UNIVERSITY OF THESSALY

SCHOOL OF SCIENCES
DEPARTMENT OF COMPUTER SCIENCE AND BIOMEDICAL INFORMATICS

PROGRAM OF UNDERGRADUATE STUDIES

DIPLOMA THESIS

A flexible state channels design to improve the scalability of public distributed ledger systems

Lydia Maria D. Negka

LAMIA

FEB 2023

ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΜΕ ΕΦΑΡΜΟΓΕΣ ΣΤΗΝ ΒΙΟΙΑΤΡΙΚΗ

ΠΡΟΠΤΥΧΙΑΚΟ ΠΡΟΓΡΑΜΜΑ ΣΠΟΥΔΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός ευέλικτης αρχιτεκτονικής state channels για τη βελτίωση της επεκτασιμότητας των δημοσίων συστημάτων αποκεντρωμένων καταλόγων

Λυδία Μαρία Δ. Νέγκα

ΛΑΜΙΑ

ΦΕΒΡΟΥΑΡΙΟΣ 2023

BACHELOR THESIS

A flexible state channels design to improve the scalability of public distributed ledger systems

Lydia Maria D. Negka

SUPERVISOR

George Spathoulas
Member of laboratory teaching stuff
University of Thessaly

THREE-MEMBER ADVISORY COMMITTEE:

George Spathoulas
Member of laboratory teaching stuff
University of Thessaly

Athanasios Kakarountas
Associate Professor
University of Thessaly

Petros Spachos
Assistant Professor
University of Thessaly

Examination Date: February 15, 2023

ABSTRACT

Blockchain technology guarantees the implementation of applications without intermediaries, making transactions secure and indisputable. Those characteristics have resulted in increasing popularity during recent years and in a significant rise in its adoption for real-world applications. This advancement has brought up a critical challenge that public blockchain systems face, which is scalability. Most of the currently deployed systems fail to cope with the constantly increasing usage. In order to provide the promised security guarantees, large delays and high usage fees are imposed for submitted transactions and thus widespread adoption of the technology is hindered. A number of different approaches have been proposed to increase the capacity of blockchain systems when it comes to processing transactions. The present work focuses on one of the most popular ones, that of state channels. A large scale survey of research in this field is performed and an extensive analysis of relevant publications is conducted. Limitations through all relevant research efforts along with the various features that differentiate proposed designs are discussed. The guidelines that emerge from said analysis are used as a basis for the development of Origami, a novel state channel design with a unique approach that achieves a significantly improved applicability range, an increased number of features and a severe decrease in the number of necessary on-chain transactions for the entirety of a blockchain network. The support of non turn-based state channel applications, the introduction of flexibility in participant sets and the use of cryptographic accumulators for local storage optimisation are some of the prominent contributions of the Origami design.

SUBJECT AREA: Public blockchain systems

KEYWORDS: blockchain, layer 2, scalability, state channels, DLT

ΠΕΡΙΛΗΨΗ

Η τεχνολογία των αποκεντρωμένων καταλόγων (Blockchain), εξελίσσεται σε μία ταχύτατα αναπτυσσόμενη τεχνολογία, η οποία έχει συγκεντρώσει το ενδιαφέρον της ερευνητικής κοινότητας, καθώς έχει εισχωρήσει σε πολλές πρακτικές εφαρμογές, που μέχρι τώρα ελέγχονταν ή/και συντονίζονταν από μία κεντρική οντότητα. Το Blockchain εγγυάται την υλοποίηση εφαρμογών χωρίς διαμεσολαβητές, καθιστώντας τις συναλλαγές ασφαλείς και αδιαμφισβήτητες. Η δημιουργία αυτής της κατανεμημένης μορφής συναλλαγών με εμπιστοσύνη και ασφάλεια, οδήγησε στην ευρεία υιοθέτηση των αποκεντρωμένων καταλόγων και αποκάλυψε τα εγγενή προβλήματα που ανακύπτουν από την αυξημένη χρήση του Blockchain. Τα περισσότερα υφιστάμενα δίκτυα παρουσιάζουν σημαντικές δυσκολίες, όσον αφορά την δυνατότητά τους να επεκταθούν/κλιμακωθούν.

Η επεκτασιμότητα/κλιμάκωση (Scalability) του Blockchain αναφέρεται στην ικανότητα ενός δικτύου Blockchain να ανταποκρίνεται αποτελεσματικά και αποδοτικά σε αυξανόμενη ποσότητα συναλλαγών, χρηστών και δεδομένων, καθώς αυξάνεται η χρήση και ο αριθμός των συμμετεχόντων σε αυτό. Προκειμένου να διατηρηθούν οι εγγυήσεις που προσφέρει το δίκτυο σχετικά με την ασφάλεια των συναλλαγών, δημιουργούνται μεγάλες καθυστερήσεις και υψηλά κόστη για κάθε αλληλεπίδραση με αυτό, εμποδίζοντας την ευρεία υιοθέτηση της τεχνολογίας.

Ο παραπάνω περιορισμός αποτελεί ένα από τα κύρια αναχώματα για την καθολική ενσωμάτωση της τεχνολογίας Blockchain στην καθημερινή ανθρώπινη δραστηριότητα και έχει οδηγήσει στην ανάπτυξη νέων μηχανισμών με σκοπό την επίλυσή του. Έχουν προταθεί ποικίλες προσσεγγίσεις για την επέκταση των blockchain δικτύων που θα αναφερθούν στην παρούσα εργασία, η οποία επικεντρώνεται σε μία από αυτές, τα state channels.

Στα πλαίσια της παρούσας εργασίας διενεργήθηκε εκτενής ανάλυση των υπαρχόντων σχεδιασμών σε αυτό το πεδίο και εντοπίστηκαν οι περιορισμοί, που αυτοί αντιμετωπίζουν, καθώς και οι μεταξύ τους διαφορές. Η συγκριτική ανάλυση των υφιστάμενων τεχνικών και οι πληροφορίες που προέκυψαν από αυτή, χρησιμοποιήθηκαν ως οδηγός για την ανάπτυξη ενός νέου σχεδιασμού state channels με την ονομασία Origami. Ο προτεινόμενος σχεδιασμός, Origami, ανήκει στην κατηγορία των Layer2 τεχνικών καθώς χρησιμοποιεί μια μοναδική προσέγγιση που ελαχιστοποιεί την αλληλεπίδραση με το blockchain δίκτυο, επιφέροντας σημαντική βελτίωση όσον αφορά τη δυνατότητα εφαρμογής του

και μεγάλο πλήθος ωφέλιμων χαρακτηριστικών, τα οποία θα παρουσιαστούν αναλυτικά.

ΘΕΜΑΤΙΚΗ ΠΕΡΙΟΧΗ: Δημόσια συστήματα αποκεντρωμένων καταλόγων

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: blockchain, layer 2, scalability, state channels, DLT

ACKNOWLEDGMENTS

I would like to thank PhD candidate Angeliki Katsika for her significant contribution to the development of the Origami design.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

1.1 Problem Statement and Motivation

It is an undeniable fact that the popularity of blockchain technology popularity has recently seen a wild surge [56]. Even though this was predicted, it is still evident that most of the public blockchain systems are not ready for widespread use, since they suffer from very low capacity with respect to transactions' processing rate (the number of transactions that can be processed per second). The first blockchain system to be proposed, Bitcoin [45] has a transaction rate of around 4 tx/sec as of September 2021, while in the case of Ethereum [14], the second most popular blockchain platform, the transaction rate is slightly better, approximately 30 tx/sec. The main idea behind the proof of work blockchain consensus mechanism requires that every transaction is processed by every network node before it is confirmed and published. On top of that, due to security reasons the block generation rate and the upper size limit for blocks are restricted, thus a hard upper limit has to be imposed on the transactions rate. While this approach is efficient in terms of security, it directly clashes with blockchain systems scalability [58], as it hinders the use of such systems by a large number of users.

Hence, the matter of addressing scalability issues in blockchain systems has popped up as an issue of unprecedented urgency. The latency of transactions and the imposed high usage cost (transaction fees) that permissionless blockchains suffer from, usually outweigh the benefits of using them and many possible applications become irrelevant.

More than one approach has been explored to remedy those issues. It is a prevailing practice to divide such efforts into two large categories. Layer 1 solutions aim to directly enhance the blockchain functionality and employ alternative protocols[54, 31] or sharding [4, 37]. Those approaches do come with their own set of drawbacks since they require fundamental adaptations of core blockchain systems' components to be applied. Especially for deployed systems that are already in use, the transition to a modified layer 1 is a very challenging process.

Layer 2 approaches are developments on top of an underlying blockchain without requiring significant modifications of that [55]. The most popular proposals in this category are sidechains [7, 52], plasma [49], rollups [2] and state channels. They are generally preferred to Layer 1 solutions, as they are easier to implement directly on top of existing

platforms and they do not come to the expense of fundamental concepts like decentralisation and security.

The focus of this work is on the State Channels approach [5]. It is one of the leading solutions proposed against the scalability problem and allows the trust-less communication of parties that wish to mutually execute a contract while allowing them to dodge high cost, high latency and any privacy issues that often come up through using a public blockchain system. However, there presently has been no extensive study focusing on this solution. State channels have only been studied along with other Layer 2 solutions and this has led to overlooking their unique characteristics mostly in favour of their close counterparts, Payment Channels. Additionally, while a very promising approach, certain drawbacks in state channel designs have kept them from being adopted on as large a scale as they could have been.

The motivation behind this work is to carry out an extensive study of existing state channel designs, to identify desirable and harmful characteristics, and to narrow down the necessary steps to improve state channels and increase their usability. The complete goal is to make use of that information to develop a state channel design that overcomes the obstacles that have so far kept this technology from being widely applicable.

1.2 Contributions

The contributions of this work come in two main categories. Firstly, the state-of-the-art analysis through which all existing state channel efforts have been recorded, analysed and compared achieves the following:

- Records all existing state channels approaches
- Analyses how those have ultimately contributed to the ecosystem
- Identifies the main requirements those protocols set to operate
- Identifies the main features those protocols offer
- Compares existing approaches
- Sets the main focus points for future research

The results of the aforementioned analysis were utilised and led to the proposal of a novel state channel design, titled Origami. This design builds upon the positives of existing efforts while taking necessary steps to avoid their shortcomings. Novelties introduced in Origami are:

- Origami is the first state channel design to provide an extensive analysis of how unordered communication can be achieved in a state channel, and thus enable the off-chain execution of non-turn-based apps.
- Origami is the first state channel design that allows complete flexibility in a channel's participant set. Once in the ecosystem, members can join and leave groups upon request without any on-chain interaction.
- Origami does not force groups and channels to close as a result of unresolved inactivity disputes. The channel adapts in order to progress with the interaction of the rest of the members (excluding the misbehaving member).
- Origami offers a single point-of-entry into an ecosystem that will then allow participants to run any number of applications amongst any composition of Origami members. This ecosystem has the ability to infinitely scale, and therefore the ability to eventually support an environment of infinite state channel users that interact with each other without the cost and burden of on-chain interactions. Origami is also exempt from restrictions placed by virtual channels, like an upper bound in funds that can be committed into a channel and the requirement that the participants must have a common intermediary.

2. BACKGROUND

2.1 Blockchain Scalability

2.1.1 Blockchain Scalability Problem

The unforeseen popularity of blockchain technology has highlighted the major scalability issues of the original design. Public blockchain platforms have been unable to cope with the exponential increase of transactions, that has been triggered by the increase in their usage. A limitation known as the blockchain trilemma[60], has been extensively discussed in recent years. Three desirable properties for a blockchain system are security, decentralization and scalability, but it is very difficult, at least given existing approaches, to maximise all three of those for a given system. Various factors combine to cause major issues and current systems have to prioritise at most two of the mentioned properties and make a compromise with respect to the level on which the third property is supported. The common approach that has been followed in the ecosystem is to retain a high level of decentralization and security and limit the scalability of the public blockchain systems.

The problem stems from the root of the design. The data verification process that gives blockchain the robustness it is famous for allows the network to be only as fast as its slowest node since blocks need to be propagated at the very least to the vast majority of nodes before the generation of the next block.

Additionally, the consensus mechanism most commonly used, called Proof of Work (PoW), with the intensive, expensive calculating process it demands from the miners, plays a big role in terms of network latency. A block can contain a finite number of transactions, and miners will prefer to include in their prospective blocks transactions that offer higher fees. Hence, those that do not provide as high a profit get pushed back in this transaction confirmation process. The obvious solution to this matter, increasing block size, is not optimal in many aspects. Beyond being just a temporary measure that would have to be re-adopted every time latency shows up again, a bigger block size also decreases decentralization and increases the chance of forks occurring in the chain, since block size affects block propagation time. Another action with the same goal that has been proposed is to reduce the size of transactions so that more of them can fit in a block. Such approaches have been implemented in Bitcoin Unlimited [51] and by Bitcoin's Segregated Witness [53], but neither of those methods has brought significant improvements to the

situation.

2.1.2 Blockchain Scalability Measures

Various approaches have been proposed to increase the capacity of existing blockchain systems with respect to transaction processing. Those are categorised into two broad categories as layer 1 or layer 2 solutions.

2.1.2.1 Layer 1

Proposals on this layer refer to fundamental alterations to the design of the blockchain system under consideration. A change in the consensus mechanism[11, 9], like the case of Ethereum 2.0 replacing Proof of Work with Proof of Stake, can measurably improve scalability, but changes of this scale are risky and difficult to implement and enforce. Moreover, the alternative consensus mechanisms may come with compromises with respect to security or decentralization.

Another popular approach on the same level is sharding[24, 57, 19]. This concept refers to replacing the requirement for all peers to sequentially process transactions. Instead, it suggests breaking up the blockchain network into individual segments (or shards). Each shard would hold a unique set of smart contracts and account balances. Nodes are assigned to individual shards to verify transactions and operations, instead of each being responsible for verifying every transaction on the entire network, while secure communication between shards is enabled to provide a global view of all data.

2.1.2.2 Layer 2

Solutions that do not require fundamental changes to the design of blockchain systems, but rather enable increased scalability through supporting mechanisms that function on top of the main chain of the system are broadly labeled as layer 2 scalability solutions [3].

Sidechains: Sidechains are schemes on which additional chains operate in a parallel and independent way to the main-chain. They do not have to share the main chain's consensus protocol but enable secure assets to transfer back and forth to the main chain. Even though sidechains are a tried and tested approach, they demand a trade-off between scalability and security.

Plasma: Plasma is an Ethereum specific solution that can be described as a separate blockchain that operates beside it. The plasma chain is secured through periodic commits of its state to the main chain. It has increased transaction throughput and reduced costs, but is limited in what transactions it supports, as it is not compatible with generic smart contracts. Additionally, it brings on other kinds of delays to enable challenges and ensure fund security.

Rollups: Rollups enable transactions to be executed off-chain and have their data stored on-chain. They require a stake deposit and a contract that can monitor off-chain execution when need be, and inherit the security guarantees of the underlying blockchain. They are further divided into two subcategories: Zero Knowledge (ZK) and Optimistic Rollups. Their main difference lies in the transaction validation process. ZK rollups require each one to be accompanied by a fraud-proof bond, while Optimistic rollups assume all transactions are valid until a challenge is issued. ZK rollups offer instant finality, but demand intense calculations for the validity proofs and are not always EVM compatible. Optimistic rollups are prone to delays due to challenges but offer full smart contract support.

Payment and State Channels: Payment and State channels are based on the same fundamental concept, performing the majority of transactions completely off-chain in the case of no malicious activity. Payment channels apply this logic for the purpose of conducting payments and materializing an unlimited series of such through a pair of deposit and withdrawal transactions. State channels extend this functionality to include the execution of code and more arbitrary state transitions. Both require an initial fund deposit from participants to operate, and in the optimistic case communicate with the blockchain system only during the channel's opening and closing.

2.2 State Channels Overview

The main reason behind the aforementioned scalability issues of blockchain systems is the fact that each update of the state of a smart contract is required to go on-chain and be executed by all nodes of the network, even if it is relevant to a limited subset of users. State channels effectively counter this phenomenon, by limiting the on-chain operations through the means of a session of off-chain interaction between the interested users, and thus reduce the corresponding overhead.

State channels were first mentioned by Jeff Coleman [22], but the concept was first tested in a chess game implementation around the same time[32]. Even though they have proven to be an undoubtedly useful mechanism for complex interactions on the blockchain, they have been greatly inspired by the simpler idea of payment channels [1], that only focus on payments. Payment channels were the first approach that introduced the idea of keeping blockchain users' interaction off-chain, when this is feasible, without undermining the security of the said interaction. Payment channels are confining in terms of possible use cases, as they emerged to solve Bitcoin's significant scalability problem[50], and only focus on conducting off-chain payments. Payment channels have been successful in general[39], and have been applied to other blockchain platforms with different functional requirements (e.g. Raiden in Ethereum [12]). Despite the restricted application domain of payment channels, they are one of the main focus points of current blockchain scalability research. As a result, they have been already analysed much more extensively[33, 34, 36] than state channels, their more broadly applicable counterparts. The present analysis focuses on state channels but it has to be noted that any advancement observed in the payment channel domain does indirectly benefit state channels since many ideas can be carried over from payment channels and adjusted to the state channels' requirements.

The state channels approach is essentially the extension of the payment channels approach, to go from simple payments to arbitrary state transitions. This extension aims at allowing the off-chain execution of smart contracts of generic functionality (without interacting with the blockchain system) while still reaping all of its security guarantees.

When two or more users interact with a smart contract, they sequentially update the contract's state. The underlying blockchain guarantees that the content of those updates (also noted as state transitions) is common for all participants and that the order under which those transitions are submitted is strictly defined. Because of the consensus mechanism used, it is not feasible (or it is at least extremely hard in practice) for any user to claim that a state transition has not been applied, that a state transition has a different content, or that past state transitions should be in a different order than the one those actually appeared in.

The main idea behind state channels is to achieve the same security guarantees but at the same time minimise the number of required on-chain transactions. In practice, there are two distinct cases with respect to state channels operation; (a) the optimistic one in which users do not intend to cheat others and off-chain exchanging of state updates between users is sufficient and (b) the pessimistic one in which there are users that may

attempt to cheat others and so the communication with the blockchain system is required to resolve disputes over malicious users' behaviour.

State channels aim at reducing to minimum on-chain interactions between participants, but they do have a great dependency on the liveness of the underlying blockchain system, since, to remain secure, they have to assume any transaction happening on the channel can always be published on-chain when required. The blockchain system is necessary to take on the role of a trusted arbitrator that will function as a guarantee for honest parties and settle any occurring disputes. To that end, state channel protocols require participants that want to collaboratively execute an app, to bind (lock) a predetermined set of assets (funds, tokens) on-chain. Those assets can be retrieved (with a different distribution for users) through the finalisation of the state channel. Through this constraint, the state channel can satisfy the requirement for establishing trust between users and it keeps parties incentivised to operate according to the protocol, a condition that enables the off-chain collaborative execution of the app. The main factor that forces participants to behave according to the rules is the fact that they have an amount of assets put in as stake on-chain, and that any misbehaviour will result in them losing their amount of assets.

The activity in a state channel can be described as a session of interactions between participating actors and a blockchain system (usually through a coordination smart contract). Participating actors mainly interact with each other, while they sporadically interact with the blockchain system to (a) initiate the state channel (b) terminate the state channel and (c) resolve any disputes that may come up. A state channel can be in one of the following four phases throughout its operation:

Opening Phase: At this phase participants have to go through a protocol to initiate the state channel. Usually, it involves the funding of the channel, during which participants commit assets on-chain that will function as a stake for the duration of the channel existence. Additionally, all participants sign an initial state for the channel, to which they all agree upon, to set the starting point of the state channel.

Update Phase: This phase is the one in which the core functionality of the state channel takes place. Participants typically communicate with each other (and not with the blockchain system) through cryptographically signed messages. Through such messages, they propose updates of the state channel and they also accept status updates proposed by others. The main concept is that a state update that has been signed by all participants is of equal finality with submitting this state on-chain. The channel remains in

the update phase as participants behave according to the protocol.

Dispute Phase: If users' behaviour deviates from the protocol then the state channel transitions to the dispute phase.

Participants need to be in full consensus regarding the state channel's state updates. If this is not the case, because of a participant remaining silent, or proposing invalid state updates then a dispute arises and participants revert to the communication with the blockchain system to ensure a fair outcome. The dispute phase is the phase for which most differences amongst the various state channel implementations can be spotted. As the dispute is resolved the state channel may return to the update phase or progress to the closing phase.

Closing Phase: This is the phase during which state channel finalises and locked assets are returned to the participants. In the optimistic case, closing occurs when participants have concluded their interaction and all agree to publish its outcome to the blockchain system, to finalise the app execution and release locked funds accordingly. In the pessimistic case, the state channel might be forced to transit to this phase through an unresolved dispute. In that case, to protect the assets of the honest parties, the state channel is closed and the locked assets are distributed to them according to the last agreed-upon state and/or in favour of the honest parties.

The main requirements for a functional state channel could be summarized in a few straightforward and fundamental concepts that have been carried over from traditional blockchain systems and are still the pillars of the whole scheme: trustlessness and finality. Trustlessness relates to the guarantee that a participant is not significantly endangering their stake regardless of the behaviour of other parties, and finality relates to the fact that a given channel state has the same validity as an on-chain state. Of course, state channels are also expected to serve their main purpose, which is to reduce the number of times that participants need to go on-chain and thus improve the load of usage a traditional blockchain system can serve.

A significant improvement in the state channels ecosystem was introduced with the development of state channel networks [20, 30], which enable the establishment of a virtual state channel on top of multiple existing state channels, without requiring any on-chain transaction. Through state channel networks the interaction with the blockchain system is further reduced, while a higher level of privacy is achieved since the on-chain publicly available footprint of state channels is even smaller. Moreover, further optimisation

of state channels has taken place to handle multiple state updates in parallel and also support multiple virtual channels on top of a single existing channel without imposing any latency. Those are the main directions to which research in the domain has been targeted during recent years and the corresponding results are analysed in Chapter 3.

2.2.1 State Channels Abstract Definition

In the present Subsection, an abstract definition of state channels is given and the main concepts that exist in most of the current state channel implementations are analysed. State Channels can only be relevant in the context of a Turing complete blockchain, therefore let's assume a blockchain platform that supports smart contracts' execution and a set of users (accounts) that want to interact with a deployed smart contract. For every interaction with the contract that alters its state, the users have to submit an on-chain transaction. This creates significant implications with respect to time delays, as the transaction rate of public blockchain systems is limited, and with respect to finality time, as even after the transaction is processed, the users have to wait for a couple of blocks to be mined, to be sure that the transaction is not reverted. On top of that, the use of public blockchain systems requires the payment of fees for each transaction. The level of the fees is highly affected by the usage of the network and this is one of the main factors that limit public blockchain systems wide adoption by the users.

A large portion of deployed smart contracts are used by a limited number of users and in such cases, the aforementioned scheme is very inefficient. The imposed time delays and financial costs are disproportional to the volume of interactions, the security of which is ensured. An alternative approach proposed by the state channels ecosystem is to limit the main part of the interaction to a user to user communication (called state channel) and require users to go on-chain only when there is a substantive reason to do so. This mainly happens during setting up or closing a state channel or when one of the users behaves maliciously or goes offline.

Users interacting with a smart contract deployed on a traditional blockchain system, call its functions to update the state of the contract. The main reason behind the state channels introduction is that the established workflow for transactions creates highly problematic time and cost overheads. In return, it ensures the secure update of the contract's status, but this is of interest only to the interacting users. State channels propose that the same level of security can be ensured by off-chain communication between in-

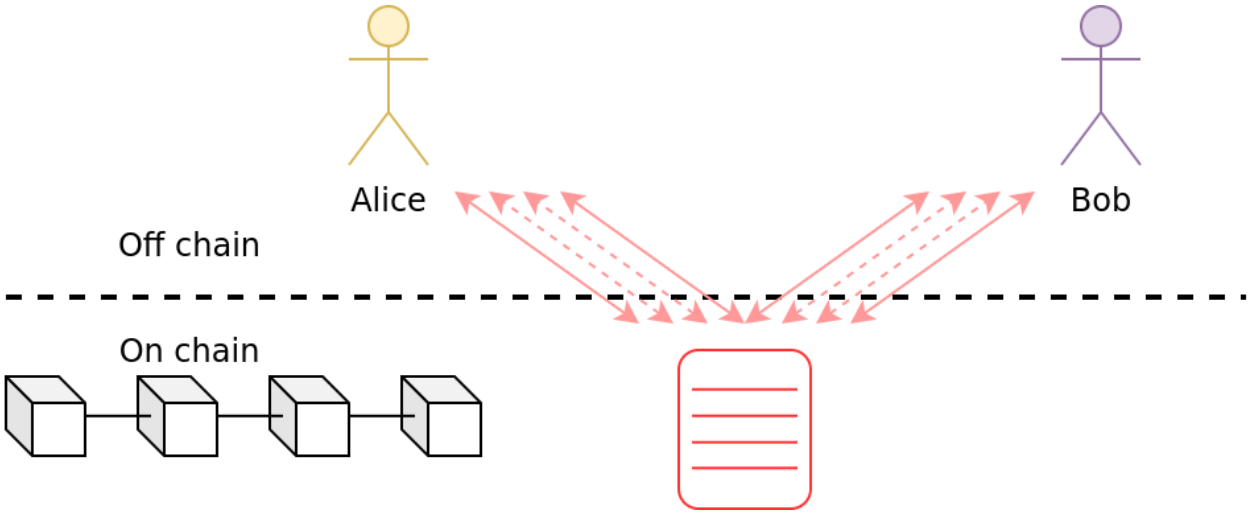

Figure 2.1: Traditional smart contracts workflow

teracting users in the optimistic case (when users are not malicious) and limited on-chain interaction in the pessimistic case (one or more users are malicious).

The main component that enables this interaction is a smart contract deployed on the blockchain that governs the operation of the state channel. While this is implemented in various formats, in each of the research efforts analysed in the next Section, its mission is standard and two-folded:

- It serves as an on-chain account to which participants deposit (or lock) funds. Those funds are eventually redistributed to participants according to the final state of the channel and ensure that users interact with the channel properly, because otherwise, they may lose their deposited funds.
- It implements functionality that validates transitions between two states of the channel. This is the basic mechanism that enables the secure operation of a state channel. It lives on-chain and it is employed by users that have been victims of the malicious behaviour of other users, to ensure a fair advancement of the state channel. On the other hand, the existence of the transition validation mechanism limits the applications that can be deployed to a state channel environment, as there is a strong requirement for an explicit description of all possible states of the application.

Figure 2.1 presents the workflow followed in the traditional context. Alice and Bob interact with a smart contract, which is deployed on-chain. Every such interaction that alters the state of the smart contract corresponds to an on-chain transaction that brings in

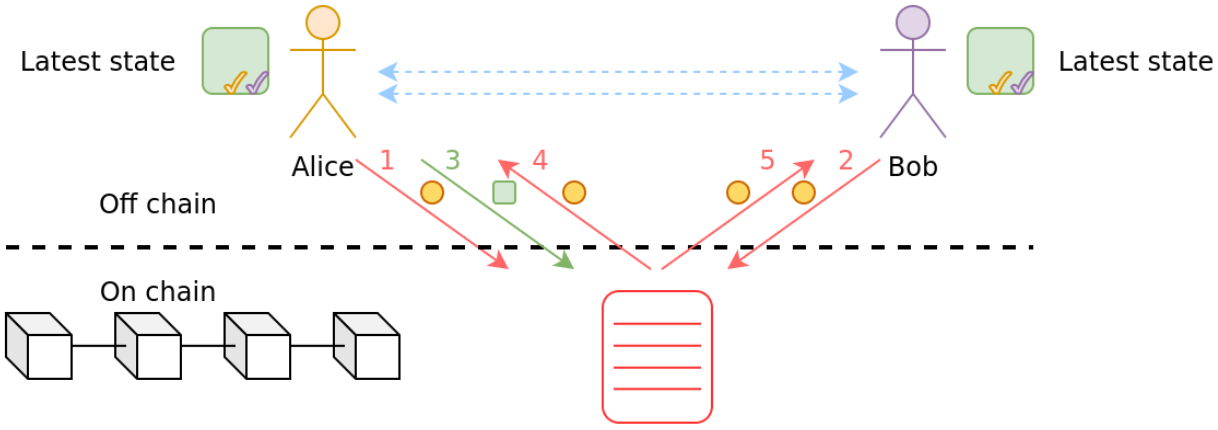

Figure 2.2: State channels workflow (optimistic case)

all the time and cost issues discussed above.

Figure 2.2 presents the general workflow that is followed by most state channel protocols and specifically shows what happens in the optimistic case. Alice and Bob need to execute the same application as previously but this time they use a state channel instead of interacting with an on-chain contract. For the sake of simplicity, we assume that there are only two users that operate a single state channel and that they deploy on it a single app. Let's assume that the state channel app is noted as app while its random state i (i stands for order) is noted as $state^{i}_{app}$. The smart contract that supports the state channel is noted as SC. The first step requires both participants Alice and Bob to fund the state channel by making an on-chain transaction of funds from their personal account to the SC contract (interactions 1,2). Those funds are locked in SC and will be redistributed back to the users according to the final $state^{i}_{app}$ of the app and under specific circumstances. The funding of the state channel brings the app to its initial state $state^{0}_{app}$.

Then comes the main operation of the app, during which the users may submit transactions to the state channel (off-chain communication between them) that update the state of the app. These transactions include a nonce i, the previous state, the updated state and an action/operation that triggers this transition and are signed by the users. The transaction is in the following format:

$$tr_i = (i, state^{i-1}_{app}, state^{i}_{app}, operation, sig_{Alice}, sig_{Bob}) \tag{2.1}$$

There are implementations that in two-party interactions only one of the parties, the one that makes the state transition signs it, but the general case is that all participants

should sign the latest app state. At the end of the transition, each participant holds the signed transition tr_i. Based on that, they can prove on-chain that this is the valid current state, unless another participant holds a more recent signed and valid state transition $tr_j, j > i$. In the optimistic case, users keep on operating the state channel and running the app off-chain until its execution reaches an end. Then one of the users submits the final state transition to the SC (interaction 3). The SC allows a time period for challenges, that are mainly related to the existence of a later valid state transition and then releases the locked funds back to the users according to the final state (interactions 4,5).

On the contrary, there are cases where one of the participants does not behave according to the protocol because the valid advancement of the app's state is not aligned with their benefit or interests or because of communication failure (e.g. the user goes off-line). Such a user may :

- Become unresponsive with respect to proposing a state transition in their turn
- Become unresponsive with respect to signing a valid state transition another participant proposes
- Propose an invalid state transition

Most state channel implementations include a challenge-response scheme, under which users are protected from counterparts the behaviour of which deviates from the defined protocol. The result of such mechanisms is that the misbehaving user ends up losing all or part of the funds they have transferred to SC during the initial funding of the channel.

The pessimistic use case is depicted in Figure 2.3. Initially, both participants fund the channel (interactions 1,2) and then start exchanging app state updates. Let's assume that at some point in time, Bob becomes inactive and does not respond with a state transition in their turn (interaction 3). Alice can challenge Bob by submitting the last valid state transition tr_i to the SC (interaction 4).

$$tr_i = (i, state_{app}^{i-1}, state_{app}^{i}, operation, sig_{Alice}, sig_{Bob}) \tag{2.2}$$

This includes Bob's signature and thus proves that Bob has agreed upon it. Then a time period is allowed for Bob to respond by submitting the next state transition tr_{i+1} to the SC.

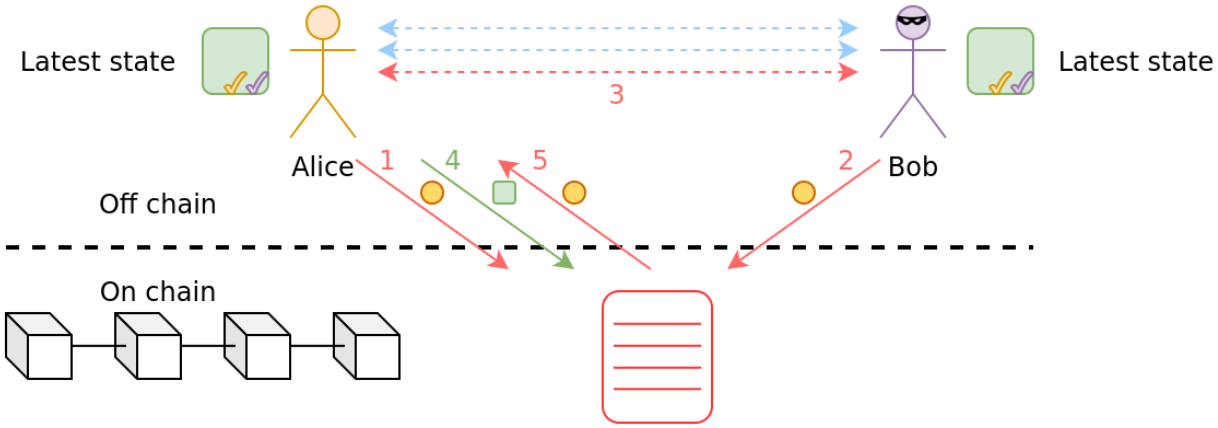

Figure 2.3: State channels workflow (pessimistic case)

$$tr_{i+1} = (i+1, state^{i}_{app}, state^{i+1}_{app}, operation, sig_{Bob}) \tag{2.3}$$

If this happens, then the state transition is checked with regards to its validity and the execution of the app can be continued (either on-chain or off-chain according to the design of the implementation). If Bob does not respond in the predefined time, then the app execution ends and the state channel is closed, with the funds being redistributed in favour of Alice (interaction 5).

Let's also assume another scenario in which Bob proposes to Alice (requesting her to sign it) for an invalid state transition for the app:

$$tr_{i} = (i, state^{i-1}_{app}, state^{i}_{app}, operation, sig_{Bob}) \tag{2.4}$$

Alice can revert to the SC contract again and report the invalid transition Bob is proposing. According to the implementation, SC may directly close the channel or again provide a time period for Bob to come up with a valid state transition.

In any case, the main concept is that when a user holds a valid state transition, signed by all participants, and also knows that it is the latest state they have signed, then they are provided with the same finality guarantees they would get on-chain. There is an additional step to be taken, which is to go on-chain and challenge the counter-parting entity, but it is definite that there may be no other result than finalising the latest signed, valid state they have access to.

2.3 Cryptographic Accumulators

An accumulator is a cryptographic tool that aims to determine whether a given object is included in a given set or excluded from it, without revealing the rest of its contents by producing and utilizing witness proofs [38]. An accumulator can be used as a cryptographic commitment scheme for a set of elements $S = (e_1, e_2, , ..e_n)$ where the prover, who stores the entire set, is enabled to convince any verifier, who only stores a succinct digest of the set, of various set relations. A number of different accumulator schemes have been developed, each best suited to different kinds of applications.

Accumulator usefulness relies mostly on the efficiency they offer in terms of used storage and time even when handling an arbitrarily large set of values since the size of the data structure remains constant. RSA accumulators specifically, offer the advantage of constant-size proofs. Additionally, accumulators can vary in a number of attributes such as whether they are symmetric or asymmetric, the size of the initial set, the type of membership proof, the demand for a trusted setup, and the required update frequency [48].

2.3.1 A basic accumulator scheme

In this section the main functionality of an accumulator that is generated through a trusted set-up is described. The accumulator consists of the following algorithms where a user responsible for an element and the corresponding membership witness acts as the prover, and a user given the relevant proof acts as the verifier at a specific verification round. [8]

- **Generation/Set up Algorithm** is performed by the manager and generates the initial set of the accumulator A_0. RSA accumulators work over hidden-order groups such as the subgroup of quadratic residues QR_N of Z_n where $N = pq$ is the product of two primes.
- **Add Algorithm** adds an element x to the accumulator and produces the updated accumulator value A_{t+1} and the membership witness proof for element x, labelled as w_x. The accumulators' users and proof holders are informed of the addition through an update message $upmsg_{(t+1)}$ that enables them to keep the witnesses of their elements up to date.
- **Del Algorithm** for the deletion of an element x from the accumulator, the algorithm

produces the updated accumulator value A_{t+1} and the non-membership witness proof, labelled as u_x, for the deleted element x, and sends an update message $upmsg_{(t+1)}$ for the accumulators' users in order to update the witnesses of their elements.

- **Create Membership/Non-membership Witness Algorithm** constructs respectively the inclusion or exclusion proof for an element. In recent works [10] techniques have been developed to compute and verify witnesses in batch, for aggregating multiple witnesses.

- **Update Membership/Non-membership Witness Algorithm** updates the membership /Non-membership witness for an element x after the addition or deletion of another element y to the accumulator. In RSA Accumulators updating membership witnesses of an element e_j upon the addition of an element e_i is given simply by adding the element e_i to the witness proof w_{j_t}.

- **Verification Algorithm** is executed by any user that preserves the latest state of the accumulator against which the existence of an element x in the accumulator can be verified using its membership witness w_x. Respectively a non-membership witness can also be verified.

2.3.2 Using RSA accumulator for state channels

The use of cryptographic accumulators, as described in Chapter 4, is based upon the state channel scheme presented in [46] that details the use of accumulators for the purpose of enabling channel states that contain large, growing sets of elements to remain at a fixed size. Therefore, the RSA accumulator was considered the most suitable scheme as it provided contant-sized constructions. The following steps describe these constructions:

- **Setup:** We assume the initial accumulator generated of such a hidden-order group G. Initialization of the group with a generator $g \in G$ where the Strong RSA assumption holds. Let H_{prime} a hash function that maps any element x_i to a unique odd prime number e_i and g the generator of every element $x \in$ S , S $= \{x_1, x_2, ..., x_{(n)}\}$ and :

$$e_i = H_{prime}(x_i) \tag{2.5}$$

The accumulator of S is produced using the representatives of the initial elements after being mapped with the hash function H_{prime}.

$$A = acc(S) = g^{\Pi_{i \in [n]} e_i} \tag{2.6}$$

- **Add:** Addition of an element to the current accumulator value A_t and calculation of the new value of the accumulator A_{t+1}, such that

$$A_{t+1} = (A_t)^{e_i} \tag{2.7}$$

- **Del:** Provided that the owner of an accumulated element e_i maintains the corresponding membership witness w_i, which equals the value of the accumulator before the aggregation of the element, the process of updating the accumulator is presented in Equation 2.8.

$$A_{t+1} = w_{it} \tag{2.8}$$

- **MemWitCreate:** Membership witness w_j for an element e_j can be computed as:

$$w_j = (A_t)^{-e_j} \tag{2.9}$$

Nevertheless, exponentiation of the accumulator by e_j^{-1} cannot be efficiently executed in a hidden-order group. To integrate accumulators into state representation for state channels, the user that adds an element to an accumulator is burdened with the responsibility to maintain its previous value A_{t-1}.

- **NonMemWitCreate:** The non-membership witness u_i of an element e_i that is not included in an accumulator A can be calculated when the product of all the accumulated elements is known. The non-membership witness can be produced using the Bezout coefficients.

- **MemWitUpdateAdd:** As stated in the basic scheme, updating membership witnesses of an element e_j upon the addition of an element e_i is given simply by adding the element e_i to the witness proof w_{j_t}.

$$w_{it+1} = w_{it}^{e_i} = A_t^{e_i}$$

- **MemWitUpdateDel:** Updating membership witnesses for e_i after the removal of an element e_j requires the computation of e_jth root of A_t which corresponds to the updated witness. After computing the Bezout coefficients a, b we can produce the updated membership witnesses according to Equation

$$w_{it+1} = w_{it}^{b} A_{t+1}^{a} \tag{2.10}$$

- **VerMem:** For the verification of a membership proof w_i of an element e_i given the current accumulator state A_t one exponentiation in G is required. Adding the element e_i to the set accumulated in w_i and checking if the result equals A_t, verifies the proof.

$$A_t = (w_i)^{e_i} \tag{2.11}$$

- **VerNonMem:** The Equation 2.12 verifies a given non-membership witness $u_i = (a, g^b)$ for an element e_i.

$$A^a * (g^b)^{e_i} = g \tag{2.12}$$

3. RELATED WORK

In the present Section, based on a state channel survey [47], significant research efforts in the state channels domain are analysed. Those are discussed in four distinct groups, formed with each publication's focus as a criterion. Early state channel implementations may have a payment channel design as their main contribution but are credited with the earliest significant state channel implementations. General state channel designs have the development of a widely applicable state channel framework as their focal point. General state channel networks implementations have extended the idea of state channels to networks and virtual channels. Finally, there are application-specific implementations that while being centred around a specific domain and its needs, still present interesting findings.

3.1 Early State Channel Implementations

In this Subsection, we present the research efforts that conceptualised the state channels approach and made the very first steps towards implementing such schemes. Those designs paved the way for the evolution of the idea of payment channels and moved from updating only the balances of participants to updating the state of smart contracts while staying off-chain. The efforts described herein have repeatedly been leaned on by later proposals that refined those very first efforts.

3.1.1 Sprites and State Channels: Payment Networks that Go Faster than Lightning

Even though the main contribution of Sprites [44] is focused on a payment channels protocol, that protocol is designed in a modular way, based on a generic state channel abstraction. Sprite payment channels are modelled on state channels as they require an adaptable connection between on and off-chain processes. They incorporate a digital signature exchange scheme through which processes handle off-chain communications.

State channels are primarily used by Sprites protocol to implement its dispute handling process and to enable the adding and withdrawal of funds from an application contract without referring to the blockchain. They are straightforwardly described as a state machine that is constantly replicated between two parties and can progress through a

transition function that is application-specific to implement. In the channel environment, a guarantee for liveness is provided, as all participating parties are guaranteed that the most recent state is identical for every other party and can be eventually finalised on-chain at any time.

An open channel will go forward in rounds, during which parties will submit inputs containing signed messages that include the current round number, the state to which the channel transitions, and the resulting blockchain output if applicable.

A dispute can be raised in the case that a party tries to submit an invalid response or becomes unresponsive, which will result in a state lacking one or more signatures. In either case, an honest party can trigger the dispute process that includes providing evidence that the previous round has already been validated and informing the rest of the participants of the dispute process's commencement. The dispute can then be cleared off-chain if evidence that contributes to the validation of the controversial round or evidence that support a later valid round is submitted. In case of insufficient evidence, resolving can happen on-chain by any party and will result in the channel exiting with the most recent committed valid state.

The layout described in the Sprites protocol with regards to the dispute process is commonly encountered in other efforts since the same or a similar approach has been also used on many later proposals on state channels.

3.1.2 Perun: Virtual Payment Channels over Cryptographic Currencies

The work of Dziembowski[28] and others introduced an innovative modification on payment channels, as they were the first to present the concept of virtual channels, where intermediary parties enable payments between ”side” parties they are already connected to without being involved in the payments. Perun virtual channels paved the way for other efforts with respect to virtual channels and their main contributions were the minimisation of interaction with the blockchain and the increased privacy for conducted payments. The same concept has been extended later on by some of the authors[30] to be applied to state channels as well. Perun protocol describes an implementation of payment channels that are built recursively over a custom, novel design of a state channel that is called multichannel. Multichannels were constructed to fit into the role of supporting virtual payment channels, but they can also be used beyond that aspect, for generic state channels.

A multichannel is identified by a unique identifier that can never be repeated. It can be created in one of two ways: (a) parties cooperatively agree on the channel parameters and the round execution begins, or (b) an initiator begins the process without consulting the other prospective participants. For every created multichannel a corresponding smart contract is deployed on the blockchain, and the multichannel remains active as long as the said contract is up and running.

A distinguishing feature of multichannels is a delicate conditional state update mechanism. This essentially means that the user is given the option to only accept some of the proposed updates, in contrast to a standard state channel in which such updates have to be treated as a group. This feature has been developed to support the functionality of creating and running in parallel many contract instances, called nanocontracts, while a mechanism that prevents parties from initiating many concurrent processes that will lead them to overspend has been put in place.

In the optimistic case, participants can update the contract state after locally executing it, without contacting the blockchain system. An on-chain transaction is required in the case of a nanocontract's registration, which happens for every contract separately, in the case of a dispute between parties, and during multichannel close. To ensure the safety of the parties' funds, and also that participants will pay their due in every case, they must lock a stake in the nanocontract. In case a dispute occurs, nanocontracts are deactivated while it is ensured that no funds are blocked. Funds cannot be added to the contract after execution has begun, although authors state that such a functionality would be possible through a global double-spending mechanism for all registered nanocontracts.

A multichannel can be closed by any party, but only if all nanocontracts supported by it are inactive. Its state can be modified if all participants agree on the outcome. Otherwise, in case of a dispute, the protocol leaves it up to the channel's contract to handle malicious or inactive behaviour. This is done through an application-specific mechanism that terminates the channel by distributing nanocontracts' funds to participants if a certain amount of time lapses with no response after the dispute has been submitted.

The authors went on to further improve their work through a more recent publication[29] that extends PERUN and also includes a security analysis.

3.2 General State Channel Designs

The majority of state channels focused research efforts aim to realise adaptable, generic designs that satisfy as many requirements as possible. There are cases in which research efforts focus more heavily on specific aspects or requirements. Collectively such implementations are the backbone of the state channels ecosystem and offer various outlooks on building protocols that enhance the efficient operation of blockchain systems.

3.2.1 ForceMove: an n-party state channel protocol

Authors of the Force-Move protocol[21] identified that conflicting state submissions, external states dependence and inactivity are the three main factors that can lead to disputes between participants in a state channel. The Force-Move protocol was designed to combat the last one, namely the unresponsiveness of participants that threatens to lock all assets in a channel indefinitely. The authors have mainly focused on turn-based applications for which information required for dispute resolution is always fully embedded into the state of the application. In this way, they eliminated the external state dependency and conflicting states issues and focused only on providing an efficient solution for dealing with the unresponsiveness case. They primarily focus on applications that are or have similarities to, games, though any other application that complies with the Force-Move specifications, like payment channels, can be implemented through it.

The main component of the protocol is the Adjudicator: a most often, but not always, an on-chain smart contract that is integral to the functionality of this design. A generic interface to be implemented by the Adjudicator component of each different application is provided. This includes methods to interpret the different types of states as defined by the protocol, to facilitate off-chain communication and to validate states. A method called ValidTransition is arguably the most important method to implement, as it is the core of the protocol, giving the ability to separate a valid state transition request from an invalid one, and eventually produce the outcome of a dispute. The functionality of this method strongly depends on the content of the application and has to be implemented accordingly. Additionally, the Adjudicator is funded with the assets that participants are putting in as stake during their interaction. Those locked funds are the main mechanism that forces participants to act according to the rules of the application, as the Adjudicator releases and distributes those funds as indicated by the game's (app's) outcome.

A Force-Move channel's id has to be unique to prevent replay attacks, and it is the participants' responsibility to define it and double-check that the uniqueness requirement holds. State structure in Force-Move protocol is configurable according to the specific application that is to run, but there are some standard attributes, such as the number of the current turn and those that define how assets would be distributed to participants if the channel was to close at the specific state.

The key countermeasure that the authors applied against the inactivity of participants is the force-move method. It is triggered by one of the participants to force an unresponsive or non-complying opponent to act. It must be noted that in this protocol an invalid move is treated as a non-existent one. The force move process will result in either the smooth continuation of the game off-chain or its immediate termination and distribution of staked assets. This depends on the validity of the response of the challengee, or the absence of one. There are four valid ways for the challenged party to reply to the force-move request; respond with a move, prove a response to an alternative move, refute challenge, provide conclusion proof. If the challenged party responds with one of those four before the expiry of the dispute time window, the game progresses normally. Otherwise, it concludes based on the last accepted state.

There are five different phases for a Force-Move channel the PreFundSetup phase, the Collaborative phase, the Challenge phase, the Concluded phase and the Terminated phase. Force move challenges can be triggered only while the channel is in the Collaborative state, and triggers its transition to the Challenge state. The funding of the game happens externally, through an Adjudicator contract.

The authors themselves point out how their design is susceptible to various cases of malicious parties' behaviour and the causing of problems through firing invalid force move challenges, that still need to be responded to.

3.2.2 Counterfactual: Generalized State Channels

A "template" to make the adoption of state channels less of a burden for applications whose developers do not want to meddle with the blockchain is presented in this framework [23]. The design also allows the expansion of the template in an open-source style, providing an easy-to-use API so that added functionality can be developed for a channel and then used by any application that fits in a similar category. This concept has the added benefit of relieving developers of solely covering deployment costs and encourages sharing between

different applications.

The counterfactual design stands upon four pillars. The Safety Criterion ensures the protection of participants' assets and trustlessness in the channel. The Finality Criterion sets the requirement for final state validity. Equally fundamental are the Responsiveness and Off-Chain Desideratums that stress the importance of encouraging participant behaviours that align with basic state channel functionality.

Similarly to most existing state channel implementations, the Counterfactual approach is dependent upon the liveness of the blockchain system it operates on top of, the availability of channel participants and the insurance that they have no incentives external to the channel, as well as the absence of errors in the application code.

Like an abundance of state channel designs, this one is also vulnerable to griefing attacks and stale update posters. The first can be halfway mitigated by practices that either sacrifice privacy or require a very large stake to be effective, while the second is the main reason most frameworks only address turn-based applications, since they depend on this concept to punish the posters.

The name of the protocol comes from the counterfactual concept that the approach is centred around. Authors take the idea of an event that, even though it has not happened yet, it can happen at any time and everyone can behave as if it already has, and apply it with slight variations to the three main components of a state channel construction:

- Counterfactual Outcomes: finalizable channel outcomes that have not yet been finalized but participants can act as if they have been.
- Counterfactual States: states that can be brought on-chain by any participant that is involved with them, but still only exist in the channel.
- Counterfactual contract Instantiations: a new process of instantiating a contract to a channel without involving the blockchain. It requires a registry that maps the counterfactual code to its Ethereum equivalent and also predetermines how its Ethereum address will occur.

Along with the aforementioned registry, the other functionality that has to be implemented on-chain is participants' deposit holding. Authors claim that such functionality can be adequately implemented by a multi-signature wallet as long as it is secure enough. On

top of that, the authors state this approach brings in preferable characteristics, such as increased privacy and upgradability.

The proposal has been influenced by and made to fit on top of Ethereum's object-oriented design, resulting in high compatibility between the two layers. Every channel has a separate counterfactual state and functionality, and this approach facilitates the implementation of conditional payments and the enabling of instant on-chain fund deposits and withdrawals.

Channels that can be formed between users through a common intermediary are also implemented and are named Metachannels. Through this, the protocol supports the important functionality of virtual channels, which enables less interaction with the chain and has been consistently proposed in other research efforts since then. Also, popular third-party services like watchtowers [42], that aim at reducing the requirement for constant connectivity for participants, have been taken into account so that the framework is compatible with them.

3.2.3 Celer Network: Bring Internet Scale to Every Blockchain

The Celer Network [25] is an assortment of technologies aiming to improve scalability by any means possible. State channels along with sidechains are employed to that end, along with a value transfer routing method that increases throughput and a redesigned cryptoeconomic model.

The term network is used to describe the relationship between the various layers of the design, the base of which is the state channel and sidechain layer which is very relevant to the scope of this work. Celer uses state channels, called cChannels, to accomplish their goal of enabling fast off-chain interactions that can run on top of any blockchain system that supports smart contracts.

Because the design is destined to be blockchain-agnostic, authors identified common points between blockchain platforms that support the execution of Turing-complete programs, and incorporated such points in channels. One of those necessary components is an off-chain address translator that maps off-chain features to blockchain functions, through the use of a unique identifier dubbed as an off-chain address. Another key point, is the use of a Hash Time Lock Registry, which is implemented on-chain and is required for the implementation of atomic transactions that happen between multiple channels.

The concept of conditional updates and dependencies amongst states is greatly stressed in this work. The focus however seems to be on the more specific case of conditional payments. A detailed analysis is given for a General Payment Channel design as an example that could be of great use and fits the state channel specification. Based on common requirements that exist for state channels, authors have implemented several optimisations that are beneficial for their operation. There is an option for cooperative parties to sign every state as the final one and hence be able to close the channel in one transaction, and similarly, there exists a process that allows the opening of a channel in only one transaction as well.

Dependencies between states could bring great latency when claiming for a final state since it would be necessary to await the cooperation of every party that is involved with a depended-upon state. To avoid this overhead, a final state can be directly claimed by submitting a fraud-proof stake.

Celer also contains an alternative state channel model that is based on sidechains instead of the main chain of a blockchain platform. While it comes with added benefits concerning stake deposits and a reduced number of necessary on-chain transactions, it also has a set of drawbacks, namely the finality delay enforced by sidechains which will affect data availability and the absolute necessity of a fraud-proof bond to be deposited by the block proposer. Those disadvantages are recognised by the authors but no corresponding countermeasures are provided.

In general, no extensive security analysis has been done, and for the most part, the proposal seems focused on a payment channel analysis, without providing sufficient details on the dispute process that is required for the secure operation of state channels.

3.2.4 Two-Party State Channels with Assertions

State Assertion Channels[13] were presented, in an effort to disburden honest parties from shouldering the costs of issuing a challenge in case of inactivity or of an invalid state submission. The design of State Assertion Channels proposes an alternate dispute settling process. A party issues a challenge when a counter-party is inactive. After that, the counter-party shall respond with a state assertion (state's hash) within a predefined time window. A response to that state assertion is a second state assertion issued by the first party. Either the first party responds with the next state assertion, hence stating that they accept the previous one, or they challenge it by submitting to the contract the

full state in plaintext. If the assertion was indeed invalid, then the honest party gets the other's bond as a refund for the process.

Even though the requirement to stay connected to respond with assertions exists, the design is not compatible with outsourcing frameworks like Pisa[42]. Additionally, timers and countdowns are difficult to implement on blockchain applications without sacrificing security. The applications that can implement this state channel design are quite restricted since it only supports those that involve strictly two parties, interacting in a turn-based scenario that only allows single transition functions and never for an exception to be thrown.

3.2.5 You Sank My Battleship! A Case Study to Evaluate State Channels as a Scaling Solution for Cryptocurrencies

Kitsune [43] is another state channel design by the authors off [13]. It supports channels of n parties regardless of the application they intend to run and combines features from previous implementations[44, 28] in an effort to integrate their features.

The authors provide an application contract template to facilitate the deployment of any app within a state channel. This allows the application contract to cease its operations on-chain, transitioning to a "locked" state, given the fact that participants' consent to move the app to a state channel. The state contract is instantiated and the list of participants and duration of the dispute period are defined. While the app is running off-chain, any participant can submit a state update that will only become valid if signed by all parties. Close can occur optimistically, if all parties sign a closing state, or through a dispute.

Once a dispute is initiated, which can be done by any party, a dispute timer starts to force participants to submit the latest valid state they hold along with state identification information and corresponding signatures. The channel contract will store the most recent valid state and after the time-out of the dispute timer, the channel is set to OFF state. The dispute's duration and its outcome are stored on-chain, while the execution of the process returns on-chain. The application contract can also be deactivated if the state channel remains inactive for long time periods.

The authors attempt to analyse the performance of their approach, along with any challenges faced and also estimate the actual cost of running a battleship application on it. They conclude that the proposed state channels scheme is an optimal solution only for applications without a strong dependency on liveness. Also in the case of participants that are not cooperative by default, the usage costs outweigh any of the benefits.

The framework allows non-turn-based submission of states but does not address how a simultaneous submission is handled. Additionally, it is common for a party to start a dispute once they do not receive all requested signatures on time (with respect to a local clock) and end up moving the application execution back on-chain. This approach seems like it will complicate and delay the progression of the channel. The dispute processes only possible outcome seems to be closing the channel, and returning the application on-chain, even in cases where disputes could be resolved and the application execution could have been allowed to continue off-chain.

3.2.6 Hydra: Fast Isomorphic State Channels

In this proposal [17] the concept of isomorphic state channels is introduced, to replace the sequential state processing that is predominant in state channel approaches. Hydra channels address applications that are run by multiple parties but require those parties to remain connected throughout the process, having liveness as an important condition. The protocol's main advantages are funds' security, high performance and preservation of the smart contract capabilities on the state channel.

Hydra makes use of the Extended UTXO model [16] to provide support for general state machines and therefore make Bitcoin Turing complete and able to support state channels. This enables the Hydra channels, called heads, to make use of the underlying blockchain's state representation without any translations or modifications being required. Thanks to the EUTXO use, heads can boast faster confirmation times, simultaneous transaction processing and full asynchrony in the optimistic case, along with smaller round complexity. The authors provide simulation results to assess the efficiency of their design and compare it to baseline approaches, to lay out the performance related characteristics.

Creating a head follows a commitment process similar to other state channels. Any party can take on the initiator role and announce the identities that are invited to be head members. Public key information is exchanged between parties through authenticated channels to be used for on-chain transaction authentication and off-chain multi-signature based validation of state updates. An initial transaction establishes the head, initialises the state machine and forges the participation tokens for every party, which will be integral to the state verification procedure.

Since the channel progresses and transactions are processed, it is common for parties to have conflicting views of the head state. Snapshots are an effective conflict

resolution mechanism that also improves on storage load since any transaction included in them can be discarded. Snapshot leaders are tasked with creating and getting snapshots signed by all participants, in order to create points that can resolve conflicting state cases.

Any head party can procure a certificate for the current UTXOs in the channel at any time and use it to initiate the head's closing. Members are given a contestation period to upload UTXO certificates and the most recent valid one gets finalised on-chain with the end of the contestation period.

Beyond the basic protocol, Hydra functionality has been extended to incorporate, among other features, the support of committing and decommitting UTXOs in the head without blockchain interaction and the ability to execute an optimistic close without a contestation period.

3.2.7 Pisa: Arbitration outsourcing for state channels

Pisa [42] builds upon the concepts first expressed in the Monitor [26] and Watchtowers proposals, with the same end goal, eliminating the requirement for state channel's participants to be constantly online and synchronised with the chain, in order not to be vulnerable to disputes with unfortunate timing and execution fork attacks by malicious parties. Pisa introduces the concept of bringing in a third party, named custodian, to prevent the above incidents from taking place while a participant, dubbed a customer, is unable to do so themselves. The customer is also provided with evidence in the form of a receipt that can be used to punish the custodian in case they do not fulfil their obligations.

The implementation is supported on three contracts, that of the channel, that of the custodian, and one based on a simpler variation of the Sprites model[44]. Any channel participant wanting to hire a custodian to watch over their channel can submit payments in real-time to the custodian's smart contract, wired through a payment channel. In return, the customer receives a receipt that counts as evidence of the appointment and the accountability of the custodian. Custodians receive payment for every hash of state they are provided with, to be used to prevent an execution fork attempt. Note that the custodian gets a salted hash of the state and not the state itself to ensure channel members' privacy. Additionally, the nature of payment channels protects the customer from having their payment stolen, and the receipt protects them from a misbehaving custodian. However, this receipt has to be ratified before it is received, and the custodian will only do so after checking the validity of the conditional transfer sent by the customer, guaranteeing

fair exchange on both sides. Protocol design protects the custodian from being framed by parties or any efforts by them to increase the storage load. It does not, however, offer an efficient way to protect a customer from a custodian colluding with the rest of the participants if the benefit offered is enough to make the custodian willing to waive their stake.

This solution only works if the participant knows and has planned to be offline for a while, not in the case of an unexpected crash unless they are proactively paying a custodian. Also, it does not secure against a challenge addressed to the customer while they are online, only against the attempt for an execution fork attack.

3.2.8 Brick: Asynchronous State Channels

The authors of this work [6] have correctly identified the assumption of a synchronous network as a security weakness that most state channel implementations suffer from. Malicious entities are provided with valuable information, like the duration a network would need to be under attack for a successful disruption, and it is an obvious motive to attempt to censor honest parties during dispute periods so that they cannot respond to or receive challenges and hence timeouts can be manipulated to go off to the attacker's benefit.

The authors of Brick implement a system that needs to make no assumptions regarding message delivery to claim that it is secure. By redesigning the dispute handling process to involve a committee of third-party members, they have introduced intermediaries that enable the architecture to work on an asynchronous network and removed the necessity for parties to remain online.

In Brick, a valid state is defined as a state that has been signed by all members, is the most recent state (freshest) and has not been invalidated by the committee. A committed state has to have been signed by a sufficient number of committee members or be part of a block on the chain.

The phases of a Brick state channel's life cycle are divided into various sub-protocols. The initial phase is the Open phase, during which the channel is funded, its closing fee is determined, hashes of members' public keys are stored, and the collateral for committee members is deposited. The channel then moves on to the Update phase, during which the members create the announcement for the state that they agreed to follow. The sub-protocol that follows is the Consistent Broadcast phase when the said announcement is

sent by every party to the committee members in the form of a hash. Finally comes the Close phase, instantiated as the Optimistic Close, in which all members sign and publish the freshest state or the Pessimistic Close, for which a party requests committee signatures on the freshest state.

The authors of the paper execute an extensive analysis to prove they live up to the conditions they claim Brick meets. Security is achieved through the guarantee that a channel can only close in the freshest state, and liveness is assured since every valid operation will eventually be committed if not invalidated. To maintain privacy, there is no way for unauthorised external entities to gain information about the state of the channel before the initiation of a close protocol.

Incentives are provided to encourage honest behaviour assuming rational entities that aim to increase their profit. Committee members, apart from locking an initial collateral into the contract, also receive guaranteed fees for providing signatures and participating in the closing of channels. Therefore, the honest behaviour of rational committee members is guaranteed, and that also guarantees the security of the network.

Brick aims to get rid of the necessity for state channel participants to remain online for the entirety of the procedure. Consequently, the risk of parties losing their funds or otherwise finding themselves susceptible to attacks that would interfere with their connectivity is also eliminated. The authors lay down the groundwork by re-imagining the dispute handling process so that a committee is involved in it, guaranteeing network activity without relying on time windows. Apart from the intended result, this has the additional benefit of eliminating the requirement to measure time in the blockchain system, a process that is known to be problematic. A possible negative point of this design is the financial rewards that have to go out to responsive committee members that weigh on the channel participants.

3.3 General State Channel Networks Designs

Research efforts analysed in the previous Subsection have established the building blocks for developing state channel designs. To enhance the benefits of those designs, there have been subsequent attempts that aim at combining multiple state channels into networks. The concept of virtual channels has been integral to the state channel environment since it commenced the advancement beyond mere channel designs to channel networks

that could further benefit the scalability prospects of blockchain systems. State channel networks have attracted various efforts to develop robust and secure protocols for off-chain set up of state channels between blockchain users on top of already established state channels.

3.3.1 General State Channel Networks

In this work [30] the authors set a course to provide a formal definition for a state channel network, along with security specifications, that had been missing from the environment until that point. They present a design for virtual channel constructions, that are built on top of existing channels, funded on-chain and noted as ledger channels. Through this approach, it is feasible to operate a state channel without the requirement to commit on-chain transactions during its opening and closing and also to potentially resolve disputes through intermediaries.

Virtual channels are recursively built on top of multi-contract ledger channels, which enables every ledger channel to support many virtual ones simultaneously and every party to involve themselves in the off-chain execution of more than one contract at a time. There is no limit on the number of intermediaries a channel can span across, while there is an assumption for a synchronous communication network.

The described implementation allows the building of virtual channels even between seemingly incompatible cryptocurrencies, under the single requirement that they support smart contracts. Virtual channels also increase privacy as they function on a peer-to-peer communication scheme. The existence of an intermediary improves security, adding an extra step before forcing participants to resort to the blockchain.

The intermediary is contacted and if they agree to host the virtual channel, they optimally need to be contacted only during open and close phases, while they are not required to interact with the blockchain. In the pessimistic case, when a dispute occurs it is handled differently than on a ledger channel. Since the intermediary, contrary to an on-chain contract, cannot be trusted, the consensus has to be reached through the execution of contract instances on the underlying ledger channel of each participant with the intermediary. This enables the parties to settle on the last state signed by both. The intermediary is required to lock funds for every virtual channel they support for the channel to be functional, and those funds are protected by the contract instances shared with the parties that make use of the channel.

In terms of efficiency, the creation and optimistic execution of channels happen in a constant number of rounds, that does not depend on the channel's length. The execution of the channel in the pessimistic scenario requires a number of rounds that are proportional to the length of the channel.

The design is limited in that it only accommodates 2-party state channels, and that execution delay may be increased in comparison to on-chain execution since the execution happens in rounds and not in real-time and that triggers communication delays. Additionally, the requirements placed on an intermediary, locking coins for the duration of contract execution and the responsibility to be available to mediate disputes, may discourage a party from undertaking that role. Authors suggest that this shall be balanced with a service fee, but this would have to be sufficient to motivate the party while at the same time still providing a cheaper alternative than running the application on-chain.

3.3.2 Multi-Party Virtual State Channels

Two major additions in the state channel environment were introduced by the same authors [27] in a later stage. They expanded the 2-party virtual state channels design to support multi-party virtual state channels, while they also redesigned the dispute process, so that the time complexity of the channel, in the worst-case scenario, is independent of its length.

The implementation of multi-party virtual state channels happens by layering them on top of networks, formed out of 2-party ledger channels, that connect every participant to all other participants. That allows any party connected to the network to easily set up a state channel with any subset of the rest of the network participants. Those multi-party channels can execute off-chain contracts that concern more than two parties. To guarantee that the outcome of those contracts' execution will be reflected on the sub-channels, on which the multi-party channel is built upon, corresponding contract instances have to be instantiated on each one of the sub-channels. Those instances also protect the safety of the funds provided by the intermediary regardless of the behaviour of the rest of the participants.

The dispute board is the integral component of this redesigned dispute process. This process requires honest parties to rely on the ledger every time malicious behaviour is detected, in contrast to relying on the channel intermediaries. The dispute board functions as a state registration point on which all parties can register their latest valid view. In

case of a dispute, the honest party detects and uploads a valid state to the board while it allows for other parties to respond in a given time frame, measured in rounds. Submitted states are then compared and the valid one is finalised. In case of a malicious party not responding or trying to enforce a false view, all other participants should register their own instances on the dispute board. This new approach can significantly decrease the worst-case time complexity, even if it creates a larger overhead for the underlying blockchain. Also, because the dispute process results are published, they can be securely used by other processes/contracts.

Virtual channels that support this model are dubbed ”hybrids” since they adopt the same update and execution process as ledger channels but open and close as virtual ones. Channels that utilise direct and indirect disputes can cooperate seamlessly, in a balance that is according to the demands of each respective application.

The protocol makes some assumptions about the network, such as synchronous communication that guarantees the delivery of messages within a single round. This has as a result that adversaries, while being able to see and to reorder communications sent in the same round, cannot alter, delay, drop or add any. In terms of security, channel creation and update can only happen with the total consent of participants and as long as a single honest party exists, they can always ensure the execution of a state. Regarding the framework's efficiency, the creation, as well as the optimistic update and close of a channel happen in a limited number of rounds, while in the pessimistic case, those suffer a delay that is not dependant on the length of the channel. The authors claim that their design can guarantee high levels of fairness and efficiency in cases where a single malicious participant exists.

The proposed method is strongly coupled with the authors' previous work [30]. This is the reason why they recursively build long channels on top of two participants' channels and do not opt for the creation of channels with greater length from scratch.

3.3.3 Nitro Protocol

The Nitro protocol[20], extends an earlier scheme, ForceMove [21] and mainly relies upon three fundamental concepts; namely finalisation, redistribution and unbeatable strategies.

The first two are directly related to the process of channel termination, which should invoke the distribution of assets locked in the channel to its participants. Finalisation rep-

resents the storing of the state channel's outcome on-chain, while redistribution is an on-chain process (succession of operations) that enforces the redistribution of assets, that are locked in the channel according to this finalised outcome.

Participants operate on the basis of unbeatable strategies, which stand for the cases in which a participant can be sure that they can force the finalization of the channel to a specific state, given the data they hold, irrespective of the actions other participants may make. The series of actions the participant has to make is called an unbeatable strategy and is theoretically equal to a final event on-chain. Unbeatable strategies are defined with a slight difference between Turbo and Nitro, the two protocols analysed by the authors.

The simpler Turbo protocol allows for a single type of channel outcome and operation. Allocation outcomes essentially define the asset distribution between participants at every point in the channel. Turbo introduces manually set priority orders that govern the distribution of channel funds. The implementation of a channels' network is mainly based on Ledger channels, which are funded on-chain, and their sub-channels which operate completely off-chain and are funded by Ledger channels. All Ledger channels implement the Consensus Game, an operation that essentially enables participants to progress the state of the channel through finalisable outcomes they all agree on (called universally finalisable outcomes).

Nitro protocol builds on the foundations set by the Turbo protocol. It introduces guaranteed channel outcomes that define the priority order for an allocation outcome of a channel. It is possible to have more than one guarantee addressing an allocation, and therefore obtaining an unbeatable strategy is a more intricate process, as there are not only different possible outcomes for every channel, but also different ways to distribute value for every outcome, thanks to the guarantees.

Proper virtual channels can be created by the use of the Nitro Protocol. Any number of participants can create a virtual channel as long as there is a common intermediary. That requires the creation of a single allocation channel, and a guaranteed channel per participant targeting that allocation.

3.4 Application Specific Implementations

State channels research has been focused on developing generic protocols that could benefit the blockchain and network designs and be used for every possible application. However, this Section focuses on the special case of more targeted state channel implementations, that have been developed to support a specific application.

3.4.1 Æternity blockchain

Æternity is a blockchain platform that aims to support decentralised applications, while it integrates oracle services and scalability enhancements[35]. State channels have been integrated into Æternity, to make the architecture highly scalable. The only transactions that are recorded on-chain are those concerning finalised channels or settling disputes, and since channels function independently from each other, those few transactions can be processed by the blockchain in parallel.

An Æternity channel's lifecycle follows a pretty standard pattern.

To open, participants submit an on-chain opening transaction with the amount of funds they will commit to the channel. The starting participant essentially leaves the counterparty with a free option to co-sign and activate the channel at any point, which leaves their initial deposit in a vulnerable state. To mitigate this security issue, channels are funded through a series of interactive steps to protect the funds of honest parties.

During the operation of the channel, the two parties exchange signed channel states that are accompanied by a nonce that represents the states' order and enables the identification of the most recent state in case of a dispute.

Channel closing can happen by cooperatively signing a closing transaction, or through a dispute that allows a time period for the other party to challenge it with a newer state. Disputes are charged to the initiator, which initially unburdens the party that is usually at fault, but authors expect this to be balanced by the redistribution of funds that will occur if the dispute is valid.

Æternity blockchain platform operates only on the basis of state channels. Distribution of funds, as well as oracle message interpretation, are handled by Æternity smart contracts that provide the basis upon which state channels are built. Those contracts do not store state, but only securely output it after a transition takes place. Contracts are

maintained locally by channels' participants and are deployed on-chain only when a dispute occurs. This provides enhanced privacy for every interaction. Hashlocks are used in Æternity to enable functionality that resembles virtual channels, allowing parties to interact with each other even if they do not have an active channel established between them, as long as they are connected by a path of state channels. However, this is a bothersome process for the intermediaries that will have to lock an immense amount of funds, and will probably need to be compensated by a fee, making this a sub-optimal option, especially if parties wish to interact more than once.

3.4.2 FunFair Technology Roadmap and Discussion and A Reference Implementation of State Channel Contracts

Funfair technologies attempted to enhance the casino gaming experience, by decreasing costs and latency through the use of state channels[41]. Their state channel design, named Fate channels, operates much like most other state channels, but focuses mainly on serving the needs of online gambling and thus focuses on actions/events that could potentially happen during such a game.

The lifetime of a fate channel is equal to a single game session, during which parties, usually a client and a server that is the house, can communicate in rapid succession through a channel they have placed a stake in. The user is charged with the opening fees of the channel. Updates happen while participants sequentially sign new states. The client can terminate the channel whenever they wish through a cash-out option, while the house covers the closing fees.

Fate channels were described in more detail[40] in a later stage. Funfair reintroduced their bidirectional 2-party channels and went on to analyse the technical processes through which they operate.

Funding is based on ERC20 tokens and it takes advantage of their built-in multi-sig capability. Participants co-sign a single transaction that transfers tokens from both of their accounts to a third account, which funds the channel. Participant balances are stored on the channel, but the channel's total funds are kept on-chain to prevent inconsistencies while multiple channels are simultaneously running.

To open a channel, parties have to deposit funds and sign the opening commitment. A timestamp is used on the commitment so it cannot be maliciously exploited at a later date. The application's code exists on the state machine so that it can be accessible

and verifiable. The first, or initial state of the channel has to be signed by both parties before the opening of the channels because at least one reference point is necessary for a dispute or the channel's close.

An action is the only way to advance the state, and only a single party can be taking one at a time. Once an action is submitted, the state machine checks that the suggested state is valid and the occurring balances non-negative before approving it.

The protocol's fundamental rules require that participants react in a reasonable time after receiving a state update. Upon receiving a state update, they have to perform necessary validation checks, sign it and send it back. A dispute can be raised by a party when the counter-party strays from that setup. Disputes can be resolved by a valid response by the counter-party, if they notice it, or by a timeout. Dispute initiators with malicious intent, that may challenge with a stale state or try to change their move through a dispute, are punished.

At the channel's close, parties send a signed close action on-chain with all the necessary parameters so that the state can be deemed valid and finalisable.

The State machine maintains a contract for each game or game type.

3.4.3 Game Channels: state channels for the gambling industry with built-in PRNG

Game Channels[18] proposal also targets the implementation of gambling and casino games through state channels, with authors targeting 2-party fraud-proof channels. The process for game channel use is quite straightforward and along with the basic principles of most state channel frameworks.

Typically the two parties are a player and a dealer since everything is discussed in the scope of casino games. The player initiates the process to create a channel, but both parties have to agree and sign an initial state, and that has to be verified by the on-chain contract for the process to move forward. The player subsequently submits the tokens to be held in escrow to the dealer, and the contract confirms both parties are in full consent for the creation of the channel and realises it.

The state advancement process happens in rounds during which participants take actions regarding the game and communicate those to the dealer, who in turn provides an answer. As long as there is no cause for dispute, the channel keeps advancing to the next round.

Every time the state of a game channel is updated, a transaction with game-related information is sent to the blockchain, which partly hinders the supposed state channel functionality of reducing transaction load on the chain.

Either participant can place the call to close the channel and activate the protocol that corresponds to the situation. Specifically, the approach to closing differs whether it is a cooperative close, funds have run out, the channel has reached its expiration point or if a party is faced with a maliciously behaving (non-responding or submitting fraudulent data) party. In case of dispute, the validity of the data comprising the claim is checked by the smart contract.

Authors have modified their base protocol to expand its use cases and include channels between two players with no dealer involved, and also a setup where a third party can observe the game without participating in it.

3.4.4 Cryptopoly: Using Ethereum State Channels for Decentralized Game Applications

Through the Cryptopoly implementation [59], the author aims to prove that developing a complex game like Monopoly on the blockchain, given that it is turn-based, is both feasible and cost-effective.

After choosing Ethereum as a platform, the necessity to address scalability issues led them to state channels as the optimal option with regards to latency and usage fees. The design adopts the dispute mechanism from Force Move [21], as well as the random number generation process used in Fate Channels [41].

A state machine with an approach that is as general as possible was designed since Monopoly is a highly customisable game. An Ethereum contract to handle the functionality of the channels (open, close, dispute) was also developed. Because validation processes differ per application, a generic applyChange method is used for its adaptability compared to hard coding the parameters. The specific effort has limited contribution as it only aims at applying already developed approaches to a specific application.

3.5 Analysis of implementations

3.5.1 General Analysis

Through the extensive study of the existing research efforts in the state channels' domain, various aspects to be commented on have emerged. Figures 3.1 and 3.2 depict observations with regards to the general landscape as this has been formed by all papers presented in Chapter 3. In both figures, a division of the efforts according to their field of origin can be discerned. The blockchain ecosystem is a diverse space and this classification provides insight regarding which sector (academia/industry) was motivated to conduct research and also what the level of interaction between the two sectors is. Additionally, this distinction also provides data as to which designs are functional purely within an academic environment and which expand beyond that and into available and usable products. The categories are the following:

- **Academia:** includes papers that have been published by academic researchers, affiliated with a university/research institution.
- **Industry:** includes any research done by commercially oriented groups, e.g. startup companies, usually concerning a product.
- **Combined:** includes any work published either by authors affiliated with both aforementioned fields, or a group of authors that have a different field of origin.

It should be noted that the focus of the analysis is state channels. Therefore, for schemes that partially included state channels[28, 44, 25, 35], any evaluation attempted only concerns the relevant parts of the overall designs.

In Figure 3.1 the categorization of the evaluated papers according to the divisions is especially useful since it enables observations regarding which sector was more active per time frame.

To visually illustrate the progression and evolution of the state channels ecosystem since the concept's introduction in 2015, Figure 3.1 depicts a timeline that shows all the major milestones and every state-channel related publication. It can be observed that for the first period from 2015 through the beginning of 2017, the only noteworthy advancement is only a state channel proof of concept implementation[32]. In the first quarter of 2017 the first specification for a state channel design as a part of a larger payment channel

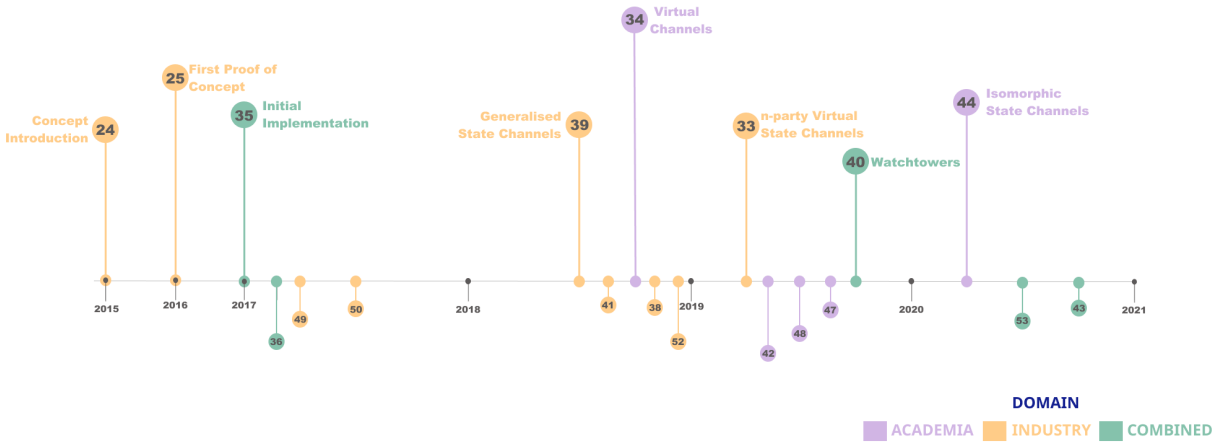

Figure 3.1: Timeline

scheme is analysed in Sprites [44], while Perun [28] follows the same year on a similar note. Counterfactual [23] provided a very detailed design, that was the first to be exclusively state channel related, in early 2018. In the second half of 2018, state channel networks and the concept of virtual channels is introduced by Generalised State Channel Networks [30]. At the start of 2019 comes the noteworthy extension of the concept to n-party channels, with Nitro [20] being the first protocol to build on the idea. In the second half of 2019, Pisa [42] focuses on resolving the common security assumption according to which all state channels' participants are required to be constantly online, extending the concept of Watchtowers that had been introduced earlier. Another work focused on the same problem but with a different approach that comes around the same time is the design proposed by Brick[6]. The most recent noteworthy contribution regarding state channels is the Hydra [17] proposal that presents isomorphic channels, a very interesting enhancement.

Figure 3.2 also utilises the distinction of proposals according to their sector of origin introduced in Figure 3.1. This allows the comparative evaluation between this factor and every other aspect of the radar.

All designs analysed have been evaluated for their technical maturity. In Figure 3.2, efforts are depicted on a radar with those with a more well-rounded and advanced implementation in the outer ring. This allows the evaluation of the average quality of implementation per sector of origin. It can be seen that the vast majority of papers stemming from the industry and startup sector average a high level of technical maturity, while aca-

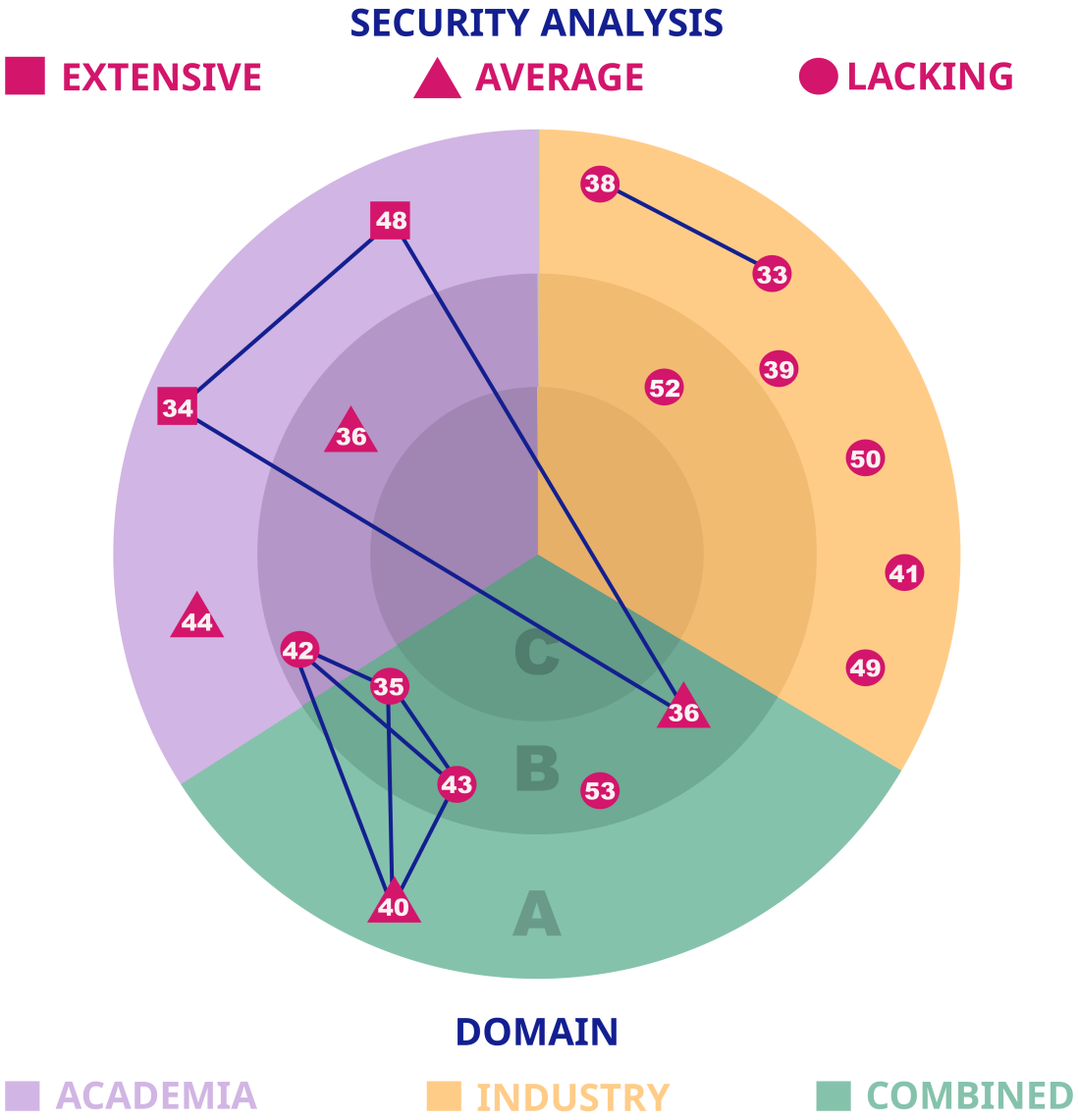

Figure 3.2: Radar depicting publication source, implementation quality, correlations and security analysis extent

demic papers are split between the two most mature rings of the graph. Proposals that occurred from the combined efforts of academia and the industry sector tend to fall on the second ring. It is a fortunate observation that the works analysed herein score a high average on the quality and extent of their practical implementations, with all providing a more or less extensive technical analysis in their appendices or through a corresponding code repository.

Another dimension in Figure 3.2 is the extent of the security analysis present in each research proposal. According to the thoroughness and detail of the analysis, three distinct ranks have emerged.

- **Lacking:** The majority of designs seem to find it sufficient to mention the security properties state channels inherit from the underlying blockchain system, or to claim security guarantees fundamental to the state channel, such as liveness and fund safety. They do not go as far as to analyse how those concepts are achieved, how the blockchain system's security extends to the protocol, and how security vulnerabilities occur because of the properties of the specific proposal. For example, more often than not, the way the dispute process is designed in ways that leave honest parties vulnerable to griefing attacks, or the channel's security becomes obsolete in the case of a software error in the contract. Methods to secure against those phenomenons are not provided by designs that fall in this category.

- **Average:** Designs that have been deemed to adequately cover the security analysis aspect by providing proofs for the security claims they make, have been placed in this category. Authors usually do that according to a defined threat model that seems to cover possible cases to a satisfactory degree.

- **Extensive:** There exist a couple of designs that provide very rigorous security analysis and use established protocols like the Universal Composability framework[15] or similar variants to evaluate the security of their design to the utmost extent.

It can be noticed that papers originating from the industry domain seem to be lacking a formal security analysis, while publications from academia tend to score a higher average on this front. Papers from the third domain appear to be falling in between.

Additional information that can be drawn from Figure 3.2 is the correlations between published works. The edges coloured in dark blue join papers that share at least one

Table 3.1: State channel implementations comparison

author. This enables evaluation of the diversity in the state channel research field as well as observation of interconnections between works that seem to have a different sector of origin. It has to be noted, that there is significant interaction between academia and industry on the state channels domain.

While it may seem like an obvious categorisation in terms of supported blockchain platforms has been omitted, the vast majority of designs address Ethereum or Ethereum-like platforms that support smart contracts. The only exceptions can be found in Hydra [17] that means to be implemented on Bitcoin running EUTXO protocol and Aeternity [35] that is in itself a blockchain platform.

3.5.2 Comparative Analysis

Through the analysis of the state of the art, common points between research efforts have been identified. Through this process, the main restrictions imposed by authors for their protocols to be functional along with the main features those offer have been summarized and are presented in Table 3.1.

Many proposals impose restrictions on the applications their state channel design

supports. Those restrictions may be limiting but at the same time, they enable designs that are robust and can securely support a specific subset of apps. On the other hand, it is not uncommon for proposals to omit to place such restrictions and then fail to adequately analyze how the issues that may occur from the allowed scenarios are handled. The first section of Table 3.1, titled Requirements, depicts said restrictions that are present or absent in each of the works analysed. The Table indicates which are the specific conditions each research effort places on applications in order to support them. When this is not the case the Table indicates whether or not each scenario has been explored to a satisfactory degree.

The second section of Table 3.1, entitled Features, presents positive characteristics supported by each design. It is desirable for each effort to have as many checks as possible on this end of the Table, as that corresponds to offering a wide variety of features and covering an abundance of scenarios. All in all, an ideal design would score zero marks in the Requirements section and all marks in the Features one. Of course, no such proposal exists, but a detailed comparison between existing proposals can be conducted based on Table 3.1.

3.5.2.1 Requirements

A thorough explanation of the meaning of each set requirement follows:

- **Turn-based Applications:** To facilitate the handling of the dispute process and the assignment of blame, most designs limit their applicability to strictly turn-based applications. This has the added benefit of not getting caught up in trying to handle simultaneous state update proposals and facilitating the process of punishing stale update posters. There exist a number of applications that do not impose or do not address this parameter. However, most of those designs do not explain how to efficiently and securely handle applications that do not adhere to the turn-based paradigm.

- **Timers:** In the great majority of cases, dispute handling is based on challenge-response schemes and thus requires a time measuring operation, to manage a time-out period. This is necessary to allow an interval during which the dispute can be contested, while also ensuring the channel's progression will not stall indefinitely. However, there still is no provably secure way to measure time on the blockchain.

Using block time is sub-optimal as it is subject to change and also bounds the granularity of time measurement.

- **Predefined Participant Sets:** A requirement that has been present in every one of the works discussed in this analysis is that the set of participants forming a state channel is predefined and remains unchanged through the channel's entire lifespan. The reason this is such a widespread assumption is that changes to the participant set would complicate the funding process and require communication with the blockchain for the addition or removal of a party. While this assumption simplifies the development of the protocols, it substantially limits the applications that can be served by such designs.

- **Constant Connectivity:** This assumption is derived from the conditions related to dispute resolution. Since, during this phase, tends to be a time limit on the ability to respond to/contest a dispute, a party that gets offline for any reason at an unfortunate moment can face severe consequences. Because an online connection can be unpredictable, having such significant stakes depending on it, is a major security issue. Pisa [42] and Brick [6] recognise the importance of this matter and focus their designs on the effort of mitigating it. Because those two efforts are rather specific and aim to diminish only this specific requirement, they cannot be directly compared to other solutions and they have not been included in the comparison of Table 3.1.

3.5.2.2 Features

- **Multi-party Channels:** The initial payment and state channel designs could only accommodate interactions between two predefined participants. The same is true for the initial virtual channel designs, that could only be formed between two parties. Proposals further down the road enabled communication via state channels, virtual or not, amongst an unlimited number of participants, given that those were predefined. Such a feature allows a wider set of applications to be deployed on top of state channels.

- **Off-chain Channel Creation:** In order to further limit the transaction load on the blockchain, attempts were made to enable the creation of state channels without any interaction with the chain. The result of these efforts was Virtual Channels, otherwise called State Channel Networks ([20], [30], [27]). The primary feature of

such designs is that they allow parties to form state channels off-chain, as long as each party has already formed one with a common intermediary. This can also be extended, providing the ability to form channels off-chain as long as there is any path of preformed state channels that can connect the participants. The drawbacks of those designs stem from the necessity in most designs, that an intermediary exists. Given Users A,B and C, and assuming B will be acting as an intermediary between A and C, it is necessary that B has enough funds committed in their state channel with A to cover C's bond, and vise versa. Essentially, User B's funds are now locked and unusable. On top of that, the funds either A or C can use are limited to the amount B has in their respective channels. This limitation increases in severity if the path between A and C is even more extended.

- **Smart Contract Compatibility:** An important point to consider when deploying an application to a state channel is related to developing a smart contract to support the application. The installation of the application has to cope with the requirements set by the state channels design used in terms of the contract that needs to be running on-chain. The least the development effort required to port an application from on-chain to a state channel the better. Low compatibility refers to the requirement for a contract to be written from scratch to be state channels compatible. Medium compatibility corresponds to the majority of the cases, where a contract is expected to go through some alterations (make use of a library, implement a specific function) to be state channel ready. High compatibility is only achieved in a handful of cases that allow the deployment of an existing contract to a state channel with no development requirements.

- **Parallel Contract Execution:** Virtual channels are built on top of ledger channels. Ledger channels need to have a contract instantiated in them for each virtual channel they support. Therefore, a channel with the ability to process more than one contract at a time also enables the support of more than a single virtual channel at a time. This feature enables more efficient use of deployed ledger channels as it provides a greater magnitude of application capacity for a given set of established channels.

- **Local Storage Optimisation:** A design must prioritise keeping the storage requirements to a minimum by preventing the state from endlessly expanding and relieving the users from the necessity of storing a vast number of past states in order to be able to enforce the design's resolution mechanisms.

4. ORIGAMI-PROPOSED DESIGN

4.1 Concept

Origami design is based on the concept of building off-chain interaction protocols (state channels) upon other similar protocols, in order to allow the creation of constructions characterized by the required level of flexibility. There are three types of off-chain interaction protocols, multiple instances of which are used to construct the origami state channels environment. In Figure 4.1 an abstract schematic view of the Origami environment is depicted. Each one of the blocks represents an instance of off-chain interaction protocols and it is based upon the block directly underneath it. The base protocol (single instance) is similar to a traditional state channel and enables users to deposit funds through on-chain transactions and take part into the Origami system. The group protocol is similar to the base one with the main difference that it has to be funded on the single base protocol instance or on another instance of the group protocol. The main functionality offered by the group protocol instances is that they enable participants to form smaller and more flexible subsets of users according to their needs, without having to do any on-chain interaction. Finally, the app protocol instances are based upon the single base protocol instance or an instance of the group protocol and are application specific thus enabling users to run applications off-chain.

The Origami approach facilitates this hierarchical construction of multiple off-chain interaction protocols to support multiple features that are missing from existing schemes, while keeping the required on-chain interaction of the users at a bare minimum. The name of the design (Origami) comes from the Japanese art of creating paper structures through multiple foldings of a piece of paper, a process that our multiple off-chain protocols resemble to. The intuition is that a large number of users can come into the broad user base of the system, by taking part in the base protocol instance. As subsets of those users want to interact, they will tend to form instances of group protocol and eventually instances of the app protocol, for which the set of participants is going to be driven by the requirement for interaction. The lesser the participants in a group/app instance are, the more efficient it gets in terms of operation (e.g. number of required signatures to progress state, probability of inactivity or bad behavior, etc.) The state of both base and group instances is strictly defined and relates to a balance-sheet with the funds of each participant in the specific protocol and the protocol instances on top of that to which part

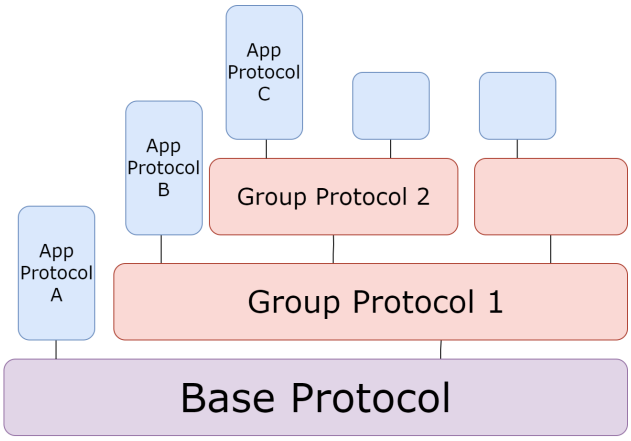

Figure 4.1: Origami concept.

of the funds have been committed. On the other hand, the state of the app instances is application specific and is defined in an ad-hoc way according to the application that is going to be deployed on top of such an instance.

The proposed design aims to support an ecosystem of users that only go through the process of joining the system once (through one on-chain transaction), but can then run any number of state channel compatible applications with any subset of the rest of the participants (without on-chain interaction in the optimistic case).

Central to the architecture are the following protocols:

- **The base protocol** is used as the initial step that allows users to join the ecosystem and deposit funds.
- **The group protocol** is a scaling mechanism that aims to decrease any overhead that is bound to occur as the ecosystem grows.
- **The app protocol** through which a subset of members of the ecosystem can operate an independent channel, created with the intent of running a state channel application.

Apart from the base protocol that is directly attached to an on-chain smart contract and requires users to deposit funds in it, the rest of the protocols operate completely off-chain in the optimistic case and with minimal on-chain interactions in the pessimistic case. The functionality of each protocol, along with the interactions between their instances, is extensively analysed in the present Section. The terms channel and protocol are used interchangeably. Additionally, the terms supporting channel and overlying channel are used

in the text to describe the relationships between protocols of different levels. Presenting Figure 4.1 as reference, the Base Protocol is the supporting channel for Group Protocol 1 and App Protocol A. Successively, they are overlying channels to the base protocol. Similarly, Group Protocol 1 is the supporting channel with Group Protocol 2 and App Protocol B as overlying channels, and those terms can be applied to any pair of protocols that differ by one level.

In the subsequent subsections, the three different protocols are presented. As the protocols are quite similar and their differences are restricted to specific parts of their functionality, the base protocol is described in detail while subsequently, we present the group and app protocols on the basis of their differences from the base protocol.

4.2 Base Protocol

An important building block for Origami is the base protocol, an instance of which is identified as the base channel and operates similarly to the conventional notion of a state channel. It functions as the entryway for those who wish to join the ecosystem. All users that want to participate in Origami must become members of the base channel after depositing funds into its corresponding smart contract.

Every phase and process of the base channel will be presented in this subsection.

4.2.1 State representation

Base channel state updates pertain to management related actions in the ecosystem, such as the addition and removal of participants, and to supporting the operation of instances of the higher levels protocols, such as group and app channels. The state of the base channel is updated whenever there is a change in the participants set (a user is added/removed from the ecosystem) or there is a change regarding the group/app channels that are deployed directly upon the base channel (a channel is opened or closed). Therefore, the form of the state of a base state channel adheres to the following template, also seen in Figure 4.2:

The state of a base channel consists of three static fields and one dynamic field the form of which is dependent on the nature of the last state update. The five fields are:

- **Universal Channel State:** The Universal Channel State (UCS) is a set of channel-

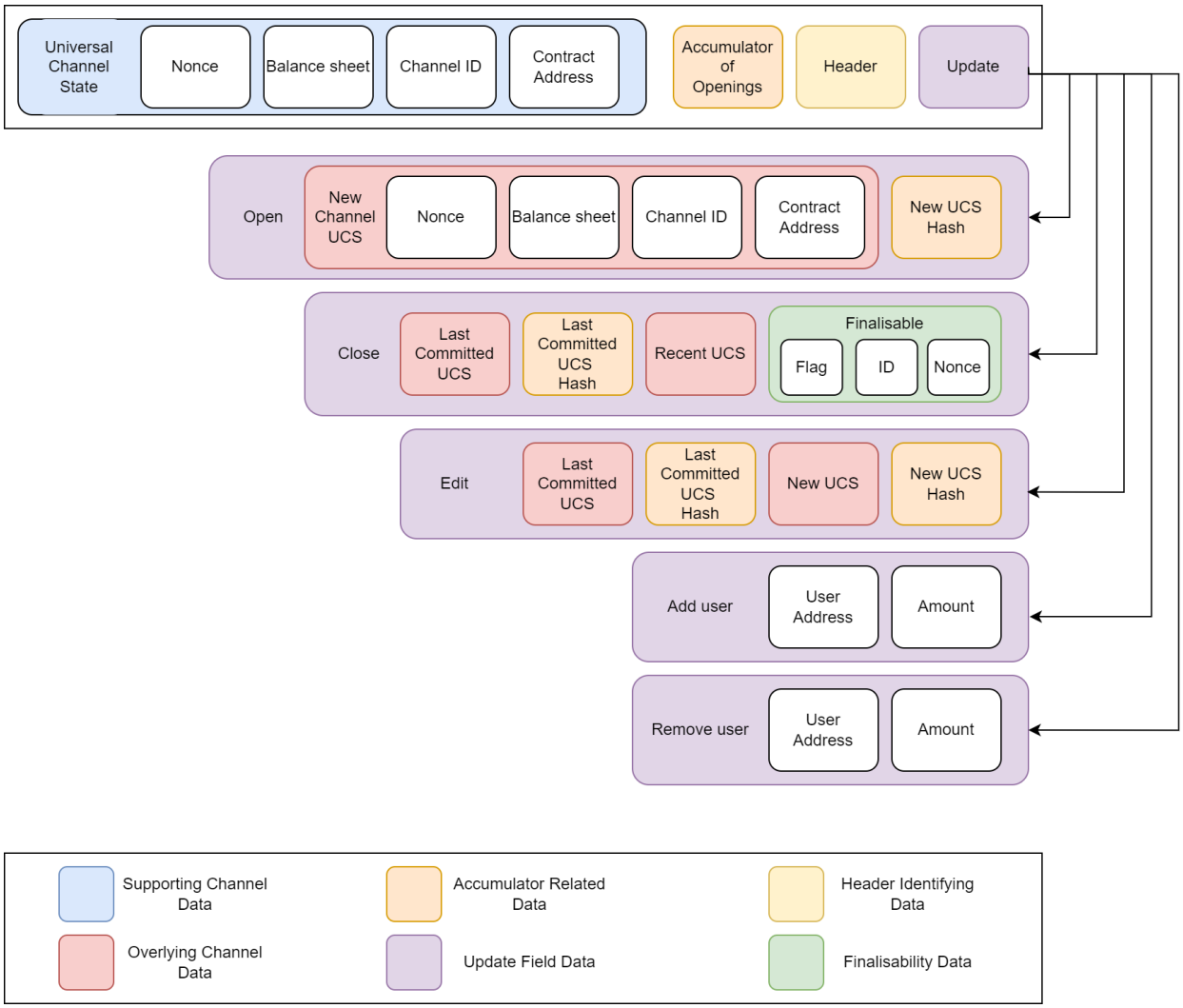

Figure 4.2: Base channel state structure

defining data that facilitates the transferring of information between protocols in different layers. It is separately signed before it is included in the state and contains the following information:

  - **Nonce:** A counter that is used to order states in a channel and is useful when trying to prove a state is stale (a more recent valid state exists).
  - **Balance sheet:** A mapping of the members of the channel to the amount of available funds they possess in the channel. Balance sheet, apart from defining available funds, is also used as a point of reference for the list of members of a channel.
  - **Channel ID:** Each channel is assigned a unique identifier for management purposes. For the base channel (which is the only one across the ecosystem) the channel id is specifically set to 0.
  - **Contract Address:** Each channel, irrespective of its type (base, group, or app) is based upon a deployed smart contract, that holds all the required functionality to support the channel (e.g. validation of state transitions). When a new channel is instantiated, the address of the corresponding smart contract is specified in this field.

- **Accumulator of Openings:** A structure that stores commitments to all open channels on top of the state base channel. For efficiency reasons and in order to keep the size of the state representation fixed the Openings structure is implemented as an RSA cryptographic accumulator. For every new channel that is opened upon the base channel an element (new channel's UCS) is added to the accumulator. If such a channel is closed then the corresponding element is removed from the accumulator. The new channel's participants are responsible for storing the element on their side and also updating the corresponding membership proof of the accumulator for that element.

- **Header:** Origami follows a unique communication design that requires setting a member as a Header for each round. This enables unordered state updates from channel participants. A state indicates who will be acting as the Header for the next state update round. The relevant process is analysed in Section 4.2.2.

- **Update:** This is a complex field that indicates the type of state update being proposed and also holds all the required information for that update. There are four

update instances that are related to (a) opening a new channel (b) closing an existing channel (c) editing an existing channel, (d) adding a user or (e) removing a user.

According to the type of the update, the update field is structured as follows.

- For opening a new channel the sub-fields of the update field are :
  - **New Channel's Universal Channel State:** The Universal Channel State (UCS) is a set of channel-defining data that facilitates the transferring of information between protocols in different layers. It contains the following information:
    - **Nonce:** A counter that is used to order states in a channel and is useful when trying to prove a state is stale (a more recent valid state exists).
    - **Balance sheet:** A mapping of the members of the channel to the amount of available funds they posses in the channel. It reflects the assets members commit to the newly created protocol.
    - **Channel ID:** A unique identifier assigned to every protocol upon creation for management purposes.
    - **Contract Address:** Each channel, irrespective of its type (base, group or app) is based upon a deployed smart contract, that holds all the required functionality to support the channel (e.g. validation of state transitions). When a new channel is instantiated the address of the corresponding smart contract is specified in this field.
  - **New Channel UCS Hash:** a hash of all the fields in the UCS to form an element that can be added to the Accumulator and facilitate verification of the state transition.
- For closing an existing channel the sub-fields of the update field are :
  - **Last committed UCS:** The last Universal Channel State (UCS) that has been committed (through an open or an edit state update) for the channel to be closed in the Accumulator of Openings.
  - **Recent UCS:** The UCS that corresponds to the last state of the channel that will be closed.
  - **Hash of last committed UCS**
  - **Finalisability** A set of three values:

- **Finalisable Flag** is a value set to true only for states deemed as appropriate to close the channel on.
- **Nonce** must be identical to the nonce of the most recent state the channel wants to close on.
- **Channel ID** must correspond to the ID of the channel to be closed.

– For editing an overlying channel (removing or adding a user) the sub-fields of the update field are :

* **Last committed UCS:** The last Universal Channel State (UCS) that has been committed (through an open or an edit state update) for the channel to be edited in the Accumulator of Openings.
* **New UCS:** The UCS that corresponds to the updated data of the edited channel.

– For adding a new user, given that the user has deposited an amount in the smart contract of the base channel, sub-fields of the update field are :

* **User address:** The address of the user to be added.
* **Amount:** The amount of funds the user has deposited and are being added for them in the channel's balance sheet.

– For removing an existing user, given that the user holds a specific amount of the funds in the channel, the sub-fields of the update field are :

* **User address:** The address of the user to be removed.
* **Amount:** The amount of funds the user has and will be withdrawn from the channel.

4.2.2 Communication model and disputes

Existing state channel designs assume that the ordering according to which the participants may propose state updates is strictly predefined. This mainly happens because having a single valid state proposer at each round significantly simplifies the process of validating the state proposals.

Because Origami design aims to eliminate such ordering restrictions in terms of which participant is entitled to make the next update proposal and lift the universal restriction of running strictly turn-based applications within the state channel, we have adopted a

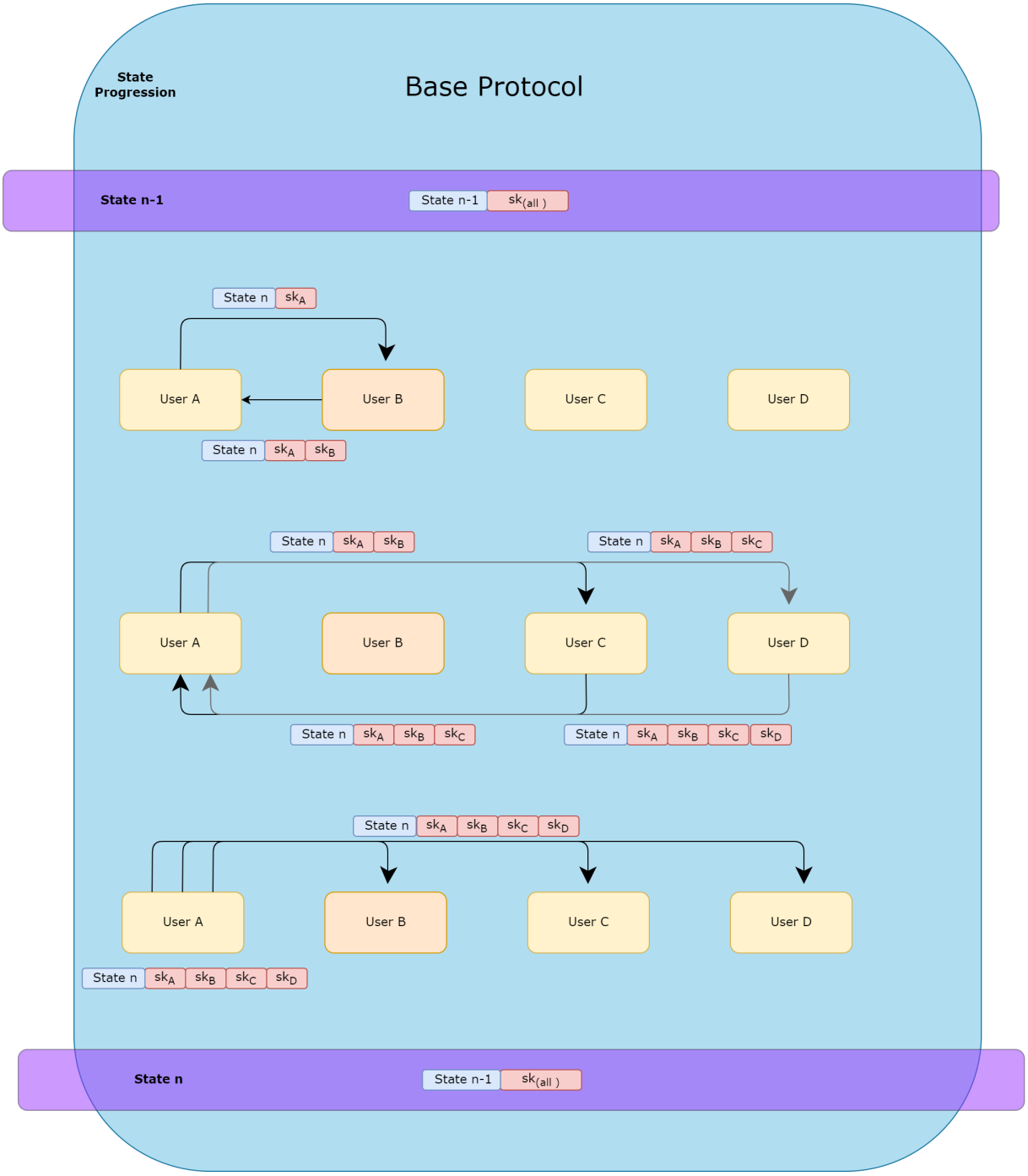

Figure 4.3: Depiction of the off-chain communication between participants.

novel way to regulate in-channel communication. The process can be observed in Figure 4.3, and the analysis is based on the notations of the same Figure.

Lets define a round as the duration it takes for a new state to be established, ergo the process of going from state $n-1$ to state n. For each round, a channel participant acts as a header, in the sense that an update proposal must first pass through them. The header is determined through the order of the member entries in the balance sheet of the respective protocol, starting from the first entry and moving along by one each round. This ensures an equal distribution of this position among the participants.

The process of establishing a new state can be described in the following steps:

Step 1: Assuming User B is acting as the header, and User A intends to send an state update proposal, then User A must sign this proposal and first send it to User B. If this state update is valid, the header signs it and sends it back to the proposer.

Step 2: User A now broadcasts the proposal (signed by the header) to the rest of the participants. They, in turn, sign it and send it back to User A, only if they can see it has already been signed by the round header.

Step 3: User A, having collected all necessary signatures on their proposal, broadcasts it as the new state.

This method ensures that, while there is no sole participant able to suggest the next update, there will also be no confusion or conflict if various participants concurrently propose their own state update. In case the header receives more than one valid state update proposal in one round, then they are entitled to choose which one to progress with.

It is important for the rest of the channel participants to be able to distinguish between the header opting for another proposal and the header being inactive. Therefore, as shown in Figure 4.4, the header has to provide all proposers with the chosen proposal, so that all involved parties know the channel is progressing even with an alternative state.

In order to support the described protocol Origami includes two types of disputes, the header inactivity dispute and the channel member inactivity dispute.

Header Inactivity Dispute

If a state proposer does not receive a reply from the round header (either their or another participant's state proposal signed by the header), then they are entitled to go on chain, and initiate a header inactivity dispute, as depicted in Figure 4.5. The process is

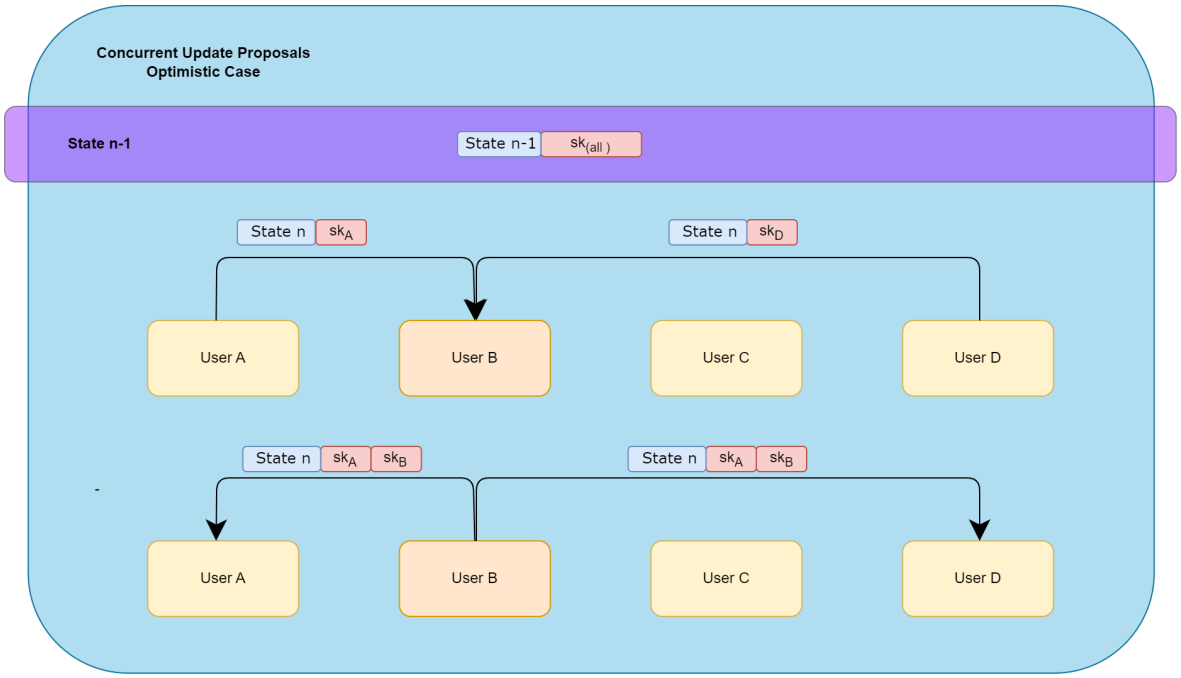

Figure 4.4: Optimistic case protocol for concurrent update proposals.

as follows:

Step 1: A channel participant, in this case User D, sends a proposed state update to the round's header and receives no reply. User D is therefore not aware that User A has also sent a proposal, and they are entitled to start a dispute through the on-chain contract.

Step 2: This dispute initiates a first countdown during which the header must provide a signed proposal they are proceeding with, whether that is the one from User A or User D. If this happens, then a second countdown is initiated to allow this proposal to be challenged in case it is stale.

Step 3: If the round header provides to the contract a valid response to the dispute, then the successful proposer can retrieve their signed proposal from the contract and broadcast it to the other channel members and continue with the progression of the channel. Alternatively, if the round header never provides a valid reply to resolve the dispute, then they suffer a penalty and the contract moves the channel on to the next header.

Channel Member Inactivity Dispute

If a member exhibits inactive behaviour by refusing to sign a valid state, a dispute against them can be triggered by any participant. The process is shown in Figure 4.6 and described by the following steps.

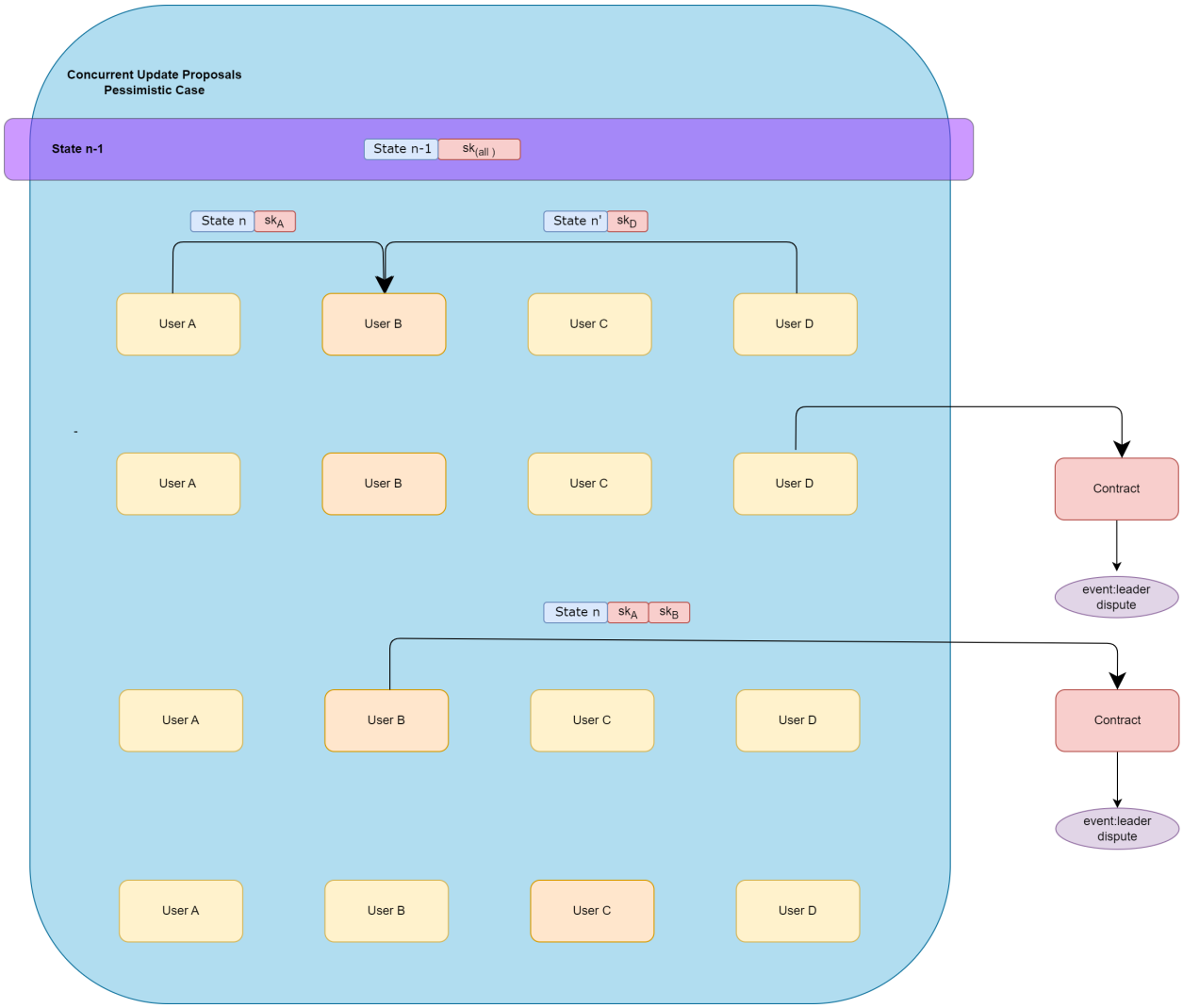

Figure 4.5: Pessimistic case protocol for concurrent update proposals.

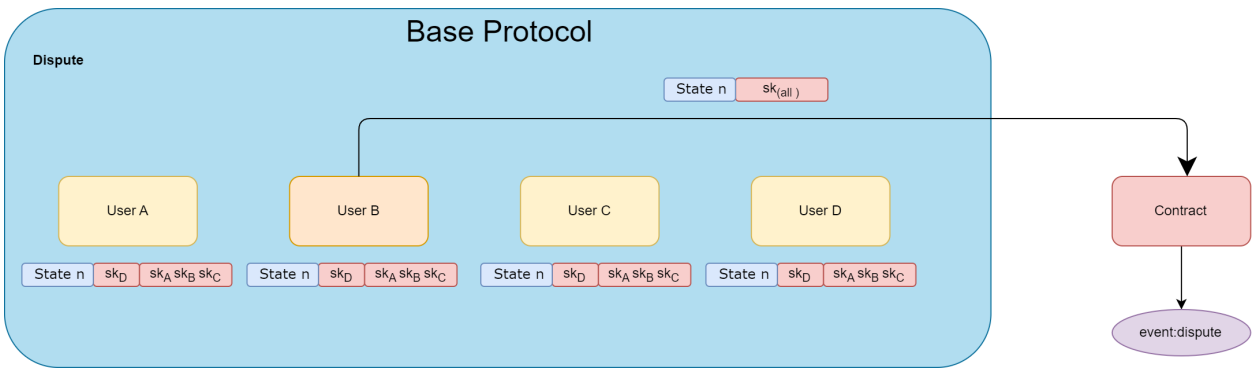

Figure 4.6: Dispute process against an inactive participant.

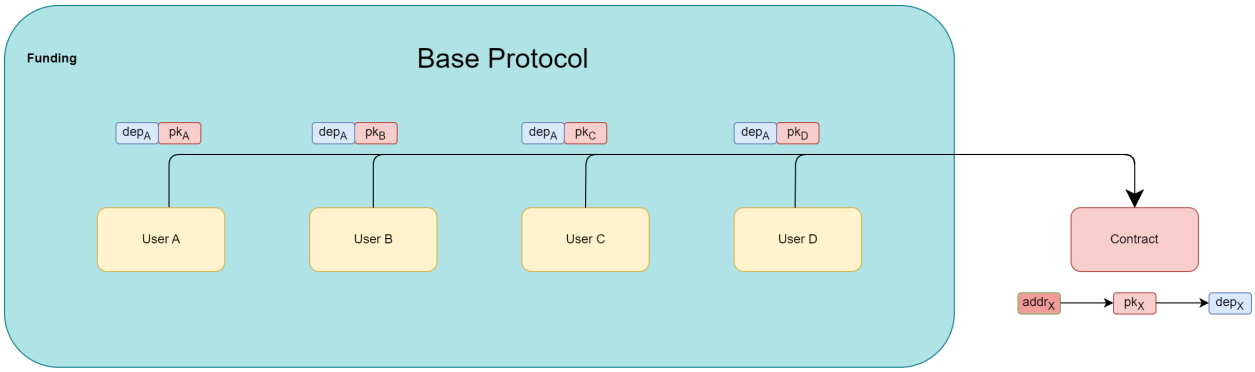

Figure 4.7: Depiction of the base channel funding process.

Step 1: The member most likely to start this process is the one that made the state update proposal. Said member, therefore, provides the contract with their header-signed state proposal and the last valid state. The contract allows a window for either of those submissions to be challenged as stale.

Step 2: After deeming the state proposal is valid, the contract starts a timeout allowing a window for the inactive member to respond with a signed version. If this happens, then the dispute is resolved. If it does not, the channel removes the absent party from the process by no longer requiring their signature to deem a state as valid. The inactive party also suffers a penalty. This process is described extensively in Section 4.2.3.6.

Part of resolving an inactivity dispute is judging whether or not the non-signing member is justified in their inactivity, which comes down to whether the proposed update is a valid transition from the previous state.

The validity of the modification of the accumulator elements is easy to verify since the process can be replicated with the information provided on the state proposal. However, since the base channel updates consist of openings and closings of groups/apps channels, it must also be ensured that the distribution of funds upon those processes correctly matches the outcome of said group. This is achieved through the mechanisms described in Section 4.4.3.

4.2.3 Functionality

4.2.3.1 Funding Phase

Before the channel can begin its operation, every participant must provide their blockchain address and deposit funds that will be enough to function both as a stake and as funds to use in the process of running applications. Those values are stored in the Base Channel contract, mapped to each participant's address. The process is graphically represented in Figure 4.7. The record of a participant's funds deposit is a prerequisite for the corresponding state update that adds the user to the base channel balance sheet.

4.2.3.2 Opening a new protocol

A core feature of the Origami design is the recursive building of protocols on top of each other. The process of opening a new protocol on top of the current one is broken down in the following steps:

Step 1: Any channel participant that wants to create a new protocol can initiate this process. User A for this example, creates a new state of the Update: Open variety. To assemble this state, User A must fill in all the corresponding fields as those are defined in 4.2.1. That includes:

1. The creation of the new protocol's UCS through the following process:
    - Setting the initial nonce to zero.
    - Composing a Balance sheet that reflects the asset/fund commitments made by the members to the new protocol.
    - Creating the new protocol's Channel ID by hashing the id of the base channel concatenated with the current state nonce for the base channel.
    - Filling in the contract address that corresponds to the smart contract the new protocol is based on.
2. User A must include in the state proposal the UCS Hash, and also add this hash to the Openings Accumulator of the base protocol.
3. The balance sheet of the current protocol must be updated to reflect that the funds committed to the new protocol are no longer spendable in this one.

Step 2: User A, having created the proposal, must get it signed by the potential protocol members first. That means that the users who are included in the Balance Sheet for the new protocol must sign the state before it reaches the Header. At this point, the same users must also separately sign the new channel's UCS.

Step 3: The proposed state update can now be sent to the round's Header, who, along with the usual checks as those are described in Section 4.2.4, must ensure the state has already been signed by all the potential members of the new protocol.

Step 4: The state is then propagated as described in Section 4.2.2.

The process of opening a protocol is identical whether the protocol to be opened is a group or app protocol.

As the Opening of a group protocol does not happen on chain, it does not involve funding in the traditional sense.

Members that wish to establish a group amongst themselves on top of the base protocol, propose a state update as is defined in Section 4.3.1.

4.2.3.3 Closing a Protocol

App protocols have a finite purpose that once served, calls for them to be closed, and at the same time, they are the only protocols that closing applies to. The following steps describe the closing process that happens through a state update of the Update: Close variety in the base protocol:

Step 1: Any member of the protocol to be closed can commence this process. User A for this example is the one to create the closing state update which is comprised of:

1. The Last Committed UCS that is currently representing the protocol in the Accumulator of Openings, whether that is UCS-0 or a more recent one through an Edit.
2. The hash of the aforementioned Last Committed UCS must also be included in the state.
3. The most Recent UCS of the protocol, which is the one it is going to close on.
4. The updated Accumulator of Openings from which the Last Committed UCS Hash representing the closing protocol has been removed.

5. The balance sheet of the supporting protocol's UCS must be updated to reflect the redistribution of funds as they result from the most recent UCS of the closing protocol.

6. The finalisability set that must be separately signed before it is appended to the state.

Step 2: The update is sent to all members of the closing channel to be signed. The signed update is sent to the supporting channel's Header and the communication process proceeds as described in Section 4.2.2.

4.2.3.4 Editing a Protocol

If Protocol B has been opened on top of the base protocol, and Protocol B's participant set changes, then the underlying protocol must be informed. This happens through a state update of the Update: Edit variety.

Step 1: Any member of the edited protocol can start off this process by creating the Edit state update in the supporting layer, in this case, the base protocol. This state update includes:

1. The Last Committed UCS: UCS currently represents the protocol being edited in the accumulator of openings. If this is the first time the protocol is edited, then this is UCS_0, otherwise, it is the UCS that resulted from the previous edit.

2. The hash of that Last Committed UCS

3. The new UCS of the edited group that has the updated balance sheet and other protocol data, as well as its hash.

4. The hash of the new UCS that will replace the Last Committed UCS Hash in the Accumulator.

5. The Accumulator of Openings that has been updated by removing the Last Committed UCS and adding the New UCS in its place

Step 2: The state update proposal is sent to the Header of the round and the communication process resumes normally as described in 4.2.2.

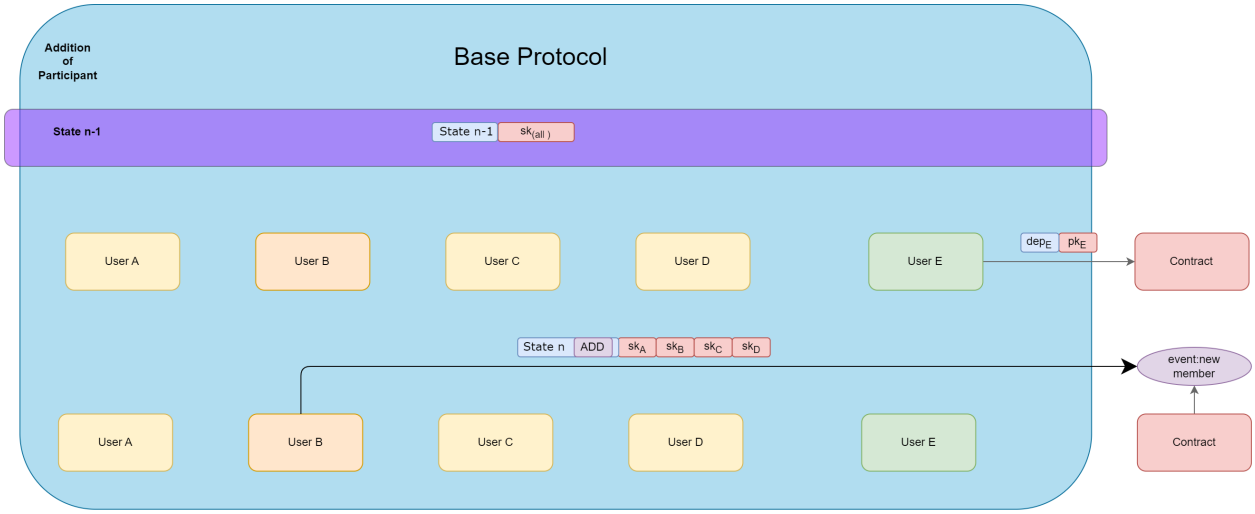

Figure 4.8: Base channel participant set expansion process.

4.2.3.5 Addition of participants

Origami allows the modification of the participant set of a channel after its creation. If a participant wants to join the base channel after it has been initially formed, then the following procedure, depicted in Figure 4.8 is triggered.

It is assumed the channel currently has 4 participants, users A through D, and user E wants to join.

Step 1: User E interacts with the channel's on-chain contract to deposit an amount of funds and this is recorded on the contract's storage. In this deposit, a small amount is included, dubbed as an ”entrance fee”, and functions as an incentive for the channels' participants to move along the process of including user E.

Step 2: The base channel contract emits an event to inform the participants that a joining request has been made by user E.

Step 3: For user E to become a member, the cooperation of those already in the channel is required. Any user A through D can initiate the process of proposing a state update of type ”Update: Add user”, which includes the address of User E and defines the funds they committed to the contract and will be bringing into the channel.

Step 4: Given the fact that this state update is valid, it is signed by all channel members and sets the new base channel state that includes E as a member. The user that initiated the update brings the new signed state update to the contract and the contract transfers to their account the entrance fee. The contract now has an updated member list and knows to take into account the signature of user E when judging whether a state is valid or not.

4.2.3.6 Removal of participants

- **Upon Request:** Any participant may decide they want to remove themselves from the base channel and retrieve their funds.

  Step 1: Said user has to prepare and propose an **Update:Remove User**, which removes themselves from the balance sheet.

  Step 2: The user being removed from the balance sheet cannot be disputed against for inactivity, therefore no user can remove another from the balance sheet through this process without consent.

Step 3: When that state becomes valid, they can bring it to the contract to retrieve their funds. Through this process, the contract is also made aware that the particular participant's signature is no longer needed.

- **Unresolved Dispute:** It is common practice in state channel designs after an unresolved dispute to be forced to terminate the channel since a valid state can no longer be procured (not all members are present to sign). Wanting to eliminate this scenario, in Origami, the necessary on-chain interaction is taken advantage of in more than one way when an inactivity dispute remains unresolved: The contract, after the expiration of the time out, applies the inactivity penalty to the unresponsive party. Additionally, the contract will now stop requiring the inactive member's signature for a valid update. Therefore the channel can progress.

4.2.4 State update validation rules

Open state update validation

Given a **Open** state update is proposed by a channel member, the rest of the members shall validate the update. During this update, a sub-set of the members of the channels are going to commit part of their funds to a new channel (group/app) based upon the base channel. The new state shall be validated according to the following rules:

- **UCS:** Regarding the fields included in the UCS the necessary conditions are:
  - **Nonce** value has to be increased by 1 with respect to the nonce of the previous state.
  - **Balance sheet** shall be identical to the balance sheet of the previous state with the required modifications (balances decrease) according to the **Balance Sheet** in the **new UCS**.
  - **Channel ID** must remain unchanged and correspond to the channel the update is taking place in.
  - **Contract Address** has to correspond to the management contract of the protocol.
- **New UCS:** Regarding the fields included in the new protocol's UCS the necessary conditions are:

  - **Nonce** value has to be set to 0 with respect to the nonce of the previous state.
  - **Balance sheet** shall include only the new channel's members and the funds they commit to the new protocol. Those funds must be equal to or less than their available funds in the base protocol.
  - **Channel ID** is a new value produced by the hash value of the concatenation of the base channel id and the nonce value.
  - **Contract Address** has to correspond to the management contract of the to-be formed protocol.

- **New UCS Hash** is the hash of all the contents of the new protocol's UCS.
- **Accumulator of Openings** shall be the product of adding the new UCS Hash to the accumulator of the previous state.
- The **Header** value shall be calculated according to the mechanism defined in Subsection 4.2.2.

Close state update validation

A **Close** state update can be proposed by any member of an app protocol that wants to terminate but must be validated by all members of the supporting channel before the protocol is permitted to close. Upon the closing of the channel, the assets committed to it are returned to the supporting protocol and distributed according to the outcome as expressed by the closing state. The validity of the closing state is determined according to the following rules:

- **UCS:** Regarding the fields included in the UCS the necessary conditions are:
  - **Nonce** value has to be increased by 1 with respect to the nonce of the previous state.
  - **Balance sheet** shall be identical to the balance sheet of the previous state with the required modifications (balances increase) according to the **Balance Sheet** in the **Recent UCS**.
  - **Channel ID** must remain unchanged and correspond to the channel the update is taking place in.
  - **Contract Address** has to be a valid contract address that corresponds to the management contract of the protocol.

- **Finalisability:** Regarding the fields included in the finalisability set, the necessary conditions are:
  - **Finalisable Flag** must be set to true.
  - **Nonce** value has to be the same as the nonce value included in the Recent UCS.
  - **Channel ID** must remain unchanged and correspond to the channel that is to be closed.
  - This set must have valid signatures by all members of the closing protocol.
- **Last Committed UCS:** must include the data that produces the hash that currently represents the protocol to be closed in the Accumulator of Openings.
- **Last Committed UCS Hash:** must be identical to the element to be removed from the Accumulator of Openings, and the product of hashing the data contained in the Last Committed UCS.
- **Recent UCS** is the UCS identical to the one included in the most recent state update of the protocol to be closed that expresses the outcome of said protocol.
- **Accumulator of Openings** shall be the product of removing the new Last Committed UCS Hash from the accumulator of the previous state.
- The **Header** value shall be calculated according to the mechanism defined in Subsection 4.2.2.

Edit state update validation

- **UCS:** Regarding the fields included in the UCS the necessary conditions are:
  - **Nonce** value has to be increased by 1 with respect to the nonce of the previous state.
  - **Balance sheet** shall be identical to the balance sheet of the previous state with the required modifications (balances increase) according to the **Balance Sheet** in the **Recent UCS**.
  - **Channel ID** must remain unchanged and correspond to the channel the update is taking place in.

  - **Contract Address** has to be a valid contract address that corresponds to the management contract of the protocol.

- **Last Committed UCS:** must include the data that produces the hash that currently represents the protocol to be closed in the Accumulator of Openings.
- **Last Committed UCS Hash:** must be identical to the element to be removed from the Accumulator of Openings, and the product of hashing the data contained in the Last Committed UCS.
- **New UCS** is the UCS that includes the changes in the member list and fund distribution of the edited channel and will replace the current UCS representing it in the Accumulator of Openings.
- **New UCS Hash** is the hash of the data included in the new UCS and the element to be added in the Accumulator of Openings.
- **Accumulator of Openings** shall be the product of removing the new Last Committed UCS Hash from the accumulator of the previous state and replacing it with the New UCS Hash.
- The **Header** value shall be calculated according to the mechanism defined in Subsection 4.2.2.

Add user state update validation

Given an Add user state update is proposed by a channel member, the rest of the members shall validate the update. The new state shall be validated according to the following rules:

- **UCS:** Regarding the fields included in the UCS the necessary conditions are:
  - **Nonce** value has to be increased by 1 with respect to the nonce of the previous state.
  - **Balance sheet** shall be identical to the balance sheet of the previous state with two exceptions. Firstly, the addition of an entry with the new user and the corresponding balance, according to the deposit in the base channel contract. Additionally, the payment of the entrance fee from the new member to the proposer of the state must be reflected in the balance sheet.

  - **Channel ID** must remain unchanged and correspond to the channel the update is taking place in.
  - **Contract Address** has to correspond to the management contract of the protocol. (Origami contract for base and group protocols, respective application contract for app protocols)

- The values in the sub-fields **User address** and **Amount** of the **Update** field must correspond to the new entry and fund addition reflected in the updated balance sheet.
- **Accumulator of Openings** shall be identical to the one of the previous state.
- **Header** value shall be calculated according to the mechanism defined in Subsection 4.2.2.

Remove user state update validation

Given a **Remove user** state update is proposed by a channel member, the rest of the members shall validate the update. The new state shall be validated according to the following rules:

- **UCS:** Regarding the fields included in the UCS the necessary conditions are:
  - **Nonce** value has to be increased by 1 with respect to the nonce of the previous state
  - **Balance sheet** shall be identical to the balance sheet of the previous state with the exception of the removal of the entry of the user to be removed.
  - **Channel ID** must remain unchanged and correspond to the channel the update is taking place in.
  - **Contract Address** has to correspond to the management contract of the protocol. (Origami contract for base and group protocols, respective application contract for app protocols)
- The values in the sub-fields **User address** and **Amount** of the **Update** field shall be set to the address of the user to be removed and the balance they had in the previous state.
- **Accumulator of Openings** shall be identical to the one of the previous state.
- **Header** value shall be calculated according to the mechanism defined in Subsection 4.2.2.

4.3 Group Protocol

Origami is meant to remain active indefinitely. However, the addition of participants to the base protocol in a boundless manner would undoubtedly result to high latency, eventually rendering the channel unusable.

The group protocol has been designed as a scaling mechanism to prevent this scenario. Subsets of base protocol members create these instances that can stack infinitely on top of each-other, limiting the number of signatures required for a state update. The goal is that as the number of Origami participants escalates, the system scales upwards and the lower, densely populated protocols are rarely active.

The group protocol shares most of its functionality with the base protocol, since it is based on the same smart contract and serves the same purpose. group protocols can support group or app protocols opening on top of them.

Specifically, the following processes are identical between the two protocols:

- *Representation of the State:* The varieties and contents of the channel states are shared between the base and group protocols.
- *Communication model:*Since group protocols are governed by the same smart contract as the base protocol, they also follow the same, non-turn based communication scheme that is described in Section 4.2.2.
- *Disputes* Disputes are also handled by the same contract and therefore in the same manner.
- *Opening, Closing and Editing another Protocol functionalities: The manner in which a channel is opened, closed or edited does not differ between it happening on top of a base protocol or a group protocol.*

The present section will be elaborating on the functionalities that differ between the base and group protocols.

4.3.1 Group Protocol Opening

For the opening of a group protocol, actions in two layers are necessary. Part of the process happens in the supporting channel (the base or group protocol underneath) and

part of it in the new channel that is being opened. Actions in the supporting channel are those described in 4.3: **Opening a new protocol**. In this section, the actions taken in the new channel during its creation are analysed.

After establishing the new group protocol in the supporting channel, the members must adhere to the following process:

Step 1: A starting state must be propagated. This state includes:

- The new channel's UCS that was signed by the members during the opening of the group as described in Section 4.2.3.
- An empty Accumulator of Openings.
- The header for the next round is defined as the first entry on the balance sheet.

The creation of the starting state is done by the protocol member that appears first on the balance sheet, for coordination reasons.

4.3.2 Addition of Participant

Group protocols also allow the modification of their participant set. However, the process differs fundamentally from the base protocol one, since no on-chain action takes place to alert participants of the to-be member's request.

The addition of a participant to an existing group protocol is modeled as such:

Step 1: It is assumed that the group protocol has 4 participants, Users A through D, and User E aims to join. User E communicates their entry request to any member of the group protocol.

Step 2: The group protocol member that adds User E, let us say User A, must create two state updates:

- *Group Protocol State Update:* This update is propagated inside the group protocol. It adds User E and their funds to the protocol's balance sheet, and also reflects the payment of the entrance fee from User E to User A. This state update is what is defined as an **Add User** type of update in Section 4.2.1.
- *Supporting Layer State Update:* Any group protocol has an underlying layer that must also be updated after any change to the participant set. The new state is

meant to replace the original opening state of the group protocol, showing the new participant set and the new balance sheet. This is necessary to keep all participants as well as the smart contract aware of the alterations. This state update is what is defined as an **Edit** type of update in Section 4.2.1.

4.3.3 Removal of Participant

A participant's removal from a group protocol is called upon in the same cases as the base protocol, but is handled differently because they do not equate the participant's removal from the entire ecosystem.

Upon Request: A participant that wants to voluntarily leave a group protocol they are in, has to create two new states, a *Group Protocol State Update* of the **Remove** variety, and an Underlying Layer state update of the **Edit** variety. Those serve the same purposes and have the same contents as described in Section 4.3.2 but now indicate a participant's removal and the resulting transfer of funds from the group protocol to its underlying layer.

Unresolved Inactivity Dispute: To prevent the stagnation of the protocol processes in the event of an unresolved Inactivity Dispute, the party at fault is removed. The necessary state updates, in both the group protocol and underlying layer, will lack the party at fault's signature, and therefore the unresolved state of the dispute must be visible in the contract to make those updates valid.

4.4 App Protocol

The app protocol maintains the purpose of a traditional state channel, which is to run a blockchain application off-chain while only involving directly interested parties. App protocols are opened on top of a group or base protocol any time a subset of participants wishes to run a state channel application amongst themselves.

Due to being higher layers in the origami protocol, app protocols share a number of functionalities with group protocols. Those are:

Opening: Opening an app protocol happens through the same process as a group protocol, by requiring a state update of the Open variety to take place in the supporting protocol. The functionality within the new channel is defined by the smart contract of the application. Participant set Modification: The option to modify

Figure 4.9: Depiction of the base channel and App Protocols.

the participant set is left upon the designers of the application smart contract an app protocol is running, because it may or may not fit into the nature of the application. If desired, however, the application smart contract can incorporate the modification of its participant set as described in Sections 4.3.2 and 4.3.3.

The rest of this section will elaborate on the functionalities unique to app protocols.

An app protocol is opened any time a subset (or the full set) of the base channel's participants want to run a state channel application amongst themselves. The benefits of such a scheme are multi-dimensional. It enhances privacy, usability and lowers communication overhead in the channel, all by removing the need for participants to sign an update that is irrelevant to them. The ability to run any application amongst any combination of origami participants can eliminate the need to ever form another state channel, therefore making this scheme very convenient and also reducing channel creation costs and transaction load on the blockchain.

How the app protocols relate to the base channel can be seen in Figure 4.9. The participant sets of each app protocol (m,k,l,h as shown in Figure **??**) can be completely unrelated, identical, or subsets of each other.

4.4.1 App Protocol Communication

In-group communication follows whatever pattern the application developers have designed. The non-turn based communication protocol described in Section 4.2.2 can be incorporated into an application, but it is not necessary in case turn-based communication is preferred.

4.4.2 App Protocol Inactivity Dispute

An inactivity dispute for an app protocol is triggered through the application's contract. Because the contract is not contacted unless there is misbehaviour, it is not aware of the members of the group or any other specific information. Therefore, the data included in the app protocol's opening state is provided to the application contract when a dispute is initiated. How the dispute is resolved also depends on the smart contract specific to the

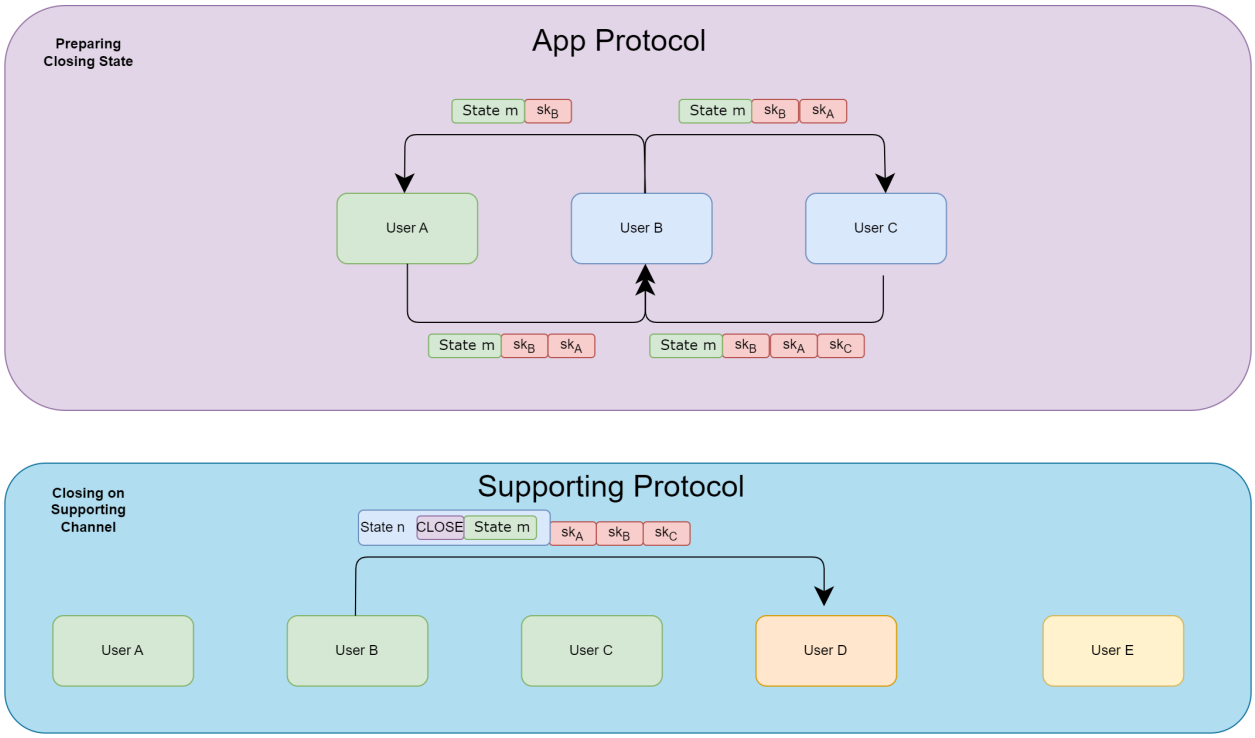

Figure 4.10: App Protocol closing process.

application.

4.4.3 App Protocol Closing

When the application run in the app protocol has fulfilled its purpose, the participants can close the channel and release the funds locked in it back in the underlying channel. The process to close an app protocol is described in Figure 4.10 and detailed in the following steps:

- **Step 1** Any member of the app protocol can initiate the process of closing it as long as the most recent state of the protocol is a finalisable state, which is a state appropriate to close upon. Finalisability is determined by the individual app contract and signaled through the independently signed Finalisable set as this is defined in Section 4.2.1.
- **Step 2** Closing an app protocol is synonymous with informing the channel it was opened on of its dissolution. The channel members must prepare and sign a closing

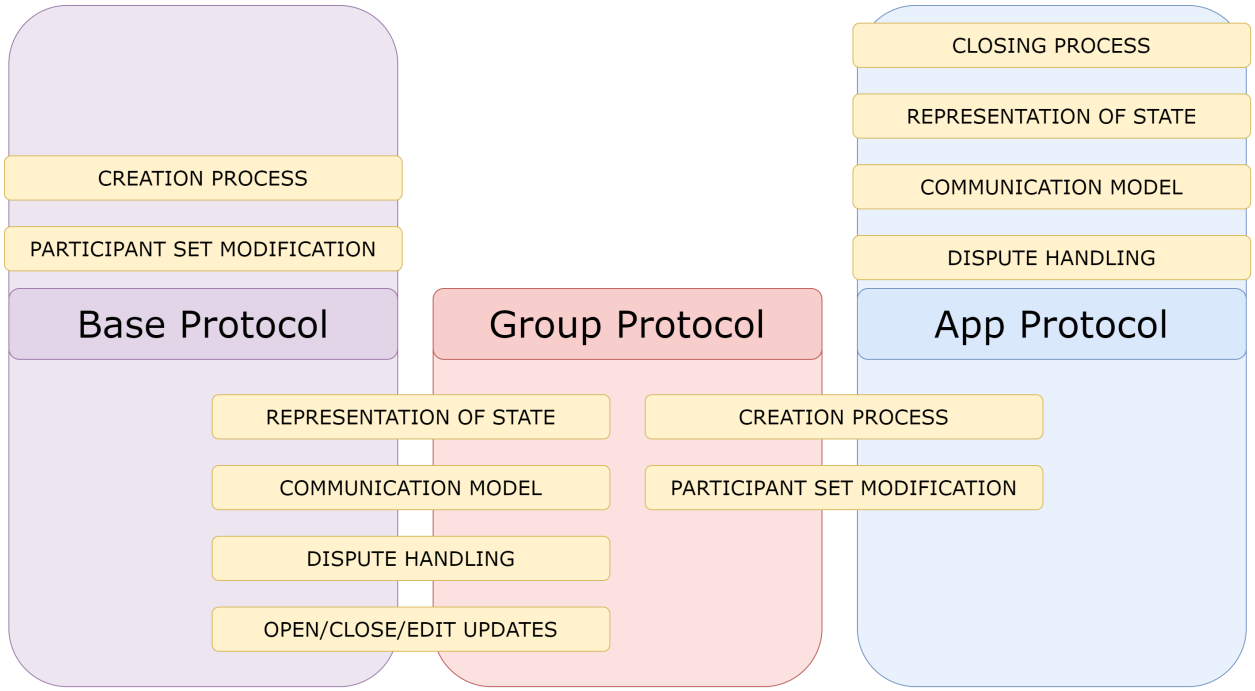

Figure 4.11: Functionalities per protocol.

state for the supporting base or group channel that is of the **Close** variety that will, among other values, include the UCS of the state to close upon and the Finalisable set in the Update field that will allow the Origami management contract to determine the validity of the state update proposal.

4.5 Protocol Relationships

In Figure 4.11 an overview of all Origami functionalities and how those are divided per protocol can be seen.

The base protocol has a unique creation process and way of handling modifications in the participant set. This stems from the fact that, as the entryway to the Origami ecosystem, interacting with the base channel sometimes requires interacting with the blockchain network.

App protocols are the only protocols where closing is applicable as app channels most often have a finite purpose. The communication model an app protocol uses can either have a more traditional turn-based approach or employ the communication scheme introduced by Origami in Section 4.2.2, depending on the needs of the particular appli-

cation. Dispute handling is also case-specific and determined by the application smart contract. The same stands for representation of the state with the exception of being required to include the Finalisability set as that is described in Section 4.2.1.

Group protocols are distinct in the sense that they do not have any unique functionalities. The communication model employed is the one analysed in Section 4.2.2, which also holds true for the base protocol. Dispute handling as well as the representation of the state are also shared between base and group protocols. Since both group and app protocols are higher layers that only require on-chain interaction for disputes, they have the same creation process and implement participant set modification in identical ways.

4.6 Origami Smart Contracts

Every state channel is managed by a smart contract that governs its funds and settles its disputes. The base and group protocols share a single management contract, called the Origami Management Contract. Each app protocol though corresponds to a state channel application contract that governs the specific application which are dubbed Origami app contracts.

Apps that aim to be compatible with the Origami design must make minimal modifications to their smart contract in order to be able to take advantage of its functionalities. Origami app contracts must therefore meet the following conditions:

- Conventional state channel contracts are not usually usable until the funding phase has been completed. In contrast, Origami app contracts are not meant to be interacted with unless a dispute occurs, and even in that case, they do not accept disputes only by a single channel, but by any app protocol that is running the particular application. Therefore, instead of a funding phase that sets the list of participants, Origami apps must accept the opening state information that is unique for each app protocol.
- In order to enable the handling of disputes at the lower levels, the Universal Channel State has been defined as an independently signed data set that enables the flow of channel data between app, group and base protocols. Origami app contracts must take care to include this separate set along with their state updates.
- Finally, every Origami App state must include the independently signed flag set that

includes the flag, the channel ID and the channel nonce and indicates whether a state is finalisable or not, as described in Section 4.2.1. The app smart contract must also check whether the value of the flag is appropriate for the specific state (eg. if the state whose finalisable flag value is true is indeed finalisable).

5. EVALUATION

In Chapter 3, a comparative evaluation of the majority of state channel designs was performed. This effort helped identify the common shortcomings of works in the state channel research field, as well as the more desirable characteristics. The main motivation behind Origami was creating a design with minimum negative characteristics and maximum beneficial features, while also maintaining a low number of necessary on-chain transactions. In this Chapter, Origami is evaluated alongside other state channel designs across the same measures set in [47].

5.1 Attribute Comparison

Table 5.1 is the modification of Table 3.1 to include Origami. Hereby follows an analysis of how the proposed design compares to the existing ones.

5.1.1 Requirements

A higher number of cells checked in the Requirements section of Table 5.1 means a more limited pool of applications is compatible with the design, while some of the demands decrease usability. Therefore, the fewer cells checked on the Requirement section, the broader the scope of applications a design can support.

- **Turn-based Applications:** Limiting the nature of supported applications to ones that function with turn-based communication is a matter of accountability distribution. While a few of the analysed schemes do not explicitly place the turn-based requirement, Origami is the only one that extensively elaborates on how non-turn based communication is realised and how concurrent update proposals would be handled, as analysed in Section 4.2.2.
- **Timers:** Disputes in state channels are structured as a challenge-response mechanism. A challenge is issued and there must be a response before the time limit goes by, both for potential contentions and for a successful resolution. Because there is no provable way to both securely and accurately measure time on the blockchain, this is not a preferred practice. However, as no workaround to this method has been developed, this is a requirement shared by all state channel designs.

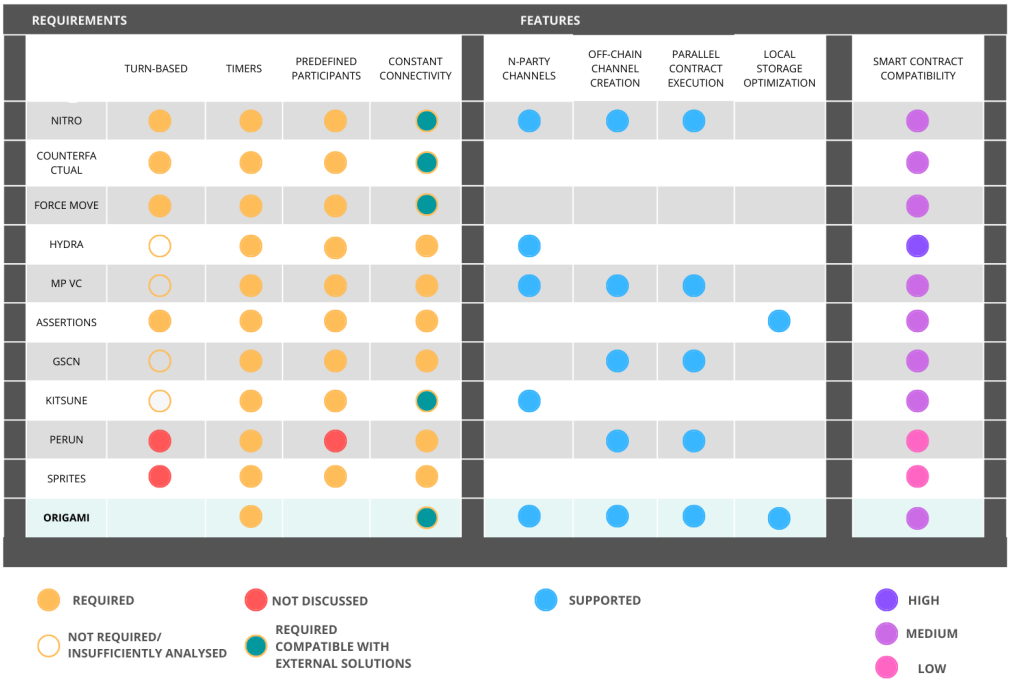

Figure 5.1: Comparative evaluation of Origami and other state channel designs.

- **Predefined Participant Sets:** To avoid complicating the funding process, and also limit interactions with the smart contract, state channel designs enforce static participant sets. This further enforces the need to create a separate state channel if there is a desire for an addition to the initial set, and the need to close the channel when a participant wishes to leave. In contrast, Origami allows total flexibility in the participants of the base channel or any of the higher levels. Mechanisms to join and leave on demand are outlined in Sections 4.2 and 4.3. The only on-chain transaction in the whole process is funding the base channel, a one time requirement to then freely enter, form and leave any higher layer. Unresolved disputes also result in member ”expulsions” to avoid forcing the group or channel to close when singular members become unavailable or misbehave.

- **Constant Connectivity:** Inactivity disputes are integral to the state channel functionality. They apply penalties to any member that obstructs state progression with their absence. The repercussions are applied regardless of whether the absence is the result of malicious behaviour or inability to respond. Therefore, by extent, channel members cannot risk being unavailable under any circumstances, or they stand to suffer consequences or lose their chance to contest a dispute. Designs like Pisa

[42] and Brick [6] are solutions that allow some form of delegation of signatures from a member that intends to be absent to an external entity. However, these solutions are not compatible with all state channel designs.

5.1.2 Features

The second section of Table 5.1 titled ”Features” lists a number of desirable functionalities a state channel design can offer. Whether it is an increase in efficiency, usability, or a reduction in cost, multiple checks in this section elevate the design that can boast them.

- **Multi-party Channels:** The earlier state channel designs were built for two party interaction. Origami, like a number of more recent state channel designs, places no restriction on the number of members in a channel or group, and even allows the participant set composition to be altered after channel establishment.
- **Off-chain Channel Creation:** In order to further limit the transaction load on the blockchain, attempts were made to enable the creation of state channels without any interaction with the chain. The result of these efforts was Virtual Channels, otherwise called State Channel Networks have as a primary feature the ability for parties to form state channels off-chain, as long as they have each already formed one with a common intermediary. This can also be extended, providing the ability to form channels off-chain as long as there is any path of preformed state channels that can connect participants. The weakness of such designs lays in the fact that an intermediary is necessary and often burdened by that process. Origami creates an ecosystem where participants can open and close groups that function as channels without on-chain interaction. However, since there is no requirement for an intermediary, users are only limited by their own deposit, and nobody has to lock their funds to enable the communication of third parties.
- **Smart Contract Compatibility:** This column refers to the level of modifications that need to be made in an application in order to make it compatible with a state channel design. Since Origami requires places minimal conditions on smart contracts it achieves medium compatibility.
- **Parallel Contract Processing:** A channel that can process more than a single contract at a time can also support more than a single application running at a time. In

	OPENING	UPDATE	DISPUTE INITIATION	DISPUTE RESOLUTION	CLOSING
NITRO	0	0	-	-	0
COUNTERFACTUAL	P+1	0	1/2*	-	-
FORCE MOVE	P+1	0	1	1	1
HYDRA	1	0	-	-	1
MP VC	0	0	1	P-1	0
ASSERTIONS	P+1	0	2	1	1*
GSCN	0	0	0/1	0/P-1	0
KITSUNE	-	0	1	P-1	1/P*
PERUN	P+2*	0	1	-	2
SPRITES	P	0	1	1/2	1/2
ORIGAMI	0	0	1	1	0

(-) ABSENCE OF DATA (*) LIMITED DATA p=NUMBER OF CHANNEL PARTICIPANTS BEST CASE/WORST CASE

Table 5.1: Comparison of state channel designs regarding on-chain transaction load

Origami, there is no limit on how many App Groups can be run at one time, since they do not require any kind of contract instantiation.

- **Local Storage Optimisation:** By making use of cryptographic accumulators, Origami severely cuts down on the local storage demands its states impose on its users. Additionally, the design eliminates the need to be maintaining any more than the most recent state for each channel a user participates in and the opening state of that channel.

5.2 Efficiency

Since the main goal of state channel designs is to reduce the transaction load and keep the, often hefty, fees associated with on-chain transactions to a minimum, it would be optimal to evaluate every proposal according to the degree of success regarding the aforementioned goals. However, an assessment for fees is impossible, since fees have an absolute dependency on factors beyond the state channel implementation (blockchain application, blockchain system, state of contract etc.) and cannot be consistently calculated for all approaches.

A noteworthy effort to eliminate the correlation between fees and the several factors it relies on is made by the authors of [13]. This is achieved through introducing a static form for the dispute submission irrespective of the nature of the application and eliminating the need for the valid state to be computed by the contract. However, this still only applies to the optimistic case of a dispute where the initial request is obviously valid and not challenged further.

Table 5.1 represents the closest to a fair efficiency evaluation between the state-channel proposals by calculating the number of on-chain transactions necessary in each major phase of the state channel as described in Section **??**. Those phases are Opening, Update, Dispute, and Closing and in most of the cases the number of transactions is expressed in relevance to the number of channel participants denoted on the table as p.

Opening: A typical state channel opening process includes the deployment of the channel contract and its funding. There is some differentiation in the funding process between designs. Creating a virtual channel requires no on-chain interaction, but it does require that ledger channels already exist, at least as many as the virtual channel's participants. When marking the opening transactions required as 0, this refers to the creation of the virtual channel and does not take into account the transactions needed for the ledger channels that are preexisting. By the same logic, Origami group and app protocols require zero on-chain transactions to be formed once in the ecosystem. In stark contrast to virtual channels though, the single transaction required to join the ecosystem guarantees any subsequent groups will be formed off-chain, regardless of whether there is a direct connection with other participants or not.

Update: It is generally expected, for a state channel to be beneficial, that during optimistic state progression, there will be no contact with the blockchain.

Dispute: Even though this is the phase where designs diverge the most from one another, it is most often in the form of the dispute transaction submitted rather than the number of transactions necessary. To better analyse the Dispute phase we have split it into two sub-phases on Table **??**, Dispute Initiation and Dispute Resolution.

Closing: Similarly to the opening phase, closing also has a universally adopted form. It signals the distribution of assets after either a dispute (resolved or unresolved) or a cooperative update to a finalisable state. Virtual Channels and Origami allow this process to happen off-chain, and Origami does not enforce the closing of a protocol after an unresolved dispute.

It must be pointed out that the information extracted for setting up Table 5.1 was very disproportionate from one work to another. Some designs gave enough detail to depict both a best and worst-case scenario, while others presented only a general idea. The values on the table marked with an asterisk (*) are our best interpretation from the limited descriptions and might not be entirely accurate. In a handful of situations, the data was insufficient or absent and those cases have been marked with a dash (-).

6. CONCLUSIONS AND FUTURE WORK

The relatively short history of state channels makes it feasible to comment on the evolution of the domain from its initiation up to today. At the present point, it can be said that the state channel approach is quite effective when applied to applications that adhere to specific rules. However, as it has stood so far, the application scope has been rather narrow. It is easily discernible from Table 3.1 that there is a high number of requirements maintained amongst proposals, for a state channel design to be effective and beneficial. Such specifications vastly limit the applicability of state channel technology resulting in a downplay of its effectiveness on the overall improvement of blockchain scalability. The commentary and classifications that can be found in this work make it significantly more feasible to pinpoint the weaknesses in the current state of the art and further points that need to be addressed by research.

The scalability problem in public blockchain systems can be said to derive from two factors (a) actual low transactions processing capacity and (b) misuse of given capacity. While other layer 2 proposals that impose less restrictions have been deemed preferable to state channels, they all aim at improving the transactions processing capacity. State channels aim at minimizing the on-chain interaction and therefore the utilization of the limited on-chain resources. While a blockchain system is the source of ground truth, it is inefficient to make use of it continuously. State channels are the only protocol that aims at enhancing the way apps utilise the given capacity of a blockchain system, they will remain relevant even if another layer 2 solution succeeds in significantly multiplying processing capacity. Further benefit could even come from the combined use of state channels and other layer 2 solutions.

With that in mind, it was deemed beneficial to focus on the development of a state channel scheme that could circumvent the existing obstacles and enable the reaping of the full benefits of the technology. Origami is meant to fill what has been missing from this research field, a design with a better balance between the absence of requirements and the presence of features.

It can be confidently concluded that the goal has been achieved. Through Origami, applications can benefit from state channels regardless of whether they employ turn-based communication schemes or not. Unnecessary termination of channels and therefore the necessity to interact with the blockchain to set new ones up have been eliminated. Users

can join a boundlessly scaling ecosystem that reduces on-chain transactions. Origami, in the optimistic case, requires a single on-chain interaction per person indefinitely, as opposed to existing designs that require setting up a new state channel for every use or alteration of the participant set.

The Origami design even prioritises keeping the storage requirements to a minimum by preventing the state from endlessly expanding and burdening the user nodes. Future work intentions include the development of a built-in mechanism that bypasses the constant connectivity requirement without relying upon an external service, as well as a high-level extensive security analysis for the Origami design.

REFERENCES

[1] Payment channels. https://en.bitcoin.it/wiki/Payment_channels. [Online; accessed 29-March-2021].

[2] LAYER 2 ROLLUPS. https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/, 2021. [Online; accessed 16-May-2021].

[3] LAYER 2 SCALING. https://ethereum.org/en/developers/docs/layer-2-scaling/, 2021. [Online; accessed 29-April-2021].

[4] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *arXiv preprint arXiv:1708.03778*, 2017.

[5] Ian Allison. Ethereum's vitalik buterin explains how state channels address privacy and scalability, 2016.

[6] Georgia Avarikioti, Eleftherios Kokoris Kogias, and Roger Wattenhofer. Brick: Asynchronous payment channels. *arXiv preprint arXiv:1905.11360*, 2019.

[7] Adam Back, Matt Corallo, Luke Dashjr, Mark Friedenbach, Gregory Maxwell, Andrew Miller, Andrew Poelstra, Jorge Timón, and Pieter Wuille. Enabling blockchain innovations with pegged sidechains. *URL: http://www. opensciencereview. com/papers/123/enablingblockchain-innovations-with-pegged-sidechains*, 72, 2014.

[8] Foteini Baldimtsi, Jan Camenisch, Maria Dubovitskaya, Anna Lysyanskaya, Leonid Reyzin, Kai Samelin, and Sophia Yakoubov. Accumulators with applications to anonymity-preserving revocation. Cryptology ePrint Archive, Report 2017/043, 2017. https://ia.cr/2017/043.

[9] Seyed Mojtaba Hosseini Bamakan, Amirhossein Motavali, and Alireza Babaei Bondarti. A survey of blockchain consensus algorithms performance evaluation criteria. *Expert Systems with Applications*, page 113385, 2020.

[10] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In Alexandra Boldyreva and

Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 561–586, Cham, 2019. Springer International Publishing.

[11] Sarah Bouraga. A taxonomy of blockchain consensus protocols: A survey and classification framework. *Expert Systems with Applications*, 168:114384, 2021.

[12] Brainbot. Raiden Network. `https://raiden.network/`, 2021. [Online; accessed 28-March-2021].

[13] Chris Buckland and Patrick McCorry. Two-party state channels with assertions. In *International Conference on Financial Cryptography and Data Security*, pages 3–11. Springer, 2019.

[14] Vitalik Buterin et al. A next-generation smart contract and decentralized application platform. *white paper*, 3(37), 2014.

[15] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*, pages 136–145. IEEE, 2001.

[16] Manuel MT Chakravarty, James Chapman, Kenneth MacKenzie, Orestis Melkonian, Michael Peyton Jones, and Philip Wadler. The extended utxo model. In *International Conference on Financial Cryptography and Data Security*, pages 525–539. Springer, 2020.

[17] Manuel MT Chakravarty, Sandro Coretti, Matthias Fitzi, Peter Gazi, Philipp Kant, Aggelos Kiayias, and Alexander Russell. Hydra: Fast isomorphic state channels. *IACR Cryptol. ePrint Arch.*, 2020:299, 2020.

[18] Alisa Chernyaeva, Ilya Shirobokov, and Alexander Davydov. Game channels: State channels for the gambling industry with built-in prng. *IACR Cryptol. ePrint Arch.*, 2019:362, 2019.

[19] Sherman SM Chow, Ziliang Lai, Chris Liu, Eric Lo, and Yongjun Zhao. Sharding blockchain. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1665–1665. IEEE, 2018.

[20] Tom Close. Nitro protocol. *IACR Cryptol. ePrint Arch.*, 2019:219, 2019.

[21] Tom Close and Andrew Stewart. Forcemove: An n-party state channel protocol, 2018.

[22] Jeff Coleman. State Channels. `https://www.jeffcoleman.ca/state-channels/`, 2015. [Online; accessed 29-March-2021].

[23] Jeff Coleman, Liam Horne, and Li Xuanji. Counterfactual: Generalized state channels. *Acessed: Nov*, 4:2019, 2018.

[24] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*, pages 123–140, 2019.

[25] Mo Dong, Qingkai Liang, Xiaozhou Li, and Junda Liu. Celer network: Bring internet scale to every blockchain. *arXiv preprint arXiv:1810.00037*, 2018.

[26] Thaddeus Dryja and Scaling Bitcoin Milano. Unlinkable outsourced channel monitoring. *Scaling Bitcoin Milan*, 2016.

[27] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, Julia Hesse, and Kristina Hostáková. Multi-party virtual state channels. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 625–656. Springer, 2019.

[28] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment channels over cryptographic currencies. *IACR Cryptol. ePrint Arch.*, 2017:635, 2017.

[29] Stefan Dziembowski, Lisa Eckey, Sebastian Faust, and Daniel Malinowski. Perun: Virtual payment hubs over cryptocurrencies. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 106–123. IEEE, 2019.

[30] Stefan Dziembowski, Sebastian Faust, and Kristina Hostáková. General state channel networks. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 949–966, 2018.

[31] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59, 2016.

[32] Paul Grau. Lessons learned from making a Chess game for Ethereum. https://medium.com/@graycoding/lessons-learned-from-making-a-chess-game-for-ethereum-6917c01178b6, 2016. [Online; accessed 29-March-2021].

[33] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Off the chain transactions. *IACR Cryptol. ePrint Arch.*, 2019:360, 2019.

[34] Lewis Gudgeon, Pedro Moreno-Sanchez, Stefanie Roos, Patrick McCorry, and Arthur Gervais. Sok: Layer-two blockchain protocols. In *International Conference on Financial Cryptography and Data Security*, pages 201–226. Springer, 2020.

[35] Zackary Hess, Yanislav Malahov, and Jack Pettersson. Æternity blockchain. *Online]. Available: https://aeternity. com/aeternity-blockchainwhitepaper. pdf*, 2017.

[36] Maxim Jourenko, Kanta Kurazumi, Mario Larangeira, and Keisuke Tanaka. Sok: A taxonomy for layer-2 scalability related protocols for cryptocurrencies. *IACR Cryptol. ePrint Arch.*, 2019:352, 2019.

[37] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[38] Amrit Kumar, Pascal Lafourcade, and Cédric Lauradoux. Performances of cryptographic accumulators. In *39th Annual IEEE Conference on Local Computer Networks*, pages 366–369, 2014.

[39] Seungjin Lee and Hyoungshick Kim. On the robustness of lightning network in bitcoin. *Pervasive and Mobile Computing*, 61:101108, 2020.

[40] Jeremy Longley. A Reference Implementation of State Channel Contracts. https://funfair.io/a-reference-implementation-of-state-channel-contracts/, 2019. [Online; accessed 29-March-2021].

[41] Jeremy Longley and Oliver Hopton. Funfair technology roadmap and discussion. Technical report, Technical Report. Funfair. 6 pages. chrome-extension …, 2017.

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

[43] Patrick McCorry, Chris Buckland, Surya Bakshi, Karl Wüst, and Andrew Miller. You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies. In *International Conference on Financial Cryptography and Data Security*, pages 35–49. Springer, 2019.

[44] Andrew Miller, Iddo Bentov, Ranjit Kumaresan, Christopher Cordi, and Patrick McCorry. Sprites and state channels: Payment networks that go faster than lightning. *arXiv preprint arXiv:1702.05812*, 2017.

[45] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[46] Lydia Negka, Angeliki Katsika, Georgios Spathoulas, and Vassilis Plagianakos. Blockchain state channels with compact states through the use of rsa accumulators. *Blockchain: Research and Applications*, page 100114, 2022.

[47] Lydia D Negka and Georgios P Spathoulas. Blockchain state channels: A state of the art. *IEEE Access*, 2021.

[48] Ilker Ozcelik, Sai Medury, Justin Broaddus, and Anthony Skjellum. An overview of cryptographic accumulators. *arXiv preprint arXiv:2103.04330*, 2021.

[49] Joseph Poon and Vitalik Buterin. Plasma: Scalable autonomous smart contracts. *White paper*, pages 1–47, 2017.

[50] Joseph Poon and Thaddeus Dryja. The bitcoin lightning network: Scalable off-chain instant payments, 2016.

[51] Peter R. Rizun. The Excessive-Block Gate: How a Bitcoin Unlimited Node Deals With “Large” Blocks. `https://medium.com/@peter_r/the-excessive-block-gate-how-a-bitcoin-unlimited-node-deals-with-large-blocks-22a.riqy7lm36`, 2016. [Online; accessed 16-May-2021].

[52] Amritraj Singh, Kelly Click, Reza M Parizi, Qi Zhang, Ali Dehghantanha, and Kim-Kwang Raymond Choo. Sidechain technologies in blockchain networks: An exam-

ination and state-of-the-art review. *Journal of Network and Computer Applications*, 149:102471, 2020.

[53] Amritraj Singh, Reza M Parizi, Meng Han, Ali Dehghantanha, Hadis Karimipour, and Kim-Kwang Raymond Choo. Public blockchains scalability: An examination of sharding and segregated witness. In *Blockchain Cybersecurity, Trust and Privacy*, pages 203–232. Springer, 2020.

[54] Yonatan Sompolinsky, Yoad Lewenberg, and Aviv Zohar. Spectre: A fast and scalable cryptocurrency protocol. *IACR Cryptol. ePrint Arch.*, 2016:1159, 2016.

[55] Josh Stark. Making sense of ethereum's layer 2 scaling solutions: State channels, plasma, and truebit. *URl: https://medium. com/l4-media/making-sense-of-ethereums-layer-2-scalingsolutions-state-channels-plasma-and-truebit-22cb40dcc2f4*, 2018.

[56] Horst Treiblmaier and Trevor Clohessy. *Blockchain and Distributed Ledger Technology Use Cases*. Springer.

[57] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 41–61, 2019.

[58] Yang Xiao, Ning Zhang, Wenjing Lou, and Y Thomas Hou. A survey of distributed consensus protocols for blockchain networks. *IEEE Communications Surveys & Tutorials*, 22(2):1432–1465, 2020.

[59] Michael Xu. Cryptopoly: Using ethereum state channels for decentralized game applications. 2020.

[60] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. Solutions to scalability of blockchain: A survey. *IEEE Access*, 8:16440–16455, 2020.