollup is active, and as a *finality platform* once a party leaves the rollup. Rollups (e.g., [12,13,40,41,42]) are essentially a special case of commit-chains. They are of lesser interest for blockchains with restricted scripting capabilities such as Bitcoin, where the storage of L1, not its computation, corresponds to the lion's share of the cost.

Extensions to Adaptor Signatures. The technique of [43] extends adaptor signatures to two pre-signers, who collaborate to pre-sign. Given then a single adapted pre-signature, they can extract the witness. In contrast, 2-AS has a single pre-signer that needs two adapted pre-signatures to extract the witness.

		PCH (LN)	NOCUST	Bitcoin Clique	Snappy (m statekeepers)
off-chain payment costs	Network (messages)	8	3	3	$3 + 2m$
	per-payment storage (user/operator)	$O(\log(\text{max pays}))$ / $O(\log(\text{max pays}))$	312b/841b (ephemeral)	1 sig + 5 pks/ 5 pks (ephemeral)	0 / 0
	fixed storage (user/operator)	2 ints + 2 pks/ $2n$ ints + $2n$ pks	529b/ $5n$ ints + n pks	n pks + n ints / n pks + n ints	0 / 0
on-chain overhead (txs)	startup	n	n	2	$n + m$
	pessimistic teardown	$2n$	$2n$	$2n$	$n + m$
	per epoch	—	1	1	0
	per payment	0	0	0	1
Works w/o Turing-complete SC		✓	✗	✓	✗
Allows any-to-any payments		✓	✓	✓	✗
user collateral (total payments of up to c coins/epoch)		—	0	0	c
operator collateral (c coins/user)		$n(n-1)c$	nc	nc	—
statekeeper collateral (insuring up to nc coins/epoch)		—	—	—	nc

Table 1. Comparison of PCHs based on LN [6], NOCUST [35], Bitcoin Clique, and Snappy [38] for n users. Ephemeral data is deleted after each epoch.

2 Preliminaries

A *digital signature scheme*, first formalized in [44], is an established cryptographic primitive that enables efficient message authentication. It provides (i) Gen, a *probabilistic polynomial time* (PPT) algorithm that generates a secret-public key pair, (ii) Sign, a PPT algorithm that, on input a secret key and an arbitrary message, produces a signature and (iii) *deterministic polynomial time* (DPT) Vrfy which, on input a public key, a message and signature, it returns whether the signature is valid. The security property ensures that, without knowledge of the secret key, one cannot forge a valid signature.

Consider next a security parameter $k \in \mathbb{N}$ and a *relation* $\mathcal{R}$, i.e., a set of statement-witness pairs (Y, y), where $Y, y \in \{0,1\}^*$. Let $L_{\mathcal{R}}$, the *language* of the relation, be the set of statements for which a valid witness exists: $L_{\mathcal{R}} = \{Y \mid \exists y \text{ s.t. } (Y, y) \in \mathcal{R}\}$. We further say that $\mathcal{R}$ is a *hard* relation if: (i) there exists a PPT algorithm $\mathsf{RGen}(1^k)$ that produces new (Y, y) pairs in $\mathcal{R}$, (ii) one can check efficiently (i.e., in polynomial in k time) whether a given (Y, y) pair is in $\mathcal{R}$ (i.e., $\mathcal{R}$ is decidable) and (iii) there is no PPT algorithm that, given Y, produces a witness y such that $(Y, y) \in \mathcal{R}$ with more than negligible probability in k.

Adaptor Signatures (AS). This scheme, formalized in [45], is built on a digital signatures scheme and a hard relation $\widetilde{\mathcal{R}}$. It enables the atomic exchange of (i) a valid signature on a message of interest $m \in \{0,1\}^*$ with (ii) a valid witness of a pre-agreed statement. In addition to the 3 algorithms of the underlying signatures, adaptor signatures provide 4 new ones: pSign, pVrfy, Adapt and Ext. In this work we leverage AS to build two-shot adaptor signatures (2-AS, Section 5).

The typical AS scenario involves two parties: Alice, who generates the pair $(Y, y) \in \widetilde{\mathcal{R}}$, keeps the witness y secret, and publishes Y, and Bob, who controls the signing keypair (sk, pk). Initially, Bob calls $\mathsf{pSign}(sk, m, Y)$ in order to *pre-sign* m, then sends the resulting *pre-signature* $\widetilde{\sigma}$ to Alice. She verifies that $\widetilde{\sigma}$ is valid by checking that $\mathsf{pVrfy}(pk, m, \widetilde{\sigma}, Y)$ returns 1. $\widetilde{\sigma}$ is however *not* a valid signature (i.e., $\mathsf{Vrfy}(pk_, m, \sigma) = 0$, where Vrfy is the verification algorithm of the underlying signature scheme). Nevertheless, Alice can call $\mathsf{Adapt}(pk, \widetilde{\sigma}, y)$ (note the use of her witness y) to obtain the desired valid signature σ: now it is $\mathsf{Vrfy}(pk, m, \sigma) = 1$. Alice then broadcasts σ (usually on a blockchain). The adapted signature σ is special: Bob can *extract* Alice's witness y from it by running $\mathsf{Ext}(\sigma, \widetilde{\sigma}, Y)$. The atomic exchange of σ for y is now complete.

The adapted signature σ thus serves a double role: It both proves that Bob indeed signed m and discloses Alice's witness to him.

A motivating application for this scheme is the atomic sale over a blockchain of a secret that satisfies a specific constraint, e.g., is the secret key of a specific public key: The seller Bob sends the statement (his public key) to the buyer Alice. She prepares a transaction that pays Bob, pre-signs it and sends him its pre-signature. Bob adapts it and publishes the transaction with the resulting signature. Lastly Alice extracts the witness (Bob's secret key) from the signature.

AS offer the following functional and security properties: (i) Bob cannot obtain a signature without adapting, (ii) if he adapts he will always obtain a valid signature and (iii) Alice can always extract the witness from an adapted signature. Thus, if Bob gets paid, then Alice learns the witness, ensuring atomicity.

CTV. We now provide some intuition on CTV, the proposed Bitcoin opcode [15] that we make heavy use of in this work. At a high level, it allows us to constrain the future use of coins. This new restriction ability enables complex ownership structures of coins, bringing to Bitcoin a large and useful subset of the smart contracts possible in blockchains with Turing-complete scripting languages [14] with a minimal, well-scrutinized modification to the Bitcoin Script.

Its mechanics are relatively simple: the CTV opcode is included in a transaction output and fully specifies every piece of data of the spending transaction, exept for the content of its inputs. At an intuitive level, it is enough to think that a CTV dictates the outputs of the next transaction.

For example, consider a transaction output θ' of c coins that is spendable by Alice, as well as another transaction output θ with $2c$ coins, encumbered only with a CTV that commits to a transaction with a single θ' output. This means that anyone can spend θ, as long as the spending transaction has a single output, θ'. The interpretation of this setup is that Alice has to pay a fee of c coins to the miners to gain c coins.

Let us examine a more useful example: Alice keeps her coins in an output encumbered with a CTV that specifies a single output. The latter is either spendable by her "hot wallet" key after a delay, or by her "cold wallet" key immediately. To pay, she first spends the CTV-encumbered output, then waits for the delay and finally uses the payment transaction. If however her hot wallet is compromised (which can presumably happen more easily than to her cold wallet), she still can salvage her coins with the cold wallet key within the delay window.

Observe that in the common case of no compromise, her cold wallet secret key is never used. This way to secure funds is currently impossible in Bitcoin.

More complex applications of CTV, such as Bitcoin Clique, implicate multiple mutually distrustful parties. Without CTV, all involved outputs would have to be signed by all parties, otherwise any missing party could be cheated out of its coins by the rest. This however does not scale, as it requires the active participation of all parties for any state update. Even worse, a single inactive party can lead to the protocol stalling, effectively locking honest party coins forever. CTV removes these pitfalls by fixing where the coins of involved outputs will go without new signatures by all parties on every update.

With regards to notation, consider a transaction $\mathtt{tx}$. We denote a CTV that commits to a transaction with the outputs of $\mathtt{tx}$ with $\mathsf{CTV}(\mathtt{tx})$: An output with spending condition $\mathsf{CTV}(\mathtt{tx})$ can only be spent by a transaction with the outputs of $\mathtt{tx}$ and no other transactions. For efficiency and privacy, a short commitment to the relevant $\mathtt{tx}$ data, generated with a hash function, is stored with $\mathsf{CTV}(\mathtt{tx})$.

3 Model

3.1 Blockchain and Transaction Model

In this work we focus on blockchains based on the Unspent Transaction Output (UTXO) model, such as Bitcoin. Under this model, coins are held in *outputs*. Formally, an output θ is a tuple (cash, φ), where cash denotes the amount of coins associated to the output and φ defines the conditions (also known as script) that need to be satisfied to spend the output. Our modeling is inspired by [46,16].

A *transaction* transfers coins across outputs, meaning that it consumes one or more existing outputs and creates a list of new outputs. A transaction has one *input* for each output it spends, which carries the *witness* that satisfies the script of the output being spent (typically one or more signatures). In other words, each transaction input is tied with exactly one previously unspent output of an older transaction. Thus, the transactions of a UTXO-based blockchain are organized in a directed, acyclic *transaction graph*. Formally, a transaction $\mathtt{tx}$ is a tuple of the form $(\mathsf{txid}, \mathsf{In}, \mathsf{Out}, \mathsf{Witness})$, where $\mathsf{txid} \in \{0,1\}^*$ is the unique identifier of $\mathtt{tx}$ and is calculated as $\mathsf{txid} := \mathcal{H}(\mathsf{In}, \mathsf{Out})$, where $\mathcal{H}$ is a hash function, commonly modeled as a random oracle. In is a vector of pointers to the outputs being spent and $\mathsf{Out} = (\theta_1, \dots, \theta_n)$ is a vector of the new outputs. The sum of coins of the new outputs must not exceed the sum of coins of the spent outputs. $\mathsf{Witness} \in \{0,1\}^*$ contains the witnesses that satisfy the scripts of the old outputs.

A valid transaction can be added to a single *block* of the blockchain (or *ledger*, $\mathcal{G}_{\text{Ledger}}$). A block consists of a number of transactions. There is a unique block for each *height* $\in \mathbb{N}$ and new blocks are continuously created. As explained below, the height of the block in which it is included can be leveraged by the script(s) of a transaction via a *timelock*. The *liveness* property guarantees that an honest transaction has to wait for at most $u \in \mathbb{N}$ blocks from submission to inclusion. One can of course store a transaction locally (a.k.a. *off-chain*) along with (some of) its witnesses in order to publish it later on-chain if needed.

Let us now enumerate the five types of spending conditions of an output used in this work. The most common spending condition is a public key. To satisfy it, the spending transaction must be signed with the corresponding secret key. Two more spending conditions are *absolute* and *relative timelocks.* These conditions make the output unspendable *before* a certain point in time. An absolute timelock is a block height after which the output can be spent. A relative timelock is the number of blocks that the output must stay on-chain before it can be spent. All timelocks in this work last for strictly longer than the liveness parameter u. The fourth spending condition is the threshold signature, which allows a subset of specific size of a designated set of keys to spend the output (this functionality is implemented with the `OP_CHECKSIGADD` opcode[3]). The last spending condition type is CTV, which has been introduced in Section 2.

We introduce our notation through examples: The spending condition $pk_B \wedge \mathsf{CTV}(\mathtt{tx}_2) + t_1$ of an output of $\mathtt{tx}_1$ can be spent by $\mathtt{tx}_2$ signed by sk_B, only after $\mathtt{tx}_1$ has been on-chain for t_1 blocks ("$+t$" denotes *relative timelock*). $(pk_C \wedge pk_D) \wedge t_2$ can be spent by a transaction signed by both sk_C and sk_D, only after block t_2 ("$\wedge t$" denotes *absolute timelock*).

3.2 Commit-chain model

A commit-chain protocol is executed among a set of users $\mathcal{P}$, an operator Op and $\mathcal{G}_{\text{Ledger}}$. We break the execution down into three phases: the *transaction*, the *exit*, and the *healing* phase. In the *transaction phase* users can transfer coins off-chain to one another and in the *exit phase* users can withdraw their rightful coins on-chain. Users that want to continue the Clique enter the *healing phase.*

Transaction phase. During this phase each user $P_i \in \mathcal{P}$ can send a message of the form $(P_i, P_j, v, \mathtt{aux})$ to the operator Op indicating that P_i wants to send v coins to user $P_j \in \mathcal{P}$. At the end of this phase each user $P \in \mathcal{P}$ attempts to compute a tuple of the form (v, e, π), where v is P's balance in epoch e and π is a *balance proof.* The protocol should ensure that a user can send coins to and receive coins from multiple users during this phase. Balances are not updated immediately but only at the end of the transaction phase. This property is referred to as *late* or *eventual finality.* Due to late finality, it could indeed be the case that an honest user cannot calculate the latest π at every moment. In this case the user will use its previous balance proof to exit the system if she so wishes without loss of funds.

Op is tasked with processing payments and updating user balances. Some commit-chain protocols require Op to send one or more on-chain transactions to $\mathcal{G}_{\text{Ledger}}$ to commit to the latest state of the system at the end of each epoch.

Exit phase. This phase can be triggered by any user $P \in \mathcal{P}$. It is carried out by submitting one or more suitable transactions to $\mathcal{G}_{\text{Ledger}}$. If Op misbehaves, P will detect it and exit in time, securely recovering all its coins on-chain.

[3] `github.com/bitcoin/bips/blob/master/bip-0342.mediawiki#cite_note-5`

Healing phase. Some commit-chain protocols require a restoration process by the users and *Op* to revert to the transaction phase after an exit phase is completed.

3.3 Communication and adversarial assumptions

Let us now discuss the communication and adversarial assumptions in our modeling. A commit-chain protocol is executed in the presence of a PPT adversary who can corrupt up to all but one parties. The corrupted parties are then controlled by the adversary, i.e., they can deviate from the protocol description and act in an arbitrary and possibly coordinated fashion.

We also assume that parties are connected via authenticated channels, i.e., the adversary can read, delay, replay or drop messages sent between parties but cannot modify their content. All parties have read and write access to $\mathcal{G}_{\text{Ledger}}$. The adversary cannot drop messages sent by an honest party to $\mathcal{G}_{\text{Ledger}}$, but it can delay them for up to a fixed period of time.

3.4 Security and Performance Guarantees

We here provide intuition for the intended guarantees of Bitcoin Clique. We refer the reader to Appendix C for the formal security definitions.

Transaction Phase Correctness. We say that a transaction is valid if the sender owns in the commit-chain more coins than the amount to be paid. During the transaction phase, if *Op*, the sender $P_i \in \mathcal{P}$ and the receiver $P_j \in \mathcal{P}$ of a valid transaction $(P_i, P_j, v, \texttt{aux})$ are honest, then either P_i's balance is reduced by v and P_j's balance is increased by v, or both balances remain unchanged (if the adversary drops or delays a message too much).

Exit Phase Correctness. If an honest user exits the commit-chain system, she is removed from the user set $\mathcal{P}$. For simplicity we assume that a user always exits with all her coins.

Balance Security. In the presence of *any number* of malicious parties, including *Op*, an honest user does not lose any coins at any stage of the protocol, i.e., an honest user is able to always exit with her entire balance. We note that due to late finality, this property essentially states that users will either be able to exit with their balance from the current or the previous epoch.

Operator Balance Security. An honest operator does not lose the collateral she deposited in the commit-chain, even in presence of *any number* of malicious users. Furthermore, she is able to exit the Clique at any time.

Formal security properties are given in Section 6 (Theorems 2 and 3).

Efficiency. Let t denote the duration of an epoch and c the per-epoch communication of Op with $\mathcal{G}_{\text{Ledger}}$. A commit-chain protocol is efficient if $t, c \in O(1)$, i.e., the duration of an epoch and the per-epoch communication of Op with $\mathcal{G}_{\text{Ledger}}$ independent of the number of both users and payments.

Efficiency is the reason why a commit-chain protocol is useful, as it guarantees that its payment fees are drastically lower than on-chain transaction fees.

4 Protocol and Primitive Overview

In this section we go over the Bitcoin Clique protocol and Two-Shot Adaptor Signatures in an informal but detailed manner, providing the necessary intuition behind both.

4.1 Bitcoin Clique

Consider users $\mathcal{P}$ with $|\mathcal{P}| = n$ and an operator Op running a Bitcoin Clique protocol. Under the current design, users can only own and exchange coins in a single, fixed denomination. Adding more denominations is relatively straightforward, but left as future work – discussion to that direction can be found in Section 7. In the current section we limit the total number of coins to be a power of 2 and we assume that each user owns 1 coin for ease of exposition; these limitations are not present in the formal protocol.

This subsection is organized as follows: We start with the protocol flow during normal operation, which includes payments and *epoch* changes. We then explain the off-chain tree of transactions that is the core of the construction. Subsequently the exit phase is discussed. Afterwards we elaborate on the mechanism which guarantees that epoch changes respect balance security; this is where two-shot adaptor signatures and the need for operator collateral come into play. Then the Clique setup procedure is presented, tying everything together. Lastly we discuss the healing mechanism, which is formally presented in Appendix D.

Transaction Phase. During normal operation, Alice $\in \mathcal{P}$ can send her coin to Bob $\in \mathcal{P}$ by sending him a single signed message, who in turn generates some keys and sends them, together with Alice's message, to Op. The latter then signs and publishes these messages to all Clique users. In practice, this last step is efficient, as Op can simply post them on, e.g., its website. Honest users should check that their payments appear there and initiate the exit phase if they do not appear within a reasonable length of time.

Periodically, i.e., at the end of each *epoch*, Op publishes to $\mathcal{G}_{\text{Ledger}}$ a specially crafted `step` transaction with 1 input and 1 output that carries the sum of all Clique coins and commits to the latest coin distribution. This transaction spends a previous `step` transaction. This is efficient: a transaction of minimal, constant size safeguards all epoch payments, irrespective of their number or the amount of users. Looking ahead, in order to move its coins back on-chain, any user can unilaterally start the exit phase by spending on-chain the `step` transaction.

Transactions structure. The central structure of a Clique is a binary tree of transactions with one leaf per coin, which exists entirely off chain during normal operation (i.e., until the exit phase). The root transaction of the tree has a single input that spends the step transaction and has two outputs, each with half the total coins. Each non-leaf transaction spends one of the two outputs of its parent and in turn provides two outputs, each with half the coins. Looking forward, a user can exit unilaterally by publishing to $\mathcal{G}_{\text{Ledger}}$ the branch of transactions that connect the root to its leaf, which contains $O(\log(n))$ transactions.

A parent transaction specifies its children using CTV. Crucially, CTV guarantees that *Op* can generate this tree locally, without interacting with the users, just by using their public keys. This avoids costly interactions and prevents a single user from stalling the protocol by inaction, ensuring the protocol is practical. Since CTV uses hashes, the resulting structure is a *Merkle tree* of transactions. This structure ensures logarithmic on-chain complexity for each user. An example Merkle tree can be seen in Figure 1.

Exit phase. If an honest user $P \in \mathcal{P}$ decides to move its coins back on-chain or detects misbehavior by *Op*, – slow response times, invalid responses, or an incorrect step transaction on-chain – it triggers the exit phase. As alluded to previously, P accomplishes this by publishing the root transaction that corresponds to the last valid step transaction, along with the $\log(n) + 1$ transactions that constitute its own branch of the Merkle Tree. In particular, each non-root transaction that P publishes spends one of the two outputs of its parent. This is the only way to spend this output without a timelock – the child transaction is specified via CTV.

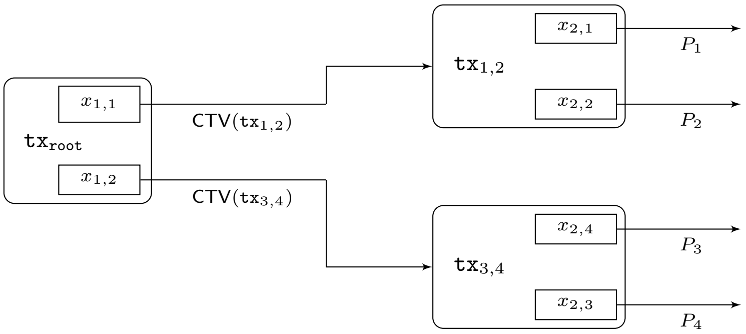

Fig. 1. Merkle tree for 4 users. The usage of CTV is exemplified.

The leaf transaction has 2 outputs as well, one of which concerns P. This output has a different spending condition: it requires a two-shot adaptor signature, pre-signed by *Op* and adapted by P — we will promptly explain why. P

spends the leaf output using an out transaction, which finally gives P access to its coins after a timelock.

For example, if $n = 128$ and P is the only exiting user, it has to publish the root transaction, another 6 Merkle tree transactions and the out transaction to exit, i.e., 8 constant-size transactions in total.

Once the step transaction is spent by P, it prompts all other Clique users to either follow the same on-chain procedure to retrieve their coins on-chain within a fixed timeframe — this is the timelock of the Merkle tree transactions we alluded to before — or join the healing phase (discussed below), otherwise their coins can be confiscated by Op. The latter is required to guarantee operator balance security. Note that a user R exiting after P needs to publish less than $\log(n) + 1$ transactions on-chain, since part of the tree has already been published. More specifically, if R exits after P and shares $1 \leq m \leq \log(n)$ levels of the Merkle tree with P, then R only needs to publish $\log(n) - m + 1$ transactions to exit.

Some details that are omitted here for simplicity can be found in Section 6.

Updating step transactions. One crucial question has been left unanswered: How does Op securely supersede the step transaction at each epoch change? On the one hand, if Op can freely spend the step transaction, it can simply steal all Clique coins without recourse. On the other hand, future payments are not known when the step transaction is generated, thus CTV cannot be used. Of course, requiring signatures by all users for each epoch update is impractical.

To resolve this quandary, the following solution is employed: *Two* step transactions are active and unspent at each instant. Each carries the entirety of the Clique coins. The first set of coins is initially provided by the users, whereas the second is provided by Op as *collateral*. At the end of each epoch, a timelock on the older one expires and Op can freely spend it. If Op is honest, it will use the next step transaction, as discussed earlier. If however it steals the coins or stays inactive, users exit via the other active step transaction – the CTV spending method, which requires the root transaction, is not timelocked. Op cannot steal the newer step transaction, as it is still timelocked. This technique ensures *balance security* for the users.

This solution however creates yet another problem: What prevents the users from simply exiting via *both* step transactions? This would effectively double each user's coins by stealing Op's collateral. This is where the two-shot adaptor signature shines. As alluded to above, $P \in \mathcal{P}$ has to publish an out transaction after the leaf transaction and wait for a timelock to access its coins. The out transaction needs a signature that P can only obtain by *adapting* a specially crafted pre-signature by Op using a specific AS witness. If Op learns *two* adapted signatures by P on out transactions of consecutive epochs, it can *extract* two AS witnesses, sum them to obtain the 2-AS witness and use the latter to confiscate the coins of one or both out transactions before P's timelock expires. Therefore P can claim its coins from either step transaction securely, but not from both. This technique provides *operator balance security.* We refer the reader to Figure 6 for a complete illustration.

Special care needs to be taken when coins change hands between epochs. In order to maintain operator balance security, the payee needs a 2-AS signature by the payer to spend its coin. This is so that Op can punish the payer if both payer and payee try to exit with the same coin.

Clique Setup. At last, all building blocks are in place. They are put together during the setup procedure as follows: Parties exchange keys and pre-signatures, then calculate the initial Merkle tree of transactions. Fixed conventions are used (e.g., lexicographic ordering of public keys) so that all parties agree on exactly the same tree. Each user then moves its c on-chain coins to the first step transaction, which exceptionally has n inputs. Its output commits to the Merkle root via CTV. Simultaneously Op moves its collateral (equal to the total Clique coins) to a step transaction that commits to the same Merkle root. As discussed before, Op can also spend them, but only after a timelock. The timelock of the second one is longer by t blocks. We say that t is the length of an epoch.

Observe that no user nor Op can lose coins during setup. Users only move their coins into the step transaction after ensuring that its output is the expected one and that they can spend their entire branch up to and including the out transaction (which needs the correct pre-signature). Likewise Op verifies that it can extract the required key and punish any user that attempts to take its coins from both step transactions.

Healing Phase. After one or more users exit, one or both step transactions are spent and part of the Merkle tree is on-chain. The remaining users need a mechanism to restore suitable unspent step transactions to carry on. We design a method by which the active users collaborate among them and with Op to consolidate the outputs of each Merkle tree into a new step transaction. This is achieved by including one more spending method to each output of each tree transaction. This method does not use a CTV, since the exiting users are not known when the tree is built and foreseeing all possible exit combinations leads to an exponential blowup. It instead needs a signature by Op and all users that have their coins in said output. At a high level, active users try to gather the needed signatures for the consolidating transaction. If some users that have not exited are inactive, the active users that share a tree output with them publish the minimum tree transactions needed to exclude the inactive users from the tree outputs and then try to consolidate again. Once the consolidating transaction is fully signed, it is published to $\mathcal{G}_{\text{Ledger}}$. The Clique is healed. A full description can be found in Appendix D.

4.2 Two-Shot Adaptor Signatures

As we saw in Section 2, a (simple) adaptor signature scheme (AS) [16] ties together the signature of a message (in our case a transaction) and the revelation of a secret value (a.k.a. witness). In a bit more detail, a pre-signer first generates a *pre-signature*, the publisher adapts this pre-signature using its witness, and upon publishing the resulting full signature the pre-signer can extract the publisher's witness using the pre- and full signatures. In order to ensure compatibility with Bitcoin, we instantiate AS with Schnorr adaptor signatures (we refer the reader to [16] for its details).

We extend this scheme to require *two* signatures for extraction. In particular, a two-shot adaptor signature scheme (2-AS) guarantees that extraction is impossible under a single valid signature and *prevention* of extraction is impossible under two valid signatures.

The construction of this primitive is an efficient modification of AS. Given the relation $\widetilde{\mathcal{R}}$ of the underlying AS, the relation $\mathcal{R}$ of 2-AS consists of two copies of $\widetilde{\mathcal{R}}$. Pre-signing, pre-signature verification, and adapting are done like in AS. Extraction performs the action of AS on the two signatures and returns the two extracted witnesses.

In order to ensure compatibility with Bitcoin, we instantiate AS with Schnorr adaptor signatures (we refer the reader to [16] for its details). We leave the proof that Schnorr adaptor signatures satisfy are secure as per the definition of [45] as future work.

The next two sections describe our two contributions in depth. We first present two-shot adaptor signatures in Section 5, as they are a prerequisite for Bitcoin Clique, which is presented subsequently in Section 6.

5 Two-Shot Adaptor Signatures

5.1 Primitive Definition

For the construction of Bitcoin Clique we require a primitive that enables a party to extract a secret after two signatures are posted on the blockchain. To this end we define and instantiate a more generalized adaptor signature scheme called *two-shot adaptor signature scheme* (2-AS). This primitive allows for the extraction of a witness given two pre-signatures and two full signatures.

Definition 1 (2-AS Syntax). *Let $k \in \mathbb{N}$ be the security parameter. A* two-shot adaptor signature signature scheme 2-AS *is defined with respect to a hard relation $\widetilde{\mathcal{R}}$ which can be sampled efficiently with* $\widetilde{\mathsf{R}}$Gen *and a digital signature scheme of which the algorithms* Gen *and* Vrfy *are provided as part of the interface. It also provides the following algorithms:*

2-pSign(sk, m, Y): a PPT algorithm that on input a secret key, a message $m \in \{0,1\}^*$ and $Y \in L_{\widetilde{\mathcal{R}}}$, outputs a pre-signature $\widetilde{\sigma}$.

2-pVrfy$(pk, m, \widetilde{\sigma}, Y)$: a DPT algorithm that on input a public key pk, a message $m \in \{0,1\}^*$, a pre-signature $\widetilde{\sigma}$, and a statement $Y \in L_{\widetilde{\mathcal{R}}}$, outputs 1 if the pre-signature is valid and 0 otherwise.

2-Adapt$(pk, \widetilde{\sigma}, y)$: a DPT algorithm that on input a public key pk, a pre-signature $\widetilde{\sigma}$, and a witness y, outputs a full signature σ.

2-Ext$(\sigma_1, \sigma_2, \widetilde{\sigma}_1, \widetilde{\sigma}_2, Y_1, Y_2)$: a DPT algorithm that on input two pairs of messages, full signatures and pre-signatures, outputs a tuple of two witnesses $\mathbf{y}$ or $\perp$ otherwise.

We provide the algorithms Gen and Vrfy of the underlying signatures scheme as part of the $\mathsf{2\text{-}AS}$ primitive unchanged in order to ensure that a full signature resulting from $\mathsf{2\text{-}Adapt}$ is compatible with protocols that use the signatures scheme but not the $\mathsf{2\text{-}AS}$ scheme, such as Bitcoin (cf. Definition 2 and Figure 2 below).

We define $\mathsf{RGen}(1^k)$ as:

$$[(Y_1, y_1) \xleftarrow{\$} \widetilde{\mathsf{R}}\mathsf{Gen}(1^k); (Y_2, y_2) \xleftarrow{\$} \widetilde{\mathsf{R}}\mathsf{Gen}(1^k); \textbf{return } ((Y_1, Y_2), (y_1, y_2))]$$

We now define the *Correctness* of a $\mathsf{2\text{-}AS}$ scheme, as well as its two security properties: *Pre-signature adaptability and non-extractability* and *Full Extractability.*

Correctness states that if a pair of pre-signatures are adapted using the correct witnesses, the resulting signatures are valid and one can extract the full witness given the pair of pre-signatures and signatures.

Definition 2 (Correctness). *A* $\mathsf{2\text{-}AS}$ *scheme is correct if* $\forall k \in \mathbb{N}$ *and any three messages* $m_1, m_2 \in \mathcal{M}$ *it holds:*

$$\Pr\left[\begin{array}{c|l} & (sk, pk) \xleftarrow{\$} \mathsf{Gen}(1^k) \\ \mathsf{2\text{-}pVrfy}(pk, m_1, \widetilde{\sigma}_1, Y_1) = 1 & ((Y_1, Y_2), (y_1, y_2)) \xleftarrow{\$} \mathsf{RGen}(1^k) \\ \wedge \mathsf{2\text{-}pVrfy}(pk, m_2, \widetilde{\sigma}_2, Y_2) = 1 & \widetilde{\sigma}_1 \xleftarrow{\$} \mathsf{2\text{-}pSign}(sk, m_1, Y_1) \\ \wedge \mathsf{Vrfy}(pk, m_1, \sigma_1) = 1 & \widetilde{\sigma}_2 \xleftarrow{\$} \mathsf{2\text{-}pSign}(sk, m_2, Y_2) \\ \wedge \mathsf{Vrfy}(pk, m_2, \sigma_2) = 1 & \sigma_1 \leftarrow \mathsf{2\text{-}Adapt}(\widetilde{\sigma}_1, y_1) \\ \wedge ((Y_1, Y_2), \mathbf{y}) \in \mathcal{R} & \sigma_2 \leftarrow \mathsf{2\text{-}Adapt}(\widetilde{\sigma}_2, y_2) \\ & \mathbf{y} \leftarrow \mathsf{2\text{-}Ext}(\sigma_1, \sigma_2, \widetilde{\sigma}_1, \widetilde{\sigma}_2, Y_1, Y_2) \end{array}\right] = 1.$$

The *adaptability & non-extractability* property roughly states that (i) if a pre-signature is valid then one can adapt it into a valid signature given the corresponding witness and (ii) a witness for a statement cannot be extracted when only a single signature that resulted from adapting a pre-signature is known.

```
Game 2-ADP-NEXT^A(1^k)
$(Y, (y_1, y_2)) \xleftarrow{\$} \mathsf{RGen}(1^k)$
$(pk, m, \widetilde{\sigma}, b, \mathsf{aux}) \leftarrow \mathcal{A}(Y)$
if $\neg\mathsf{2\text{-}pVrfy}(pk, m, \widetilde{\sigma}, Y)$ then return $0$
$\sigma \leftarrow \mathsf{2\text{-}Adapt}(\widetilde{\sigma}, y_b)$
if $\neg\mathsf{Vrfy}(pk, m, \sigma)$ then return $1$ // adaptability
$y^* \leftarrow \mathcal{A}(\sigma, \mathsf{aux})$
return $(Y, y^*) \in \mathcal{R}$ // non-extractability on single signature
```

Fig. 2. Game for Adaptability & Non-Extractability under single signature

Definition 3 (Pre-signature adaptability & non-extractability). *A* $\mathsf{2\text{-}AS}$ *scheme is* $\mathsf{2\text{-}ADP\text{-}NEXT}$*-secure if*

$$\forall k \in \mathbb{N}, \forall \mathcal{A} \in \mathsf{PPT}, \Pr[\mathsf{2\text{-}ADP\text{-}NEXT}^{\mathcal{A}}(1^k) = 1] < \text{negl}(k) \ .$$

We will now define the remaining property, Full Extractability, which is defined using a cryptographic game. It is based on the corresponding notion of [45]. At a high level, it ensures that the adversary cannot output a pair of valid signatures that has neither been published by the honest signer nor can be produced by adapting a suitable pair of honestly produced pre-signatures using a witness known to the adversary.

The 2-FEXT game captures this requirement by providing the adversary with a signing and a pre-signing oracle, as well as an oracle that outputs a random statement but not its witness. The adversary outputs two message-signature pairs and wins if they are both valid, neither message has been queried to the signing oracle and its output cannot be used to extract a valid witness. The oracles are defined in Figure 3 and 2-FEXT is formally defined in Figure 4.

```
Game Oracle $\mathcal{O}_{\mathsf{2pS}}$
$\mathcal{O}_{\mathsf{2pS}}(m, Y)$:
    $\widetilde{\sigma} \xleftarrow{\$} \mathsf{2\text{-}pSign}(sk, m, Y)$
    $\mathcal{T}(m) \xleftarrow{\cup} (Y, \widetilde{\sigma})$
    return $\widetilde{\sigma}$
```

```
Game Oracle $\mathcal{O}_{\mathsf{2S}}$
$\mathcal{O}_{\mathsf{2S}}(m)$:
    $\sigma \xleftarrow{\$} \mathsf{Sign}(sk, m)$
    $\mathcal{Q} \xleftarrow{\cup} \{m\}$
    return $\sigma$
```

```
Game Oracle $\mathcal{O}_{2\mathcal{R}}$
$\mathcal{O}_{2\mathcal{R}}()$:
    $(\cdot, Y) \xleftarrow{\$} \mathsf{RGen}()$
    $\mathcal{C} \xleftarrow{\cup} \{Y\}$
    return $Y$
```

Fig. 3. Oracles for the 2-FEXT game

```
Game 2-FEXT$^{\mathcal{A}}(1^k)$
    $\mathcal{Q} \leftarrow \emptyset$; $\mathcal{C} \leftarrow \emptyset$; $\mathcal{T} \leftarrow$ empty $\mathcal{M} \mapsto 2^{L_{\mathcal{R}} \times \text{pre-sigs}}$ map; $(sk, pk) \xleftarrow{\$} \mathsf{Gen}(1^k)$
    $(m_1^*, m_2^*, \sigma_1^*, \sigma_2^*) \leftarrow \mathcal{A}^{\mathcal{O}_{\mathsf{2pS}}, \mathcal{O}_{\mathsf{2S}}, \mathcal{O}_{2\mathcal{R}}}(pk)$
    $b_f \leftarrow \mathsf{Vrfy}(pk, m_1^*, \sigma_1^*) \wedge \mathsf{Vrfy}(pk, m_2^*, \sigma_2^*) \wedge m_1^* \notin \mathcal{Q} \wedge m_2^* \notin \mathcal{Q}$
    $b_w \leftarrow \nexists Y_1, Y_2, \widetilde{\sigma}_1, \widetilde{\sigma}_2 : (Y_1, \widetilde{\sigma}_1) \in \mathcal{T}(m_1^*) \wedge (Y_2, \widetilde{\sigma}_2) \in \mathcal{T}(m_2^*) \wedge Y_1 \notin \mathcal{C} \wedge Y_2 \notin \mathcal{C}$
        $\wedge((Y_1, Y_2), \mathsf{2\text{-}Ext}(\sigma_1^*, \sigma_2^*, \widetilde{\sigma}_1, \widetilde{\sigma}_2, Y_1, Y_2)) \in \mathcal{R}$
    return $b_f \wedge b_w$
```

Fig. 4. Game for Full Extractability

Definition 4. *A Two-Shot Digital Signature scheme is* 2-FEXT-secure *if*

$$\forall k \in \mathbb{N}, \forall \mathcal{A} \in \texttt{PPT}, \Pr[\mathsf{2\text{-}FEXT}^{\mathcal{A}}(1^k) = 1] < \operatorname{negl}(k) \ .$$

5.2 Construction

Let AS be a secure adaptor signature scheme w.r.t. a hard relation $\widetilde{\mathcal{R}}$. We now provide a construction of 2-AS based on AS. The relation $\mathcal{R}$ used in the construction is a tuple $((Y_1, Y_2), (y_1, y_2))$ where $\forall b \in \{1, 2\}, (Y_b, y_b)$ form an $\widetilde{\mathcal{R}}$ tuple. In other words, we have:

$$\mathcal{R} = \{((Y_1, Y_2), (y_1, y_2)) | (Y_1, y_1) \in \widetilde{\mathcal{R}} \wedge (Y_2, y_2) \in \widetilde{\mathcal{R}} \wedge y_1 \neq y_2\} \ . \tag{1}$$

Our construction of 2-AS is defined in Figure 5.

```
$\mathsf{2\text{-}pSign}(sk, m, Y)$
    return $\mathsf{AS.pSign}(sk, m, Y)$
```

```
$\mathsf{2\text{-}Ext}(\sigma_1, \sigma_2, \widetilde{\sigma}_1, \widetilde{\sigma}_2, Y_1, Y_2)$
    $y_1 \leftarrow \mathsf{AS.Ext}(\sigma_1, \widetilde{\sigma}_1, Y_1)$
    $y_2 \leftarrow \mathsf{AS.Ext}(\sigma_2, \widetilde{\sigma}_2, Y_2)$
    return $(y_1, y_2)$
```

```
$\mathsf{2\text{-}pVrfy}(pk, m, \widetilde{\sigma}, Y)$
    return $\mathsf{AS.pVrfy}(pk, m, \widetilde{\sigma}, Y)$
$\mathsf{2\text{-}Adapt}(pk, \widetilde{\sigma}, y)$
    return $\mathsf{AS.Adapt}(pk, \widetilde{\sigma}, y)$
```

Fig. 5. Two-Shot Adaptor Signature construction

Theorem 1 (Security). *Assume that* AS *is a secure adaptor signature from [16] with respect to $\widetilde{\mathcal{R}}$, and let $\mathcal{R}$ be a hard relation defined as in Equation (1) w.r.t. $\widetilde{\mathcal{R}}$. Then the construction defined in Figure 5 is a secure two-shot adaptor signature scheme in the random oracle model.*

To prove Theorem 1 we have to prove the following 3 Lemmas.

Lemma 1 (Correctness). *Under the assumptions of Thm. 1, the construction of Figure 5 satisfies correctness (Def. 2).*

Lemma 2 (Adaptability). *Under the assumptions of Thm. 1, the construction of Figure 5 satisfies pre-signature adaptability (Def. 3).*

Lemma 3 (2-FEXT Security). *Under the assumptions of Thm. 1, the construction of Figure 5 satisfies* 2-FEXT *security (Def. 4).*

We refer the reader to Appendix A for the proofs of Lemmas 1, 2, and 3. Our scheme can be easily extended to a k-shot adaptor signature for $k > 2$, however we do not currently know of any applications for such a primitive and as such we have focussed on $k = 2$.

6 Bitcoin Clique Protocol

We now present our protocol in more detail. An illustration of the CTV-based Merkle tree can be seen in Figure 6. Thanks to CTV, the root transaction is the only transaction that can spend the on-chain $\mathtt{tx}_{\mathsf{step}}$.

To update the balances of the users at the end of each epoch, this Merkle tree and the associated $\mathtt{tx}_{\mathsf{step}}$ need to be updated by *Op*. As we saw earlier, after the end of the epoch *Op* has to be able to freely spend the current $\mathtt{tx}_{\mathsf{step}}$ and replace it with the desired next $\mathtt{tx}_{\mathsf{step}}$. As discussed, to prevent *Op* from abusing this power and stealing all Clique coins, two step txs exist on-chain at any time. To protect *Op* from losing its collateral by a user that spends both step txs, the aforementioned 2-AS-based technique is employed. In Figure 6, P_1's secret y_1 is revealed if P_1 exits from both trees (i.e., by spending the $\mathtt{tx}_{\mathsf{step}}$ of two consecutive epochs) and *Op* can use it on the $pk_{Op} \wedge Y_1$ spending condition of $\mathtt{tx}_{\mathsf{out},e,1}$ to punish P_1. *Op* is not in a race with P_1, since the latter cannot spend the coins immediately but needs to wait until block t_p (spending condition $pk_{\mathsf{out},1} \wedge t_p$ of $\mathtt{tx}_{\mathsf{out},e,1}$).

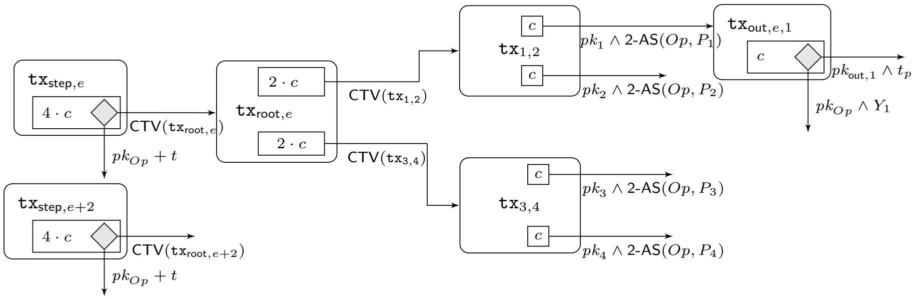
$\texttt{tx}_{\mathsf{step},e}$
$4 \cdot c$
CTV($\texttt{tx}_{\mathsf{root},e}$)
$pk_{Op} + t$
$\texttt{tx}_{\mathsf{step},e+2}$
$4 \cdot c$
CTV($\texttt{tx}_{\mathsf{root},e+2}$)
$pk_{Op} + t$
$2 \cdot c$
$\texttt{tx}_{\mathsf{root},e}$
$2 \cdot c$
CTV($\texttt{tx}_{1,2}$)
CTV($\texttt{tx}_{3,4}$)
c
$\texttt{tx}_{1,2}$
c
$pk_1 \wedge$ 2-AS(Op, P_1)
$pk_2 \wedge$ 2-AS(Op, P_2)
c
$\texttt{tx}_{3,4}$
c
$pk_3 \wedge$ 2-AS(Op, P_3)
$pk_4 \wedge$ 2-AS(Op, P_4)
$\texttt{tx}_{\mathsf{out},e,1}$
c
$pk_{\mathsf{out},1} \wedge t_p$
$pk_{Op} \wedge Y_1$

Fig. 6. Illustration of a Bitcoin Clique with 4 users, showing the transactions that can be published on-chain for the step transaction of epoch e, $\texttt{tx}_{\mathsf{step},e}$. 2-AS($Op, P_j$) represents a spending condition that requires a signature generated via a two-shot adaptor signature, where Op is the pre-signer and P_j the adapter. The diamond notation represents an OR spending condition, e.g., $\texttt{tx}_{\mathsf{out},e,1}$ can be spent either by P_1 after block t_p or by Op if she knows y_1 such that $(Y_1, y_1) \in \mathcal{R}$. Op can learn y_1 only if P_1 maliciously publishes the $\texttt{tx}_{\mathsf{out}}$ of two consecutive epochs. The $\texttt{tx}_{\mathsf{out}}$ of *two* epochs are unspent at any point during a transaction phase, here only one is shown. The outputs of $\texttt{tx}_{\mathsf{root},e}$, $\texttt{tx}_{1,2}$ and $\texttt{tx}_{3,4}$ can be spent by Op after a timelock, thus preventing a coalition of malicious users from indefinitely blocking Op's collateral. These timelocked spending methods however are omitted here for conciseness.

In order for P to pay R, the latter generates a new statement-witness pair for 2-AS along with new keys for the tree and out txs. All users are informed by Op about the new keys, so that they can take them into account when computing the tree of the next epoch. Simply switching from P's to R's keys at the new epoch however would expose Op to an attack: P takes its output in the old epoch and R takes its output in the new epoch, thus Op loses an equal collateral. As alluded to earlier, the protection is as follows: When the current epoch ends, P receives a pre-signature from Op, adapts it, and gives the complete signature to R. R needs this extra signature to obtain its coins during the next epoch. Therefore, if both P and R try to obtain the same coin, Op will learn P's secret and retrieve its collateral from P's out tx in the current epoch.

To sum up, at any time there are two unspent $\texttt{tx}_{\mathsf{step}}$ on-chain, representing the last two epochs. Each can be spent by the corresponding tree of transactions, or by Op after a timelock. The two timelocks are staggered, so that Op cannot spend both $\texttt{tx}_{\mathsf{step}}$ simultaneously. At the end of the e-th epoch, Op spends one $\texttt{tx}_{\mathsf{step},e}$ with a new $\texttt{tx}_{\mathsf{step},e+2}$, alternating between the two series of step txs on every epoch. If some users exit, the rest can actively collaborate to heal the Clique by signing and publishing a single tx which moves all available coins to a new step output and carry on with the protocol.

We next provide the protocol pseudocode. We refer the reader to Appendix B for the full protocol code, to Appendix C for its security proof, and to Appendix D for the healing subprotocol.

Bitcoin Clique

Constants: N users with c coins each, operator Op, each epoch lasts t blocks.

Setup Phase.

1. Public keys distribution:
   - Op and users exchange normal and 2-AS keys.
2. Initial transactions preparation:
   - $\texttt{tx}_{\mathsf{step},1}$ is funded by the N users and has a t-block timelock.
   - $\texttt{tx}_{\mathsf{step},2}$ is funded by Op with Nc collateral coins and has a $2t$-block timelock.
3. $\texttt{out}$ transactions preparation:
   - Op pre-signs the two $\texttt{out}$ txs of each user using as statement the 2-AS keys of the user and sends the two pre-signatures to the user for verification.
4. Setup Finalization:
   - Users sign $\texttt{tx}_{\mathsf{step},1}$ and Op signs $\texttt{tx}_{\mathsf{step},2}$, the two txs are published to $\mathcal{G}_{\text{Ledger}}$.

Payment Phase (P transfers an output to R).

1. P sends to R a signed message with the output, R's id and the next epoch.
2. R sends to Op new normal and 2-AS keys, along with P's message.
3. Op generates a new 2-AS key for this output and sends it to all N users, along with P's message and R's keys.
4. When the current epoch ends, P adapts the pre-signature by Op, gets a valid signature and sends it to R for verification, who needs this signature to spend the corresponding coins (see l. 1 of Epoch Finalization & l. 3 of User Exit).

Epoch Finalization Phase. When the timelock of $\texttt{tx}_{\mathsf{step},e}$ expires:

1. Op generates the $(e+2)$-th tx tree and $\texttt{tx}_{\mathsf{step},e+2}$ and publishes the latter, which spends $\texttt{tx}_{\mathsf{step},e}$. For each output that has been transfered during epoch e, Op uses the 2-AS keys of both the sender and the receiver to build the new tx tree. This means that signatures from both parties are needed to spend the *leaf* tx of this output at epoch $e+1$.
2. Op pre-signs the new $\texttt{out}$ tx of each user using as statement the 2-AS key of the user and sends the pre-signature to the user.
3. Each user verifies that the epoch change has taken place in a timely manner, with the expected tx tree, and that the pre-signature is valid.

User Exit Phase. P must exit when it detects any dishonest behavior. The procedure below is repeated for each of P's outputs.

1. P signs and publishes all txs that constitute the path from the root to its leaf of the latest Merkle tree, spending the latest unspent $\texttt{tx}_{\mathsf{step}}$.
2. P adapts the relevant pre-signature and adds the resulting signature to $\texttt{tx}_{\mathsf{out},e}$.
3. If P received its output at the latest epoch, P also adds the previous owner's signature to $\texttt{tx}_{\mathsf{out},e}$ and publishes it.

4. P stops any action related to this output except for further use of its now on-chain coins. This prevents accidentally adapting another pre-signature and disclosing P's secret keys to Op.

Operator Exit Phase. Op needs to receive Nc coins to recover its collateral.

1. Op tries to get the coins of a $\mathtt{tx}_{\mathsf{step}}$ of which the timelock has expired.
2. If this fails (because both step txs are spent by the root tx of the corresponding tx tree), Op tries to take c coins per user:
   - If the timelock of any tx in the tx tree expires, Op gets its funds from it (thus receiving value equal to the sum of coins that are owned by the users that have tx in their path).
   - For every user P that has published both its out txs (and thus no timelock on either of its paths is left to expire), Op extracts both P's AS secrets from the signatures using 2-Ext.
   - Op spends at least one of P's two out txs using its own secret key and the sum of P's two secrets, thus taking c coins from P as desired.

The two central balance security theorems follow, where an *environment* $\mathcal{E}$ may order any party to exit at any time:

Theorem 2 (User balance security). $\forall$ *honest* $P \in \mathcal{P}$ *that owns a set of outputs* O *in the protocol, if it is instructed by* $\mathcal{E}$ *to exit (Figure 16) then it will eventually exclusively own all outputs in* O *on-chain.*

This theorem also covers any case of emergency exit or response to someone else's exit, since in such a case P must have already safeguarded or be in the process of safeguarding its outputs when it receives $\mathcal{E}$'s exit instruction. It holds because an honest user can retrieve its coins on-chain after a failed setup, it can unilaterally put exactly one out tx on-chain any time after a successful setup, and the timelock of the out tx will always expire, giving the user access to its funds on-chain.

Theorem 3 (Operator balance security). *If honest* Op *is instructed by* $\mathcal{E}$ *to exit (Figure 18) then eventually* Op *will exclusively own at least the sum of all players' outputs (which is equal to* Op*'s collateral).*

This theorem also implicitly covers any case in which a response to someone else's exit is needed. As discussed, it holds because Op can always claim the coins back, either from an expired timelock of a step or tree tx, or by punishing a user that published two out txs (and thus leaked its secret to Op).

Formal proofs for both theorems can be found in Appendix C. Transaction and exit phase correctness as well as efficiency can be verified by simple inspection of the protocol.

7 Future Work

Several future work directions remain open. To begin with, only unilateral closure was considered. This however has a high aggregate on-chain cost and, in case of closure of a big Clique, could create on-chain congestion. Our protocol can be extended in a straightforward manner to efficiently handle cooperative exiting of a subset of the users. This is doable by moving the exiting users' outputs from the leaves of the Merkle tree to the next step transaction. This solution only needs the cooperation of Op, not of all Clique users, maintaining practicality.

Furthermore, the current construction is not privacy-preserving, as all parties learn all payments. Per-epoch mixing techniques can be used to bolster privacy.

Additionally, removing the fixed-denomination payment value limitation and the need for operator collateral would greatly improve usability and practicality. A simple extension of our protocol can provide multiple denominations by including one Merkle tree per denomination. Fiat cash exemplifies how this approach could be sufficient for practical use.

Operators introduce centralization concerns. Nevertheless, since many Cliques with different operators can coexist and compete, operators are dissuaded from providing poor service, and balance security ensures users only rely on the operator for quality of service, not for funds safety. Operator power can be further limited by (i) adding a voting mechanism among users to replace the operator and (ii) enabling inter-Clique payments. These are left as future directions.

What is more, the tree structure need not necessarily be binary. It is possible that other structures are in practice more efficient, e.g., tertiary trees. Complementarily, leaf transactions with more than 2 users can be leveraged, trimming a few levels from the tree. Such optimizations are left as a concern for a possible future production-level implementation.

Last but not least, to the best of our knowledge, the security of Schnorr adaptor signatures with respect to the definitions of [45] has not been proven. We leave this proof as future work.

Acknowledgements. This work was partly supported by the German Federal Ministry of Education and Research and the Hessen State Ministry for Higher Education, Research and the Arts within their joint support of the National Research Center for Applied Cybersecurity ATHENE.

References

1. Croman K., Decker C., Eyal I., Gencer A. E., Juels A., Kosba A., Miller A., Saxena P., Shi E., Sirer E. G., et al.: On scaling decentralized blockchains. In International Conference on Financial Cryptography and Data Security: pp. 106–125: Springer (2016)
2. Nakamoto S.: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)
3. Gudgeon L., Moreno-Sanchez P., Roos S., McCorry P., Gervais A.: SoK: Layer-Two Blockchain Protocols. In Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers: pp. 201–226: https://doi.org/10.1007/978-3-030-51280-4_12: URL https://doi.org/10.1007/978-3-030-51280-4_12 (2020)
4. Bitcoin Wiki: Payment Channels. https://tinyurl.com/y6msnk7u

5. Decker C., Wattenhofer R.: A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In Stabilization, Safety, and Security of Distributed Systems 2015: pp. 3–18 (2015)
6. Poon J., Dryja T.: The Bitcoin Lightning Network: Scalable Off-chain Instant Payments. https://tinyurl.com/q54gnb4 (2016)
7. Update from the Raiden team on development progress, announcement of raidEX. https://tinyurl.com/z2snp9e (2017)
8. Dziembowski S., Eckey L., Faust S., Malinowski D.: Perun: Virtual Payment Hubs over Cryptocurrencies. In 2019 IEEE Symposium on Security and Privacy: pp. 106–123: IEEE Computer Society Press: https://doi.org/10.1109/SP.2019.00020 (2019)
9. Tairi E., Moreno-Sanchez P., Maffei M.: A^{2}L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In 2021 IEEE Symposium on Security and Privacy: pp. 1834–1851: IEEE Computer Society Press: https://doi.org/10.1109/SP40001.2021.00111 (2021)
10. Qin X., Pan S., Mirzaei A., Sui Z., Ersoy O., Sakzad A., Esgin M. F., Liu J. K., Yu J., Yuen T.: BlindHub: Bitcoin-Compatible Privacy-Preserving Payment Channel Hubs Supporting Variable Amounts. In 2023 IEEE Symposium on Security and Privacy (SP): pp. 2462–2480: IEEE Computer Society, Los Alamitos, CA, USA: https://doi.org/10.1109/SP46215.2023.10179427: URL https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.10179427 (2023)
11. Poon J., Buterin V.: Plasma: Scalable autonomous smart contracts (2017)
12. Buterin V.: On-chain scaling to potentially 500 tx/sec through mass tx validation. https://ethresear.ch/t/on-chain-scaling-to-potentially-500-tx-sec-through-mass-tx-validation/3477
13. Optimism: Optimistic rollup overview. https://github.com/ethereum-optimism/optimistic-specs/blob/0e9673af0f2cafd89ac7d6c0e5d8bed7c67b74ca/overview.md
14. Wood G.: Ethereum: A secure decentralised generalised transaction ledger
15. Rubin J.: Bitcoin Improvement Proposal 119. https://github.com/bitcoin/bips/blob/master/bip-0119.mediawiki
16. Aumayr L., Ersoy O., Erwig A., Faust S., Hostáková K., Maffei M., Moreno-Sanchez P., Riahi S.: Generalized Channels from Limited Blockchain Scripts and Adaptor Signatures. In M. Tibouchi, H. Wang (editors), ASIACRYPT 2021, Part II: vol. 13091 of *LNCS*: pp. 635–664: Springer, Heidelberg: https://doi.org/10.1007/978-3-030-92075-3_22 (2021)
17. Eckey L., Faust S., Hostáková K., Roos S.: Splitting Payments Locally While Routing Interdimensionally. IACR Cryptol. ePrint Arch.: p. 555: URL https://eprint.iacr.org/2020/555 (2020)
18. Malavolta G., Moreno-Sanchez P., Schneidewind C., Kate A., Maffei M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019: The Internet Society: URL https://www.ndss-symposium.org/ndss-paper/anonymous-multi-hop-locks-for-blockchain-scalability-and-interoperability/ (2019)
19. Tairi E., Moreno-Sanchez P., Maffei M.: A^{2}L: Anonymous Atomic Locks for Scalability in Payment Channel Hubs. In 42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021: pp. 1834–1851:

IEEE: https://doi.org/10.1109/SP40001.2021.00111: URL https://doi.org/10.1109/SP40001.2021.00111 (2021)
20. Thyagarajan S. A. K., Malavolta G., Schmidt F., Schröder D.: PayMo: Payment Channels For Monero. IACR Cryptol. ePrint Arch.: p. 1441: URL https://eprint.iacr.org/2020/1441 (2020)
21. Malavolta G., Moreno-Sanchez P., Kate A., Maffei M., Ravi S.: Concurrency and Privacy with Payment-Channel Networks. In B.M. Thuraisingham, D. Evans, T. Malkin, D. Xu (editors), ACM CCS 2017: pp. 455–471: ACM Press: https://doi.org/10.1145/3133956.3134096 (2017)
22. Malavolta G., Moreno-Sanchez P., Schneidewind C., Kate A., Maffei M.: Anonymous Multi-Hop Locks for Blockchain Scalability and Interoperability. In NDSS 2019: The Internet Society (2019)
23. Avarikioti Z., Litos O. S. T., Wattenhofer R.: Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In J. Bonneau, N. Heninger (editors), Financial Cryptography and Data Security - 24th International Conference, FC 2020, Kota Kinabalu, Malaysia, February 10-14, 2020 Revised Selected Papers: vol. 12059 of *Lecture Notes in Computer Science*: pp. 346–366: Springer: https://doi.org/10.1007/978-3-030-51280-4_19: URL https://doi.org/10.1007/978-3-030-51280-4_19 (2020)
24. Avarikioti Z., Litos O. S. T.: Suborn Channels: Incentives Against Timelock Bribes. In I. Eyal, J.A. Garay (editors), Financial Cryptography and Data Security - 26th International Conference, FC 2022, Grenada, May 2-6, 2022, Revised Selected Papers: vol. 13411 of *Lecture Notes in Computer Science*: pp. 488–511: Springer: https://doi.org/10.1007/978-3-031-18283-9_24: URL https://doi.org/10.1007/978-3-031-18283-9_24 (2022)
25. Kiayias A., Litos O. S. T.: A Composable Security Treatment of the Lightning Network. In IEEE CSF 2020: pp. 334–349
26. Canetti R.: Universally Composable Security: A New Paradigm for Cryptographic Protocols. In 42nd FOCS: pp. 136–145: IEEE Computer Society Press: https://doi.org/10.1109/SFCS.2001.959888 (2001)
27. Dziembowski S., Faust S., Hostáková K.: General State Channel Networks. In D. Lie, M. Mannan, M. Backes, X. Wang (editors), ACM CCS 2018: pp. 949–966: ACM Press: https://doi.org/10.1145/3243734.3243856 (2018)
28. Dziembowski S., Eckey L., Faust S., Hesse J., Hostáková K.: Multi-party Virtual State Channels. In Y. Ishai, V. Rijmen (editors), EUROCRYPT 2019, Part I: vol. 11476 of *LNCS*: pp. 625–656: Springer, Heidelberg: https://doi.org/10.1007/978-3-030-17653-2_21 (2019)
29. Miller A., Bentov I., Bakshi S., Kumaresan R., McCorry P.: Sprites and State Channels: Payment Networks that Go Faster Than Lightning. In I. Goldberg, T. Moore (editors), FC 2019: vol. 11598 of *LNCS*: pp. 508–526: Springer, Heidelberg: https://doi.org/10.1007/978-3-030-32101-7_30 (2019)
30. Chakravarty M. M. T., Coretti S., Fitzi M., Gazi P., Kant P., Kiayias A., Russell A.: Hydra: Fast Isomorphic State Channels. Cryptology ePrint Archive, Report 2020/299: https://eprint.iacr.org/2020/299 (2020)
31. Buterin V.: Minimal Viable Plasma. https://tinyurl.com/y2s9grpd (2018)
32. Floersch K.: Plasma Cash Simple Spec. https://tinyurl.com/yxdp2rqr (2018)
33. Plasma Debit. https://tinyurl.com/yx936xzk (2018)
34. Plasma snapp. https://tinyurl.com/yxbza3pl (2018)
35. Khalil R., Zamyatin A., Felley G., Moreno-Sanchez P., Gervais A.: Commit-Chains: Secure, Scalable Off-Chain Payments. Cryptology ePrint Archive, Report 2018/642: https://eprint.iacr.org/2018/642 (2018)

36. Erwig A., Faust S., Riahi S., Stöckert T.: CommiTEE: An Efficient and Secure Commit-Chain Protocol using TEEs. In 2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P): pp. 429–448: IEEE Computer Society, Los Alamitos, CA, USA: https://doi.org/10.1109/EuroSP57164.2023.00033: URL https://doi.ieeecomputersociety.org/10.1109/EuroSP57164.2023.00033 (2023)
37. Nick J., Poelstra A., Sanders G.: Liquid: A Bitcoin Sidechain (2020)
38. Mavroudis V., Wüst K., Dhar A., Kostiainen K., Capkun S.: Snappy: Fast On-chain Payments with Practical Collaterals. In 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020: The Internet Society: URL https://www.ndss-symposium.org/ndss-paper/snappy-fast-on-chain-payments-with-practical-collaterals/ (2020)
39. Ng L. K. L., Chow S. S. M., Wong D. P. H., Woo A. P. Y.: LDSP: Shopping with Cryptocurrency Privately and Quickly under Leadership. In 2021 IEEE 41st International Conference on Distributed Computing Systems (ICDCS): pp. 261–271: https://doi.org/10.1109/ICDCS51616.2021.00033 (2021)
40. Whitehat B.: Roll up. https://github.com/barryWhiteHat/roll_up
41. Donno L.: Optimistic and Validity Rollups: Analysis and Comparison between Optimism and StarkNet. CoRR: vol. abs/2210.16610: https://doi.org/10.48550/arXiv.2210.16610: URL https://doi.org/10.48550/arXiv.2210.16610 (2022)
42. Kalodner H. A., Goldfeder S., Chen X., Weinberg S. M., Felten E. W.: Arbitrum: Scalable, private smart contracts. In W. Enck, A.P. Felt (editors), 27th USENIX Security Symposium, USENIX Security 2018, Baltimore, MD, USA, August 15-17, 2018: pp. 1353–1370: USENIX Association: URL https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner (2018)
43. Erwig A., Faust S., Hostáková K., Maitra M., Riahi S.: Two-Party Adaptor Signatures from Identification Schemes. In J.A. Garay (editor), Public-Key Cryptography - PKC 2021 - 24th IACR International Conference on Practice and Theory of Public Key Cryptography, Virtual Event, May 10-13, 2021, Proceedings, Part I: vol. 12710 of *Lecture Notes in Computer Science*: pp. 451–480: Springer: https://doi.org/10.1007/978-3-030-75245-3_17: URL https://doi.org/10.1007/978-3-030-75245-3_17 (2021)
44. Katz J., Lindell Y.: Introduction to Modern Cryptography, Second Edition. CRC Press: ISBN 9781466570269 (2014)
45. Dai W., Okamoto T., Yamamoto G.: Stronger Security And Generic Constructions For Adaptor Signatures. In Progress in Cryptology – INDOCRYPT 2022: 23rd International Conference on Cryptology in India, Kolkata, India, December 11–14, 2022, Proceedings: p. 52–77: Springer-Verlag, Berlin, Heidelberg: ISBN 978-3-031-22911-4: https://doi.org/10.1007/978-3-031-22912-1_3: URL https://doi.org/10.1007/978-3-031-22912-1_3 (2023)
46. Erwig A., Faust S., Riahi S., Stöckert T.: CommiTEE: An Efficient and Secure Commit-Chain Protocol using TEEs. Cryptology ePrint Archive, Report 2020/1486: https://eprint.iacr.org/2020/1486 (2020)
47. Badertscher C., Maurer U., Tschudi D., Zikas V.: Bitcoin as a transaction ledger: A composable treatment. In Annual International Cryptology Conference: pp. 324–356: Springer (2017)
48. Badertscher C., Gaži P., Kiayias A., Russell A., Zikas V.: Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security: pp. 913–930: ACM (2018)

49. Nick J., Ruffing T., Seurin Y.: MuSig2: Simple Two-Round Schnorr Multi-signatures. In Malkin and Peikert [54]: pp. 189–221: https://doi.org/10.1007/978-3-030-84242-0_8: URL https://doi.org/10.1007/978-3-030-84242-0_8 (2021)
50. Maxwell G., Poelstra A., Seurin Y., Wuille P.: Simple Schnorr multi-signatures with applications to Bitcoin. Des. Codes Cryptogr.: vol. 87(9), pp. 2139–2164: https://doi.org/10.1007/s10623-019-00608-x: URL https://doi.org/10.1007/s10623-019-00608-x (2019)
51. Nick J., Ruffing T., Seurin Y., Wuille P.: MuSig-DN: Schnorr Multi-Signatures with Verifiably Deterministic Nonces. In J. Ligatti, X. Ou, J. Katz, G. Vigna (editors), CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020: pp. 1717–1731: ACM: ISBN 978-1-4503-7089-9: https://doi.org/10.1145/3372297.3417236: URL https://doi.org/10.1145/3372297.3417236 (2020)
52. Komlo C., Goldberg I.: FROST: Flexible Round-Optimized Schnorr Threshold Signatures. In O. Dunkelman, M.J.J. Jr., C. O'Flynn (editors), Selected Areas in Cryptography - SAC 2020 - 27th International Conference, Halifax, NS, Canada (Virtual Event), October 21-23, 2020, Revised Selected Papers: vol. 12804 of *Lecture Notes in Computer Science*: pp. 34–65: Springer: ISBN 978-3-030-81651-3: https://doi.org/10.1007/978-3-030-81652-0_2: URL https://doi.org/10.1007/978-3-030-81652-0_2 (2020)
53. Garillot F., Kondi Y., Mohassel P., Nikolaenko V.: Threshold Schnorr with Stateless Deterministic Signing from Standard Assumptions. In Malkin and Peikert [54]: pp. 127–156: https://doi.org/10.1007/978-3-030-84242-0_6: URL https://doi.org/10.1007/978-3-030-84242-0_6 (2021)
54. Malkin T., Peikert C. (editors): Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I: vol. 12825 of *Lecture Notes in Computer Science*: Springer: ISBN 978-3-030-84241-3: https://doi.org/10.1007/978-3-030-84242-0: URL https://doi.org/10.1007/978-3-030-84242-0 (2021)

A Proof of theorem 1

We now provide the proof of Theorem 1. This is a direct result of Lemmas 1, 2, and 3, which we prove below.

Proof (Lemma 1). The proof consists mainly of a direct substitution of the definition of 2-AS correctness (Definition 2) with the provided 2-AS construction (Figure 5).

$$\begin{gathered}(sk, pk) \xleftarrow{\$} \mathsf{Gen}(1^k), ((Y_1, Y_2), (y_1, y_2)) \xleftarrow{\$} \mathsf{RGen}(1^k) \\ \widetilde{\sigma}_1 = \mathsf{pSign}(sk, m_1, Y_1), \widetilde{\sigma}_2 = \mathsf{pSign}(sk, m_2, Y_2), \\ \sigma_1 = \mathsf{Adapt}(\widetilde{\sigma}_1, y_1), \sigma_2 = \mathsf{Adapt}(\widetilde{\sigma}_2, y_2), \\ (y_1', y_2') \leftarrow (\mathsf{Ext}(\sigma_1, \widetilde{\sigma}_1, Y_1), \mathsf{Ext}(\sigma_2, \widetilde{\sigma}_2, Y_2))\end{gathered}$$

Due to the assumed correctness of the underlying AS scheme ([45], Figure 1, Right), it is:

$$\begin{gathered}
\mathsf{2\text{-}pSign}(sk, m_1, Y_1) = \mathsf{pSign}(sk, m_1, Y_1) = 1 \\
\wedge\ \mathsf{2\text{-}pSign}(sk, m_2, Y_2) = \mathsf{pSign}(sk, m_2, Y_2) = 1 \\
\wedge\ \mathsf{Vrfy}(pk, m_1, \sigma_1) = 1 \wedge \mathsf{Vrfy}(pk, m_2, \sigma_2) = 1 \\
\wedge (Y_1, y_1) \in \widetilde{\mathcal{R}} \wedge (Y_2, y_2) \in \widetilde{\mathcal{R}}
\end{gathered}$$

The relation $\widetilde{\mathcal{R}}$ is hard and thus the same witness can be generated by $\widetilde{\mathsf{R}}\mathsf{Gen}$ twice only with negligible probability in k — otherwise, a PPT adversary given a statement could simply generate a new pair and produce the desired secret witness with non-negligible probability in k. Therefore $y_1 \neq y_2$ with overwhelming probability in k, thus $((Y_1, Y_2), (y_1, y_2)) \in \mathcal{R}$. The proof is complete.

Proof (Lemma 2). This proof follows from two facts. Firstly, the fact that the signature verification of line 5 of Figure 2 does not return 1 with overwhelming probability stems from the adaptability of AS (according to the adaptability definition of [16]). Concretely, AS adaptability means that a single pre-signature can be adapted to a valid full signature given y where $(Y, y) \in \widetilde{\mathcal{R}}$ with overwhelming probability, which is the exact same requirement of line 5 in the $\mathsf{2\text{-}ADP\text{-}NEXT}$ game.

Secondly, the fact that the relation membership check of line 7 of Figure 2 does not return 1 with overwhelming probability stems from the definition of the relation $\mathcal{R}$, the hardness of the underlying relation $\widetilde{\mathcal{R}}$ and the fact that the challenger has adapted with only one of the two underlying witnesses. More specifically, we can reduce $\mathsf{2\text{-}ADP\text{-}NEXT}$ to the 1-one-wayness game of $\widetilde{\mathcal{R}}$ as defined in [45]: An adversary $\mathcal{A}$ that wins the $\mathsf{2\text{-}ADP\text{-}NEXT}$ game due to line 7 with non-negligible probability α can be used by an adversary $\mathcal{B}$ to win $\mathbf{G}^{\text{1-ow}}_{\widetilde{\mathcal{R}}}$ with probability $\alpha/2$. $\mathcal{B}$ is called with $Y \in L_{\widetilde{\mathcal{R}}}$. $\mathcal{B}$ samples (Y', y') with $\widetilde{\mathsf{R}}\mathsf{Gen}$, flips a fair coin and assigns $Y_1 \leftarrow Y, Y_2 \leftarrow Y'$ in case of heads, $Y_1 \leftarrow Y', Y_2 \leftarrow Y$ otherwise. $\mathcal{B}$ then passes (Y_1, Y_2) to $\mathcal{A}$ — since both Y and Y' are sampled from $\widetilde{\mathsf{R}}\mathsf{Gen}$, the input to $\mathcal{A}$ follows the same distribution as in $\mathsf{2\text{-}ADP\text{-}NEXT}$ and its output b is independent of $\mathcal{B}$'s coin flip. This means that $\mathcal{A}$ will request for the pre-signature to be adapted with y' half of the times, which $\mathcal{B}$ can do. In this scenario, if $\mathcal{A}$ wins the $\mathsf{2\text{-}ADP\text{-}NEXT}$ game, the element y^*_{3-b} of its output is a witness to Y, thus $\mathcal{B}$ can output it and win the $\mathbf{G}^{\text{1-ow}}_{\widetilde{\mathcal{R}}}$ game. In total, $\mathcal{B}$ wins $\mathbf{G}^{\text{1-ow}}_{\widetilde{\mathcal{R}}}$ with non-negligible probability $\alpha/2$ given an adversary $\mathcal{A}$ that wins $\mathsf{2\text{-}ADP\text{-}NEXT}$ with non-negligible probability α.

Proof (Lemma 3). The proof of this lemma follows via a reduction of $\mathsf{2\text{-}FEXT}$ to the $\mathbf{G}^{\text{fext}}$ of the AS scheme [45]. We now describe a concrete adversary $\mathcal{B}$ that plays the $\mathbf{G}^{\text{fext}}$ game and can simulate all queries made by the adversary of the $\mathsf{2\text{-}FEXT}$ game, $\mathcal{A}$. To this end, when called with input pk ([45], Figure 3, line 2) by the $\mathbf{G}^{\text{fext}}$ game, $\mathcal{B}$ first initializes an empty map $\mathcal{T}$ that maps messages to pre-signed statement/pre-signature pairs, an empty set of signed messages $\mathcal{Q}$, and an empty set of oracle-generated statements $\mathcal{C}$. It then forwards pk to $\mathcal{A}$ (Figure 4, line 2). Furthermore, $\mathcal{B}$ simulates the oracles $\mathcal{O}_{\mathsf{2pS}}, \mathcal{O}_{\mathsf{2S}}, \mathcal{O}_{\mathsf{2}\mathcal{R}}$ of 2-FEXT (which are made available to $\mathcal{A}$) by leveraging the respective oracles of $\mathbf{G}^{\text{fext}}$ as follows:

$\mathcal{O}_{\mathsf{2pS}}(m, Y \in L_{\widetilde{\mathcal{R}}})$: Invoke the pre-signing oracle of $\mathbf{G}^{\text{fext}}$ on m and obtain $\widetilde{\sigma}$ ($\widetilde{\sigma} \leftarrow \mathcal{O}_{\text{pSign}}(m,\, Y)$). Add $(Y, \widetilde{\sigma})$ to $\mathcal{T}(m)$, then output $\widetilde{\sigma}$.

$\mathcal{O}_{\mathsf{2S}}(m)$: Add m to $\mathcal{Q}$, then output the signature on m, which is obtained by the signing oracle of $\mathbf{G}^{\text{fext}}$, $\mathcal{O}_{\text{Sign}}(m)$.

$\mathcal{O}_{\mathsf{2}\mathcal{R}}()$: Obtain a new random statement by invoking the statement-producing oracle of $\mathbf{G}^{\text{fext}}$ twice ($Y_1 \leftarrow \mathcal{O}_{\text{NewY}}()$, $Y_2 \leftarrow \mathcal{O}_{\text{NewY}}$), add both to $\mathcal{C}$ and output the pair $(Y_1,\, Y_2)$.

Observe that the map and sets tracked by the simulated oracles contain the exact same elements with the corresponding map and sets of the $\mathbf{G}^{\text{fext}}$ oracles after every invocation. Finally, $\mathcal{B}$ chooses b from $\{1, 2\}$ uniformly at random and outputs (m^*_b, σ^*_b), where the output of $\mathcal{A}$ is $(m^*_1, m^*_2, \sigma^*_1, \sigma^*_2)$.

We can see that the inputs and oracle responses that $\mathcal{A}$ receives follow the same distribution as in a standalone 2-FEXT game, therefore its probability of winning its game (say, α) is maintained when used internally by $\mathcal{B}$ as described above.

Every time $\mathcal{A}$ wins 2-FEXT, the returned σ^*_1 and σ^*_2 are valid signatures for the returned m^*_1 and m^*_2 respectively (i.e., $\mathsf{Vrfy}(pk, m^*_1, \sigma^*_1) \wedge \mathsf{Vrfy}(pk, m^*_2, \sigma^*_2)$), it has not queried the messages to the oracles (i.e., $m^*_1, m^*_2 \notin \mathcal{Q}$) and yet it is impossible to use the forged signatures to extract a witness for any statement that was used to pre-sign with $\mathcal{O}_{\mathsf{2pS}}$ and was not produced by $\mathcal{O}_{\mathsf{2}\mathcal{R}}$ (i.e., ($\nexists\, Y_1$, Y_2, $\widetilde{\sigma}_1$, $\widetilde{\sigma}_2$: $(Y_1,\, \widetilde{\sigma}_1) \in \mathcal{T}(m^*_1) \wedge (Y_2,\, \widetilde{\sigma}_2) \in \mathcal{T}(m^*_2) \wedge Y_1 \notin \mathcal{C} \wedge Y_2 \notin \mathcal{C} \wedge ((Y_1, Y_2)$, $\mathsf{2\text{-}Ext}(\sigma^*_1,\, \sigma^*_2,\, \widetilde{\sigma}_1,\, \widetilde{\sigma}_2,\, Y_1,\, Y_2)) \in \mathcal{R}$). In that case, output (m^*_b, σ^*_b) of $\mathcal{B}$ wins $\mathbf{G}^{\text{fext}}$ at least half of the times: As we saw, the signature is valid ($\mathsf{Vrfy}(pk, m^*_b, \sigma^*_b)$ outputs 1), thus the assertion of line 3 of $\mathbf{G}^{\text{fext}}$ succeeds. Next, the message has not been queried to $\mathcal{O}_{\mathsf{2S}}$ (as $m^*_b \notin \mathcal{Q}$), thus it has not been queried to $\mathcal{O}_{\text{Sign}}$, thus the assertion of line 4 of $\mathbf{G}^{\text{fext}}$ succeeds as well. As for the alignment of $\mathcal{B}$'s output with the requirement of line 6 of $\mathbf{G}^{\text{fext}}$, we argue that if the requirement could fail no matter which b was chosen, then $\mathcal{A}$ would lose, which is a contradiction. Failure of $\mathcal{B}$'s requirement of line 6 for both possible values of b means that $\forall b \in \{1, 2\}, \exists Y_b, \widetilde{\sigma}_b : (Y_b, \widetilde{\sigma}_b) \in T[m^*_b] \wedge Y_b \notin C \wedge (Y_b, \mathsf{Ext}(Y_b, \widetilde{\sigma}_b, \sigma^*_b)) \in \widetilde{\mathcal{R}}$. Due to the definitions of $\mathcal{O}_{\mathsf{2pS}}$, $\mathcal{O}_{\mathsf{2}\mathcal{R}}$, and 2-Ext, this means that $\forall b \in \{1, 2\}, (Y_b, \widetilde{\sigma}_b) \in \mathcal{T}(m^*_b) \wedge Y_b \notin \mathcal{C} \wedge ((Y_1, Y_2), (\mathsf{Ext}(\widetilde{\sigma}_1, \sigma^*_1, Y_1), \mathsf{Ext}(\widetilde{\sigma}_2, \sigma^*_2, Y_2))) \in \mathcal{R}$, dicrectly contradicting the last requirement for $\mathcal{A}$ to win $\mathbf{G}^{\text{fext}}$.

Thus, at least one of the two b values results in $\mathcal{B}$ winning. Since b is selected independently, $\mathcal{B}$ wins its game with probability at least $\alpha/2$. This concludes the reduction.

B Bitcoin Clique Construction

Let $N \leftarrow |\mathcal{P}|$. Using $\mathcal{G}_{\text{Ledger}}$ [47,48] with parameter $\kappa = \texttt{windowSize}$ to model the blockchain, one of [49,50,51] (a.k.a. `MuSig`) for n-of-n multisignatures and one of [52,53] for t-of-n threshold signatures (abbreviated here as $\texttt{TSig}(t$, set of keys)). For example, the spending condition $\texttt{TSig}(2, \{pk_E, pk_F, pk_G\})$ can be spent by a transaction signed by any two out of sk_E, sk_F, sk_G. Spending conditions can be combined with a logical OR ($\vee$).

Let the *inclusion delay*, denoted with s, be a (derived) system parameter that specifies the maximum number of blocks between the submission of a tx and its inclusion on-chain. In detail, if P submits a valid $\texttt{tx}$ when her chain is of height h and if no competing tx is submitted, then $\texttt{tx}$ will be included in a block of height between $h+1$ and $h+s$ (inclusive). Note that s depends entirely on the details of $\mathcal{G}_{\text{Ledger}}$.

Definition 5 (Epoch update deadline). *The* epoch update deadline*, denoted with p, is the number of blocks within which protocol parties expect Op to update each epoch. In detail, consider an honest $P \in \mathcal{P}$ that sees a chain of height h' and an unspent $\texttt{tx}_{\mathsf{step}}$ with a timelock t that is included in the block of height h. If $h' \geq h + t + p$, then P assumes that Op is faulty and exits unilaterally.*

Note that p can be negotiated between the protocol parties. In this work we assume p is a fixed system parameter.

Definition 6 (Epoch update slack). *The* epoch update slack*, denoted with w, is the maximum number of blocks within which an honest Op must update each epoch. In detail, if Op sees an unspent $\texttt{tx}_{\mathsf{step},i}$ with a timelock t that is included in the block of height h, then she has to publish $\texttt{tx}_{\mathsf{step},i+2}$ (which spends $\texttt{tx}_{\mathsf{step},i}$) until block $h + t + w$ to avoid triggering the honest party unilateral exit of Def. 5.*

Lemma 4 (Epoch update). *Let s be the inclusion delay as described above, p be the epoch update deadline (Def. 5) and w be the epoch update slack (Def. 6). It is*

$$w = p - s - 1 \ .$$

Proof (Lemma 4). We here adopt the notation of Def. 6. If Op submits $\texttt{tx}_{\mathsf{step},i+2}$ when its block height is $h + t + w$, then it will be included at most in block of height $h+t+w+s$. In order to avoid triggering the honest party unilateral exits of Def. 5, it must be $h + t + w + s < h + t + p \Leftrightarrow w < p - s$. Since w is defined as the maximum value that prevents these exits, it is $w = p - s - 1$.

Definition 7 (End of epoch). *Let the* end-of-epoch buffer*, denoted with r, be a number of blocks. Let h be the block height in which the timelock of the second-latest $\texttt{tx}_{\mathsf{step},i}$ expires. The* end of an epoch *is the moment in which Op sees a chain of height $h - r$. An honest Op uses in $\texttt{txTree}_{i+2}$ the balances of the parties as they are calculated when taking into account all payments that have been signed off and sent by Op to all parties the end of the epoch.*

The end-of-epoch mechanism ensures that all parties have the chance to receive all payments and calculate the new $\texttt{txTree}$ before the new $\texttt{tx}_{\mathsf{step}}$ is added to a block. Like p, r can be negotiated between the protocol parties and should correspond to a timespan that is larger than the slowest possible roundtrip time between Op and any party. Once again we assume r is a fixed system parameter.

Process Notation table

- s, inclusion delay: Maximum blocks between tx submission and inclusion
- p, epoch update deadline: Blocks within which Op must update epoch
- t: Blocks per epoch
- $t_{\mathsf{leave},i}$: Block before which a user can exit if another user initiates exit during epoch i
- $t_{\mathsf{punish},i}$: Block before which Op can punish a malicious user that tries to take the same output from epochs $i-1$ and i, or i and $i+1$
- t_{rest}: Number of blocks Op delayed publishing the last $\mathtt{tx}_{\mathsf{step}}$. Used to correct block drift between epochs.

- $\mathtt{tx}_{\mathsf{step},i}$, step $\mathtt{tx}$of epoch i: On-chain tx that carries all Clique funds. Can be spent by $\mathtt{txTree}_i$ to initiate exit. One such tx is published every epoch by Op, spending $\mathtt{tx}_{\mathsf{step},i-2}$.
- $\mathtt{txTree}_i$: A binary tree of txs of epoch i. Its root spends $\mathtt{tx}_{\mathsf{step},i}$, has one leaf per user output.
- $\mathtt{tx}_{i,j}$: The j-th tx of the i-th level of the current $\mathtt{txTree}$. If it is a leaf, it corresponds to an output and is spendable by a simple and an adaptor signature by the current output owner, if this output has been transfered during the last epoch, it additionally needs a 2-adaptor signature by the previous output owner. The 2-adaptor signatures of two successive epochs for the same epoch leak to Op the secret needed to spend the $\mathtt{tx}_{\mathsf{out}}$ that spends this leaf tx. If it is not a leaf, it has a single input and is the only transaction that can spend one of the two outputs of its parent (via CTV). It also has 2 outputs that specify its children via CTV. If it is the root, it spends $\mathtt{tx}_{\mathsf{step}}$. Publishing the root tx starts the exit phase.
- $\mathtt{tx}_{\mathsf{out},i,j}$, "out" $\mathtt{tx}$ of epoch i and output j: Tx that spends a leaf tx. Can be spent by Op if it knows the punishing secret, or by the output owner after timelock t_{punish} elapses.

- (pk_A, sk_A): public-secret keypair. Same subscript means that sk_A can create signatures that are valid by pk_A.
- $pk_{\mathsf{2\text{-}AS},i,j,Op}$, Op's key for epoch i, output j: Op pre-signs "out" txs with this key.
- $pk_{\mathsf{2\text{-}AS},i,j,P,b}$, P's key for epoch i, output j, $b \in \{0,1\}$: this is the statement used when Op presigns "out" txs, alternating between $b=0$ and $b=1$ every epoch. Op can spend the "out" tx if it knows $sk_{\mathsf{2\text{-}AS},i,j,P,0}$ and $sk_{\mathsf{2\text{-}AS},i,j,P,1}$.
- $pk_{\mathsf{in},i,j}$, "in" key of epoch i and output j: Used to sign payments during normal operation.
- $pk_{\mathsf{mid},i,j}$, "mid" key of epoch i and output j: Can spend any non-leaf tx on the path to $\mathtt{tx}_{\mathsf{out},i,j}$.
- $pk_{\mathsf{out},i,j}$, "out" key of epoch i and output j: Allows the owner to spend the "out" tx after timelock t_{punish}.

- $pk^*_{\textsf{2-AS},i,j,Op}$: Op's key for epoch $i-1$, output j, useful only when the output changed hands in epoch $i-1$.
- $pk'_{\textsf{2-AS},i,j,P}$, P's punishing key for epoch i, output j: The sum of the two P's keys. Can be used by Op to punish P if the latter misbehaves.

- $P_{i,j}$: owner of output j at epoch i
- σ: signature
- $\widetilde{\sigma}$: pre-signature

Fig. 7. Bitcoin Clique construction – Notation table

Process Rules for constants

- $2s > p$
- $t \geq p + s$
- $\forall i \in \mathbb{N}, t_{\textsf{leave},i} > s$
- $\forall i \in \mathbb{N}, t_{\textsf{punish},i} > t_{\textsf{leave},i+1} + s$ (cf. all locations in which the two timelocks are calculated, namely ll. 16, 18 of Fig. 9, l. 14 of Fig. 12 and l. 19 of Fig. 13)

Fig. 8. Bitcoin Clique construction – Rules for constants

Process Bitcoin Clique construction – Security parameter k – Setup

```
Run by everyone:
 1: Agree with everyone on current block height. The current block height, $t_0$, is
    taken to be the minimum block height proposed by any party, as long as it is
    at least $t_{\textsf{us}} - \kappa$, where $t_{\textsf{us}}$ is the block height given by $\mathcal{G}_{\text{Ledger}}$ to ourselves // no
    honest party can have a chain more than $\kappa$ blocks shorter than ours, due to
    the $\mathcal{G}_{\text{Ledger}}$ guarantees
 2: for all $j \in [N]$ do
Run by $Op$:
 3:     $(sk_{\textsf{2-AS},1,j,Op}, pk_{\textsf{2-AS},1,j,Op}) \leftarrow (sk_{\textsf{2-AS},2,j,Op}, pk_{\textsf{2-AS},2,j,Op}) \leftarrow \textsf{Gen}(1^k)$
 4:     Send $pk_{\textsf{2-AS},1,j,Op}$ to all parties
Run by $P_j$:
 5:     $((pk_{\textsf{2-AS},1,j,P,0}, pk_{\textsf{2-AS},1,j,P,1}), (sk_{\textsf{2-AS},1,j,P,0}, sk_{\textsf{2-AS},1,j,P,1})) \leftarrow$
        $((pk_{\textsf{2-AS},2,j,P,0}, pk_{\textsf{2-AS},2,j,P,1}), (sk_{\textsf{2-AS},2,j,P,0}, sk_{\textsf{2-AS},2,j,P,1})) \leftarrow \textsf{RGen}(1^k)$
 6:     Send $(pk_{\textsf{2-AS},i,j,P,k})_{i \in \{1,2\}, k \in \{0,1\}}$ to all parties
 7: end for

Run by everyone:
 8: Exchange public keys $(pk_{\textsf{in},1,j} = pk_{\textsf{in},2,j})_{j \in [N]}$, $(pk_{\textsf{mid},1,j} = pk_{\textsf{mid},2,j})_{j \in [N]}$,
    $(pk_{\textsf{out},1,j} = pk_{\textsf{out},2,j})_{j \in [N]}$, $pk_{\textsf{cont},Op}$, and outpoints $(O^*_j)_{j \in [N]}$ with other parties
    and $Op$ // the last key subscript specifies key owner
```

```
Ensure that each outpoint $O^*_j$ corresponds to an on-chain unspent P2WPKH output owned by the public key $pk_{\mathsf{in},j}$ with $c$ coins // $c$ is the constant value of all outputs
for $j \in [N]$ do
    $pk_{x,j,y} \leftarrow pk_{x,i+1,j,y}$
    $pk_{\mathsf{2\text{-}AS},j,Op} \leftarrow pk_{\mathsf{2\text{-}AS},1,j,Op}$
    $pk^*_{\mathsf{2\text{-}AS},j,Op} \leftarrow \bot$
    $pk'_{\mathsf{2\text{-}AS},j,P} \leftarrow pk_{\mathsf{2\text{-}AS},1,j,P,0} + pk_{\mathsf{2\text{-}AS},1,j,P,1}$
end for
Let $O \leftarrow (O^*_j)_{j \in [N]}$, $t_{\mathsf{rest}} \leftarrow 0$, $t_{\mathsf{leave}} \leftarrow t_{\mathsf{leave},1} \leftarrow t_0 + t + s$, $t_{\mathsf{punish}} \leftarrow t_{\mathsf{punish},1} \leftarrow t_{\mathsf{leave}} + s + 1$ and obtain $\mathtt{tx}_{\mathsf{step},1}$, $\mathtt{txTree}_1$, $(\mathtt{tx}_{\mathsf{out},1,j})_{j \in [N]}$ according to Fig. 14

Run by $Op$:
Send to every $P \in \mathcal{P}$ an output $O'_2$ which has $Nc$ coins and does not spend any outpoint $O^*_j$

Run by everyone:
Let $O \leftarrow O'_2$, $t_{\mathsf{rest}} \leftarrow t$, $t_{\mathsf{leave}} \leftarrow t_{\mathsf{leave},2} \leftarrow t_0 + 2t + s$, $t_{\mathsf{punish}} \leftarrow t_{\mathsf{punish},2} \leftarrow t_{\mathsf{leave}} + s + 1$ and obtain $\mathtt{tx}_{\mathsf{step},2}$, $\mathtt{txTree}_2$, $(\mathtt{tx}_{\mathsf{out},2,j})_{j \in [N]}$ according to Fig. 14

Run by $Op$:
for all $j \in [N]$ do
    $\widetilde{\sigma}_{\mathsf{out},1,j} \leftarrow \mathsf{2\text{-}pSign}(sk_{\mathsf{2\text{-}AS},1,j,Op}, \mathtt{tx}_{\mathsf{out},1,j}, pk_{\mathsf{2\text{-}AS},1,j,P,0})$
    $\widetilde{\sigma}_{\mathsf{out},2,j} \leftarrow \mathsf{2\text{-}pSign}(sk_{\mathsf{2\text{-}AS},2,j,Op}, \mathtt{tx}_{\mathsf{out},2,j}, pk_{\mathsf{2\text{-}AS},2,j,P,1})$
    Send $\widetilde{\sigma}_{\mathsf{out},1,j}, \widetilde{\sigma}_{\mathsf{out},2,j}$ to $P_j$
end for

Run by $P_j \in \mathcal{P}$:
Ensure $\mathsf{2\text{-}pVrfy}(pk_{\mathsf{2\text{-}AS},1,j,Op}, \mathtt{tx}_{\mathsf{out},1,j}, \widetilde{\sigma}_{\mathsf{out},1,j}, pk_{\mathsf{2\text{-}AS},1,j,P,0}) = 1$ and $\mathsf{2\text{-}pVrfy}(pk_{\mathsf{2\text{-}AS},2,j,Op}, \mathtt{tx}_{\mathsf{out},2,j}, \widetilde{\sigma}_{\mathsf{out},2,j}, pk_{\mathsf{2\text{-}AS},2,j,P,1}) = 1$
$\sigma_{\mathsf{start},j} \leftarrow \mathsf{Sign}(sk_{\mathsf{in},j}, \mathtt{tx}_{\mathsf{step},1})$; send $\sigma_{\mathsf{start},j}$ to $Op$

Run by $Op$:
for all $j \in [N]$ do
    Ensure $\mathsf{Vrfy}(pk_{\mathsf{in},j}, \mathtt{tx}_{\mathsf{step},1}, \sigma_{\mathsf{start},j}) = 1$; add $\sigma_{\mathsf{start},j}$ to $\mathtt{tx}_{\mathsf{step},1}$
end for
$\mathsf{Sign}()$ $\mathtt{tx}_{\mathsf{step},1}$ with the key that spends $O'_2$ and add signature to $\mathtt{tx}_{\mathsf{step},2}$
Submit $\mathtt{tx}_{\mathsf{step},1}$ and $\mathtt{tx}_{\mathsf{step},2}$ to $\mathcal{G}_{\text{Ledger}}$

Run by $P_j \in \mathcal{P}$:
Wait for $\mathcal{G}_{\text{Ledger}}$ to be extended by $p$ blocks
while neither of $\mathtt{tx}_{\mathsf{out},1,j}$, $\mathtt{tx}_{\mathsf{out},2,j}$ is on $\mathcal{G}_{\text{Ledger}}$ do
    if both $\mathtt{tx}_{\mathsf{step},1}$ and $\mathtt{tx}_{\mathsf{step},2}$ are on $\mathcal{G}_{\text{Ledger}}$ then
        Break "while"
    else // at least one of $\mathtt{tx}_{\mathsf{step},1}$, $\mathtt{tx}_{\mathsf{step},2}$ not on $\mathcal{G}_{\text{Ledger}}$, unilateral exit
        if $\mathtt{tx}_{\mathsf{step},1} \notin \mathcal{G}_{\text{Ledger}}$ then respend $O^*_j$ // prevents future use of $\mathtt{tx}_{\mathsf{step},1}$
        if $\mathtt{tx}_{\mathsf{step},i} \in \mathcal{G}_{\text{Ledger}}$ then exit protocol as in Fig. 16 with $\mathtt{tx}_{\mathsf{step},i}$
    end if
end while
Assume the role of $P_j$ // setup successful, $P_j$ owns $O_j^*$

Run by everyone:
for all $i \in [2]$ do
    Let $h_i$ be the height of the block in which the timelock of $\texttt{tx}_{\mathsf{step},i}$ expires
end for
$e \leftarrow 1$ // $e$ is the current epoch number
```

Fig. 9. Bitcoin Clique construction – Setup

Process Bitcoin Clique construction – Op enters end-of-epoch buffer

```
Let $i$ be the minimum number such that $\texttt{tx}_{\mathsf{step},i}$ is unspent
Let $h$ be the current block height
if $h \geq h_i - r \wedge e \leq i$ then // epoch just changed
    Let $S$ be the set of party indexes that have transferred their output during
    epoch $e$ (i.e., all $j \in [N] : P_i$ ran Fig. 11)
    $\{P_{e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{P_{e+1,j}\}_{j \in [N] \setminus S}$
    $\{pk_{\mathsf{2\text{-}AS},e+2,j,P,k}\}_{j \in [N] \setminus S, k \in \{0,1\}} \leftarrow \{pk_{\mathsf{2\text{-}AS},e+1,j,P,k}\}_{j \in [N] \setminus S, k \in \{0,1\}}$
    $\{pk_{\mathsf{2\text{-}AS},e+2,j,Op}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{2\text{-}AS},e+1,j,Op}\}_{j \in [N] \setminus S}$
    $\{pk_{\mathsf{in},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{in},e+1,j}\}_{j \in [N] \setminus S}$
    $\{pk_{\mathsf{mid},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{mid},e+1,j}\}_{j \in [N] \setminus S}$
    $\{pk_{\mathsf{out},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{out},e+1,j}\}_{j \in [N] \setminus S}$
    Forget data from epoch $e - 1$ // optimization
    $e \leftarrow i + 1$ // if all goes well, before assignment it is $i = e$
end if
```

Fig. 10. Bitcoin Clique construction – Epoch end

Process Bitcoin Clique construction – P_j gives R its output o_j

We assume that, during the interactions of Fig. 11, no new blocks are added to the chain for all parties.

```
Run by $Op$, $P_j$ and $R$:
Update epoch if needed, as in Fig. 10 for $Op$ and Fig. 13 for $P_j$, $R$
Let $h$ be the current block height
if $h_e - r - \kappa \leq h \leq h_e - r + \kappa$ then return // no payments allowed within
    end-of-epoch buffer

Run by $Op$:
Send $e + 1$ to $P_j$ and $R$

Run by $P_j$:
```

```
Assign received value to $l'$
if $h < h_e - r - \kappa$ then ensure $l' = e + 1$
if $h_e - r + \kappa \leq h$ then ensure $l' = e + 2$
$m \leftarrow (j, R, l')$
$\sigma_P \leftarrow \mathsf{Sign}(m, sk_{\mathsf{in},l',j})$
When we reach epoch $l'$ (i.e., when we run l. 23 of Fig. 13 with $e + 1 = l'$),
send $\sigma^*_{\mathsf{out},l'+1,j} \leftarrow \mathsf{2\text{-}Adapt}(pk_{\mathsf{2\text{-}AS},l',j,Op}, \widetilde{\sigma}_{\mathsf{out},l'+1,j}, sk_{\mathsf{out},l',j,P,(l'+1) \bmod 2})$ to $R$.
Then, keeping the rest of our current roles, give up the role of $P_j$. // We just
noticed that $\mathtt{tx}_{\mathsf{step},l'+1}$ is on-chain. This is the moment when the outgoing
payment is committed.
Send $(m, \sigma_P)$ to $R$
```

Run by R:

```
Assign value received by $Op$'s message to $l'$
Ensure $m = (j, R, l')$
if $\mathsf{Vrfy}(pk_{\mathsf{in},l',j}, m, \sigma_P) = 0$ then return
When we reach epoch $l'$, (i.e., when we run l. 24 of Fig. 13 with $e + 1 = l'$),
wait for $\sigma^*_{\mathsf{out},l'+1,j}$ from $P_j$ and ensure that
$\mathsf{Vrfy}(pk_{\mathsf{out},l',j,Op}, \mathtt{tx}_{\mathsf{out},l'+1,j}, \sigma^*_{\mathsf{out},l'+1,j}) = 1$. Then, additionally to our current
roles, also adopt the role of $P_i$ // We just noticed that $\mathtt{tx}_{\mathsf{step},l'+1}$ on-chain.
This is the moment when the incoming payment is committed.
$((pk_{\mathsf{2\text{-}AS},l',j,P,0}, pk_{\mathsf{2\text{-}AS},l',j,P,1}), (sk_{\mathsf{2\text{-}AS},l',j,P,0}, sk_{\mathsf{2\text{-}AS},l',j,P,1})) \leftarrow \mathsf{RGen}(1^k)$
$(sk_{\mathsf{in},l',j}, pk_{\mathsf{in},l',j}) \leftarrow \mathsf{Gen}(1^k)$; $(sk_{\mathsf{mid},l',j}, pk_{\mathsf{mid},l',j}) \leftarrow \mathsf{Gen}(1^k)$;
$(sk_{\mathsf{out},l',j}, pk_{\mathsf{out},l',j}) \leftarrow \mathsf{Gen}(1^k)$
Send $(pk_{\mathsf{2\text{-}AS},l',j,P,0}, pk_{\mathsf{2\text{-}AS},l',j,P,1}, pk_{\mathsf{in},l',j}, pk_{\mathsf{mid},l',j}, pk_{\mathsf{out},l',j}, m, \sigma_P)$ to $Op$
```

Run by Op:

```
Parse message by $R$ as
$(pk_{\mathsf{2\text{-}AS},e+1,j,P,0}, pk_{\mathsf{2\text{-}AS},e+1,j,P,1}, pk_{\mathsf{in},e+1,j}, pk_{\mathsf{mid},e+1,j}, pk_{\mathsf{out},e+1,j}, m, \sigma_P)$ and $m$ as
$(j, R', l')$
if $\mathsf{Vrfy}(pk_{\mathsf{in},e+1,j}, m, \sigma_P) = 0 \vee l' = e + 1 \vee R' \neq R$ then return // ensure
sender is payee
Let $P_{e+1,j} \leftarrow R$
$(sk_{\mathsf{2\text{-}AS},e+1,j,Op}, pk_{\mathsf{2\text{-}AS},e+1,j,Op}) \leftarrow \mathsf{Gen}(1^k)$
Append $pk_{\mathsf{2\text{-}AS},l',j,Op}$, $pk_{\mathsf{2\text{-}AS},e+1,j,P,0}$, $pk_{\mathsf{2\text{-}AS},e+1,j,P,1}$, $pk_{\mathsf{in},e+1,j}$,
$pk_{\mathsf{mid},e+1,j}, pk_{\mathsf{out},e+1,j}$ and $e + 1$ to $m$
$\sigma_{Op} \leftarrow \mathsf{Sign}(m, sk_{\mathsf{cont},Op})$
Send $(m, \sigma_P, \sigma_{Op})$ to every $P \in \mathcal{P}$
```

Run by $P \in \mathcal{P}$:

```
try:
Verify epoch update if needed, as in Fig. 13
    Parse $m$ as $(j,\ P_{l',j},\ pk_{\mathsf{2\text{-}AS},l',j,Op},\ pk_{\mathsf{2\text{-}AS},l',j,P,0},\ pk_{\mathsf{2\text{-}AS},l',j,P,1},\ pk_{\mathsf{in},l',j},$
    $pk_{\mathsf{mid},l',j},\ pk_{\mathsf{out},l',j}, l')$
    Ensure $\mathsf{Vrfy}(pk_{\mathsf{cont},Op}, m, \sigma_{Op}) = \mathsf{Vrfy}(pk_{\mathsf{in},j}, (j, R), \sigma_P) = 1$
    Let $h$ be the current block height
    if $h < h_e - r - \kappa$ then ensure $l' = e + 1$
    if $h_e - r - \kappa \leq h \leq h_e - r + \kappa$ then ensure $l' \in \{e + 1, e + 2\}$
    if $h_e - r + \kappa < h$ then ensure $l' = e + 2$
```

```
34: if any of ll. 28-33 fail then exit as in Fig. 16 with $\texttt{tx}_{\mathsf{step},e+1}$
35: if $P = P_j$ then mark output $o_j$ as not owned by self // payment complete
```

Fig. 11. Bitcoin Clique construction – Payment during normal operation

Process Bitcoin Clique construction – Op updates epoch

```
Let $i$ be the maximum integer such that $\texttt{tx}_{\mathsf{step},i}$ is on-chain
Let $h$ be our view of the current block height
Ensure that the timelock of $\texttt{tx}_{\mathsf{step},i-1}$ has expired // i.e., that $h \geq h_{i-1}$
// It should be $h \leq h_{i-1} + w$ to prevent honest party unilateral exit
for $j \in [N]$ do
    $pk_{x,j,y} \leftarrow pk_{x,i+1,j,y}$
    if $P_{i+1,j} = P_{i,j}$ then // Output not transferred during previous epoch
        $pk^*_{\mathsf{2\text{-}AS},j,Op} \leftarrow \bot$
    else // Output transferred during previous epoch
        $pk^*_{\mathsf{2\text{-}AS},j,Op} \leftarrow pk_{\mathsf{2\text{-}AS},i,j,Op}$
    end if
    $pk'_{\mathsf{2\text{-}AS},j,P} \leftarrow pk_{\mathsf{2\text{-}AS},i+1,j,P,0} + pk_{\mathsf{2\text{-}AS},i+1,j,P,1}$ // "+" is the public key group operation
end for
Let $O \leftarrow \texttt{tx}_{\mathsf{step},i-1}.\texttt{output}$, $t_{\mathsf{rest}} \leftarrow h_i - h$, $t_{\mathsf{leave}} \leftarrow t_{\mathsf{leave},i+1} \leftarrow h_i + t + s$, $t_{\mathsf{punish}} \leftarrow t_{\mathsf{punish},i+1} \leftarrow t_{\mathsf{leave}} + s + 1$ and obtain $\texttt{tx}_{\mathsf{step},i+1}$, $\texttt{txTree}_{i+1}$, $(\texttt{tx}_{\mathsf{out},i+1,j})_{j \in [N]}$ according to Fig. 14 // using $t_{\mathsf{rest}} = h_i - h$ to correct any time drift
for all $j \in [N]$ do
    $\widetilde{\sigma}_{\mathsf{out},i+1,j} \leftarrow \mathsf{2\text{-}pSign}(sk_{\mathsf{2\text{-}AS},i+1,j,Op}, \texttt{tx}_{\mathsf{out},i+1,j}, pk_{\mathsf{2\text{-}AS},i+1,j,P,(i+1) \bmod 2})$
    Send $\widetilde{\sigma}_{\mathsf{out},i+1,j}$ to $P_j$
end for
Add $\mathsf{Sign}(\texttt{tx}_{\mathsf{step},i+1}, sk_{\mathsf{cont},Op})$ to $\texttt{tx}_{\mathsf{step},i+1}.\texttt{input}$
Submit $\texttt{tx}_{\mathsf{step},i+1}$ to $\mathcal{G}_{\text{Ledger}}$
Let $h_{i+1}$ the height of the block in which the timelock of $\texttt{tx}_{\mathsf{step},i+1}$ expires
Send $h_{i+1}$ to every $P \in \mathcal{P}$
```

Fig. 12. Bitcoin Clique construction – Move to next epoch

Process Bitcoin Clique construction – $P_k \in \mathcal{P}$ verifies epoch update

```
Let $h$ be our view of the current block height
if $h > h_e + t - s$ then mark ourselves as negligent and return
Ensure $h \geq h_e + p$ // give time to $Op$ to spend $\texttt{tx}_{\mathsf{step},e}$
try:
    Ensure we have received $h_{e+2}$ by $Op$
```

```
6:      Ensure $h_{e+1} + t + 1 \leq h_{e+2} \leq h_{e+1} + t + s$ // ensure next epoch will expire within prescribed window
7:      Ensure that at least one output of $\texttt{tx}_{\mathsf{step},e}$ with $Nc$ coins is spent
8:      Assign the spending tx to $\texttt{tx}_{\mathsf{step},e+2}$
9:      Assign the block height of $\texttt{tx}_{\mathsf{step},e+2}$ to $h'_{e+2}$
10:     for $j \in [N]$ do
11:         $pk_{x,j,y} \leftarrow pk_{x,e+2,j,y}$
12:         if $P_{e+2,j} = P_{e+1,j}$ then // Output not transferred during previous epoch
13:             $pk^*_{\mathsf{2\text{-}AS},j,P} \leftarrow \bot$
14:         else // Output transferred during previous epoch
15:             $pk^*_{\mathsf{2\text{-}AS},j,P} \leftarrow pk_{\mathsf{2\text{-}AS},e+1,j,P,(e+2) \mod 2}$
16:         end if
17:         $pk'_{\mathsf{2\text{-}AS},j,P} \leftarrow pk_{\mathsf{2\text{-}AS},e+2,j,P,0} + pk_{\mathsf{2\text{-}AS},e+2,j,P,1}$ // "+" is the public key group operation
18:     end for
19:     Let $O \leftarrow \texttt{tx}_{\mathsf{step},e}.\texttt{output}$, $t_{\mathsf{rest}} \leftarrow h_{e+2} - h'_{e+2} - t$, $t_{\mathsf{leave}} \leftarrow t_{\mathsf{leave},e+2} \leftarrow h_{e+1} + t + s$, $t_{\mathsf{punish}} \leftarrow t_{\mathsf{punish},e+2} \leftarrow t_{\mathsf{leave}} + s + 1$ and obtain $\texttt{tx}_{\mathsf{step}}$, $\texttt{txTree}_{e+2}$, $(\texttt{tx}_{\mathsf{out},e+2,j})_{j \in [N]}$ according to Fig. 14
20:     Ensure $\texttt{tx}_{\mathsf{step}} = \texttt{tx}_{\mathsf{step},e+2}$
21:     Ensure $\texttt{txTree}_{e+2}.\texttt{root}$ can spend an output of $\texttt{tx}_{\mathsf{step},e+2}$ with $Nc$ coins
22:     Ensure we have received $\widetilde{\sigma}_{\mathsf{out},e+2,k}$ and that $\mathsf{pVrfy}(pk_{\mathsf{2\text{-}AS},e+2,k,Op}, \texttt{tx}_{\mathsf{out},e+2,k}, \widetilde{\sigma}_{\mathsf{out},e+2,k}, pk_{\mathsf{2\text{-}AS},e+2,k,P,(e+2) \mod 2}) = 1$
23:     Commit any pending outgoing payments according to Fig. 11, l. 10 // commit outgoing before incoming to avoid deadlocks
24:     Wait at most until block height is up to $h_e + t - s$ for the commitment of any pending incoming payments according to Fig. 11, l. 15
25: if any of ll. 5-24 fail then exit as in Fig. 16 with $\texttt{tx}_{\mathsf{step},e+1}$
26: Let $S$ be the set of party indexes that have transferred their output during epoch $e$ (i.e., all $j \in [N] : P_j$ ran Fig. 11 and we received $m = (j, \_, \_, \_, e)$ of l. 28)
27: $\{P_{e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{P_{e+1,j}\}_{j \in [N] \setminus S}$
28: $\{pk_{\mathsf{2\text{-}AS},e+2,j,P,k}\}_{j \in [N] \setminus S, k \in \{0,1\}} \leftarrow \{pk_{\mathsf{2\text{-}AS},e+1,j,P,k}\}_{j \in [N] \setminus S, k \in \{0,1\}}$
29: $\{pk_{\mathsf{2\text{-}AS},e+2,j,Op}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{2\text{-}AS},e+1,j,Op}\}_{j \in [N] \setminus S}$
30: $\{pk_{\mathsf{in},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{in},e+1,j}\}_{j \in [N] \setminus S}$
31: $\{pk_{\mathsf{mid},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{mid},e+1,j}\}_{j \in [N] \setminus S}$
32: $\{pk_{\mathsf{out},e+2,j}\}_{j \in [N] \setminus S} \leftarrow \{pk_{\mathsf{out},e+1,j}\}_{j \in [N] \setminus S}$
33: Forget data from epoch $e - 1$ // optimization
34: $e \leftarrow e + 1$
```

Fig. 13. Bitcoin Clique construction – Epoch update verification

Process Generate step and tree transactions

Build a complete binary tree $\mathtt{txTree}$ of $N-1$ transactions, each with 1 input and 2 outputs. The i-th tx of the j-th layer is denoted with $\mathtt{tx}_{j,i}$. It is $j \in [\lceil \log_2 N \rceil]$ and $i \in [2^{j-1}]$ when $j \in [\lceil \log_2 N \rceil - 1]$, or $i \in [N - 2^{\lceil \log_2 N \rceil - 1}]$ when $j = \lceil \log_2 N \rceil$. Also build a step transaction that is spendable by the root transaction of $\mathtt{txTree}$ and one "out" transaction for each non-$\mathtt{TSig}$ output in $\mathtt{txTree}$ (i.e., for each leaf tx). $pk'_{\mathsf{2\text{-}AS},x,P}$ keys are used for enabling punishment of P by Op. $pk^*_{\mathsf{2\text{-}AS},x,Op}$ keys are used in outputs that have been just transferred and belong to the previous owner. All $pk^*_{\mathsf{2\text{-}AS},x,Op}$ keys that are equal to $\perp$ correspond to outputs that have not been transferred and are omitted from the script.
The notation $pk + t$ is used for relative timelocks, while $pk \wedge t$ is used for absolute timelocks.

- There are $\lfloor \frac{N}{2} \rfloor$ leaf transactions, $v := N - 2^{\lceil \log_2 N \rceil - 1}$ in the last level and $u := 2^{\lceil \log_2 N \rceil - 1} - \lceil \frac{N}{2} \rceil$ in the second-last level.
  - Last level leaf and "out" txs: For each $i \in [v]$, the "out" tx of party P_{2i-1} ($\mathtt{tx}_{\mathsf{out},2i-1}$) has a single $(c, (pk_{\mathsf{cont},Op} \wedge pk'_{\mathsf{2\text{-}AS},2i-1,P}) \vee (pk_{\mathsf{out},2i-1,P} \wedge t_{\mathsf{punish}}))$ output. Likewise the "out" tx of party P_{2i} ($\mathtt{tx}_{\mathsf{out},2i}$) has a single $(c, (pk_{\mathsf{cont},Op} \wedge pk'_{\mathsf{2\text{-}AS},2i,P}) \vee (pk_{\mathsf{out},2i} \wedge t_{\mathsf{punish}}))$ output. Also the i-th tx of the last level $\mathtt{tx}_{\lceil \log_2 N \rceil,i}$ has 2 outputs, namely $(c, (pk_{\mathsf{2\text{-}AS},2i-1,Op} \wedge pk^*_{\mathsf{2\text{-}AS},2i-1,Op} \wedge \mathsf{CTV}(\mathtt{tx}_{\mathsf{out},2i-1})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$ and $(c, (pk_{\mathsf{2\text{-}AS},2i,Op} \wedge pk^*_{\mathsf{2\text{-}AS},2i,Op} \wedge \mathsf{CTV}(\mathtt{tx}_{\mathsf{out},2i})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$.
  - If N is odd, then the "out" tx of party P_{2v+1} ($\mathtt{tx}_{\mathsf{out},2v+1}$) has a single $(c, (pk_{\mathsf{cont},Op} \wedge pk'_{\mathsf{2\text{-}AS},2v+1,P}) \vee (pk_{\mathsf{out},2v+1,P} \wedge t_{\mathsf{punish}}))$ output. The second output of the $\frac{v+1}{2}$-th tx of the second-last level ($\mathtt{tx}_{\lceil \log_2 N \rceil - 1, \frac{v+1}{2}}$) is $(c, (pk_{\mathsf{2\text{-}AS},2v+1,Op} \wedge pk^*_{\mathsf{2\text{-}AS},2v+1,Op} \wedge \mathsf{CTV}(\mathtt{tx}_{\mathsf{out},2v+1})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$.
  - Second-last level leaf and "out" txs: For each $i \in \{\lceil \frac{v}{2} \rceil + 1, \ldots, 2^{\lceil \log_2 N \rceil - 2}\}$, the "out" tx of party P_{2i+v-1} ($\mathtt{tx}_{\mathsf{out},2i+v-1}$) has a single $(c, (pk_{\mathsf{cont},Op} \wedge pk'_{\mathsf{2\text{-}AS},2i+v-1,P}) \vee (pk_{\mathsf{out},2i+v-1,P} \wedge t_{\mathsf{punish}}))$ output. Likewise the "out" tx of party P_{2i+v} ($\mathtt{tx}_{\mathsf{out},2i+v}$) has a single $(c, (pk_{\mathsf{cont},Op} \wedge pk'_{\mathsf{2\text{-}AS},2i+v,P}) \vee (pk_{\mathsf{out},2i+v,P} \wedge t_{\mathsf{punish}}))$ output. Also the i-th tx of the second-last level $\mathtt{tx}_{\lceil \log_2 N \rceil - 1,i}$ has 2 outputs, namely $(c, (pk_{\mathsf{2\text{-}AS},2i+v-1,Op} \wedge pk^*_{\mathsf{2\text{-}AS},2i+v-1,Op} \wedge \mathsf{CTV}(\mathtt{tx}_{\mathsf{out},2i+v-1})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$ and $(c, (pk_{\mathsf{2\text{-}AS},2i+v,Op} \wedge pk^*_{\mathsf{2\text{-}AS},2i+v,Op} \wedge \mathsf{CTV}(\mathtt{tx}_{\mathsf{out},2i+v})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$.
- Second-last level non-leaf txs: For each $i \in [\lfloor \frac{v}{2} \rfloor]$, the i-th tx of the second-last level $\mathtt{tx}_{\lceil \log_2 N \rceil - 1,i}$ has 2 outputs, namely $(2c, (\mathsf{CTV}(\mathtt{tx}_{\lceil \log_2 N \rceil,2i-1}) \wedge \mathtt{TSig}(1, \{pk_{\mathsf{mid},4i-3}, pk_{\mathsf{mid},4i-2}\})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$ and $(2c, (\mathsf{CTV}(\mathtt{tx}_{\lceil \log_2 N \rceil,2i}) \wedge \mathtt{TSig}(1, \{pk_{\mathsf{mid},4i-1}, pk_{\mathsf{mid},4i}\})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$.
- Non-leaf txs of remaining levels: For each j from $\lceil \log_2 N \rceil - 2$ to 1, $i \in [2^{j-1}]$, the i-th tx of the j-th level $\mathtt{tx}_{j,i}$ has 2 outputs, namely $(\sum_{k=1}^{2} \mathtt{tx}_{j+1,2i-1}.\mathtt{outputs}[k].\mathtt{value}, (\mathsf{CTV}(\mathtt{tx}_{j+1,2i-1}) \wedge$ $\mathtt{TSig}(1, \bigcup_{k=1}^{2} \mathtt{tx}_{j+1,2i-1}.\mathtt{outputs}[k].\mathtt{TSigPubkeys})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$ and

$(\sum_{k=1}^{2} \texttt{tx}_{j+1,2i}.\texttt{outputs}[k].\texttt{value}, (\mathsf{CTV}(\texttt{tx}_{j+1,2i}) \wedge$
$\texttt{TSig}(1, \bigcup_{k=1}^{2} \texttt{tx}_{j+1,2i}.\texttt{outputs}[k].\texttt{TSigPubkeys})) \vee (pk_{\mathsf{cont},Op} \wedge t_{\mathsf{leave}}))$.

- Every tx has $\texttt{nLockTime} = 0$, $\texttt{nVersion} = 2$ and 1 input with $\texttt{nSequence} = \texttt{0xffffffff}$, empty $\texttt{scriptSig}$ and empty $\texttt{witness}$. // setting $\texttt{nLockTime} = 0$ avoids having to sync with other parties on which is the current block, but does not help prevent fee sniping[a]
- Step tx $\texttt{tx}_{\mathsf{step}} \leftarrow$
  - input(s): O
  - output: $(Nc, (\mathsf{CTV}(\texttt{tx}_{1,1}) \wedge \texttt{TSig}(1, \bigcup_{k=1}^{2} \texttt{tx}_{1,1}.\texttt{outputs}[k].\texttt{TSigPubkeys})) \vee$ $(pk_{\mathsf{cont},Op} + (t_{\mathsf{rest}} + t))$
- The input of the root tx spends the unique output of $\texttt{tx}_{\mathsf{step}}$ (i.e., has $\texttt{prevout} = \mathcal{H}(\texttt{tx}_{\mathsf{step}}) + \texttt{"/0"}$).
- For each j from 2 to $\lceil \log_2 N \rceil - 1$, for each $i \in [2^{j-1}]$, as well as for $j = \lceil \log_2 N \rceil$ and for each $i \in [N - 2^{\lceil \log_2 N \rceil - 1}]$ the input of the i-th tx of the j-th level $\texttt{tx}_{j,i}$ spends the $2 - (i \mod 2)$-th output of $\texttt{tx}_{j-1,\lceil i/2 \rceil}$ (i.e., has $\texttt{prevout} = \mathcal{H}(\texttt{tx}_{j-1,\lceil i/2 \rceil}) + \texttt{"/1} - (i \mod 2)\texttt{"}$).
- For each $i \in [v]$ each of the 2 corresponding "out" txs ($\texttt{tx}_{\mathsf{out},n_{2i-1}}$ and $\texttt{tx}_{\mathsf{out},n_{2i}}$) has a single input that spends the first and the second output of $\texttt{tx}_{\lceil \log_2 N \rceil,i}$ respectively (i.e., with $\texttt{prevout} = \mathcal{H}(\texttt{tx}_{\lceil \log_2 N \rceil,i}) + \texttt{"/0"}$ and $\mathcal{H}(\texttt{tx}_{\lceil \log_2 N \rceil,i}) + \texttt{"/1"}$ respectively).
- If N is odd, then the "out" tx of party $P_{n_{2v+1}}$ ($\texttt{tx}_{\mathsf{out},n_{2v+1}}$) has a single input that spends the second output of $\texttt{tx}_{\lceil \log_2 N \rceil - 1, \frac{v+1}{2}}$ (i.e., with $\texttt{prevout} = \mathcal{H}(\texttt{tx}_{\lceil \log_2 N \rceil - 1, \frac{v+1}{2}}) + \texttt{"/0"}$).
- For each $i \in \{\lceil \frac{v}{2} \rceil + 1, \dots, 2^{\lceil \log_2 N \rceil - 2}\}$ each of the 2 corresponding "out" txs ($\texttt{tx}_{\mathsf{out},n_{2i+v-1}}$ and $\texttt{tx}_{\mathsf{out},n_{2i+v}}$) has a single input that spends the first and the second output of $\texttt{tx}_{\lceil \log_2 N \rceil - 1,i}$ respectively (i.e., with $\texttt{prevout} = \mathcal{H}(\texttt{tx}_{\lceil \log_2 N \rceil - 1,i}) + \texttt{"/0"}$ and $\mathcal{H}(\texttt{tx}_{\lceil \log_2 N \rceil - 1,i}) + \texttt{"/1"}$ respectively).
- **return** $\texttt{tx}_{\mathsf{step}}$, the set of all $\texttt{tx}_{j,i}$ and the set of all $\texttt{tx}_{\mathsf{out},i}$

[a] https://bitcoinops.org/en/topics/fee-sniping/

Fig. 14. Step and Tree transactions generation

Process $\texttt{myTXs}(\texttt{txTree}, P_j)$

```
v ← $2^{\lceil \log_2 N \rceil - 1}$
res ← $\emptyset$
if $j \leq 2(N - v)$ then
    for $i$ from 1 to $\lceil \log_2 N \rceil$ do
        add $\texttt{tx}_{\lceil \log_2 N \rceil + 1 - i, \lceil \frac{j}{2^i} \rceil}$ of txTree to res
    end for
else // $j > 2(N - v)$
```

```
    for $i$ from 1 to $\lceil \log_2 N \rceil - 1$ do
        add $\mathtt{tx}_{\lceil \log_2 N \rceil - i, \lceil \frac{j-(N-v)}{2^i} \rceil}$ of $\mathtt{txTree}$ to $\mathtt{res}$
    end for
end if
return $\mathtt{res}$
```

Fig. 15. $\mathtt{myTXs}(\mathtt{txTree}, P_i)$

Process $P_j \in \mathcal{P}$ exits unilaterally via $\mathtt{tx}_{\mathsf{step},i}$

Run when P_j is instructed by $\mathcal{E}$ to exit

```
1: if no $\mathtt{tx}_{\mathsf{step},i}$ is specified then set it to $\mathtt{tx}_{\mathsf{step},e+1}$
2: Sign with $sk_{\mathsf{mid},j}$ all transactions in $\mathtt{myTXs}(\mathtt{txTree}_i, P_j)$ that are not on $\mathcal{G}_{\text{Ledger}}$
   // $P_j$ claims coins with at most $O(\log N)$ on-chain txs, each of size $O(1)$
3: Add $\mathsf{2\text{-}Adapt}(pk_{\mathsf{2\text{-}AS},i,j,Op}, \widetilde{\sigma}_{\mathsf{out},i,j}, sk_{\mathsf{2\text{-}AS},i,j,P,i \bmod 2})$ to $\mathtt{tx}_{\mathsf{out},i,j}$
4: if $P_{i-1,j} \neq P_{i,j}$ /*received $j$-th output at epoch $i$*/ then add $\sigma^{*}_{\mathsf{out},i,j}$ to
   $\mathtt{tx}_{\mathsf{out},i,j}$
5: Submit to $\mathcal{G}_{\text{Ledger}}$ all txs signed in ll. 2-4
6: Halt execution of role $P_j$ // ensures we cannot reuse $sk_{\mathsf{2\text{-}AS},i,j,P,i \bmod 2}$, which
   would reveal it to $Op$, giving her our coins
```

Fig. 16. Unilateral (emergency) exit for party

Process $P \in \mathcal{P}$ responds to Unilateral Exit

If a $\mathtt{tx}_{\mathsf{step}}$ is spent by the root transaction of the corresponding $\mathtt{txTree}$, then exit as in Fig. 16 with this $\mathtt{tx}_{\mathsf{step}}$

Fig. 17. Claim of a party's coins when unilateral exit has started by other party

Process Op exits unilaterally

```
Let $\mathtt{tx}_{\mathsf{step},i}$, $\mathtt{tx}_{\mathsf{step},i+1}$ be the two step txs that have not been spent by a step tx
for $\mathtt{tx} \in \{\mathtt{tx}_{\mathsf{step},i}, \mathtt{tx}_{\mathsf{step},i+1}\}$ do
    Wait for timelock of $\mathtt{tx}$ to expire
    Move all coins from $\mathtt{tx}$ to own key
    if moving fails then run Fig. 19 for $\mathtt{tx}$ // failure only happens if $P \in \mathcal{P}$
    spends $\mathtt{tx}$ with the corresponding root tx
end for
```

Fig. 18. Unilateral (emergency) exit for operator

```
Process Op responds to Unilateral Exit
if we have not run Fig. 19 for $t_{\mathsf{punish}} - s - \kappa$ blocks then mark ourselves as negligent and return
if $\mathtt{tx}_{\mathsf{step},i}$ is spent by the root transaction of the corresponding txTree then
    move every txTree output with an expired timelock to own key // Every tx in every txTree can be spent by $Op$ after a timelock
    for each $i, j \in \mathbb{N}$ such that the leaf transaction of the $j$-th party of $\mathtt{txTree}_i$ is spent using a signature $\sigma_1$ that has not been generated by us do
        if no tuple $(\sigma_2, pk_{\mathsf{2\text{-}AS},i,j,P,(i+1) \bmod 2}, \mathtt{tx}_2, i')$ is stored locally then
            store $(\sigma_1, pk_{\mathsf{2\text{-}AS},i,j,P,i \bmod 2}, \mathtt{tx}_1, i)$ locally
        else // a tuple $(\sigma_2, pk_{\mathsf{2\text{-}AS},i,j,P,(i+1) \bmod 2}, \mathtt{tx}_2, i')$ is stored locally
            for each output $o_3$ spendable by $pk_{\mathsf{2\text{-}AS},i,j,Op} \wedge (pk_{\mathsf{2\text{-}AS},i,j,P,i \bmod 2} + pk_{\mathsf{2\text{-}AS},i,j,P,(i+1) \bmod 2})$ do // "+" is the public key group operation
                Retrieve corresponding pre-signatures $\widetilde{\sigma}_1 = \widetilde{\sigma}_{\mathsf{out},i,j}, \widetilde{\sigma}_2 = \widetilde{\sigma}_{\mathsf{out},i',j}$ // generated in Fig. 9 l. 20/l. 21 or Fig. 12 l. 16
                Build a tx that spends $o_3$ using this spending method and transfers coins to own key and sign it with $sk_{\mathsf{2\text{-}AS},i,j,Op}$
                Add $\mathsf{Sign}(\sum \mathsf{2\text{-}Ext}(\sigma_1, \sigma_2, \widetilde{\sigma}_1, \widetilde{\sigma}_2, pk_{\mathsf{2\text{-}AS},i,j,P,0}, pk_{\mathsf{2\text{-}AS},i,j,P,1}), \mathtt{tx})$ to tx // "$\sum$" is the secret key group operation
                Publish tx to $\mathcal{G}_{\text{Ledger}}$
            end for
        end if
    end for
end if
```

Fig. 19. Claim of operator's coins when unilateral exit has started by a party

C Bitcoin Clique Security

Lemma 5 (Setup security). *For an honest, non-negligent $P_i \in \mathcal{P}$ that runs the setup of the protocol, at least one of the following will be true:*

- *It will own output O_i,*
- *It will be able to exclusively spend the output of $\mathtt{tx}_{\mathsf{out},1,i}$ or $\mathtt{tx}_{\mathsf{out},2,i}$, one of which will be on-chain,*
- *It will be able to exclusively spend output O_i^*,*
- *It will exclusively own the output of the transaction that spent O_i^*.*

Note that all aforementioned outputs carry c coins.

Proof (Lemma 5). The relevant logic is in Fig. 9. If any of the agreement of l. 1, exchanges of ll. 4, 6, 8, 17, 22 or the checks of ll. 9, 24 fail, then P_i will stop the protocol, never executing l. 25 and thus not losing ownership of O_i^*, a situation that matches the third Lemma case.

In case the condition of l. 36 succeeds, then P_i tries to respend O_i^* – if this succeeds then the situation matches the fourth Lemma case. The only way in which this can fail is if $\mathtt{tx}_{\mathsf{step},1}$ spends O_i^*, since, other than the respending one, the only transaction that can spend O_i^* and that P_i has ever signed is $\mathtt{tx}_{\mathsf{step},1}$. In case $\mathtt{tx}_{\mathsf{step},1}$ wins the race but $\mathtt{tx}_{\mathsf{step},2}$ is not on-chain, then the condition of l. 37 triggers for $\mathtt{tx}_{\mathsf{step},1}$. Due to the same argument made in the proof of Lemmas 6 and 7 as to why P_i can always consume the output of a valid $\mathtt{tx}_{\mathsf{step},k}$ and produce a $\mathtt{tx}_{\mathsf{out},k,i}$ of which the output is exclusively spendable by P_i, the exit of l. 37 will succeed. This situation corresponds to the second case of the Lemma.

Lastly, if P_i runs the setup subprotocol of Fig. 9 to completion, then it owns O_i^* by definition (l. 40), a situation that corresponds to the first Lemma case.

Lemma 6 (Leaf tx on-chain). *If an honest, non-negligent P that owns output O is instructed by $\mathcal{E}$ to exit, then eventually at least one output by a leaf $\mathtt{tx}$ which can be spent by P's $\mathtt{tx}_{\mathsf{out}}$ will be included on-chain.*

Proof (Lemma 6). If P has already halted when it receives the exit instruction, then P has already executed l. 6 of Fig. 16, since this is the only location in which the protocol can halt. The $\mathtt{tx}_{\mathsf{step},j}$ chosen in l. 1 of Fig. 16 is on-chain and has previously been validated by P in ll. 20-22 of Fig. 13 – this can be proven by induction, as failure to validate $\mathtt{tx}_{\mathsf{step},j}$ would have triggered an exit with an earlier, already validated $\mathtt{tx}_{\mathsf{step},k}, k < j$, as in l. 25, Fig. 13 and the base case is guaranteed by Lemma 5, first bullet if P has owned O since the beginning of the protocol or by the fact that, if P has received O after protocol start, P checks then the validity of the then-current step tx (Fig. 11, l. 1). Then l. 2 of Fig. 16 signs all transactions in the path between $\mathtt{tx}_{\mathsf{step},j}$ and $\mathtt{tx}_{\lceil \log N \rceil - y, \lfloor \frac{i}{2} \rfloor}$ for $y = \chi_{>N-2^{\lceil \log_2 N \rceil - 1}}(i)$, where i is the party's index, and subsequently submits them to $\mathcal{G}_{\text{Ledger}}$ (Fig. 16, l. 5). As one can verify by inspection of Figs. 14 and 15, each of these transactions except for the last one can be spent by the next one without a timelock (due to the CTV rules of non-leaf txs) with a signature by $sk_{\mathsf{mid},i}$.

We will now show that the timelocked spending option of $\mathtt{tx}_{\mathsf{step},j}$ or of any tx of the aforementioned path (the ones that are not encumbered with a CTV) cannot be used. If a party other than P has already put the root tx on-chain, then the race against the timelock of $\mathtt{tx}_{\mathsf{step},j}$ has been already won, likewise for any tx on the path. If P is exiting because it ran l. 37 of Fig. 9, then the root tx needs until block $t_0 + p + s$ to be included (cf. Fig. 9, l. 31), whereas the timelock of $\mathtt{tx}_{\mathsf{step},j}$ expires at block $t_0 + t + 1$ at the earliest (i.e., in case $j = 1$ and $\mathtt{tx}_{\mathsf{step},j}$ is included in the first block after the protocol starts). Since $t \geq p + s$ (Fig. 8), the root tx will always enter the chain before the timelock expires. Regarding any tx on the path, it is submitted at most at block $t_0 + p$ together with all txs connecting it to $\mathtt{tx}_{\mathsf{step},j}$ and thus it will be included by $t_0 + p + s$, which is before the expiration of the timelock $t_{\mathsf{leave},j}$ ($t_{\mathsf{leave},1} = t_0 + t + s > t_0 + p + s$ and $t_{\mathsf{leave},2} = t_0 + 2t + s > t_0 + p + s$ cf. ll. 16, 18 of Fig. 9 and Fig. 8).

In case P is exiting because it ran l. 34 of Fig. 11, P has already updated its epoch (Fig. 11, l. 1 or l. 27 depending on whether P is implicated in the output transfer or not) and no blocks have been mined since then (cf. assumption of Fig. 11), thus the current block height is at most $h_{e-1} + t - s$. We furthermore can see that $h_{e+1} \geq h_e + t + 1 \geq h_{e-1} + 2t + 2$ (Fig. 13, l. 6 and Fig. 9, ll. 16, 18), therefore the root tx is published in time to be on-chain before h_{e+1}. Regarding the txs on the path, they are also published at most at block $h_e + t - s$. The timelocks of those txs are all $t_{\mathsf{leave},e+1}$. If $e = 1$ then $t_{\mathsf{leave},2} = t_0 + 2t + s$ (l. 18, Fig. 9) and the txs are published at block $h_1 + t - s$ at the latest (l. 2, Fig. 13). $h_1 \leq t_0 + p + t$ (ll. 16, 31, Fig. 9), thus they need at most until block $t_0 + p + 2t - s$ to be included on-chain. Indeed it is $t_0 + p + 2t - s < t_0 + 2t + s = t_{\mathsf{leave},2}$ (cf. Fig. 8), so all path txs will be included on-chain. If $e > 1$, then $t_{\mathsf{leave},e+1} = h_e + t + s$ (l. 14, Fig. 12) and P publishes the path txs at block $h_e + t - s$ at the latest. Since $t_{\mathsf{leave},e+1} = h_e + t + s > h_e + t$, all path txs will be on-chain before any of their timelocks expire.

In case P is exiting because it ran l. 25 of Fig. 13, then the block height when running l. 25 is at most $h_e + t - s$ (Fig. 13, ll. 2 and 24). Since $h_{e+1} \geq h_e + t + 1$, there are at least s blocks until the timelock of $\mathtt{tx}_{\mathsf{step},e+1}$ expires, thus the root tx will always enter the chain before expiration of their timelock. Similarly to the line of reasoning of the previous paragraph, path txs will also be submitted by block $h_e + t - s$ and $t_{\mathsf{leave},e+1} = h_e + t + s > h_e + t$, thus all path txs will be on-chain before any of their timelocks expire.

Therefore either P's transactions will be included in time or they will lose the race to identical transactions signed by other parties (identity is guaranteed due to the aforementioned CTV constraint).

If P is still active when it receives the exit instruction then it runs Fig. 16 right away. The timelock of $\mathtt{tx}_{\mathsf{step},e+1}$ has not expired, since P, being non-negligent, has run its last epoch update verification (Fig. 13) when the block height was at most $h_{e-1} + t - s$ (Fig. 13, l. 2 – the e in that line is one less than the e used here, because e is incremented at the end of Fig. 13), thus the current block height is at most $h_e + t - s$. We furthermore can see that $h_{e+1} \geq h_e + t + 1$ (Fig. 13, l. 6 and Fig. 9, ll. 16, 18), therefore the root tx is published in time to be on-chain before h_{e+1}. The rest of the analysis above applies to this case as well, therefore in every case P's leaf tx will be included on-chain, proving the Lemma.

Lemma 7 (Out tx on-chain). *If P is honest and non-negligent and at least one P's leaf $\mathtt{tx}$ output is on-chain, then P will sign and broadcast exactly one $\mathtt{tx}_{\mathsf{out}}$ that spends said output, which will eventually be included in the chain.*

Proof (Lemma 7). Let i be P's index and $\mathtt{tx}_{\mathsf{step},j}$ the tx chosen in l. 1 of Fig. 16, which is necessarily the most recent ancestor of the leaf tx of the hypothesis. We can check that $\mathtt{tx}_{\mathsf{out},j,i}$ spends the leaf tx by inspection of Fig. 14. Furthermore, as we saw in the proof of Lemma 6, the $\mathtt{tx}_{\mathsf{step},j}$ that is the ancestor of the leaf tx has been validated by P and the validation of the corresponding epoch has not failed in any other check since P has not exited with an earlier $\mathtt{tx}_{\mathsf{step},k}$, $k < j$, therefore the check of Fig. 13, l. 22, verifying that a correct $\widetilde{\sigma}_{\mathsf{out},j,i}$ has been received, had succeeded. We thus deduce that the Adapt of Fig. 16, l. 3 succeeds. Due to the pre-signature adaptability property of the underlying $\mathsf{2\text{-}AS}$ scheme (Def. 3), it is guaranteed that the output of Adapt is a valid signature on $\mathtt{tx}_{\mathsf{out},j,i}$ with respect to $pk_{\mathsf{2\text{-}AS},j,i,Op}$. Furthermore, in case P did not own the i-th output during epoch $j-1$ (in other words, if P did not fulfill the role of P_i during epoch $j-1$) it must have adopted this role for the first time for epoch j. This can only happen in Fig. 11, l. 15. The same line ensures that P has received a valid signature $\sigma^*_{\mathsf{out},j,i}$ on $\mathtt{tx}_{\mathsf{out},j,i}$ with respect to $pk_{\mathsf{out},j-1,i,Op}$. Therefore P will publish a valid $\mathtt{tx}_{\mathsf{out},j,i}$ transaction, spending the leaf tx. The same arguments made in the proof of Lemma 6 as to why the timelocked spending method of the leaf tx will not be used can be made here as well, therefore $\mathtt{tx}_{\mathsf{out},j,i}$ will be included in the chain.

Lastly, P will not publish a second "out" tx, since it halts right after publishing the first valid "out" tx (Fig. 16, l. 6) and only the P_i role stores the secret keys needed to create and complete the valid signatures needed for such a transaction. The proof of Lemma 7 is complete.

Lemma 8 (Timelock expiry). *If P is honest and non-negligent and exactly one of P's $\mathtt{tx}_{\mathsf{out}}$ is on-chain, then the timelock will expire (i.e., the non-timelocked spending method will not be used) therefore P exclusively owns the output on-chain.*

Proof (Lemma 8). Let $\mathtt{tx}_{\mathsf{out},j,i}$ be the transaction of the Lemma hypothesis. Its non-timelocked spending method needs a signature σ' valid w.r.t. the key $pk_{\mathsf{2\text{-}AS},j,i,P,0} + pk_{\mathsf{2\text{-}AS},j,i,P,1}$ (Fig. 12, l. 12). P, who generates $sk_{\mathsf{2\text{-}AS},j,i,P,0}$ and $sk_{\mathsf{2\text{-}AS},j,i,P,1}$ (Fig. 9, l. 5 or Fig. 11, l. 16) and keeps them secret, never produces such a signature. However there is a single way in which $sk_{\mathsf{2\text{-}AS},j,i,P,k}$, $k \in \{0,1\}$ can be leaked: If P publishes a signature σ produced by completing a pre-signature $\widetilde{\sigma}$ of which the "relation statement" Y is $sk_{\mathsf{2\text{-}AS},j,i,P,k}$. If one knows the public key pk that corresponds to the secret key with which the pre-signature was created, they can get $sk_{\mathsf{2\text{-}AS},j,i,P,k}$ as the output of $\mathsf{Ext}(pk, \mathtt{tx}_{\mathsf{out},j,i}, \sigma, \widetilde{\sigma})$. Indeed, by publishing $\mathtt{tx}_{\mathsf{out},j,i}$ P publishes such a signature for $k = j \mod 2$. Nevertheless, since P only publishes a single "out" tx, the other necessary secret, $sk_{\mathsf{2\text{-}AS},j,i,P,1-(j \mod 2)}$, will not be leaked in this way. Due to the unforgeability of Schnorr AS [16], $sk_{\mathsf{2\text{-}AS},j,i,P,1-(j \mod 2)}$ cannot be leaked in any other way with overwhelming probability. The unforgeability of the underlying Digital Signatures scheme prevents any other way of producing a valid signature σ'. Therefore no signature σ' can be created, so the timelock of $\mathtt{tx}_{\mathsf{out},j,i}$ will expire and its output will be exclusively spendable by P.

Proof (Theorem 2). Since $P \in \mathcal{P}$ assumes (Fig. 11, l. 15/Fig. 9, l. 40) or gives up (Fig. 11, l. 10) the role of the owner of the i-th output when it receives or offers it respectively, we only need to prove the theorem for P_i that owns only the i-th output. This is a direct result of Lemmas 6, 7 and 8.

Proof (Theorem 3). If either move of Fig. 18, l. 4 succeeds, then the theorem is satisfied, as each of these outputs carries coins equal to the required sum.

If neither move succeeds, then both $\texttt{tx} \in \{\texttt{tx}_{\textsf{step},i}, \texttt{tx}_{\textsf{step},i+1}\}$ have been spent by $\texttt{txTree}_i.\texttt{root}$ and $\texttt{txTree}_{i+1}.\texttt{root}$ respectively. Due to the $\textsf{CTV}$ rules (Fig. 14), parties other than Op can only spend the root tx using txs from the respective $\texttt{txTree}$. We will prove that for each $j \in [N]$, Op will obtain at least c coins. If for some $k \in \{i, i+1\}$, Op moves to its own key an output of a tx of $\texttt{txTree}_k$ that is the ancestor of $\{\texttt{txTree}_k.\texttt{tx}_{\textsf{out},l} | l \in \{a, \dots, b\}\}$ but not of $\{\texttt{txTree}_k.\texttt{tx}_{\textsf{out},l} | l \in [N] \setminus \{a, \dots, b\}\}$ using the timelocked spending method (Fig. 19, l. 3), then Op obtains $(b - a + 1)c$ coins, i.e., c coins per party of which the out tx has the aforementioned spent output as an ancestor. If on the other hand both $\texttt{txTree}_i.\texttt{tx}_{\textsf{out},j}$ and $\texttt{txTree}_{i+1}.\texttt{tx}_{\textsf{out},j}$ are on-chain, then Op can get the signature $\sigma_1 = \sigma_{\textsf{2-AS},i,j}$ from $\texttt{txTree}_i.\texttt{tx}_{\textsf{out},j}$ and the signature $\sigma_2 = \sigma_{\textsf{2-AS},i+1,j}$ if the j-th output has not changed owners between epochs i and $i+1$ or $\sigma_2 = \sigma^*_{\textsf{2-AS},i+1,j}$ else from $\texttt{txTree}_{i+1}.\texttt{tx}_{\textsf{out},j}$. These two signatures are valid with respect to Op's "ots" keys (cf. Fig. 14), but Op has only ever produced pre-signatures with $pk_{\textsf{2-AS},k,j,P,k \mod 2}$ as statements for $\texttt{txTree}_k.\texttt{tx}_{\textsf{out},j}, k \in \{i, i+1\}$ respectively, therefore the two signatures can only be the result of P_j adapting the pre-signatures with $pk_{\textsf{2-AS},k,j,P,k \mod 2}$ with overwhelming probability and thus, due to the extractability property of the $\textsf{2-AS}$ scheme, Op will extract with overwhelming probability a pair of secret keys that correspond to the public keys $pk_{\textsf{2-AS},i,j,P,0}$ and $pk_{\textsf{2-AS},i,j,P,1}$ (Fig. 19, l. 11), the sum of which is the key needed to use the non-timelocked spending method of $\texttt{tx}_{\textsf{out},i,j}$, in l. 11 of Fig. 19. Additionally, it is impossible for the timelock of $\texttt{tx}_{\textsf{out},i,j}$ to have expired for the next s blocks after $\texttt{tx}_{\textsf{out},i+1,j}$ enters the chain from the point of view of Op, since $t_{\textsf{punish},i} > t_{\textsf{leave},i+1} + s$ (Fig. 8). Therefore the transaction published by Op in l. 12 of Fig. 19 will enter the blockchain with overwhelming probability, giving Op the c coins that correspond to P_j.

D Bitcoin Clique Healing

In its previously described form, Bitcoin Clique is vulnerable to a DoS attack: When the exit phase is initiated by any user, the entire Clique is torn down for everyone. We here propose an extension to the protocol, named *healing*, which allows active users to reinstate the Clique securely with minimal on-chain overhead.

At a high level, healing works by enabling a new way to spend tree txs which needs the active participation of all relevant users and Op. After some users exit, some tree tx outputs remain unspent. The users that want to stay in the Clique collaborate with each other and with Op to create a single transaction that spends all remaining tree tx outputs using the new spending method and produces a suitable $\textsf{step}$ tx output. The protocol is resilient to inactive users.

D.1 Healing extension details

In more detail, the solution is as follows: Consider an output of an arbitrary tree tx, which is spendable by the subset of users $\mathcal{T} \subset \mathcal{P}$. We add an alternative spending method, named *healing*, to the tree tx. Its script is $\bigwedge_{P \in \mathcal{T}} P \wedge Op$. This modification is done to every tree tx of every epoch.

$s + 1$ blocks after an exit phase is initiated, a user P that wishes to keep its coins in the Clique first initializes $\mathcal{C} \subset \mathcal{P}$ as the set of users that have not exited (i.e., the users of whom the out tx is not on-chain) and then repeats the following steps until either healing is complete (step 2) or the need for P to exit arises (discussed after the healing steps).

1. Generate and sign a new step tx that spends all currently unspent tree outputs using the *healing* spending method and has a single output with the coins and script of a step tx for users $\mathcal{C}$ (with the same b as the step tx that was exited from). See also Fig. 14. If the current block is within the epoch update period (Fig. 12) of the exited-from step tx, then produce the successor to the exited-from step tx instead (i.e., produce the step tx that would spend the exited-from step tx, two epochs later). Gossip signatures with other users and Op.
2. Wait for $t_{\mathsf{reconcile}}$ blocks (a system-wide parameter, discussed in D.2). If all users in $\mathcal{C}$ and Op sign the new step tx as well within this period, then publish it to the ledger. Healing is complete.
3. Else:
   (a) Remove from $\mathcal{C}$ the users that have not provided the aforementioned signature.
   (b) Publish to the ledger the minimum set of tree txs on the path from the root to P's leaf so that all users that can spend the resulting tree output are in $\mathcal{C}$. (This action ostracizes inactive users on P's path.)
   (c) Wait for $s + 1$ blocks (giving time to our and other branches to finalize on-chain).
   (d) Remove from $\mathcal{C}$ all users that can spend an unspent tx tree output that can also be spent by a user in $\mathcal{P} \setminus \mathcal{C}$. (This action ostracizes users that did not ostracize inactive users on other paths by following step 3b. This is needed because the healing spending method needs the signature of all relevant users.)

The procedure needs to be repeated potentially many times because previously active users may become uncooperative in the process.

The need for P to exit arises if the new step tx has not been published by block $t_{\mathsf{leave}} - s$. In that case, P exits by publishing its branch of the tx tree and out tx as usual. This scenario can happen if Op becomes malicious and does not sign the new step tx, or if the other users maliciously classify P as inactive and do not include its tree output in the step tx. This, together with the fact that all relevant users (including P) need to sign for the healing spending method to be used and the fact that P only uses it to return to a normal step tx, guarantees that the healing extension safeguards balance security.

Op follows the same procedure as the users, apart from step 3b. Since its signature is needed for all healing spending methods and it only uses it to return to a normal step tx, operator balance security is guaranteed.

It is possible for the protocol to be executed on both active step txs simultaneously — balance security and healing are maintained.

D.2 Discussion and Future Work

Note that $t_{\mathsf{reconcile}}$ does not appear in any timelock, as it only dictates off-chain communication timeouts. It could therefore be alternatively expressed in terms of time. We here however express $t_{\mathsf{reconcile}}$ in terms of blocks for homogeneity of notation. We recommend using the shortest $t_{\mathsf{reconcile}}$ value that ensures each user has enough time to do a communication round-trip with every other user.

During healing, users might end up being too quick to assume another user is inactive and publish a tree tx that is not strictly needed. This incurs unneeded on-chain fees. A practical system would need to experiment with concrete parameters to minimize such events while promoting quick healing. Users are encouraged to be online and share as many signatures as possible as early and widely as possible to minimize such events, as well as being Bitcoin peers with each other in order to minimize discrepancies in their ledger views. To further mitigate this effect, it is possible to design a more elaborate synchronization protocol that allows users that were erroneously assumed inactive in step 3d to be re-included in the set of active users during the subsequent signature gossip step 1. We leave this as future work.

The above shows that this is a best-effort mechanism and does not benefit from uniquely attributable faults, which would in turn enable exclusion of malicious users from the healed Clique. There are specific cases in which it is possible to uniquely attribute faults, such as when a user publishes the root tx and no subsequent tree tx. We leave detecting and punishing uniquely attribute faults as future work.

Nevertheless, the healing mechanism can save a lot of on-chain transactions in many realistic scenarios of DoS attempts and always leads to reinstating and continuing the Clique with all honest, active users irrespective of the number of malicious users if Op is honest and network delays are bounded.

Let us give us two example scenarios: In case a single user unilaterally exits and everyone else cooperates, then the on-chain footprint is $\log_2(N)$ transactions of the tree, 1 out tx, and 1 healing step tx. On the other hand, if at least one user of each leaf tx is malicious and publishes its entire branch of the tx tree, but not its out tx, then healing results in putting the entire tree tx on-chain and then recreating the exact same step tx output that was initially spent, for a total of $2N$ on-chain txs. The latter is the worst case scenario. We observe that even in this case, honest users can still successfully heal.

In a practical deployment, Op can facilitate the protocol by being the primary point of contact for users and leveraging its (presumably) better network connection to enhance coordination, collect and distribute signatures, and signal which users are inactive. Still, users must not rely solely on Op for message passing, lest they want to give it the ability to suppress an honest, active user.