Brick: Asynchronous Payment 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 (channels) are a promising solution to the scalability and privacy challenges of blockchain systems. Current proposals, however, require synchrony assumptions to preserve the safety of a channel, leaking to an adversary the exact amount of time needed to control the network for a successful attack. In this paper, we introduce Brick, the first off-chain construction that remains secure under network asynchrony and concurrently provides correct incentives. The core idea is to incorporate the conflict resolution process within the off-chain channel by introducing a rational watchtower committee. Hence, if a party wants to close a channel unilaterally, it can only get the committee's approval for the last valid state. Brick provides sub-second latency because it does not employ heavy-weight consensus. Instead, Brick uses *consistent broadcast* to announce updates and close the channel, a light-weight abstraction that is powerful enough to preserve safety and liveness to honest parties. Furthermore, we consider the permissioned model of blockchains, where the additional property of auditability might be desired for regulatory purposes. We introduce Brick+, an off-chain construction that provides auditability on top of Brick without conflicting with its privacy guarantees. We formally define the properties our state channel construction should fulfill, and prove that both Brick and Brick+ satisfy them. We also design incentives for Brick such that honest and rational behavior aligns for both the committee and the parties of a channel.

1 INTRODUCTION

A promising solution to the scalability challenge [9] of large-scale blockchains, are the so-called *channels* [12, 36, 39]. The idea is that any two parties that interact (often) with each other can set up a joint account on the blockchain (i.e., a channel), and transact off-chain, relying on the blockchain as a fail-safe mechanism in case of disputes.

The security guarantees of a channel are ensured by the on-chain dispute handling mechanism which allows parties to challenge the closing of the channel on an invalid (outdated) state within a predefined time period t. Hence, channels are secure as long as all parties of the channel are frequently – at least once every t time – online and monitoring the blockchain. This design suffers from two major shortcomings. First, the channel parties must be frequently online, monitoring the blockchain. Second, the publicly-announced dispute period t leaks the exact amount of time an adversary needs to control the network in order to successfully commit fraud. Thus, to maintain security, this construction requires network synchrony and a perfect blockchain that will correctly handle disputes [20].

The community has tried to resolve the first challenge using semi-trusted third parties called *watchtowers* [3, 15, 21, 31]. However, in all existing watchtower solutions, the second challenge of demanding a synchronous network and a perfect blockchain persists. This means that a malicious party able to censor[1] transactions from an honest party or a watchtower during the dispute period can break the security of a channel by attacking the liveness[2] of the underlying blockchain. Unlike blockchain protocols where attacking liveness can slow down the system but cannot result in the loss of funds, the following attack can occur in channels: Suppose a malicious party publishes an outdated signed state of the channel that awards the party more funds than the last signed state. Further, suppose the malicious party successfully censors all on-chain transactions from the counterparty and the watchtowers during the dispute period. Then, the malicious party will successfully manage to steal channel funds from the counterparty, simply by attacking the liveness of the blockchain (see Appendix A). Unfortunately, these attacks are feasible [32], and thus the aforementioned assumptions are invalid in practice.

To address these issues we introduce Brick, a novel incentive-compatible state channel construction that does not rely on any timing assumption for the delivery of messages to be secure. Brick provides proactive security, detecting and preventing fraud before it appears on-chain. As a result, Brick can guarantee the security of channels even under censorship [32] or any liveness attack.

To achieve these properties, Brick needs to address three key challenges. The first challenge is on how to achieve this proactive check without using a trusted third party that approves every transaction. The core idea of Brick is to enable the participants of the channel to outsource the dispute arbitration to a group of watchtowers instead of a single one. As a result, if there is a dispute, the watchtowers 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. The second challenge for Brick is that for the watchtowers to simulate this trusted third party, it would need to run costly asynchronous consensus [27] for every update. Instead, in Brick we show that a light-weight consistent broadcast protocol is enough to preserve the safety and liveness of the channel.

A final challenge of Brick that we address is to not only handle Byzantine watchtowers but also align the honest behavior to the rational behavior by employing correct incentives. Unfortunately, existing watchtower solutions do not align the expected and rational behavior of the watchtower, hence a watchtower is reduced to a trusted third party. Specifically, Monitors [15], Watchtowers [21], and DCWC [3] pay the watchtower upon fraud. Given that the use of a watchtower is public knowledge, any rational channel party will not commit fraud and hence the watchtower will never be paid. Therefore, there is no actual incentive for a third party to offer a

[1] This censoring ability is encompassed by the chain-quality property [19] of blockchain systems which is rightly bound to the synchrony of the network.
[2] Liveness states that a transaction will be eventually included on-chain as long as it is provided to all honest parties for a long enough time period [19].

watchtower service. On the other hand, Pisa [31] pays the watchtower regularly every time a transaction is executed on the channel. The watchtower also locks collateral on the blockchain in case it misbehaves. However, Pisa's collateral is not linked to the channel or the party that employed the watchtower. Hence, a watchtower that is contracted by more than one channel can double-assign the collateral, making Pisa vulnerable to bribing attacks. Even if the incentives of Pisa get fixed, punishing a misbehaving watchtower in Pisa is still a sychronous protocol (longer than the dispute period). In Brick, we employ both rewards and punishment to design the appropriate incentive mechanisms such that honest and rational behavior of watchtowers align, while no synchrony assumptions are required, i.e., the punishment of misbehaving watchtowers is not conditional on timing assumptions.

We also present Brick+, a channel construction suitable for regulated environments such as supporting a real currency [42]. The challenges Brick+ needs to solve are on how to provide auditability of transactions without forfeiting privacy [20] and while preserving the accountability of the auditor. To resolve this we change Brick in two ways. First, the state updates are no longer just private but also interconnected in a tamper-evident hash-chain. Essentially, the watchtowers maintain a single hash (constant-size storage cost) which is the head of the hash-chain of the state history. Second, to provide accountability for the auditor, we require that the auditor posts the access request on-chain [24]. Only then will the committee provide the auditor the necessary metadata to verify the state history received from the parties of the channel.

To evaluate our channel construction we deploy our protocol on a large-scale testbed and show that the overhead of an update is around the round-trip latency of the network (in our case 0.1 seconds). Unlike existing channels, the parties in Brick need not wait for the dispute transaction to appear on-chain. Hence, our dispute resolution mechanism is two orders of magnitude faster than existing systems that need to wait until the transaction is finalized on-chain. The evaluation can be found in Appendix D.

In summary, this paper makes the following contributions:

- We introduce Brick, the first incentive-compatible off-chain construction that operates securely with offline channel participants unde full asynchrony with sub-second latency.
- We introduce Brick+, a modification on Brick channel construction, that enables external auditors to lawfully request access to the channels history while maintaining privacy.
- We define the desired channel properties and show they hold for Brick under rational participants (channel parties and watchtowers). Specifically, we present elaborate incentives mechanisms (rewards and punishments) for the watchtowers to maintain the channel properties under any collusion or bribing scheme.

2 BACKGROUND AND PRELIMINARIES

In this section, we provide the necessary introduction to state channels. Furthermore, we introduce the necessary distributed abstractions and cryptographic primitives that our constructions build 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 [34]. 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. Notably, Bitcoin handles at most seven transactions per second [9], 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 [11, 25, 26], 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[3] to create a "channel" on the blockchain through which they can transact fast and secure; this solution is known as *payment channel* [12, 36, 39].

Payment channels allow transactions between two parties to be executed instantly off-chain while maintaining the guarantees of the blockchain. Essentially, the underlying blockchain acts as a "judge" in case of fraud. There are multiple proposals on how to construct payment channels [10, 12, 36, 39], but all proposals share the same core idea: The two parties create a joint account on the blockchain (funding transaction). 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)[4]. 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, thus violating the safety property. Lightning channels [36] 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 releases a secret to the counterparty. That secret enables the counterparty to claim the money of the channel in case the party publishes the previous update transaction (breach remedy). However, this transaction is

[3] Note that a channel can also be created between multiple parties [7].
[4] By definition, a channel requires the participation and signature of every party of the channel to update the channel's state in order to maintain security.

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 [12] guarantee that the latest update transaction will become valid before any previous update transaction, again utilizing timelocks. In both cases, the liveness of the underlying blockchain and timelocks are crucial to the safety of the payment channel solution. Additionally, both solutions require online participants that frequently monitor the blockchain to ensure safety.

2.2 Consistent Broadcast

Consistent broadcast [37] is a distributed protocol run by a node that wants to send a message to a set of peers reliably. 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 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 the 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.

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 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 [19]. However, we do not require a "perfect" blockchain system since Brick can tolerate temporary liveness attacks. Specifically, if an adversary temporarily violates the liveness property of the underlying blockchain, this may result in violating the liveness property of channels but will not affect the safety of the 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 watchtowers and we define a threshold $t = 2f + 1$ to achieve the liveness and safety properties. The non-Byzantine part of the committee is assumed rational; we first assume t honest watchtowers to prove the protocol goals, and subsequently incentivize this honest behavior for rational watchtowers.

3.3 Brick Overview

Both parties of a channel agree on a committee of watchtowers before opening the channel. The watchtowers 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 both parties or by a transaction signed by one of the parties and a fraction of the watchtowers. Thus, the committee acts as power of attorney for the parties of the channel. Furthermore, Brick employees correct incentives for the rational parties to follow the protocol, hence it can withstand $t = 2f + 1$ rational and f Byzantine watchtowers while the channel parties are assumed to be rational.

A naive solution would then instruct the committee to run consensus on every new update, which would cost $O(n^4)$ [27] per transaction, a rather big overhead for the critical path of our protocol. Instead in Brick, consensus is not necessary for update transactions, as we only provide liveness guarantees to rational parties (if they misbehave they will lose their funds). As a result, every time a new update state occurs in the channel, the parties run a consistent broadcast protocol (cost of $O(n)$) with the committee. Specifically, a party announces to the committee that a new update has occurred. This announcement (defined below) is signed by all the parties of the channel to signal that they are in agreement. In addition, this announcement (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 freshest state. If the consistent broadcast protocol succeeds (t watchtowers acknowledge reception) then this can serve as proof for both parties that the state update is safe. After this procedure terminates correctly, both parties proceed to the execution of the off-chain state.

At the end of the life-cycle of a channel, a dispute might exist, leading to the unilateral closing of the channel. Even in this case, we can still guarantee the security and liveness of the closing with consistent broadcast. The crux of the idea is that if $2f + 1$ committee members replied at the update before receiving the closing request (hence the counterparty has committed) then at least one honest member will not accept the closing at an older state and instead reply to the party that he can only close at the new state. As a result we define a successful closing to be at the maximum of all proposed states, which guarantees safety. Although counter-intuitive, this closing process is safe because the transactions are already totally ordered and agreed to by the parties of the channel; thus, the committee simply acts as shared memory for the channel.

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 or illegal. This is a realistic case as the scalability in payment channels comes from a persistent relationship that models 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 must ensure that (a) nothing happens without the committee's approval and (b) a sufficient audit trail is left on-chain to stop regulators from misbehaving. We resolve the first issue by disabling the parties' ability to close the channel without the participation of the committee and by additionally having the committee to store the hash-chain of the state history. To prevent the auditor from misbehaving, we force the auditor to post a "lawful access request" [18] on-chain to convince the channel parties to initiate the closing of the channel for audit purposes and send the state history to the auditor.

3.5 Reward Allocation & Collateral

To avoid bribing attacks, we enforce the watchtowers to lock collateral in the channel. The total amount of collateral is proportional to the value of the channel meaning that if the committee size is large, then the collateral per watchtower is small. More details on the necessary amount of collateral are thoroughly discussed in Sections 5.2 and 8. Additionally, the committee is incentivized to actively participate in the channel with a small reward that each watchtower gets when they acknowledge a state update of the channel. This reward is given with a unidirectional channel [20], which does not suffer from the problems Brick solves. Moreover, the watchtowers 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.

3.6 Protocol Goals

To define the goals of Brick, we first need to define the necessary properties of a channel construction. Intuitively, a channel should ensure similar properties with a blockchain system, i.e., a party cannot cheat another party out of its funds, and any party has the prerogative to eventually spend its funds at any point in time. The first property, when applied to channels, means that no party can cheat the channel funds of the counterparty, and is encapsulated by *Safety*. The second property for a channel solution is captured by *Liveness*; it translates to any party having the prerogative to eventually close the channel at any point in time. We say that a channel is *closed* when the locked funds of the channel are spent on-chain, while a channel *update* refers to the off-chain change of the channel's state.

In addition, we define *Privacy* which is not guaranteed in many popular blockchains, such as Bitcoin [34] or Ethereum [41], 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 for use on a regulated 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 both 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:*

> *Both parties of the channel have signed the state s_i.*
> *The state s_i is the **freshest** state, i.e., no subsequent state s_{i+1} is valid.*
> *The committee has not invalidated the state. The committee can invalidate the state s_i if the channel closes in the state s_{i-1}.*

Definition 2. *A state of the channel is **committed** if it was signed by at least $2f+1$ watchtowers or is valid and part of a block in the persistent part of the blockchain.*

Definition 3 (Safety). *A Brick channel will only close in the freshest committed state.*

Definition 4 (Liveness). *Any valid operation (update, close) on the state of the channel will eventually[5] be committed (or invalidated).*

Definition 5 (Privacy). *No unauthorized[6] external (to the channel) party learns about the state of the channel (e.g., the current distribution of funds between the parties of a payment channel) 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*, 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 two parties. In this phase, the parties create the initial funding transaction, similarly to other known payment channels such as [12, 36]. However, in Brick we also define three additional parameters during this phase; we include in the funding transaction the hashes of the public keys of the watchtowers of the channel, denoted by $W_1, W_2, \ldots, W_n$, the threshold t, and a closing fee F. This fee will be awarded to the responsive watchtowers in the last phase, *Close*, if and only if the closing of the channel is done in collaboration with the committee. In this case, the watchtowers that assisted in closing the channel will be rewarded with the amount F divided by the number of participating watchtowers t. In addition, each watchtower locks a collateral in the Brick smart contract which will be claimed by the parties of the channel during phase *Close* if a watchtower misbehaves (e.g., receives 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 using

[5] Depending on the message delivery.
[6] Authorized parties are potential auditors of the channel, as described in Brick+.

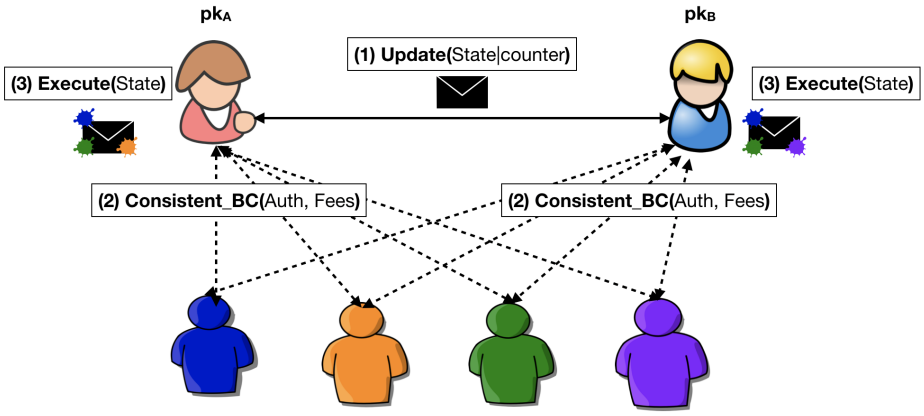

Figure 1: Typical workflow of Brick for a state update. (1) Alice and Bob agree on a new state update. (2) They individually broadcast the update to the committee (with an associated fee). (3) When a threshold of the committee replies that the broadcast update state is committed, each sender (Alice or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

Protocol 1: `Open`

Input: Channel parties A, B, watchtowers $W_1, W_2, \ldots, W_n$, initial state s_1, closing fee F.
Goal: Open a Brick payment channel.

1. Both parties A, B sign: $open^1 H^1 W_1{}^\circ, H^1 W_2{}^\circ, \ldots, H^1 W_n{}^\circ, t, s_1, F^\circ$.
2. Register to $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$ the announcement of Protocol `Update`$(A, B, s_1, null)$.
3. Execute Protocol `Consistent Broadcast`$(M, \sigma^1 M^\circ, A, B, W_1, W_2, \ldots, W_n)$.
4. Publish to the blockchain $\sigma^1 open^1 H^1 W_1{}^\circ, H^1 W_2{}^\circ, \ldots, H^1 W_n{}^\circ, t, F^{\circ\circ}$.

Protocol 2: `Update`

Input: Channel parties A, B, current state s.
Goal: Create announcement $M, \sigma^1 M^\circ$ (cloaked state signed by both parties).

1. Both parties A, B sign: $\mathtt{f}H^1 s_i, r_i{}^\circ, i\mathtt{g} = M$, where r_i is a random number and s_i the current state, thus creating the announcement $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$.

Protocol 3: `Consistent Broadcast`

Input: Channel parties A, B, watchtowers $W_1, W_2, \ldots, W_n$, announcement $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$, reward r.
Goal: Inform the committee of the new update state and verify the validity of the new state.

1. Each party broadcasts to all the watchtowers $W_1, W_2, \ldots, W_n$ the announcement $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$ alongside fee r.
2. Each watchtower W_j, upon receiving $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$, verifies that both parties' signatures are present, and the sequence number is exactly one higher than the previously stored sequence number. If the watchtower has published a closing state, it ignores the update state. Otherwise, W_j stores the announcement $\mathtt{f}M, \sigma^1 M^\circ \mathtt{g}$ (replacing the previous announcement), signs M, and sends the signature $\sigma_{W_j}{}^1 M^\circ$ to the parties that payed the update fee r.
3. Each party, upon receiving at least t signatures on the announcement M, considers the state committed and proceeds to the state transition.

`Consistent Broadcast`. The announcement is the hash of the new state (to preserve privacy), as well as the sequence number, signed by both parties of the channel[7]. This way both 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 the counterparty or the committee. At the same time, the state of the channel is not revealed to the committee. Nevertheless, if a party wants to close the channel unilaterally, the watchtowers can provide a signature on the stored announcement $\sigma_{W_i}{}^1 M, close^\circ$. Consequently, the Brick smart contract, given the correct state by the party, can verify the hash and the appropriate signatures and close the channel in the correct state.

In Protocol 3, for every state update, each party sends to all watchtowers the announcement including a small *update fee* for watching the channel. Then, each watchtower replies to every party that sent the announcement by signing the announcement. The watchtower's signature can be used later to penalize the watchtower (a party can claim the watchtower's collateral) in case the watchtower colludes with a party and signs a previous update state to close the channel.

[7]We abuse the notation of signature σ to refer to the multisig of both A and B.

The last phase of the protocol, *Close*, can be implemented in two different ways: the first is similar to the traditional approach for closing a channel (Protocol 4: `Optimistic Close`) where both parties collectively sign the freshest update state (closing transaction) and publish it on-chain. However, in case a channel party is not

Protocol 4: `Optimistic Close`

Input: Channel parties A, B, state s.
Goal: Close a channel on state s, assuming both parties are responsive and in agreement.

1. A party $p \in \{A, B\}$ broadcasts the request $close^{1}s^{0}$.
2. Both parties A, B sign the state s (if they agree) and exchange their signatures.
3. The party p (or any other channel party) publishes the signed by both parties state, $\sigma_{A,B}{}^{1}s^{0}$ on-chain.

Protocol 5: `Pessimistic Close`

Input: Party $p \in \{A, B\}$, watchtowers $W_1, W_2, \ldots, W_n$, state s_i, random number r_i.
Goal: Close a channel on state s_i with the assist of the committee.

1. Party p broadcasts to the watchtowers $W_1, W_2, \ldots, W_n$ the request $close^{1^0}$.
2. Each watchtower W_j publishes on-chain a signature on the (last) stored announcement $\sigma_{W_j}{}^{1}M, close^{0}$ and stops signing new update states.
3. Party p, upon verifying t on-chain watchtower signatures, recovers the $max^{1}i^{0}$ that is included in the announcements. Then, party p publishes state s_i on-chain along with the signatures of both parties on the announcement that corresponds to state s_i.
4. After the state is included in a (permanent) block, the smart contract closes the channel in state s_i and every watchtower whose signature is published on the blockchain gets an equal fraction of the closing fee F (fee locked in the funding transaction).

responding to new updates or closing requests, the counterparty 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 from each watchtower its signature on the stored announcement. A watchtower, upon receiving the closing request, publishes on-chain a closing announcement, i.e., the stored announcement signed along with a flag close. When t closing announcements are on-chain, the party recovers the state that corresponds to the maximum sequence number from the closing announcements. Then, the party publishes this state along with the signatures of both parties on the corresponding announcement on-chain. As soon as these data are included in a (permanent) block the Brick smart contracts verifies and closes the channel. In particular, the smart contract verifies that the submitted state corresponds to the hash with the maximum sequence number, that there are t submitted announcement that correspond to watchtower identities committed on-chain in Protocol 1, and that the signatures of both parties are present. If all verifications check the smart contract closes the channel in the submitted state and rewards $F \bullet t$ to each of the t participating watchtowers.

5 BRICK SECURITY ANALYSIS

We first prove Brick satisfies the protocol goals as defined in Section 3.1, i.e., *Safety, Liveness* and *Privacy*. Then, we design the incentive mechanisms to incentivize honest behavior to all rational players. We show that rational watchtowers will honestly follow the protocol in Appendix B.2. Last, we argue for rational channel parties thus eliminating the assumption that the participants of a channel are honest (Appendix B.3). Essentially, we show that Brick enriched with the proposed incentive mechanisms is incentive-compatible. Omitted proofs can be found in Appendix B.1.

5.1 Security Analysis under Honest Participants

Theorem 1. Brick *achieves safety.*

Proof. 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`), both parties agree on closing the channel in a specific state (which is always the freshest valid state [8]). 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. The Brick smart contract verifies that the state is valid, i.e., the signatures of both parties are present and the sequence number of that state is the maximum from the submitted announcements. Given the validity of the closing state, it is enough to show that the channel cannot close in a state with a sequence number smaller than the one in the freshest committed state. This holds because even if the channel closes in a valid but not yet committed state with sequence number larger than the freshest committed state, this state will eventually become the freshest committed state (similarly to Protocol 4).

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 freshest state agreed by both parties. We will prove safety by contradiction. For the channel to close at s_i at least $t = 2f + 1$ watchtowers have provided a signed closing announcement, and the maximum sequence number of these announcements is i. Otherwise the Brick smart contract would not have accepted the closing state as valid. According to the threat model, at most $n - t = f$ watchtowers are Byzantine, thus at least $f + 1$ honest watchtowers have submitted a closing announcement. Hence, none of the $f + 1$ honest watchtowers have received and signed the announcement of any state with sequence number greater than i. 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 $t = 2f + 1$ watchtowers. Since at most $n - {}^{1}f + 1^{0} = 3f + 1 - f - 1 = 2f \leq 2f + 1$ watchtowers have seen (and hence signed) the state s_k, the state s_k is not committed. Contradiction.

[8] 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.

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*. We say that operation close applies in a state s if this state was published on-chain either in collaboration of both parties (Protocol 4) or unilaterally by a channel party as the closing state (Protocol 5, step 3).

If the operation is close and is not committed, then either the parties did not agree on this operation (`Optimistic Close`), or a verification of the smart contract failed (`Pessimistic Close`). In both cases, the operation is not valid.

Suppose now the operation is close and never invalidated. Then, if it is an optimistic close, all the parties of 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, the state will be eventually included in a block in the persistent part of the blockchain and thus will be eventually committed. On the other hand, if it is a pessimistic close and not invalidated, the smart contact verifications were successful therefore the state was 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 watchtowers eventually received the update state (`Consistent Broadcast`). However, the update state was never committed, therefore at least $f + 1$ watchtowers did not sign the update state. We assumed at most f Byzantine watchtowers, hence at least one honest watchtower did not sign the valid update state. According to Protocol 3 (line 2), an honest watchtower does specific verifications and if the verifications hold the watchtower signs the new update state. Thus, for the honest watchtower 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 watchtower has published previously a closing announcement on-chain and ignores the update state. In this case, either (a) the closing state of the channel is the update state - submitted by another watchtower that received the update before the closing request - or (b) the closing state had a smaller sequence number from the update state. In the first case (a), the update state is committed eventually (on-chain), while in the second case (b) the update state is invalidated as the channel closed in a previous state.

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.

Theorem 3. Brick *achieves privacy.*

5.2 Incentivizing Honest Behavior

There are three incentive mechanisms in Brick:

(1) *Update Fee (r).* The parties establish a one-way channel [20] with each watchtower where they send a small payment every time they want a signature for a state update. Note that the update fee is awarded to the watchtowers at step 1 of Protocol 3.
(2) *Closing Fee (F).* If the channel closes in collaboration with the committee (`Pessimistic Close` protocol), the participating watchtowers are awarded a closing fee. Specifically, during phase `Open` (Protocol 1), the parties lock a closing fee F in the channel, which is split only among the watchtowers that participated in the consensus process in Protocol 5.
(3) *Watchtower Collateral.* During phase *Open*, each watchtower locks collateral at least equal to the amount locked in the channel v divided by f. The collateral is either returned to the watchtower at the closing of the channel or claimed by a channel party that provides a proof-of-fraud. A proof-of-fraud consists of two conflicting messages signed by the same watchtower: either (a) a signature on an announcement on a state update of the channel, and (b) a signature on an announcement for closing on a previous state of the channel, or (c) two signatures on announcements with the same sequence number but different states (hashes).
In case, a party submits the closing announcements and at most f proofs-of-fraud, to close the channel we execute a second closing process excluding the watchtowers that committed fraud. Then, the channel closes in the state with the maximum sequence number of the states/announcements submitted by the remaining parties. On the other hand, if a party submits the closing announcements and at least $f + 1$ proofs-of-fraud, the party that submitted the proofs-of-fraud claims only the collateral and the entire channel balance is awarded to the counterparty. Lastly, if no proofs-of-fraud are submitted the channel closes as described in Protocol 5.

We further demand that the size of the committee is at least $n \geq 7$, hence $f \geq 2$. As a result, we guarantee there is at least one channel party with locked funds greater than each individual watchtower's collateral, $\frac{v}{2} \geq \frac{v}{f}$. This restriction along with the aforementioned incentive mechanisms ensure resistance to collusion and bribing of the committee, meaning that following the protocol is the dominant strategy for the rational watchtowers ($2f + 1$). This way we show that Brick achieves the protocol goals even under a rational committee, while allowing f Byzantine watchtowers (Appendix B.2). We complete our analysis by showing rational parties will behave honestly in Appendix B.3.

6 BRICK+ DESIGN

In this section, we describe the Brick+ system design, while the security analysis for Brick+ can be found in Appendix C.

Brick+ consists, similarly to Brick, of three phases, *Open*, *Update*, 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. The *Audit* functionality is illustrated in Figure 2 (Appendix C) and described in detail in Protocol 6.

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,

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-chain the access request for channel c.
2. Each watchtower of the channel, upon verifying the validity of the access request, initiates the closing phase of the channel (Protocol 5), and publishes on chain the stored announcement (head of the hash-chain of state history).
3. After the execution of Protocol 5, the channel is closed on-chain in a committed state s. Then, both parties of the channel send to the auditor the state history.
4. The auditor collects from the blockchain the hashes published by the watchtowers and selects the hash that corresponds to the maximum sequence number (i.e., corresponds to the closing state s). Then, the auditor, upon receiving the state history from both parties, verifies it 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.

we modify Protocol 2 and Protocol 3 (phase *Update*) such that the watchtowers store a hash-chain of the state history instead of the hash of the freshest valid state they received. Thereby, we ensure that the parties cannot present an alternate state history to the auditor as it achieves fork-consistency [30].

The *Audit* functionality is initiated by an authorized third party, the auditor, who publishes an access request on-chain. Then, the watchtowers of the channel verify the validity of the access request and initiate the closing of the channel by publishing the closing announcements (head of the hash-chain of the state history) on-chain (Protocol 5: `Pessimistic Close`). After the execution of Protocol 5, the closing state is on-chain and both the (honest) watchtowers and parties of the channel have a consistent view of the channel history. Both parties send to the auditor the entire state history. The auditor then verifies the state history received from every party by computing the hash-chain and comparing the last hash with the hash that corresponds to the maximum sequence number published by the watchtowers (i.e., the hash that corresponds to the closing state of the channel). If the parties misbehave and send an alternate state history the auditor can pursue external punishment (e.g., legal action).

7 RELATED WORK

Payment channels were originally introduced by Spilman [39], 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. with Lightning Network [36], and Decker and Wattenhofer with Duplex Micropayment Channels [12]. 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 assumptions that sometimes fail in practice. Brick, on the other hand, is the first bidirectional channel solution that does not require timelocks and operates securely under full asynchrony.

Another safety requirement of the original payment channel proposals is that the participants of a channel are obligated to be frequently online, actively watching the blockchain. To address this issue, recent proposals introduced 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 [15], who introduced third-parties on Lightning channels, known as Monitors or Watchtowers [21]. Later, Avarikioti et al. proposed DCWC [3], a less centralized distributed protocol for the watchtower service, where every full node can act as a watchtower for multiple channels depending on the network topology. In both these works, the watchtowers are paid upon fraud. Hence, the solutions are not incentive-compatible since in case a watchtower is employed no rational party will commit fraud on the channel and thus the watchtowers will never be paid. This means there will be no third parties offering such a service unless we assume they are altruistic.

In parallel, McCorry et al. [31] proposed Pisa, a protocol that enables the delegation of Sprites [33] channels' safety to third-parties called Custodians. Although Pisa proposes both rewards and penalties similarly to Brick, it fails to secure the channels against bribing attacks. Particularly, the watchtower's collateral can be double-spent since it is not tied to the channel/party that employed the watchtower. More importantly, similarly to Watchtowers and DCWC, Pisa demands a synchronous network and a perfect blockchain, meaning that transactions must not be censored, to guarantee the safety of channels. On the contrary, Brick is provably secure under rational parties and watchtowers and operates under full asynchrony. Similarly to Watchtowers and Pisa, Brick also maintains privacy from external to the channel parties, i.e., all the off-chain update states are visible only to the parties of the channel (not to the watchtowers).

Concurrently to this work, Avarikioti et al. introduced Cerberus channels [4], a modification of Lightning channels that incorporates rational watchtowers to Bitcoin channels. Although Cerberus channels are incentive-compatible, they still require timelocks. Thus, in contrast to Brick, their security depends on the synchrony assumptions and a perfect blockchain that cannot be censored. Furthermore, Cerberus channels do not guarantee privacy from the watchtowers, as opposed to Brick and Brick+, where unauthorized external parties have no information on the state of the channel.

We summarize the comparison to previous work in Table 1. We exhibit the differences of Brick to other channel constructions, such as Lightning, as well as other watchtower solutions, such as Monitors, Pisa and Cerberus. Specifically, we observe that Brick is the only Layer 2 solution that maintains security under an asynchronous network and offline channel parties while assuming rational

[9] The party needs to be able to deliver messages and punish the watchtower within a large synchrony bound T.
[10] The watchtower needs to lock collateral per-channel, equal to the channel's value. Current implementation of Pisa does not provide this.
[11] Possible if consensus is run for closing the channel as described in Section 8.

Table 1: Comparison with previous work

Safe under	Lightning [36]	Monitors [15]	Pisa [31]	Cerberus [4]	Brick
Asynchronous Network	✗	✗	✗	✗	✓
Offline Parties	✗	✓	$T \gg t$[9]	$T \gg t$[9]	✓
Rational Watchtowers	✗	✗	~[10]	✓	✓
Censorship	✗	✗	✗	✗	✓
Congestion	✗	✗	✗	✗	✓
Forks	✗	✗	✗	✗	✓[11]
Bitcoin Compatible	✗	✓	✗	✓	✗

watchtowers. Further, Brick is secure even when the blockchain substrate is censored, and also when the network is congested. Finally, an extension to Brick that we describe in Section 8 enables protection against small scale persistence attacks making it more secure than the underlying blockchain.

Towards a different direction, Leinweber et al [29] recently proposed TEE-based distributed watchtowers to reduce the storage costs for watchtowers in Bitcoin. Simultaneously, Khabbazian et al. presented Outpost [23], a lightweight watchtower design for Bitcoin in which watchtowers can extract the secrets to revoke a fraudulent commitment transaction directly from the commitment transaction that appeared on-chain. Both these works aim to reduce the storage cost for watchtowers in Bitcoin channels. In Brick, the storage cost is constant since the watchtowers only store the hash of the last state.

Payment channels are specifically-tailored solutions that support only payments between users. To extend this solution to handle smart contracts [40] that allow arbitrary operations and computations, *state channels* were introduced [33]. Recently, multiple state channel constructions have emerged, but they mainly focus on the routing problem of channel networks. In particular, Sprites [33] 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 [17] 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 [8] 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 under rational participants and thus can be used as the foundation in all these constructions to alleviate their shortcomings.

Lastly, another solution for scaling blockchains is sidechains [5]. In this solution, the workload of the blockchain (main chain) is transferred in other chains, the sidechains, which are "pegged" to the main chain. Although the solution is kindred to channels, it differs significantly in one aspect: in channels, the states updates are totally ordered and unanimously agreed by the parties thus a consensus process is not necessary. On the contrary, sidechains must operate a consensus process to agree on the validity of a state update. Brick lies in the intersection of the two concepts; the states are totally ordered and agreed by the parties, whereas watchtowers merely remember that agreement was reached at the last state announced.

8 DISCUSSION, LIMITATIONS, AND EXTENSIONS

This section briefly outlines several of Brick's important remaining limitations, possible extensions, and discussion on design choices.

Byzantine players. In the threat model, we assume that channel parties are rational. If both parties are byzantine then the watchtowers' collateral can be locked arbitrarily long since the parties can simply crash forever. Nevertheless, the actual assumption we need is that only the richest party of the channel is rational and starts the pessimistic close protocol; the counterparty can be byzantine and Brick would still guarantee that the party does not lose money. Note that the richest channel party has locked a higher amount of funds than any watchtower, hence its "cost" is higher for holding the watchtowers hostage. The second threshold assumption of Brick is that at most f out of the $3f + 1$ watchtowers are byzantine. This threshold is necessary to maintain safety and is in accordance to well known lower bounds for asynchronous consistent broadcast.

Watchtower Unilateral Exit. As mentioned above, if both parties become malicious they might hold the watchtowers' collateral hostage. A similar situation is indistinguishable from the parties not transacting often. As a result the watchtowers might want to exit the channel. A potential extension can support this in two ways.

The first functionality we can incorporate in Brick, is committee replacement. In particular, we can allow a watchtower to withdraw its service as long as there is another watchtower willing to take its place. In this case, we simply replace the collateral and watchtower identities with an update of the funding transaction on-chain, paid by the watchtower that requests to withdraw its service.

Second, if a significant number (e.g, $2f + 1$) of watchtowers declare they want to exit, the smart-contract can release them and convert the channel to a synchronous channel [31]. The parties will now be able to close the channel unilaterally by directly publishing the last valid state. If the counterparty tries to cheat and publishes an old state, the party (or any remaining wathctower) can catch the dispute on-time and additionally claim the (substantial) closing fee.

Commi ee Selection. Each channel has its own group of watchtowers, i.e., the committee is independently selected for each channel by the channel parties. The scalability of the system is not affected by the use of a committee since each channel has its own independent committee of watchtowers. The size of the committee for each channel can vary but is constrained by the threat model. If we assume at least one honest party in the channel, a single rational watchtower is enough to guarantee the correct operation of Brick. Otherwise, we require at least 7 watchtowers to avoid hostage situations from colluding channel parties (see Section 5.2).

The main purpose of introducing a watchtower committee is to increase the fault tolerance of the protocol. The cost of maintaining the channel security for the parties is not dependent on the committee size, but on the value of the channel. If the parties chose a small committee size, the collateral per watchtower is high, thus the update fees are few but high. On the other hand, if the parties employ many watchtowers, the collateral per watchtower is low, thus the update fees are many but low. In summary, the cost of security is proportional to the value of the channel and independent of the size of the committee.

Consensus vs Consistent Broadcast. Employing consistent broadcast in a blockchain system typically implies no conflict resolution. However, in channels, the state can only be updated if both parties agree and sign the new state; otherwise, the state is invalid. Furthermore, the state updates in channels are totally ordered by the parties. Thereby, it in not the role of the watchtower committee to enforce agreement, but merely to verify that agreement was reached, and act as a shared memory for the parties. For this reason, state machine replication (or consensus) protocols are not necessary. Consistent broadcast is tailored for Brick as it offers the only necessary property, non-equivocation.

Brick Security under Execution Fork A acks. We can extend Brick to run asynchronous consensus [27] during the closing phase in order to defend against execution fork attacks [22]. This would add an one-off overhead during close but would make Brick resilient against extreme conditions [2]. For example, in case of temporary dishonest majority the adversary can attack the persistence[12] of the underlying blockchain, meaning that the adversary can double-spend funds. The same holds for all channel constructions so far; if the adversary can violate persistence, the dispute resolution can be reversed, hence funds can be cheated out of a channel party.

For Brick there is limited protection as the adversary can only close on the last committed state or the freshest valid (but not committed) state. With consensus during close, Brick maintains safety (i.e., no party loses channel funds) even when persistence is violated. A malicious party can only close the channel in the state that the consensus decides to be last, thus a temporary take-over can only affect the channel's liveness. Therefore, Brick can protect both against *liveness and persistence attacks*[13] on the underlying blockchain adding an extra layer of protection, and making it safer to transact on Brick than on the underlying blockchain.

[12]Persistence states that once a transaction is included in the permanent part of one honest party's chain, then it will be included in every honest party's blockchain, i.e., the transaction cannot change.

[13]We assume the channel to be created long before these attacks take place, so the adversary cannot fork the transaction that creates the channel.

Update Fees. Currently in Brick, the watchtowers (committee) get rewarded on every state update via a one-way channel. Ideally, these rewards would be included in the state update of the channel. However, even if we modify the update of the state to increase the fees on every update, the parties can always invoke the `Optimistic Close` protocol, and update the channel state to their favor when closing. Therefore, the incentives mechanism is not robust if the update rewards of the watchtowers are included in the update states.

Collateral. The collateral for each watchtower is $v \bullet f$, where v is the total value of the channel and f the number of byzantine watchtowers. This is slightly higher than $v \bullet {}^{1}f \ , \ 1^{0}$ that is the lowest amount for which security against bribing attacks is guaranteed when both channel parties and watchtowers are rational under asynchrony. Towards a contradiction, we can simply consider a channel where watchtowers lock a total collateral $c \ddot{Y} v$. Then any rational party that has less than $v \quad c \quad \epsilon$ value in the freshest state will bribe the watchtowers $c \ , \ \epsilon$ coins and close the channel in a previous state where the party holds the total value v of the channel. In a synchronous network, this attack would not work since the other parties would have enough time to dispute. However, under asynchrony (or simple offline parties assumption [31]) there is no such guarantee. This requirement highlights a trade-off for replacing trust: online participation with synchrony requirements or appropriate incentive mechanisms to compel the honest behavior of rational players.

Brick+ Channel Close. In Brick, the parties can close the channel in agreement at any time without the need for committee intervention (`Optimistic Close`). However, Brick+ does not allow this function; a channel can only close in collaboration with the committee. Otherwise, the channel parties can update the channel and alter the history without accountability. In case of malicious behavior of the committee (which is outside the threat model of this work), we assume that the parties can alert the auditor who can pursue external punishment to enforce the watchtowers to be responsive.

9 ACKNOWLEDGEMENTS

We would like to thank Jakub Sliwinski for his valuable discussion and feedback as well as his impactful contribution to this work.

REFERENCES

[1] Dave Appleton. 2017. Ethereum Multi-Signature Wallets. (2017). https://medium.com/hellogold/ethereum-multi-signature-wallets-77ab926ab63b

[2] Georgia Avarikioti, Lukas Käppeli, Yuyi Wang, and Roger Wattenhofer. 2019. Bitcoin Security Under Temporary Dishonest Majority. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019*. 466–483. https://doi.org/10.1007/978-3-030-32101-7_28

[3] Georgia Avarikioti, Felix Laufenberg, Jakub Sliwinski, Yuyi Wang, and Roger Wattenhofer. 2018. Towards Secure and Efficient Payment Channels. *CoRR* abs/1811.12740 (2018). arXiv:1811.12740 http://arxiv.org/abs/1811.12740

[4] Zeta Avarikioti, Orfeas Stefanos Thyfronitis Litos, and Roger Wattenhofer. 2020. Cerberus Channels: Incentivizing Watchtowers for Bitcoin. In *Financial Cryptography and Data Security (FC), Kota Kinabalu, Sabah, Malaysia.*

[5] Adam Back and authors. 2014. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains* 72 (2014).

[6] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[7] Conrad Burchert, Christian Decker, and Roger Wattenhofer. 2018. Scalable funding of Bitcoin micropayment channel networks. *Royal Society open science* 5, 8 (2018), 180089.

[8] Jeff Coleman, Liam Horne, and Li Xuanji. 2018. Counterfactual: Generalized state channels. (2018). https://l4.ventures/papers/statechannels.pdf.

[9] Kyle Croman et al. 2016. On scaling decentralized blockchains. In *International Conference on Financial Cryptography and Data Security*. Springer, 106–125.

[10] Christian Decker, Rusty Russell, and Olaoluwa Osuntokun. 2018. eltoo: A Simple Layer2 Protocol for Bitcoin. (2018). https://blockstream.com/eltoo.pdf.

[11] Christian Decker, Jochen Seidel, and Roger Wattenhofer. 2016. Bitcoin Meets Strong Consistency. In *17th International Conference on Distributed Computing and Networking (ICDCN), Singapore*. http://www.tik.ee.ethz.ch/file/ed3e5da74fbca5584920e434d9976a12/peercensus.pdf

[12] Christian Decker and Roger Wattenhofer. 2015. A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels. In *Stabilization, Safety, and Security of Distributed Systems*, Andrzej Pelc and Alexander A. Schwarzmann (Eds.). Springer International Publishing, Cham, 3–18.

[13] DeDiS Cothority 2016. DeDiS cothority. (Sept. 2016). https://www.github.com/dedis/cothority.

[14] DeterLab 2012. DeterLab Network Security Testbed. (September 2012). http://isi.deterlab.net/

[15] Tadge Dryja. 2016. Unlinkable Outsourced Channel Monitoring. Scaling Bitcoin Milan. (2016).

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

[17] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. 2017. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*. 327–344.

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

[19] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. 2015. The Bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT 2015*. Springer, 281–310. https://eprint.iacr.org/2014/765.pdf

[20] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. 2019. SoK: Off The Chain Transactions. *IACR Cryptology ePrint Archive* 2019 (2019), 360. https://eprint.iacr.org/2019/360

[21] Alyssa Hertig. 2018. Bitcoin Lightning Fraud? Laolu Is Building a 'Watchtower' to Fight It. https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud. (2018). https://www.coindesk.com/laolu-building-watchtower-fight-bitcoin-lightning-fraud.

[22] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. 2012. Double-spending fast payments in Bitcoin. In *19th ACM Conference on Computer and communications security*. ACM, 906–917. https://eprint.iacr.org/2012/248.pdf

[23] Majid Khabbazian, Tejaswi Nadahalli, and Roger Wattenhofer. 2019. Outpost: A Responsive Lightweight Watchtower. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019*. 31–40. https://doi.org/10.1145/3318041.3355464

[24] Eleftherios Kokoris-Kogias, Enis Ceyhun Alp, Sandra Deepthy Siby, Nicolas Gaillya, Philipp Jovanovic, Linus Gasser, and Bryan Ford. 2018. Hidden in Plain Sight: Storing and Managing Secrets on a Public Ledger. Cryptology ePrint Archive, Report 2018/209. (2018).

[25] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. 2016. Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing. In *Proceedings of the 25th USENIX Conference on Security Symposium*. 279–296.

[26] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. 2018. OmniLedger: A Secure, Scale-Out, Decentralized Ledger via Sharding. In *2018 IEEE Symposium on Security and Privacy*. 583–598.

[27] Eleftherios Kokoris-Kogias, Alexander Spiegelman, Dahlia Malkhi, and Ittai Abraham. 2019. *Bootstrapping Consensus Without Trusted Setup: Fully Asynchronous Distributed Key Generation.* Technical Report. Cryptology ePrint Archive, Report 2019/1015.

[28] kyber 2010 – 2018. The Kyber Cryptography Library. (2010 – 2018).

[29] Marc Leinweber, Matthias Grundmann, Leonard Schönborn, and Hannes Hartenstein. 2019. TEE-Based Distributed Watchtowers for Fraud Protection in the Lightning Network. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer International Publishing, 177–194.

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

[31] Patrick McCorry, Surya Bakshi, Iddo Bentov, Sarah Meiklejohn, and Andrew Miller. 2019. Pisa: Arbitration outsourcing for state channels. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 16–30.

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

[33] Andrew Miller, Iddo Bentov, Surya Bakshi, Ranjit Kumaresan, and Patrick McCorry. 2019. Sprites and State Channels: Payment Networks that Go Faster Than Lightning. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019*. 508–526.

[34] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).

[35] Henning Pagnia and Felix C Gärtner. 1999. *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.

[36] Joseph Poon and Thaddeus Dryja. 2015. The Bitcoin lightning network: Scalable off-chain instant payments. (2015).

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

[38] Jeremy Rubin, Manali Naik, and Nitya Subramanian. 2014. Merkelized Abstract Syntax Trees. https://github.com/JeremyRubin/MAST. (2014).

[39] Jeremy Spilman. 2013. Anti DoS for tx replacement. Bitcoin Developers Forum. (2013). Accessed: 2020-01-20.

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

[41] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. (2014).

[42] Karl Wüst, Kari Kostiainen, Vedran Capkun, and Srdjan Capkun. 2019. PRCash: Fast, Private and Regulated Transactions for Digital Currencies. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019*. 158–178. https://doi.org/10.1007/978-3-030-32101-7_11

A ATTACK AGAINST EXISTING CHANNEL CONSTRUCTIONS

Here we present a concrete attack against Pisa [31]. This attack applies to any channel construction that has a time-out dispute process. The attack is simple and relies on delaying the appearance of the dispute transaction on-chain until the dispute window expires. The attack is as follows:

(1) Alice and Bob establish a state-channel. As Pisa constructs, both Alice and Bob chose their favorite watchtower and provably assign to them the job of watching the chain.
(2) Alice sends multiple coins to Bob through state changes $S_1, S_2, ..., S_n$.
(3) Alice request the closure of the channel with state S_1 where Alice holds more coins that S_n.
(4) Bob or Bob's watchtower detects the attack and sends on the network a dispute transaction for S_n.
(5) Alice attacks Bob in one of the following ways:
    (a) Congests the blockchain so that the dispute transaction for S_n does not appear on-chain until after the dispute window expires.
    (b) Depending on the blockchain, Alice can censor the dispute transaction for S_n without congesting the network [32].
    (c) If Alice is a classic asynchronous network adversary [6], Alice can delay the delivery of the dispute transaction for S_n until after the dispute window.
(6) Since the dispute transaction for S_n appears after the dispute window expires, the safety of the channels is violated.

In Pisa this attack might cause the watchtower to be punished although the watchtower did not misbehave. In other channels [21, 33], that do not employ or do not collateralize watchtowers, this attack leads to direct value loss for Bob. These attacks do not apply in Brick and Brick+ since the adversary can only temporarily thwart liveness of the close function.

B BRICK SECURITY ANALYSIS

B.1 Honest Parties Assumption

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 the external party intercepted a message (between the parties of the channel or between the parties and the committee) or it is a watchtower. 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 watchtowers receive during the *Update* phase a message $M = \{H^1(s_i), r_i^0, i\}$ for any valid update state (assuming honest parties that do not intentionally reveal the state). If the watchtower extracts the state of the channel s_i from the $H^1(s_i), r_i^0$, the watchtower reverted the hash function, Therefore, the hash function is not pre-image resistant for a computationally-bounded adversary and hence not cryptographically-secure. This contradicts the system model, where we assumed cryptographically-secure hash functions.

B.2 Incentivizing Rational Watchtowers

In this section, we show that rational watchtowers will follow the protocol, i.e., that deviating from the honest protocol execution can only result in decreasing a watchtower's expected payoff. We consider each phase of Brick separately, and evaluate the payoff of a watchtower for each possible action.

Open. During the opening of the channel, we assume the watchtowers follow Protocol 1 and commit the requested collateral on-chain (Brick smart contract). Otherwise, the parties simply replace the unresponsive/misbehaving watchtowers.

Update. In Protocol 2 the watchtowers do not participate. Thereby, the only action by the watchtowers is step 2 of Protocol 3. A watchtower can deviate as follows:

The watchtower can simply ignore the party's request to update the state (does not reply to the party), since the update fee is already collected. At first sight, this game looks like a fair exchange game, which is impossible to solve without a trusted third party [16]. Furthermore, we cannot use a blockchain to solve it [35] 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 watchtowers want to increase their expected rewards in the long term. Thereby, they know that if they receive an update fee from a party and do not respond, then the party will stop using them (there is f fault tolerance in Brick), thus their expected payoff for the repeated game will decrease.

The watchtower sends its signature on the new update state but does not store the new announcement. In this case, the watchtower cannot verify the correctness of a later update state and could unintentionally commit fraud by signing two conflicting acknowledgements. As a result, the watchtower could lose its collateral, hence its expected payoff decreases.

The watchtower replies to the party although it has published a closing announcement on-chain. Then, the party can penalize the watchtower by claiming its collateral.

The watchtower does not perform the necessary verifications. Then, the watchtower might unintentionally commit fraud by signing an invalid state, hence allow the party to claim the collateral.

Close. Watchtowers do not participate in the optimistic close (Protocol 4). As a result, we only evaluate the possible actions a watchtower can deviate from the honest execution of Protocol 5.

The watchtower does not publish a closing announcement on-chain. In this case, the watchtower can attempt to enforce a hostage situation on the funds of the channel in collaboration with other watchtowers in order to blackmail the channel parties. However, to enforce a hostage situation on the channel's funds, at least $f + 1$ watchtowers must collude, hence at least one rational watchtower must participate. However, a rational watchtower cannot be certain that the other watchtowers will indeed maintain the hostage situation or participate in the consensus thus claim the closing fee (only the first t watchtowers get paid); therefore, we reduce our problem to the prisoner's dilemma problem. Thus, the only strong Nash equilibrium for a rational watchtower is to immediately publish a closing announcement on-chain in order to claim later the closing fee.

The watchtower signs later a new update state. Then, the watchtower allows the party receiving the signed announcement with higher sequence number than the closing announcement to create a proof-of-fraud and claim the watchtower's collateral.

The watchtower publishes on-chain a closing announcement that is not the stored one[14]. However, the watchtower has already sent a signature on an announcement with a higher sequence number, therefore the party can create a proof-of-fraud and claim the watchtower's collateral. Therefore, any rational watchtower will request as a bribe an amount at least marginally higher than the collateral to perform the fraud. We will show that no rational party will provide such a bribe to any rational watchtower. Hence, all rational watchtowers will honestly follow the protocol and submit the stored announcement for closing the channel.

To that end, let us denote by p_A the payoff function of party A (the cheating party wlog). The payoff function depends on the channel balance of party A when requesting to close the channel, denoted by c_A ($0 \leq c_A \leq v$, where v is the total channel funds), the collateral the party claims through proofs-of-fraud, and the total amount spent for bribing rational watchtowers. Formally,

$$p_A = c_A + x\frac{v}{f} - b\frac{v}{f} + \epsilon$$

where x is the number of proofs-of-fraud submitted by the party, b the number of bribed rational watchtowers, and ϵ the marginal gain over the collateral the watchtowers require

[14] We assume the watchtower will only sign as closing an announcement that used to be valid in an attempt to commit fraud. Otherwise, the party will claim the watchtower's when the smart contract verifications fail.

to be bribed. Note that each watchtower has locked $\frac{v}{f}$ as collateral. Further, note that the Byzantine watchtowers do not require a bribe but act arbitrarily malicious, meaning that they will provide a proof-of-fraud to party A without any compensation.

Next, we analyze all potential strategies for party A and demonstrate that the payoff function maximizes when the party does not bribe any rational watchtower, but closes the channel in the freshest committed state. There are four different outcomes in the strategy space of party A:

(1) The channel closes normally (Protocol 5). Then, $p_A = c_A$.
(2) Party A submits the proofs-of-fraud only from the Byzantine watchtowers and the channel closes in the freshest committed state (by the remaining t rational watchtower). Then, $p_A = c_A + f\frac{v}{f} = c_A + v$. Note that the payoff function in this case maximizes for $x = f$.
(3) Party A bribes $b > 0$ rational watchtowers and the channel closes in the freshest committed state. Then, $p_A < c_A + {}^1f + b^0\frac{v}{f} - b^1\frac{v}{f} + \epsilon^0 = c_A + v - b\epsilon < c_A + v - \epsilon$. The first inequality holds because the bribed watchtowers might be part of the second set of t closing announcements, in which case they do not contribute to the claimed collateral.
(4) Party A bribes $b > 0$ rational watchtowers and the channel closes in a state other than the freshest committed state. Let us denote by y the number of rational watchtowers that provide a proof-of-fraud to the party, and m the number of submitted proofs-of-fraud that belong to Byzantine watchtowers. Then, the possible actions in this strategy are depicted in Table 2. Note that at least $f + 1$ misbehaving watchtowers are required to close the state in a previous state since we assume there can be f slow rational watchtowers that have not yet received the update states. In this case, the payoff function is

$$p_A < v + m + y\frac{v}{f} - y + m + 1\frac{v}{f} + \epsilon = v - \frac{v}{f} + \epsilon < y + m + 1 - v\frac{v}{f} + \epsilon$$

where the first inequality holds since $0 < c_A < v$. Therefore, the payoff function maximizes in case the party follows the second strategy, i.e., when the channel closes in the freshest committed state and no rational watchtowers are bribed.

Table 2: Potential actions to close the channel in a "fraudulent" state.

Action	Proof-of-fraud	Close	Total
Byzantine	m	$f - m$	f
Bribed (rational)	y	$f + 1 - {}^1f - m^0 = m + 1$	$y + m + 1$
Total	$m + y$	$f + 1$	-

We notice that in any possible deviation of the protocols the watchtower's expected payoff decreases, thus a rational watchtower will honestly follow the protocol execution.

B.3 Incentivizing Rational Parties

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. Therefore, the security of Brick holds even under rational channel parties. We argue for every phase separately.

Open. Naturally, 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. 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 increase its profit from such behavior directly (attacking the safety of the channel). Moreover, attempting to attack the liveness of the channel is not profitable since the counterparty can always request to close in collaboration with the committee by invoking Protocol 5: `Pessimistic Close`.

Lastly, channel parties can collude and stop updating the channel (liveness attack) in order to enforce a hostage situation on the watchtowers' collateral. However, the committee size is at least $n > 7$. Thereby, from the pigeonhole principle there is at least one party in the channel that has locked funds at least equal to $\frac{v}{2} > \frac{v}{f}$. Thus, a channel party locks an amount larger than each watchtower's collateral which means that this party's cost is larger than a watchtowers. Since all parties are rational, the "richest" party of the channel at any time is incentivized to close the channel in collaboration with the committee. Thus, rational parties will not deviate from the honest execution of Protocol 2.

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

First, a party 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. This state will be considered committed for the counterparty after the execution of Protocol 3. We show a rational party cannot increase its payoff by not broadcasting the announcement to all watchtowers. To demonstrate this, we consider two cases; either the new state is beneficial to the party or not. If the new state is beneficial to party A, then this state is not beneficial for the counterparty (e.g., B payed A for a service). Thus, if the committee has not received the freshest state, party B can "rightfully" close the channel in the previous state. Hence, the expected payoff of party A decreases. On the other hand, suppose the new state is not beneficial to party A and it chooses not to send the announcement to the committee. Then, either the counterparty will have the state committed or the state will not be executed. From the safety property of the channels, party A cannot successfully close the channel in a previous state if the state was committed. Hence, party A does not increase its payoff. On the other hand, if party A does not request the signature of a watchtower that will later commit fraud, then the party

cannot construct a proof-of-fraud to claim the watchtower's collateral and therefore the party's payoff decreases.
Second, a party can broadcast different messages to the committee or parts of the committee. During the execution of Protocol 3, the watchtowers verify the parties' signatures, thus an invalid message will not be acknowledged from an honest watchtower. If the messages are valid (both parties' signatures are present), the parties have misbehaved in collaboration. This can lead to 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 previous one, where the party can only lose from following it.
Lastly, the party can choose not to proceed to the state transition. This is outside the scope of the paper and a problem of a different nature (a fair exchange problem).

Overall, a rational party cannot increase its payoff by deviating from Protocol 2 or Protocol 3.

Close. In this phase, there are two different options: `Optimistic Close` and `Pessimistic Close`.
In the first case, where Protocol 4 is executed, a party can deviate from it in the following ways:

It is the party requesting the closing of the channel in a cheating state. The counterparty will not sign the state since it is being cheated, else it would not be a cheating state. Thus, safety is guaranteed and the party cannot profit from this strategy.
It is the party requesting the closing of the channel and never publishes the signed closing state. In line 2 of Protocol 4, the signatures on the state are exchanged between the parties, hence the counterparty will eventually publish the closing state. Note that we assume that the closing party sends its signature first with the closing request.
It is the 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 payoff 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 that requested to close is not responsive, meaning that the party does not publish the closing state. In this case, either the party is the richest of the channel or not. If the party is the richest of the two then the party's cost of not responding is higher than that of the counterparty and any other watchtower's cost. Therefore, the party is not the richest of the channel. In this case, the counterparty which wants to close the channel, can use the on-chain closing announcements of the watchtowers and publish the closing state. In both cases, the party that requested close cannot increase its payoff by not revealing the closing state, but only lose from locking its channel funds for a longer period of time that necessary (since no updates are possible).
The party that requested to close the channel publishes an invalid closing state (e.g., random state, previously valid state, a committed state that is not the freshest). However, the smart contract will not close the channel in this state because some verification will fail. Therefore, the party will simply lose the blockchain fee for the on-chain transaction and will not gain any profit. Note that for a rational party to successfully commit fraud, at least one rational watchtower must be bribed to commit fraud, which is not profitable for the party as demonstrated in the analysis of Section B.2.

To summarize, if a channel party deviates from the honest protocol execution at any phase, it cannot increase its payoff. Therefore, any rational party will follow the protocol, hence honest behavior of channel parties is incentivized.

C BRICK+ SECURITY ANALYSIS

In this section, we first prove the Brick+ goals, namely *Safety, Liveness, Privacy* and *Auditability* assuming $2f + 1$ honest watchtowers, and later we argue that rational watchtowers 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 that the auditor will only deviate from the protocol to gain more profit (hence the need for a smart contract to do the fair exchange between the auditor and the committee). However, the auditor will not punish a party arbitrarily with no proof, since the auditor 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 to Protocol 6, the parties publish the closing state on-chain to close the channel (Protocol 5). 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 both the watchtowers and the parties of the channel are rational, Protocol 6 will not complete (a valid closing state will not be published) 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).

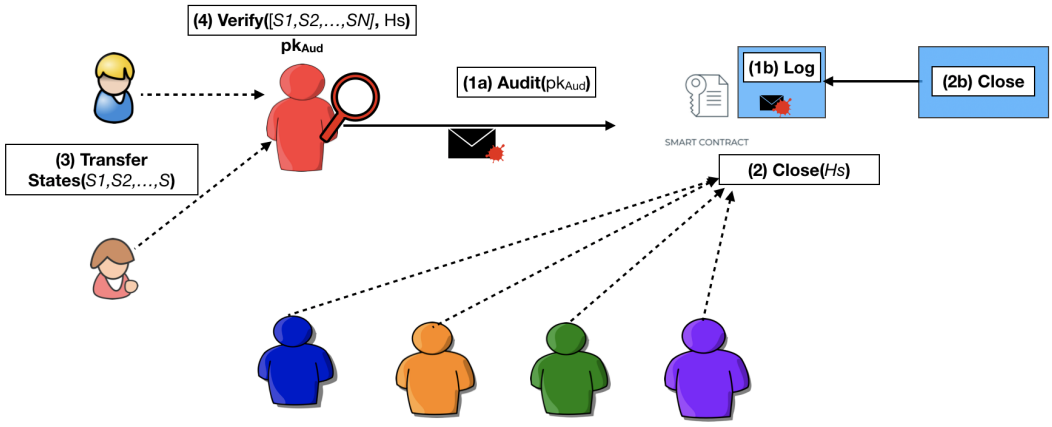

Figure 2: Typical workflow of Brick+ for an audit update. (1) The auditor starts the audit by posting the request on chain, (2) the committee closes the channel, and (3) the parties transfer the state to the auditor. Finally, (4) the auditor cross-checks the claims of the party and the committee.

This means that state s is replaced by another state s' either in the history of all parties or in the hash-chain that produced the hash-head corresponding to the closing state. If s is not part of the hash-chain, but it is committed, then the parties misbehaved and sent to at least one watchtower a different state. Note that the watchtower cannot misbehave since the announcement must have the signatures of both parties. Furthermore, two different states cannot be simultaneously committed since this would require two quorums of $2f + 1$ watchtowers that signed different state updates, thus at least $f + 1$ Byzantine watchtowers. Contradiction to our threat model. Therefore, state s is not part of the state history provided by the parties. In both cases, 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), the parties will not misbehave. Thus, every committed state is verifiable.

D EVALUATION OF BRICK

We have implemented consistent broadcast in Golang using the Kyber [28] cryptographic library and the cothority [13] framework. In Table 3 we evaluate our protocol on Deterlab [14] using 36 physical machines, each having four Intel E5-2420 v2 CPUs and 24 GB RAM. To have a realistic wide area network, we impose a 100ms roundtrip latency on the links between watchtowers and a 35Mbps bandwidth limit.

As illustrated the overhead of using a committee is almost equal to a round-trip latency (100ms). The small overhead is due to party sending the messages in sequence, hence the last message is sent with a small delay $t > 0$. This is observed in the total latency which is close to $100 + t$ ms. These numbers are three orders of magnitude faster than current blockchains. Furthermore, channels are independent and embarrassingly parallel which means that we can deploy as many as we want without significantly increasing the overhead.

As for deployment over existing blockchains, Brick exposes a classic multi-signature wallet [1] abstraction. The funding transaction transfers atomically the collateral of the committee and the state of the parties under the management of the wallet. The closing transaction commits the final state by transferring the funds of the wallet back to its rightful owners. In order to respect the rules of Brick the policy for transferring funds out of the wallet needs to say that either all public-keys of the parties sign (only for Brick) or a combination of t-out-of-n watchtowers together with one party sign. This is a straightforward implementation of a multi-signature wallet on Ethereum. Unfortunately, it can become rather costly on Bitcoin because of the linear (to the number of parties) possibilities we need to define for the closing transaction. Advancement such as MAST [38] can lift this cost if implemented on Bitcoin. However, we believe that implementing the appropriate incentive mechanisms in Bitcoin might be quite challenging if possible at all.

Table 3: Microbenchmark of Brick and Brick+

Number of Watchtowers	4	34	151
Consistent Broadcast	0.1138 sec	0.118 sec	0.1338 sec