arXiv:1905.11360v1 [cs.DC] 27 May 2019

# BRICK: Asynchronous State Channels

Georgia Avarikioti
zetavar@ethz.ch
ETH Zürich

Eleftherios Kokoris Kogias
eleftherios.kokoriskogias@epfl.ch
EPFL

Roger Wattenhofer
wattenhofer@ethz.ch
ETH Zürich

## ABSTRACT

Off-chain protocols (layer 2) are a promising solution to the scalability and privacy challenges of blockchain systems. In off-chain channels, the core idea is that state changes between two complying parties need not be published. Instead, on-chain computation is only required when the parties are in dispute; in this case the blockchain takes the role of a trusted third party. Current proposals, however, require a synchronous network to preserve the safety of the channel, leaking to an adversary the exact amount of time needed to control the network for an attack. This problem is exacerbated when lifting the assumption of a perfect blockchain substrate, as low chain quality (*i.e.* censorship probability) or congestion can break the assumption that dispute resolution transactions will appear on-chain in time even if the network is perfect.

In this paper, we introduce BRICK, the first off-chain construction that remains secure under full asynchrony. The core idea is to incorporate the conflict resolution process within the off-chain channel by introducing a watchtower committee. Every state update is consistently broadcast to the committee after approval. Hence if a party wants to unilaterally close a channel it can only get the committee's approval for the valid state. Furthermore, we consider the permissioned model of blockchains where the additional property of audibility might be desired for regulatory purposes and introduce BRICK+ an off-chain construction that can provide auditability on top of BRICK without conflicting with its privacy guarantees. We formally define the properties our state-channel construction should fulfill, prove them for BRICK and BRICK+, and design incentives for the committee such that honest and rational behavior align.

## 1 INTRODUCTION

As every node of a permissionless blockchain must learn and validate every single transaction, the throughput is often too low. So far, the most promising approach to solve the throughput bottleneck are so-called *channels* [17, 44, 50]. The idea is that any two parties that interact (often) with each other will set up a joint channel. Instead of sending payments between the two parties to the blockchain, the two parties will simply accumulate their cryptographically signed payments. The blockchain will only be used to settle a conflict between the two parties, or to close the channel if it is not needed anymore. In other words, after establishing a channel, all transactions between the two parties will happen *off-chain*. The blockchain will only be used as a fail-safe mechanism in case of disputes.

The security guarantees of a channel are ensured by the on-chain dispute handling mechanism. To briefly explain the mechanism, each party in a channel maintains a local view of the most recent channel state, with signatures. If a party aborts (or provides invalid data), the counter party must publish the most recent signed state to the blockchain to initiate the dispute process. To handle the case where a malicious party initiates a spurious dispute, the other party is given a fixed time period to resolve the dispute. This is effective as long as honest parties remain online and responsive; a node that crashes or goes offline for an extended period of time may miss the time window to participate in the dispute process. This design, however, has a significant shortcoming as it assumes a perfect blockchain that will correctly handle disputes [28]. Unfortunately, this is not a valid assumption. Malicious parties can try to censor transactions[1] from an honest party during the dispute period [38]. Proposals such as Monitors [20], Watchtowers [1, 5] or Custodians [37] do not help in such a case as they can suffer the exact same censoring as the honest client.

In this work we propose BRICK, a novel state channel construction that does not rely on any assumption for the delivery of messages to be secure. As a result it can guarantee the correctness of the channels even under censorship or execution fork attacks [29]. The core idea of BRICK is to enable the participants of the payment channel to outsource the dispute arbitration to an external committee (*e.g.* a group of watchtowers). As a result, in BRICK, if there is a dispute, the committee will make sure the correct state is the only one available for submission on-chain regardless of the amount of time it takes to make this final state visible. Additionally, our construction protects from inactive committees in the cases that the parties agree as they can exit the channel in consensus without the approval of the committee.

A secondary shortcoming of channels is that they forfeit auditability of the updates in the channel in order to provide some weak notion of privacy [28], which makes channels unsuitable for any kind of regulated process [53] such as supporting a real currency. In this work, we resolve this problem with BRICK+, by further leveraging the committees to provide audibility of channels. We construct the channel's state update to form an internal hash-chain which the committee stores. In order to preserve privacy, state updates are hashed and only then presented to the committee. Essentially, the committee maintains a hash which is the head of the hash-chain of the state history. In order to provide accountability for the auditor we require that the auditor posts the access request on-chain [31]. Only then will the committee provide the auditor the necessary metadata to verify the state history he will receive from the parties of the channel.

To evaluate our channel construction we initially define properties that a channel solution should have to be consistent with the blockchain guarantees, specifically safety and liveness. Additionally, our construction protects the privacy of the channel's parties from external adversaries that are not authorized to access it. Finally, we study the reward allocation for the external

---

[1] This censoring ability is encompassed by the chain-quality property [26] of blockchain systems which is rightly bound to the synchrony of the network.

committee and present an incentive mechanism that ensures the honest behavior of rational committee members, which in turn ensures that our payment channel construction is secure. Similarly, we show how to align the incentives of rational participants with our expected honest behavior.

In summary, this paper makes the following contributions:

- We introduce BRICK, the first off-chain construction that remains secure under asynchrony and offline channel participants.
- We introduce BRICK+, a modification on BRICK channel construction, that enables external auditors to lawfully request access to the channels history while maintaining privacy.

## 2 BACKGROUND AND PRELIMINARIES

In this section we present the necessary introduction to state channels that are designed to scale blockchains. Furthermore, we introduce the necessary distributed abstractions and cryptographic primitives that our constructions builds upon.

### 2.1 Blockchain Scalability & Layer 2

One of the major problems of blockchain protocols is the limited transaction throughput that is associated with the underlying consensus mechanism, originally introduced in Bitcoin [40]. Nakamoto consensus demands that every participant of the system verifies and stores a replica of the entire history of transactions, *i.e.* the blockchain, to guarantee the safety and liveness of the transaction ledger. However, this requirement leads to blockchain systems with limited block size and block creation time; if we increase size or decrease time, we implicitly enforce participants to verify and store more data which in turn leads to centralization and additional advantages for participants that invest in more infrastructure. Thus, blockchain systems that use the Nakamoto or similar consensus mechanisms, face a scalability problem. Particularly, Bitcoin handles at most seven transactions per second [14] while current digital monetary systems, such as Visa, handle tens of thousands.

Proposed solutions for the throughput limitation of blockchain systems can be categorized in two groups: on-chain solutions that attempt to create faster blockchain protocols [4, 16, 32, 33, 49], and off-chain solutions that use the blockchain only as a fail-safe mechanism and move the transaction load offline, where the bottleneck is the network speed. While on-chain solutions lead to the design of new promising blockchain systems, they typically require stronger trust assumptions and they are not applicable to existing blockchain systems (without a hard fork). In contrast, off-chain (layer 2) solutions are built on top of the consensus layer of the blockchain and operate independently. Essentially, off-chain solutions allow two parties[2] to create a "channel" on the blockchain through which they can transact fast and secure; this solution is known as *payment channel*, originally introduced by Spilman [50], and made bidirectional by [17, 44].

Payment channels allow transactions between two parties to be executed instantly off-chain while maintaining the guarantees of the blockchain. Essentially, the underline blockchain acts as a "judge" in case of fraud. There are multiple proposals on how to construct payment channels [15, 17, 44, 50], but all proposals share the same core idea: the two parties create a joint account on the blockchain (funding transaction), and every time the parties want to make a transaction they update the distribution of the capital between them accordingly and they both sign the new transaction as if it would be published on the blockchain (update transaction). To close the channel, a party publishes the latest update transaction either unilaterally or in collaboration with the counter-party with a closing transaction.

The various proposals differ in the way they handle disputes, *i.e.* the case where one of the parties misbehaves and attempts to close the channel with a transaction that is not the latest update transaction, and as such violating the safety property. Lightning channels [44] penalize the misbehaving party by assigning the money of the channel to the counter-party in case of fraud. To achieve this, every time an update transaction is signed, each party gives a signed transaction to the counter-party that enables the counter-party to claim the money of the channel in case the party publishes the previous update transaction, as some form of breach remedy. However, this transaction is valid only for a window of time, since the party should, in case of no fraud, eventually be able to spend his money from the channel. This dispute period is enforced with a (relative) timelock. On the other hand, Duplex channels [17] guarantee that the latest update transaction will become valid before any previous update transaction, again utilizing timelocks. In both cases, the liveness of the underline blockchain and timelocks are crucial to the safety of the payment channel solution. Additionally, both solutions require online participants that constantly monitor the blockchain to ensure safety.

To extend the concept of payment channels on an arbitrary state, *state channels* were introduced [39]. Several recent constructions exist in this direction [13, 22, 39]. In contrast with these works, we focus on designing a safe state channel protocol that is asynchronous, *i.e.* does not require timelocks, which can be used later as a *brick* in any of these constructions.

### 2.2 Consistent Broadcast

Consistent broadcast [45] is a distributed protocol run by a node that wants to reliably send a message to a set of peers. It is called consistent because it guarantees that if a correct peer delivers a message $m$ with sequence number $s$ and another correct peer delivers message $m'$ with sequence number $s$, then $m = m'$. Thus, the sender cannot equivocate. In other words, the protocol maintains consistency among the actually delivered messages with the same sender and sequence numbers, but makes no provisions that any parties do deliver the messages. In our system we only care about consistency of sequence numbers, as any party of the channel can be the sender of a message $m$ even after $m$ is correctly broadcast. We allow this in order to remove the need for parties to share the proofs, as there is no incentive to do so. The basic communication pattern of consistent broadcast can be seen in Figure 1.

---

[2] Note that a channel can also be created between multiple parties [9].

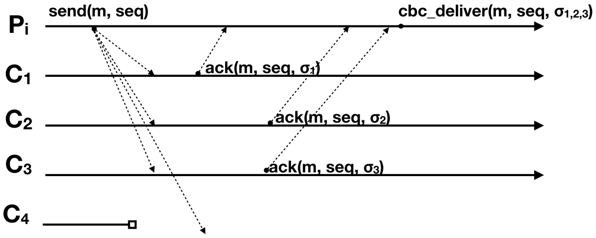

**Figure 1: Consistent broadcast communication pattern: The party $P_i$ will use consistent broadcast to send the new message and sequence number to the committee. If enough committee members acknowledge (*e.g.* 3 out of 4) then $P_i$ holds enough proof that the message will persist. Anyone willing to get the same information can either ask $P_i$ or run the same protocol.**

## 2.3 Cryptographic Secret Sharing

*Secret Sharing.* The notion of secret sharing was introduced independently by Blakely [6] and Shamir [47] in 1979. An $(t, n)$-secret sharing scheme, with $1 \leq t \leq n$, enables a dealer to share a secret $a$ among $n$ trustees such that any subset of $t$ honest trustees can reconstruct $a$ whereas smaller subsets cannot. Thus, a sharing scheme can withstand up to $t - 1$ malicious participants.

In the case of Shamir's scheme, the dealer evaluates a degree $t - 1$ polynomial $s$ at positions $i > 0$ and each share $s(i)$ is handed out to a trustee. The important observation here is that only if a threshold of $t$ honest trustees collaborates then the shared secret $a = s(0)$ can be recovered (through polynomial interpolation).

*Verifiable Secret Sharing.* The downside of these simple secret sharing schemes is that they assume an honest dealer which might not be realistic in some scenarios. Verifiable secret sharing (VSS) [12, 24] adds verifiability to those simple schemes and thus enables trustees to verify if the shares distributed by a dealer are consistent, that is, if any subset of a certain threshold of shares reconstructs the same secret.

*Distributed Key Generation.* A Distributed Key Generation (DKG) protocol removes the dependency on a trusted dealer from the secret sharing scheme by having every trustee run a secret sharing round. In essence, a $(n, t)$ DKG [30] protocol allows a set of $n$ servers to produce a secret whose shares are spread over the nodes such that any subset of servers of size greater than $t$ can reveal or use the shared secret, while smaller subsets do not have any knowledge about the secret. Pedersen proposed the first DKG scheme [43] based upon the regular discrete logarithm problem without any trusted party. We provide a summary of how Pedersen DKG works in Appendix A, and optionally modify it to meet our consensus needs below.

## 2.4 Consensus

The *Byzantine Generals' Problem* [35, 42] is a more powerful primitive than the consistent broadcast protocol where a group of $n$ processes in a distributed system reaches agreement on a value (which requires termination, unlike consistent broadcast). In BRICK the processes only need to decide whether or not to close a channel, hence we only care for the restricted problem of binary consensus in which the input is binary. Pease et al. [42] show that $3f + 1$ participants are necessary to be able to tolerate $f$ arbitrary faulty processes and still reach consensus.

It is well known that reaching consensus in full asynchrony with a single process crashing is impossible with a deterministic protocol [25] (FLP), hence we need to introduce some stronger assumption for BRICK to be functional when consensus is needed. In this section, we introduce the types of consensus algorithms we can deploy in BRICK, which do not affect the safety of the system under full asynchrony (unlike existing channel constructions), but only rely on stronger assumptions for liveness.

*Partially Synchronous Consensus.* Consensus is achievable despite the FLP impossibility result by adding timing assumptions. For this variant of consensus, we need to assume there is a time GST (General Stabilization Time) after which all messages among correct replicas arrive within a known bound $\Delta$. The system can swing between periods of synchrony and asynchrony but termination of consensus in only guaranteed during the periods of synchrony. The first practical protocol is [11], but dozens of other protocols exist, *e.g.*, [2, 32].

*Randomized Asynchronous Consensus with Trusted Setup.* A second way to circumvent the FLP impossibility is to introduce randomization so that consensus is reached with probability 1, but there can still exist a non-terminating run with probability 0. The most efficient of these protocols assume a strong shared coin meaning that all processes return the same random number when they flip the coin. In this setting, we can use VABA [3] as it is the most scalable protocol to date with $O(n^2)$ communication complexity.

It is noteworthy that the only existing constructions that implement a strong common coin require cryptographic assumptions and a trusted setup [10]. Next, we show how to circumvent the trusted assumption by assuming a perfect failure detector and aborting the setup in case any process fails. This is a more acceptable assumption than the trusted setup in our setting as the job of the committee that will reach consensus is to always be live. Therefore, if a committee fails during setup, then BRICK can use it as an indicator to use another committee.

Given this assumption, we can run a modified synchronous distributed key generation protocol by Genarro et al. [27] to produce a common coin. The details are in Appendix A. In short, we set $N = 3f + 1$, $t = 2f + 1$ in [27] and wait for all participants to reply within $\Delta$ (instead of waiting for $N - f$ responses) or fail the protocol. If the protocol finishes then a correct shared private-key is generated. This can be used to produce an unlimited number of deterministic threshold signatures (BLS [7] or RSA [48]) which can be used as common-coins [10]. The caveat of our protocol is that it fails even with one crashed committee member. However, since the main job of the committee is to be available at all times, failure to do so during setup is a good indicator to replace the committee member.

*Modified Liveness argument for setup.* In [27] the liveness of the system holds even if $f$ participants crash since the system only needs $f+1$ honest replies before terminating. Furthermore, the safety of the DKG is guaranteed since the reconstruction threshold is $t = f + 1$ and synchrony is also assumed during reconstruction. Hence, the $f + 1$ honest nodes are guaranteed to reply.

In BRICK we convert the protocol to support a fully asynchronous reconstruction phase. There we run into the problem that if our setup is asynchronous, we can only wait for $2f + 1$ nodes to reply before terminating, however $f$ of those might be malicious. This means that during the reconstruction of the common coin there is no guarantee that there will be $t = 2f + 1$ honest nodes holding valid shares and as a result liveness of the common coin is not guaranteed.

This is the underlying reason of our strong availability assumption during setup. We need to know that there are $2f + 1$ nodes that hold valid shares. As a result we need to wait for all $3f + 1$ nodes to accept their shares before terminating the setup. If BRICK's setup is done correctly we know that later on there will be $2f + 1$ honest share holders (since at most $f$ lied during setup) and when their shares are eventually delivered, then the common coin will be correctly reconstructed.

*Randomized Asynchronous Consensus.* If we do not want to make the assumption of a live setup nor of partial synchrony, BRICK resorts to reaching randomized asynchronous consensus using Bracha [8]. The expected time of this protocol is exponential with regard of the number of participants. In our case consensus is run rarely (usually once per channel at closing time), hence the overhead might be acceptable, especially if the size of the committee is small.

## 3 PROTOCOL OVERVIEW

In this section we present the system and threat model we consider, a high level overview of BRICK and BRICK+ together with the goals we want to achieve.

### 3.1 System Model

We make the usual cryptographic assumptions: the participants are computationally bounded and that cryptographically-secure communication channels, hash functions, signatures and encryption schemes exist. We assume the underlying blockchain maintains a distributed ledger that has the properties of *persistence* and *liveness* as defined in [26]. However, we do not require a "perfect" blockchain system, since BRICK can tolerate temporary extreme conditions. Specifically, if an adversary can temporarily violate the blockchain liveness property, this will effect in violating payment channels liveness property and will not affect the safety of the payment channel construction.

### 3.2 Threat Model

We initially assume honest participants in the channels to simplify the security analysis. However, later, we show that the security analysis holds even under rational channel parties that intentionally deviate from the protocol if they can increase their profit. Regarding the committee, we assume that there are at most $f$ out of $n = 3f + 1$ Byzantine nodes and we define a threshold $t = 2f + 1$ to achieve the liveness and safety properties. The non-Byzantine part of the committee is assumed honest when proving the desired properties, nevertheless, we later incentivize this honest behavior for rational committee members.

### 3.3 BRICK Overview

All parties agree on a committee before opening a channel. The committee members commit their identities on the blockchain during the funding transaction of the channel (opening of the channel). After opening the channel on the blockchain, the channel can only be closed either by a transaction published on the blockchain and signed by all parties or by a transaction singed by one of the parties and a fraction of the committee members. Thus, the committee acts as power of attorney for the parties of the channel.

Every time a new update state occurs in the channel, every party runs a consistent broadcast protocol with the members of the committee. Specifically, he announces to the committee that a new update has occurred. This announcement (defined below) is first signed by all the participants of the channel to signal that they are in consensus and (a) consists of a commitment to the state in order to preserve privacy and (b) is associated with a monotonically increasing sequence number to guarantee that this update is the most fresh state. If the consistent broadcast protocol succeeds ($t$ nodes acknowledge reception) then this can serve as proof for all parties that the state-update is safe. After this procedure terminates correctly all parties proceed to the execution of the off-chain state.

In case a party wants to close the channel in collaboration with the committee, the members of the committee run a consensus protocol to ensure there is no concurrent update state. If the consensus is successful then the committee members sign the closing state and send it to the party, otherwise they validate the new update state.

### 3.4 BRICK+ Overview

BRICK+ is designed to enable payment channels in a permissioned, regulated setting, for example a centrally-banked cryptocurrency. In such a setting, there will be an auditor (*e.g.* the IRS) that can check all the transactions inside a channel as these transactions might be taxable. This is a realistic case as the scalability in payment channels come from persistent relationship that model well B2B and B2C relationships that are usually taxed. In this setting, we assume that the auditor can punish the parties and the committee externally of the system, hence our goal is to enhance transparency even if misbehavior is detected retroactively.

In order to convert BRICK into BRICK+ we need to make sure (a) that nothing happens without the committee's approval and (b) that a sufficient audit trail is left on-chain to stop regulators from misbehaving. We resolve the first issue by disabling the ability for the parties to close the channel without the participation of the committee and by additionally having the committee to store the a hash-chain of the state history. To enable prevent the auditor from misbehaving, we force him to post a "lawfull access request" [23] on-chain in order to convince the parties of the channel to initiate the closing of the channel for audit purposes and send him the state history. To incentivize the committee to participate, this access request comes with additional closing fees for the committee members who send to the audit smart contract the head of the hash-chain.

## 3.5 Reward Allocation & Collateral

To avoid bribing attacks, we enforce the committee members to lock a collateral in the channel. The necessary amount of collateral is discussed in detail in Section 6. Additionally, the committee is incentivized to actively participate in the channel with a small reward that each committee member gets when they acknowledge a state update in the channel. This reward can be given with a unidirectional channel [28], which does not suffer of the problems BRICK solves. Moreover, the committee members that participate in the closing state of the channel get an additional reward, hence the committee is incentivized to assist a party when closing in collaboration with the committee is necessary.

Similarly to closing in BRICK, during auditing in BRICK+, we incentivize the honest behavior of rational committee members by giving them a reward (allocated by the auditor when publishing the access request), which is delivered by the audit smart contract when a committee member sends the correct head of the hash-chain.

## 3.6 Protocol Goals

To define the necessary goals of BRICK, we first need to define the necessary properties of a channel construction. Intuitively, a channel construction should ensure similar properties with blockchain systems *i.e.*, a party cannot cheat another party participating in the channel and any party has the prerogative to eventually close the channel at any point in time. The first property is encapsulated by *Safety*, while the second by *Liveness*. Additionally, we define an optional property, *Privacy*, which is not guaranteed in many popular blockchains, such as Bitcoin [40] or Ethereum [52], but constitutes an important practical concern for any functional monetary (cryptocurrency) system. Furthermore, we define another optional property, *Auditability*, which allows authorized third parties to audit the states of the channel, thus making the channel construction suitable to be used on a real currency. The first three properties are met by BRICK, while the latter is only available in BRICK+.

First, we define some characterizations on the state of the channel, namely, validity and commitment. Later, we define the properties for the channel construction.

Each state of the channel has a discrete sequence number that reflects the order of the state. We assume the initial state of the channel has sequence number 1 and every new state has a sequence number one higher than the previous state agreed by all the parties of the channel. We denote by $s_i$ the state with sequence number $i$.

DEFINITION 1. *A state of the channel, $s_i$, is **valid** if the following hold:*

- *All parties on the channel have signed the state $s_i$.*
- *The state $s_i$ is the **freshest** state,* i.e. *the previous valid state was $s_{i-1}$.*
- *The committee has not invalidated the state. The committee can invalidate the state $s_i$ if at least $2f+1$ committee members sign the state $s_{i-1}$ upon receiving a closing request.*

DEFINITION 2. *A state of the channel is **committed** if it was signed by at east $2f + 1$ committee members or is part of a block in the persistent part of the blockchain.*

DEFINITION 3 (**SAFETY**). *No channel will close in a state other than the most fresh committed state.*

DEFINITION 4 (**LIVENESS**). *Any valid operation (update, close) on the state of the channel will eventually* [3] *be committed (or invalidated).*

DEFINITION 5 (**PRIVACY**). *No unauthorized* [4] *external (to the channel) party learns about the state of the channel (eg. for a payment channel, the current distribution of funds between the channel parties) unless at least one of the parties initiate the closing of the channel.*

DEFINITION 6 (**AUDITABILITY**). *All committed states of the channel are verifiable by an authorized third party.*

# 4 BRICK DESIGN

The protocol consists of three phases: `Open`, `Update & Consistent Broadcast`, and `Close`. We assume the existence of a *smart contract* (self-executed code run on a blockchain). The BRICK smart contract has two functions, *Open* and *Close*, that receive the inputs of the protocols and verify that they adhere to the abstractly defined protocols specified below.

Algorithm 1 describes the first phase, *Open*, which is the opening of a channel between $m$ parties. In this phase, the parties create the initial funding transaction, similarly to other known payment channels such as [17, 44]. However, in BRICK we also define three additional parameters during this phase; we include to the funding transaction the hash of the public keys of the committee members of the channel, denoted by $C_1, C_2, \ldots, C_n$, the threshold $t$, and a closing fee $F$. This fee will be awarded to the responsive committee members at the last phase, *Close*, if and only if the closing of the channel is done in collaboration with the committee. In this case, the members that participated in signing the closing transaction will be rewarded with the amount $F$ divided by the number of signatures from the committee members $(2f + 1)$. In addition, each committee member locks a capital in the BRICK smart contract which will be claimed by the parties of the channel during the phase *Close* if a committee member misbehaved (*e.g.* received a bribe).

The second phase of the protocol consists of two sub-protocols, `Update` (Protocol 2) and `Consistent Broadcast` (Protocol 3). Both algorithms are executed consecutively every time a new update state occurs, *i.e.* when the state of the channel changes. During `Update`, the parties of the channel agree on a new state and create an announcement which they broadcast to the committee members using `Consistent Broadcast`.

---

[3] Depending on the message delivery.

[4] Authorized parties are potential auditors of the channel, as described in BRICK+.

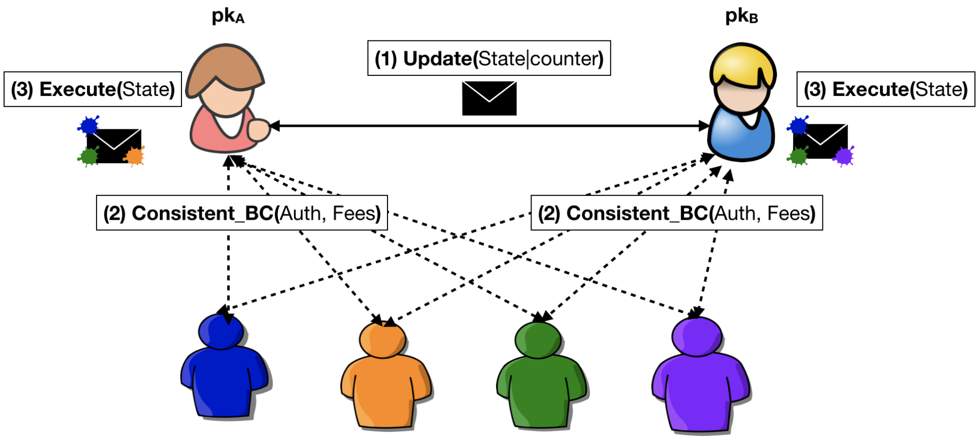

**Figure 2: Typical workflow of BRICK for a state update. (1) Alice and Bob agree on a new state update. (2) They individually broadcast the update to the committee (with an associated fee). (3) When a threshold of committee members replies that the broadcast update state is committed, each sender (Alice or Bob) executes this update as persistent.**

**Protocol 1:** `Open`

*Input:* Channel parties $P_1, P_2, \ldots, P_m$, committee members $C_1, C_2, \ldots, C_n$, initial state $s_1$ ($m$-order vector), fee for the committee $F$.

*Goal:* Open a BRICK payment channel.

1. Every party $P_1, P_2, \ldots, P_m$ signs: $open(H(C_1), H(C_2), \ldots, H(C_n), t, s_1, F)$.
2. Register to $\{M, \sigma(M)\}$ the announcement of Protocol `Update`$(P_1, P_2, \ldots, P_m, s_1, null)$.
3. Execute Protocol `Consistent Broadcast`$(M, \sigma(M), P_1, P_2, \ldots, P_m, C_1, C_2, \ldots, C_n)$.
4. Publish to the blockchain $\sigma_{P_1, P_2, \ldots, P_m}(open(H(C_1), H(C_2), \ldots, H(C_n), t, F))$.

The announcement is the hash of the new state (to preserve privacy), as well as the sequence number, signed by all the parties in the channel. This way all the parties commit to the new update state of the channel, while none of the parties can unilaterally close the channel without the collaboration of either all other parties or the committee. At the same time the state of the channel is not revealed to the committee, but in case a party wants to close the channel, the committee members, given the correct state, can verify this is the freshest update state all parties have agreed to.

In Protocol 3, for every state update, each party sends to all committee members the announcement including a small fee for watching the channel. Then, each committee member replies to every party that sent him the announcement by signing the announcement, if no closing state is in progress. This signed announcement can be used later to construct a proof-of-fraud in case the committee member colludes with a party and signs an older update state to close the channel.

**Protocol 2:** `Update`

*Input:* Channel parties $P_1, P_2, \ldots, P_m$, current state $s$.

*Goal:* Create announcement $M, \sigma(M)$ (cloaked state signed by all parties).

1. Every party $P_1, P_2, \ldots, P_m$ signs: $\{H(s_i, r_i), i\} = M$, where $r_i$ is a random number and $s_i$ the current state, thus creating the announcement $\{M, \sigma(M)\}$.

**Protocol 3:** `Consistent Broadcast`

*Input:* Channel parties $P_1, P_2, \ldots, P_m$, committee members $C_1, C_2, \ldots, C_n$, announcement $\{M, \sigma(M)\}$, reward $r$.

*Goal:* Inform the committee of the new update state and verify the validity of the new state.

1. Each party $P_i$ broadcasts to all the committee members $C_1, C_2, \ldots, C_n$ the announcement $\{M, \sigma(M)\}$ alongside a reward $r$.
2. Each committee member $C_j$, upon receiving $\{M, \sigma(M)\}$, verifies that all parties' signatures are present, the sequence number is exactly one higher than the previously stored sequence number and no party has requested to execute the protocol `Close` on an earlier state. Then, $C_j$ stores the announcement $\{M, \sigma(M)\}$ (replacing the previous announcement), then signs $M$, $(\sigma_{C_j}(M))$, and broadcasts it to every party $A_i$ that payed the reward $r$.
3. Each party $P_i$, upon receiving at least $t$ signatures on the announcement $M$, considers the state committed and proceeds to state transition.

The last phase of the protocol, `Close`, can be implemented in two different ways: the first is similar to the classical approach for closing a channel (Protocol 4: `Optimistic Close`) where all parties collectively sign the freshest update state (closing transaction) and publish it on the blockchain. However, in case some of the parties of the channel are not active or responding to sign the closing transaction a party can unilaterally close the channel in collaboration with the committee of the channel. In Protocol 5: `Pessimistic Close`, a party initiates the last phase of the protocol by requesting the signatures of the

**Protocol 4:** `Optimistic Close`

*Input:* Channel parties $P_1, P_2, \ldots, P_m$, state $s$.
*Goal:* Close a channel on state $s$, assuming all parties are responsive and in agreement.

1. A party $p \in \{P_1, P_2, \ldots, P_m\}$ broadcasts the request $close(s)$.
2. All parties $P_1, P_2, \ldots, P_m$ sign the state $s$ (if they agree) and broadcast the signed message to all other parties.
3. The party $p$ (or any other party of the channel) publishes the signed by all parties state, $\sigma_{P_1, P_2, \ldots, P_m}(s)$ in the blockchain.

committee members on the freshest state of the channel. The committee members, upon receiving this request that includes the state and the sequence number, verify this is the freshest committed state and initiate a consensus protocol amongst themselves (parties do not participate). If the consensus protocol terminates successfully, hence all committee members agree that this is the freshest committed update state, each committee member signs the closing state and sends it to the party. As soon as the party collects at least $t$ signatures on the closing state, he publishes to the blockchain the state with the multisig (or threshold signature if the committee run a DKG at setup) from the committee members. The `Pessimistic Close` protocol has stronger assumptions and might have higher communication complexity than the rest of the system as it needs consensus to guarantee liveness, however it is only executed when in dispute and does not affect the performance of the common case.

## 5 BRICK SECURITY ANALYSIS

In this section, we prove that BRICK satisfies the protocol goals as defined in Section 3.1, *i.e. Safety, Liveness* and *Privacy*.

THEOREM 1. BRICK *achieves safety.*

PROOF. We can safely assume that the channel will not close in a state that is not committed when the committee is involved, since the honest committee members will not recognize and thus agree on such a state during consensus. Furthermore, when the parties close the channel in collaboration the closing state will eventually be committed by definition. Thus, to prove safety for BRICK it is enough to show that a channel cannot close in a committed state that is not the freshest.

In BRICK there are two ways to close a channel (phase `Close`), either by invoking Protocol 4 or by invoking Protocol 5. In the first case (`Optimistic Close`), all parties agree on closing the channel in a specific state (which is always the freshest valid state [5]). As long as this valid state is published in a block in the persistent part of the blockchain, it is considered to be committed. Thus, safety is guaranteed.

In the second case, when Protocol 5: `Pessimistic Close` is invoked, a party has decided to close the channel unilaterally in collaboration with the committee. We will prove safety by contradiction. Let us denote by $s_i$ the closing state of the channel. Suppose that there is a committed state $s_k$ such that $k > i$, thus $s_i$ is not the most fresh state agreed by all parties. At least $t = 2f + 1$ committee members have signed the state $s_i$, since otherwise the BRICK smart contract would not have accepted the closing state as valid. According to the threat model, at most $n - t = f$ committee members are byzantine, thus at least $f + 1$ honest committee members have signed state $s_i$ as the closing state. However, according to Protocol 5 (line 2) all honest committee members that follow the protocol verify that $M = \{H(s_i, r_i), i\}$ , where $M$ is the announcement the committee member have stored, *i.e.* the announcement that corresponds to the freshest valid state she previously received, and initiate a consensus process. Since $s_i$ is a closing state, the consensus terminated successfully for the state $s_i$. Therefore, there are at least $f + 1$ honest committee members that consider $s_i$ the freshest committed state and thus have not received the freshest valid update state. However, in phase `Update & Consistent Broadcast`, an update state is considered to be committed, according to Protocol 3 (line 3), if and only if it has been signed by at least $t = 2f + 1$ committee members. Since at most $n - (f + 1) = 3f + 1 - f - 1 = 2f < 2f + 1$ members of the committee have seen (and hence singed) the state $s_k$, the state $s_k$ is not committed. Contradiction. Therefore, the closing state $s_i$ is the freshest committed state. □

**Protocol 5:** `Pessimistic Close`

*Input:* Party $p \in \{P_1, P_2, \ldots, P_m\}$, committee members $C_1, C_2, \ldots, C_n$, state $s_i$ ($m$-order vector), random number $r_i$
*Goal*: Close a channel on state $s_i$ with the assist of the committee.

1. Party $p$ broadcasts to the committee members $C_1, C_2, \ldots, C_n$ the request $close(s_i)$. He additionally reveals to each committee member the public keys of all committee members.
2. Each committee member verifies that $\{H(s_i, r_i), i\} = M$ (where $M$ is the announcement the committee member has stored). Then they input the final state update they have seen together with $M$ in a consensus round. If the close is decided to be valid (ordered before any new update) then every $C_j$ signs the state $s_i$, $\sigma_{C_j}(s_i)$ and sends it to party $p$.
3. Party $p$, upon receiving $t$ signatures from the committee on the state $s_i$, publishes on the blockchain $s_i$ with the $t$ signatures.
4. After the state is included in a (permanent) block, the smart contract closes the channel in state $s_i$ and each committee member whose signature is published on the blockchain gets a reward of $F/t$ (reward locked in the funding transaction).

THEOREM 2. BRICK *achieves liveness.*

PROOF. We will show that every possible valid operation is either committed or invalidated. There are two distinct operations: *close* and *update*.

---

[5] We assume that if the parties want to close the channel in a previous state, they will still create a new state similar to the previous one but with an updated sequence number.

If the operation is close and is not committed, then either the parties did not agree on this operation (`Optimistic Close`), hence the operation is not valid or the consensus did not end successfully. In the latter case, a concurrent valid update state was published and the consensus (`Pessimistic Close`) ordered the update operation first, thus invalidating the close operation.

Suppose now the operation is close and never invalidated. Then, if it is an optimistic close, all the parties in the channel have signed the closing state since it is valid. Since the parties are honest they will broadcast the transaction to the blockchain. Assuming the blockchain has liveness, eventually the state will be included in a block in the persistent part of the blockchain and thus will be eventually committed.

Suppose the operation is a valid update and it was never committed. Since the operation is valid and the parties of the channel are honest, the committee members eventually received the update state (`Consistent Broadcast`). However, the update state was never committed, therefore at least $f + 1$ committee members did not sign the update state. We assumed at most $f$ byzantine committee members, hence at least one honest committee member did not sign the valid update state. According to Protocol 3 (line 2), an honest committee member does specific verifications and if the verifications hold she signs the new update state. Thus, for the honest committee member that did not sign, one of the verifications failed. If the first verification fails, then a signature from the parties of the channel is missing thus the state is not valid. Contradiction. The second verification concerns the sequence number and cannot fail assuming the channel parties are honest. Thus, the third verification fails, which means there is a request to execute `Close` on an previous state. In this case, a consensus protocol is executed according to Protocol 5, which orders the update state and the closing state. One of the two gets committed and the other invalidated. Since the update state was never committed, it was invalidated by the committee during the consensus process.

For the last case, suppose the operation is a valid update and it was never invalidated. We will show the update was eventually committed. Suppose the negation of the argument towards contradiction. We want to prove that an update state that is not committed is either not valid or invalidated. The reasoning of the proof is similar to the previous case with minor modifications. □

THEOREM 3. BRICK *achieves privacy.*

PROOF. Suppose an external party learns about the state of the channel during the protocol execution. This means that either she intercepted a message (between the parties of the channel or between the parties and the committee) or she is a committee member. In the first case, we assume secure communication channels thus a computationally-bounded adversary cannot get any information from the messages between the parties. In the latter case, the committee members receive during the `Update & Consistent Broadcast` phase a message $M = \{H(s_i, r_i), i\}$ for any valid update state (assuming honest parties that do not intentionally reveal the state). If the committee member extracts the state of the channel $s_i$ from the $H(s_i, r_i)$, this means she reverted the hash function, hence the hash function is not pre-image resistant for a computationally-bounded adversary and thus not secure. This contradicts the system model, where we assumed cryptographically-secure hash functions. □

# 6 INCENTIVIZING HONEST BEHAVIOR

In this section, we design incentive mechanisms for rational committee members to incentivize honest behavior. Additionally, we argue for rational channel parties thus alleviating the assumption that the participants of a channel are honest.

## 6.1 Incentivizing Rational Committee Members

There are three different mechanisms in BRICK to incentivize honest behavior for rational committee members. The first one is a small fee that rewards the committee members that are responsive during the `Update & Consistent Broadcast` phase. The second one is a final fee rewarded to the committee members participating in the `Close` phase. Lastly, the third incentive mechanism has a different nature; committee members lock collateral to ensure honest behavior (resistance to colluding and bribing) is the dominant strategy for the rational part of the committee ($2f + 1$). This way we guarantee the safety of the channel construction even under a rational committee (while allowing $f$ byzantine members).

**`Update` *Rewards.*** In order to incentivize the responsiveness of the committee members the parties establish an one-way channel [28] where they send a small payment every time they want a signature. At first sight, this game looks like a fair exchange game, which is impossible to solve without a trusted third party [21]. Furthermore, we cannot use a blockchain to solve it [41] as the whole point of state channels is to reduce the number of transactions that go on-chain. Fortunately, the state-update game is a repeated game where committee members want to increase their expected rewards in the long term. As a result, they know that if they receive a micro-payment from a party and do not respond, then the party will stop using them (there is $f$ fault tolerance in BRICK) and as a result their expected rewards for the repeated game will decrease.

**`Close` *Rewards.*** The same fair exchange issue arises in the `Pessimistic Close` protocol, which is not a repeated game as the committee members know that closing is the last action that happens in a channel. Thus, in this case we use the blockchain to solve the fair exchange problem. Specifically, we split the fee $F$, which was locked during phase `Open` (Protocol 1), only among the members of the committee that signed the closing state in Protocol 5. In this phase, we assume rational channel parties, hence the parties have no incentive to bribe the committee to ignore a close request in a specific state since such a strategy cannot increase somehow the parties' profit. Therefore, a committee member has the choice to either sign the closing state and gain part of the extra fee or not. Since we assume rational committee members that aim to increase their expected profit, they will all sign the closing state (as long as the fee is sufficient enough to cover their cost of being online).

*Committee Collateral.* To enforce incentives for the committee, we demand each committee member to lock a collateral strictly higher than the amount of money locked in the channel divided by $f + 1$. To commit fraud, one of the parties can attempt to bribe some (at least $f + 1$) members of the committee to sign an older state and thus close the channel in a wrong state. However, during the bribing procedure the committee member must provide a signature on a previous than the freshest state. This signature can construct a proof-of-fraud for the rest of the channel parties, if broadcast by the bribing party. Specifically, a proof-of-fraud consists of (a) the singed older state and (b) the singed acknowledgment the same committee member sent to the party when asked for the validity of the last freshest state of the channel. These two messages ensure that the committee member was aware that the channel state was updated and thus intentionally committed fraud by signing an older state. Consequently, the channel party can eventually claim the committee member's collateral when closing the channel on chain (including this proof-of-fraud). If the collateral is strictly higher than the amount of money locked in the channel divided by $f + 1$, the party would gain more profit if he claims the collateral of all misbehaving committee members than closing the channel in any wrong state. Thus, any committee member will not provide such a proof-of-fraud, and hence will not accept any bribes.

## 6.2 Rational Parties Assumption

For the security proofs, we assumed the parties of the channel are honest. However, this assumption is not necessary. In this section, we argue that a rational party will not deviate from the protocol, thus the security of BRICK holds even under rational channel parties. We argue for every phase separately.

`Open`. It is trivial to see that if a party is not incentivized to open a channel then that channel will never be opened. We assume the parties have some business interest to use the blockchain and since transacting on channels is faster and cheaper they will prefer it. Deviating from the protocol at this phase is meaningless.

`Update & Consistent Broadcast`. During the execution of Protocol 2, any party can deviate from the protocol by not signing the hash of the new update state. In this case, the new state will not be valid and thus cannot be committed and will not be executed. No party can gain profit from such a behavior directly (attacking the safety of the channel). Moreover, attempting to attack the liveness of the BRICK channel is not profitable, since any other party can always request to close on the previously committed state in collaboration with the committee by invoking Protocol 5: `Pessimistic Close`. Thus, rational parties will not deviate from the `Update` protocol.

During the execution of `Consistent Broadcast`, a party can deviate in the following ways:

- First, he can choose not to broadcast the announcement to the committee or part of the committee. In this case, the party has signed the new update state, which is now a valid state and for the rest of the parties this state will be considered committed after the execution of Protocol 3. Therefore, the only damage this party can do is to himself: if the committee closes the channel in the previous state after the execution of the new update state, he cannot construct a proof-of-fraud because he never received the acknowledgment of the announcement from the committee members. Hence, the party cannot prove fraud and can potentially lose his money.
- Second, he can broadcast different messages to the committee or parts of the committee. During the execution of Protocol 3, the committee members verify the parties' signatures, thus an invalid message will not be acknowledged from an honest committee member. If the messages are valid (all parties' signatures are present), all parties have misbehaved in collaboration. This can lead in a permanent partition of the view of the committee regarding the state history, but at most one of the states can be committed (get the $2f + 1$ signatures). Thus, this strategy has the same caveats as the first one, where the party can only lose from following it.
- Lastly, he can choose not to proceed to the state transition. This is outside the scope of the paper and is a general problem of different nature (a fair exchange problem).

Overall, a rational party cannot increase his profit by deviating from Protocols 2 or 3.

`Close`. In this phase, there are two different options: `Optimistic Close` and `Pessimistic Close`.
In the first case, a party can deviate from the protocol in the following ways:

- He is the party requesting the closing of the channel in a cheating state. At least one of the other parties will not sign the state since at least one of the parties is being cheated, else it would not be a cheating state. Thus, safety is guaranteed and the party cannot profit from this strategy, even if he colludes with some of the other parties of the channel.
- He is the party requesting the closing of the channel and he never publishes the signed closing state. In line 2 of Protocol 4, the signed state is broadcast by all parties to all other parties, hence another party will eventually publish the closing state.
- He is a party that got the closing request and does not sign the state. In this case, the party requesting to close the channel can invoke the `Pessimistic Close` protocol and close the channel in collaboration with the committee in the freshest committed state.

Thus, any potential deviation from Protocol 4 cannot increase the profit of a party executing the protocol.
In the second case, where Protocol 5 is executed, a party can deviate from it in the following ways:

- The party requests to close the channel in a state $s_i$ that is not the freshest committed state. Then, the party's request for close will be ignored by the rational committee members (as explained in subsection 6.1) and thus he will not be able to gather $2f + 1$ signatures and close the channel in a "wrong" state.

- The party requests to close the channel on the freshest committed state, gets the necessary signatures, but never publishes the closing state (attempting to attack liveness and potentially blackmail other parties). In this case, any committee member has incentive to publish the signed pessimistic close in order to claim the closing reward, so the liveness of the channel is guaranteed. Hence, the party cannot gain any profit from deviating from the protocol in this step.

To summarize, if a channel party deviates from the honest protocol execution at any phase, he cannot gain additional profit. Therefore, any rational party will follow the protocol, hence honest behavior of channel parties is incentivized.

## 7 BRICK+ DESIGN

BRICK+ consists, similarly to BRICK, of three phases, `Open`, `Update & Consistent Broadcast`, and `Close`. However, BRICK+ has one additional functionality, *audit*, that allows an authorized third party to audit the states of a channel. Invoking this functionality though enforces the closing of the channel [6]. The audit functionality is illustrated in Figure 3 and described in detail in Protocol 6. We assume the existence of an *audit* smart contract that handles the fair exchange of the reward for the responsive committee member, similarly to the BRICK smart contact (for close).

To enable the audit functionality and therefore achieve *Auditability*, we disable Protocol 4: `Optimistic Close`, and enforce the parties to close in collaboration with the committee. This way we guarantee that all states of the channel are available to the committee and hence to the potential auditor for verification. Moreover, we modify Protocol 2 and Protocol 3 (phase `Update & Consistent Broadcast`) such that the committee members store a hash-chain of the state history instead of the hash of the freshest valid state they received. This way we ensure that the parties cannot present an alternate state history to the auditor as it achieves fork-consistency [36].

The audit functionality is initiated by an authorized third party, namely the auditor, who publishes an access request on-chain. Then, the parties of the channel verify the validity of the access request and initiate the closing of the channel by invoking Protocol 5: `Pessimistic Close`. After the execution of Protocol 5, both the honest committee members and the (honest) parties of the channel have a consistent view of the channel history. Every party sends to the auditor the entire state history. Additionally, each committee member sends to the smart contract the head of the hash-chain of the state history. The smart contract, upon receiving $2f + 1$ replicas of the hash, sends the hash to the auditor and rewards these $2f + 1$ committee members, similarly to the BRICK smart contract (phase `Close`). The auditor then verifies the state history he received from every party by computing the hash-chain and comparing the last hash with the hash he received from the smart contract. If the parties misbehave and send an alternate state history the auditor can pursue external punishment (*e.g.* legal action).

**Protocol 6:** `Audit`

*Input:* Auditor $A$ of channel $c$, audit smart contract with access on the information published on the blockchain, from Protocol 1.

*Goal*: Audit of the channel.

1. The auditor $A$ publishes on the blockchain the access request for channel $c$ and locks rewards in the audit smart contract.
2. Each party of the channel, upon verifying the validity of the access request, initiates the closing phase of the channel, *i.e.* invokes Protocol 5: `Pessimistic Close` in the last committed phase.
3. After the execution of Protocol 5, the channel is closed on-chain in a committed state $s$. Then, all parties of the channel send to the auditor the state history and each committee member sends to the audit smart contract the head of the hash-chain of all committed states (where $s$ is the last committed state).
4. The smart contract upon receiving the same hash from $2f + 1$ committee members (checks the identities with the opening state), rewards the committee members that were responsive, and publishes the hash on-chain.
5. The auditor receives the state history from all parties and reads the hash from the audit smart contract, and verifies the state history by re-creating the hash-chain. If a party does not respond to the access request or presents a different state history the auditor pursues external punishment.

### 7.1 BRICK+ Security Analysis

In this section, we first prove the BRICK+ goals, namely *Safety, Liveness, Privacy* and *Auditability*, under the assumption of $2f + 1$ honest committee members and later we argue that rational committee member will not deviate from the protocol.

Throughout this section, we assume the parties of the channel are rational players, thus they will not deviate from honest behavior if they will be discovered and punished. Furthermore, we assume the auditor is also rational, meaning he will deviate from the protocol if he can gain more profit (hence the need for a smart contract to do the fair exchange between the auditor and the committee). However, he will not punish a party arbitrarily with no proof, since he is supposed to be an external trusted authority (*e.g.* judge, regulator, tax office etc.).

THEOREM 4. BRICK+ *achieves safety.*

PROOF. It follows from Theorem 1, since all modifications on BRICK do not affect the safety property. □

THEOREM 5. BRICK+ *achieves liveness.*

PROOF. It follows from Theorem 2, since all modifications on BRICK do not affect the liveness property. □

THEOREM 6. BRICK+ *achieves privacy.*

PROOF. The audit request, which is the first step of the audit functionality, does not leak any information on the channel since the auditor is an external to the channel party. According

---

[6] Closing the channel when an audit request occurs is not necessary, as long as the committee simulates the closing process, *i.e.* the consensus on the last update state, and inform the parties. However, to avoid confusion we assume that the audit functionality enforces the closing of the channel.

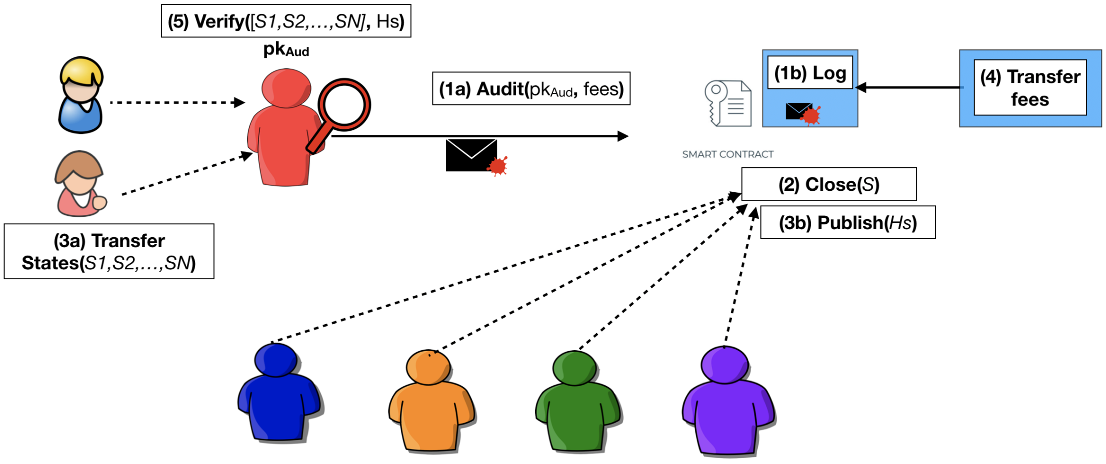

**Figure 3: Typical workflow of BRICK+ for an audit update. (1) The auditor starts the audit by posting the request on chain, (2) the committee closes the channel, and (3) the parties transfer the state to the auditor. Then (4), the committee posts the head of the hash-chain of the channel on-chain to claim their reward and finally (5) the auditor cross-checks the claims of the party and the committee.**

to Protocol 6 (line 2), the parties initiate the closing of the channel. Thus, privacy is preserved since by definition it is only guaranteed until at least one of the parties initiates the closing of the channel. □

THEOREM 7. BRICK+ *achieves auditability.*

PROOF. Since the parties of the channel are rational they will not initiate the closing process (Protocol 6, line 2) unless the request for access is valid, hence the auditor is authorized. Thus, to prove BRICK+ satisfies auditability, it is enough to prove that every committed state is verifiable by a third-party.

Towards contradiction, suppose $s$ is the earliest (least fresh) committed state that is not verifiable (since there is at least one). This means that either state $s$ does not exist or it is replaced by another state $s'$ either in the state history of all the parties or in the hash-chain that produced the hash head provided by the $2f+1$ honest committee members. If $s$ is not part of the hash-chain (latter case), but it is committed then there is a quorum of $2f+1$ committee members that produced a different hash-chain and thus a different head, therefore there are at least $f+1$ byzantine committee members. Contradiction to our threat model. Thus, the state $s$ is not included in the state history by all parties. In this case, the auditor punishes the parties (externally). And since we assume the parties of the channel are rational (and the external punishment exceeds the potential gain of cheating) they will not provide an alternate state history. Thus, every committed state is verifiable. □

## 7.2 Incentivizing honest behavior

To complete the proofs under the assumption of a rational committee, we need to show that the $2f+1$ rational committee members will provide the auditor the correct head of the hash-chain. Similarly to previously stated arguments in section 6.1, in case a member of the committee attempts to change the hash-chain during the `Pessimistic Close`, any channel party can construct a proof of fraud and claim the collateral of the committee member. Thus, any rational committee member will not attempt to commit fraud, since her loss overcomes her potential gain from bribing (from a channel party).

In addition, we incentivize the committee members to actually send the head of the hash-chain (participate) by rewarding them via the BRICK+ smart contract. This way we also enforce the auditor to pay the rewards he has committed in the smart contract to the responsive committee members.

# 8 EVALUATION OF PRIMITIVES

We have implemented consistent broadcast and consensus in Golang using the Kyber [34] cryptographic library and the cothority [18] framework. In Table 1 we evaluate our protocols on Deterlab [19] using 36 physical machines, each having four Intel E5-2420 v2 CPUs and 24 GB RAM and being arranged in a star-shaped virtual topology. In order to have a realistic wide area network we impose a 200ms roundtrip latency on the links between committee members and a 35Mbps bandwidth limit.

As illustrated the overhead of using a committee is around 0.1 seconds, while the close can take between 0.7 and 10 seconds depending both on the size of the committee and the size of the state. These numbers are acceptable compared to the latency of current blockchains, especially since channels are independent and embarrassingly parallel, meaning that we can deploy as many as we want without increasing the overhead.

# 9 RELATED WORK

Payment channels were originally introduced by Spilman [50], as a unidirectional off-chain solution, meaning that the sender can send to the receiver incremental payments off-chain via the channel, as long as the sender has enough capital on the channel. Later, bidirectional channels were introduced independently by Poon et al. [44] with Lightning Network, and Decker and Wattenhofer [17] with Duplex Micropayment Channels. Bidirectional channels allowed the capital locked in the channel to be moved in both directions from sender to receiver and back, like in a row of an abacus. All these solutions though require timelocks to guarantee safety, and thus make strong synchrony

**Table 1: Evaluation of Primitives used by BRICK and BRICK+**

| **Committee Size** | **4** | **34** | **151** |
|---|---|---|---|
| **Consistent Broadcast** | 0.1138 sec | 0.118 sec | 0.1338 sec |
| **ByzCoin Consensus (State Size: 500B)** | 0.643 sec | 1.927 sec | 1.949 sec |
| **ByzCoin Consensus (State Size: 1MB)** | 1.985 sec | 6.34 sec | 10.7 sec |

assumptions which sometimes fail in practice. BRICK, on the other hand, is the first bidirectional channel solution that does not require timelocks and operates under full-asynchrony.

Another safety requirement of the original payment channel proposals is that the participants of a channel are obligated to be constantly online and actively watching the blockchain. To address this issue, recent proposals introduce third-parties in the channel to act as proxies for the participants of the channel in case of fraud. This idea was initially discussed by Dryja [20], who introduced third-parties on Lightning channels, known as Monitors or Watchtowers [1]. Later, Avarikioti et al. [5] proposed a less centralised distributed protocol for the Watchtower service, where every full node can act as a Watchtower for multiple channels depending on the network topology. In parallel, McCorry et al. [37] proposed Pisa, a protocol that enables the delegation of Sprites [39] channels' safety to third-parties called Custodians. Simultaneously, Celer Network [46] proposed the State Guardian Network, a side chain that safeguards the off-chain transactions for a specific period of time when requested by a channel party. Similarly to these works, BRICK presents the committee who acts as a proxy for the parties of the channel. However, in this work the committee is more powerful than in the aforementioned works, since the protocol requires a fraction of responsive rational committee members to operate correctly. Furthermore, in contrast to previous work, the role of the third parties in BRICK (and BRICK+) is proactive instead of retroactive, because the committee's approval is needed before the parties of a channel execute a transaction.

Payment channels are specifically-tailored solutions that support only payments between users. This is quite limiting due to the emergence of smart contracts [51] that allow for arbitrary operations and computations. For this purpose, *State channels* were introduced [39], which are a generalized version of payment channels, with application on blockchain systems that support smart contracts. Recently, multiple state channel constructions have emerged, but they mainly focus on the routing problem of channel networks. In particular, Sprites [39] improve on the worst-case delay for releasing the collateral of the intermediate nodes on the payment network for multi-hop payments. In parallel, Perun [22] proposes a virtual payment hub, where every party can connect to and hence establish a "virtual channel" with any other party connected to the hub. In a similar manner, Counterfactual [13] presents "meta-channels", which are generalized state channels (not application-specific) with the same functionality as "virtual channels". However, all these constructions use the same foundations, *i.e.* the same concept on the operation of two-party channels. And as the fundamental channel solutions are flawed the whole construction inherits the same problems (synchrony and availability assumptions). In contrast, BRICK presents an alternative fundamental state channel solution that does not suffer from synchrony and availability assumptions to ensure safety and thus can be used as the foundation in all these constructions to alleviate their shortcomings.

## 10 CONCLUSION

BRICK is the first off-chain construction that operates securely under full-asynchrony. Furthermore, BRICK does not require online participants that maintain the blockchain to ensure safety. Instead, BRICK allows the parties of the channel to go offline and introduces a committee that acts as a proxy for the parties of the channel. To effectively remove timelocks, and thus achieve security under full asynchrony, the committee acts proactively, and guarantees that fraud cannot occur instead of acting upon the case of fraud. Overall, BRICK guarantees no fraud will occur (*safety*), and that the channel can be closed or updated at any time (*liveness*). Moreover, BRICK guarantees *privacy*, since all unauthorized external to the channel parties cannot learn any information on the state of the channel during the life of the channel.

In addition, we present BRICK+, a modification of the the main channel construction, that enables *auditability* of the channel by an authorized third-party. This functionality makes BRICK suitable for real world use where regulators can request access to the states of the channel. BRICK+ also maintains all previously stated properties, *safety*, *liveness* and *privacy*.

The security analysis of both BRICK and BRICK+ is done under the assumption that at most $f$ out of $3f + 1$ committee members are malicious and the rest are honest, while the participants of the channel are also considered honest. However, we later present incentive mechanisms, rewards and collateral, to guarantee the honest behavior of both the committee and the parties of the channel.

We evaluate the primitives used in BRICK and BRICK+, *i.e.* consistent broadcast and consensus. Specifically, we show that the overhead of using a committee is approximately 0.1 seconds per update, and can reach up to 10 seconds during the closing of the channel when both the size of the channel state and the committee are large.

## REFERENCES

[1] Merge blocks. https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud. Accessed: 2018-06-29.

[2] I. Abraham, G. Gueta, and D. Malkhi. Hot-stuff the linear, optimal-resilience, one-message BFT devil. *CoRR*, abs/1803.05069, 2018.

[3] I. Abraham, D. Malkhi, and A. Spiegelman. Validated asynchronous byzantine agreement with optimal resilience and asymptotically optimal time and

word communication. *CoRR*, abs/1811.01332, 2018.

[4] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis. Chainspace: A sharded smart contracts platform. *CoRR*, abs/1708.03778, 2017.

[5] G. Avarikioti, F. Laufenberg, J. Sliwinski, Y. Wang, and R. Wattenhofer. Towards secure and efficient payment channels.

[6] G. R. Blakley. Safeguarding cryptographic keys. In *Proceedings of the national computer conference*, volume 48, pages 313–317, 1979.

[7] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 514–532. Springer, 2001.

[8] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[9] C. Burchert, C. Decker, and R. Wattenhofer. Scalable Funding of Bitcoin Micropayment Channel Networks. In *19th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), Boston, Massachusetts, USA*, November 2017.

[10] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. In *19th ACM Symposium on Principles of Distributed Computing (PODC)*, July 2000.

[11] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, Feb. 1999.

[12] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395. IEEE, 1985.

[13] J. Coleman, L. Horne, and L. Xuanji. Counterfactual: Generalized state channels, 2018.

[14] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, et al. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*, pages 106–125. Springer, 2016.

[15] C. Decker, R. Russell, and O. Osuntokun. eltoo: A simple layer2 protocol for bitcoin.

[16] C. Decker, J. Seidel, and R. Wattenhofer. Bitcoin Meets Strong Consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN), Singapore*, January 2016.

[17] C. Decker and R. Wattenhofer. A fast and scalable payment network with bitcoin duplex micropayment channels. In A. Pelc and A. A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, pages 3–18, Cham, 2015. Springer International Publishing.

[18] DeDiS cothority, Sept. 2016. https://www.github.com/dedis/cothority.

[19] DeterLab Network Security Testbed, September 2012.

[20] T. Dryja. Unlinkable outsourced channel monitoring.

[21] S. Dziembowski, L. Eckey, and S. Faust. Fairswap: How to fairly exchange digital goods. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 967–984. ACM, 2018.

[22] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski. Perun: Virtual payment hubs over cryptographic currencies. Technical report, IACR Cryptology ePrint Archive 2017, 2017.

[23] J. Feigenbaum. Multiple Objectives of Lawful-Surveillance Protocols (Transcript of Discussion). In *Cambridge International Workshop on Security Protocols*, pages 9–17. Springer, 2017.

[24] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[25] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. Technical report, MASSACHUSETTS INST OF TECH CAMBRIDGE LAB FOR COMPUTER SCIENCE, 1982.

[26] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*, pages 281–310. Springer, 2015.

[27] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In *Eurocrypt*, volume 99, pages 295–310. Springer, 1999.

[28] L. Gudgeon, P. McCorry, P. Moreno-Sanchez, A. Gervais, and S. Roos. Sok: Off the chain transactions.

[29] G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in Bitcoin. In *19th ACM Conference on Computer and communications security*, pages 906–917. ACM, 2012.

[30] A. Kate and I. Goldberg. Distributed key generation for the internet. In *Distributed Computing Systems, 2009. ICDCS'09. 29th IEEE International Conference on*, pages 119–128. IEEE, 2009.

[31] E. Kokoris-Kogias, E. C. Alp, S. D. Siby, N. Gaillya, P. Jovanovic, L. Gasser, and B. Ford. Hidden in Plain Sight: Storing and Managing Secrets on a Public Ledger. Cryptology ePrint Archive, Report 2018/209, 2018.

[32] E. Kokoris-Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium*, 2016.

[33] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *Security and Privacy (SP), 2018 IEEE Symposium on*, pages 19–34. Ieee, 2018.

[34] The Kyber Cryptography Library, 2010 – 2018.

[35] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[36] D. Mazieres and D. Shasha. Building secure file systems out of byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*, pages 108–117. ACM, 2002.

[37] P. McCorry, S. Bakshi, I. Bentov, A. Miller, and S. Meiklejohn. Pisa: Arbitration outsourcing for state channels. *IACR Cryptology ePrint Archive*, 2018:582, 2018.

[38] A. Miller. Feather-forks: enforcing a blacklist with sub-50% hash power. https://bitcointalk.org/index.php?topic=312668.0, 2013.

[39] A. Miller, I. Bentov, R. Kumaresan, and P. McCorry. Sprites: Payment channels that go faster than lightning. *CoRR*, abs/1702.05812, 2017.

[40] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[41] H. Pagnia and F. C. Gärtner. On the impossibility of fair exchange without a trusted third party. Technical report, Technical Report TUD-BS-1999-02, Darmstadt University of Technology, Department of Computer Science, Darmstadt, Germany, 1999.

[42] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.

[43] T. P. Pedersen et al. Non-interactive and information-theoretic secure verifiable secret sharing. Springer.

[44] J. Poon and T. Dryja. The bitcoin lightning network: Scalable off-chain instant payments. 2015.

[45] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80. ACM, 1994.

[46] ScaleSphere Foundation Ltd. Celer network: Bring internet scale to every blockchain. https://www.celer.network/doc/CelerNetwork-Whitepaper.pdf. Accessed: 2019-05-08.

[47] A. Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, 1979.

[48] V. Shoup. Practical threshold signatures. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 207–220. Springer, 2000.

[49] Y. Sompolinsky, Y. Lewenberg, and A. Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptology ePrint Archive*, 2016:1159, 2016.

[50] J. Spilman. Anti dos for tx replacement. https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html. Accessed: 2019-04-17.

[51] N. Szabo. Formalizing and securing relationships on public networks. *First Monday*, 2(9), 1997.

[52] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. 2014.

[53] K. Wüst, K. Kostiainen, V. Capkun, and S. Capkun. Prcash: Centrally-issued digital currency with privacy and regulation. *Financial Cryptography and Data Security*, 2019.

# A SYCHRONOUS DISTRIBTUED KEY GENERATION

We assume the existence of $n$ servers participating in the DKG and each of them is in possession of a private-public key pair. The list of public keys is publicly known. The Pedersen DKG assumes the existence of a private channel between any pairs of participants and a broadcast channel available to all participants. Furthermore, this DKG scheme works in a synchronous setting, where an upper bound on the communication delay is fixed and known in advance. While this may be restrictive in today's global Internet era, a sufficient large timeout can simulate such synchrony. If such a synchrony assumption is too strong, Aniket et al. [30] provides a partially synchronous DKG variant, that can substitute the one described below.

We assume the existence of a cyclic group $\mathcal{G}$ of prime order $p$ and of a generator $G$ of $\mathcal{G}$.

*Key Generation:* The protocol follows several steps in order to securely generate a distributed key:

(1) Each party $P_i$ chooses a random polynomial $f_i(z)$ over $\mathbb{Z}_p^*$ of degree $t$:

$$f_i(z) = a_{i0} + a_{i1} * z + \cdots + a_{it} * z^t$$

(2) Each $P_i$ computes the individual secret shares $s_{ij} = f_i(j) \mod p$ for all $j \in \{1, \ldots, n\}$ and sends $s_{ij}$ to party $P_j$ using a confidential point-to-point channel. We denote $a_{i0}$ by $x_i$, the individual secret contribution to the distributed private key.

(3) Each $P_i$ broadcasts the commitment to the coefficients $A_{ik} = G^{a_{ik}}$ for all $k \in \{0, \ldots, k\}$ to all other participants. We denote $A_{i0}$ by $X_i$, the individual public contribution to the distributed public key.

(4) Each $P_j$ verifies the share received from the other parties by checking, for all $i \in \{1, \ldots, n\}$:

$$G^{s_{ij}} = \prod_{k=0}^{t} (A_{ik})^{j^k} \mod p \tag{1}$$

If the check fails for an index $i$, $P_j$ broadcasts a *complaint* against $P_i$.

(5) For each complaining party $P_j$, party $P_i$ reveals the corresponding share $s_{ij}$ matching (1). If any of the revealed shares fails this equation, $P_i$ is disqualified. We define the set QUAL to be the set of non-disqualified parties.

(6) The public key is defined as $X = \prod_{i \in QUAL} X_i$. Each party $P_j$ sets his share of the secret to $x_j = \sum_{i \in QUAL} s_{ij} \mod p$. The public verification values are computed as $A_k = \prod_{i \in QUAL} A_{ik}$. The distributed secret key is defined as $x = \sum_{i \in QUAL}^{t} x_j * \lambda_i$, where $\lambda_i$ is the $i$-th Lagrange coefficient.