Blockchain: Research and Applications 4 (2023) 100114

Contents lists available at ScienceDirect

Blockchain: Research and Applications

journal homepage: www.journals.elsevier.com/blockchain-research-and-applications

Research Article

Blockchain state channels with compact states through the use of RSA accumulators

Lydia Negka [a], Angeliki Katsika [a], Georgios Spathoulas [a,b,*], Vassilis Plagianakos [a]

[a] Department of Computer Science and Biomedical Informatics, University of Thessaly, 2-4 Papasiopoulou st., Lamia, 35131, Greece
[b] Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), Mail Box 191, Gjøvik, NO-2815, Norway

ARTICLE INFO

Keywords:
Blockchain
State channels
Accumulators
Layer 2
NFTs

ABSTRACT

One of the major concerns regarding currently proposed public blockchain systems relates to the feasible transaction processing rate. It is common for such systems to limit this rate to maintain the required levels of security and decentralisation. State channels are an approach to overcome this limitation, as they aim to decrease the required on-chain transactions for a given application and thus indirectly increase the capacity (in terms of applications) of public blockchain systems. In the present paper, we propose a state channel design that, through the use of RSA accumulators, operates on a compact state structure. This scheme is optimal for applications whose state includes large sets of elements. The novel state channel design is presented by analysing all state channel operations and how they have to be revised. The security of the design is discussed, while a practical use case scenario regarding the use of the design for an on-chain asset (e.g., non-fungible tokens) exchange application is also analysed.

1. Introduction

Blockchain technology has emerged as one of the most significant advancements in computer science in the past decade [1]. It facilitates the design and operation of secure distributed systems without the requirement for trusted central nodes. Even though existing designs come with several drawbacks, such as inefficiency for high loads of usage or inherent privacy issues, the vision of shifting from Web 2.0 to Web 3.0 [2] and the elimination of the requirement for central nodes for most of the existing systems maintain high expectations out of blockchain technology.

The unbounded popularity and ever-growing usage of blockchain networks have exposed the scalability problems that are built into most designs [3]. Various methods have been developed to combat this issue [4], one of them being the use of state channels [5]. State channels operate on the notion that transactions (also known as state updates) should remain off-chain and be implemented upon interaction between the directly interested nodes unless there is a necessity for blockchain communication. However, to achieve the security and finality levels of the blockchain, it must always be possible to resolve conflicts on-chain. In the vast majority of cases, this requirement is served by occasionally uploading state transitions to an on-chain smart contract that acts as an adjudicator. To that end, state channel nodes are frequently required to maintain a number of previous states. The size of these data often greatly burdens the local storage of channel participants, while it also induces significant costs when participants have to go on-chain to resolve disputes. The size of a single state as well as the level of necessity to maintain previous states in a node is an important parameter for state channel designs' efficiency and level of usability.

The present paper aims at providing a state channel scheme that will enable the secure and efficient operation of state channels for applications, the state of which can be partially or fully represented as a dynamic set of elements that tends to grow significantly in size. With the traditional state channel approach, a very large set of elements would require a boundless channel state representation, which would cause practical difficulties, as participants would be required to maintain a large data structure. According to state channels' general design, in order to resolve a potential dispute due to the inactivity or malicious behaviour of a

* Corresponding author. Department of Information Security and Communication Technology, Norwegian University of Science and Technology (NTNU), Mail Box 191, Gjøvik, NO-2815, Norway.
E-mail addresses: lnegka@uth.gr (L. Negka), akatsika@uth.gr (A. Katsika), georgios.spathoulas@ntnu.no (G. Spathoulas), vpp@uth.gr (V. Plagianakos).

https://doi.org/10.1016/j.bcra.2022.100114
Received 1 February 2022; Received in revised form 4 October 2022; Accepted 9 November 2022

participant, the need to bring the state of the state channel to an on-chain contract arises. This creates significant implications, as it is technically challenging to bring on-chain a list of elements of dynamic size, and it can only be achieved through an expensive (in gas fees) approach, e.g., sending elements one by one. Moreover, it would be required to recursively parse that list element by element in the on-chain contract to resolve the dispute. The basic concept of the proposed design is to serve applications, the state of which can be partially or fully represented as a dynamic set of elements that tends to grow significantly in size in a much more efficient way. The proposed scheme expresses the state of such a state channel with a data structure of manageable, set size that does not take up huge amounts of node storage space and does not have boundless costs when uploaded and processed on-chain. This is achieved through the use of cryptographic accumulators that enable constructions that can efficiently serve the need for proving the existence/non-existence of an element in a set, while the size of those constructions is fixed and irrelevant to the number of elements they refer to.

1.1. Research objectives

The research objectives of the present paper are as follows:

- Development of the first state channel design that can efficiently and securely handle states that may contain unbounded sets of elements with states of constant size. The proposed design applies to applications in which the states fully or partially consist of dynamic sets of unordered elements.
- Revision of existing state channel schemes to adequately handle RSA accumulators as part of the state during funding, state update, dispute, and closing phases of a state channel.
- Extensive analysis of a use case scenario that demonstrates the application of the proposed scheme to a specific real-world application.
- Security analysis of the proposed design to ensure that it provides an equal level of security with existing state channel designs.

2. Related work

2.1. On state channels

2.1.1. ForceMove: an n-party state channel protocol

ForceMove [6] pioneered a design that was subsequently adopted by the majority of state channel designs in terms of dispute handling. By focusing on turn-based applications, the authors developed a mechanism to handle inactivity-related disputes between channel participants. Their design ensures that an honest party can always retrieve its assets when confronting an uncooperative party in the channel. The main pattern for dispute handling established in this work is as follows: a challenger issues a dispute, and the challenger has a time window in which they can provide a valid response, or the channel closes according to the last valid state that usually favours the challenger.

2.1.2. Counterfactual: generalised state channels

The goal of the authors of Counterfactual [7] was to essentially create a “state channel template” that can be utilised by any application provided some use case-specific modifications are made. An easy-to-use API is provided so that it is feasible for any application meeting the requirements to make those modifications. The concept of the design centres around the term “counterfactual”, which stands for outcomes, states and contract instantiations that have not happened yet but cannot be prevented from happening and therefore have relative finality. Metachannels are also implemented, which are channels formed between two users through a common intermediary.

2.1.3. You sank my battleship! a case study to evaluate state channels as a scaling solution for cryptocurrencies

Kitsune, the framework presented in this work, is an application-agnostic, n-party state channel architecture that combines the features authors found most useful in previous implementations like Sprites and Perun. It provides a template for application contracts that facilitates the addition of state channel features into any existing application, and functions simply by freezing app functionality on-chain and transferring it to the channel, and resuming it when the channel comes to a close. During the running of the app on the channel, any party can propose an update and has to collect signatures from all other parties for it to be considered valid. The paper does not address how a scenario where updates are proposed simultaneously is handled. The channel can be closed cooperatively or through dispute. Disputes can be started by any participant and initiate a timer for parties to submit state information. Anyone can resolve the dispute, and it seems the only possible outcome is giving the state to the application contract and resuming on-chain.

The authors conducted an experiment during which they analyse the process, challenges, costs, and usefulness of implementing a Battleship game through state channels, concluding that any application that has a liveness requirement and not all participants are willing to cooperate is not particularly suited to this technology.

2.1.4. Hydra: fast isomorphic state channels

The Hydra design forfeits the sequential transaction processing that state channel designs usually function by in favour of concurrent processing enabled through the use of the extended unspent transaction output (UTXO) model. The isomorphic channels the authors describe, called heads, function by moving a set of UTXOs the participants decide on off-chain, evolving them there and then making sure the latest state of the channel at close is reflected on the chain. The process begins with an initial transaction through which the parameters necessary for the channel (e.g., participant list) are defined and participation tokens necessary for the state validation process are forged for each participant. State propositions are validated through multi-signatures and periodically collected into snapshots to align the channel view of the head parties, which often differs due to the concurrent transaction confirmation scheme. Snapshot leaders are responsible for this gathering of states and for conflict resolution that may be needed. At the channel’s closing, a contestation takes place during which channel members submit valid states more recent than any already submitted, and the latest one gets reflected on-chain after the finalisation of the head. This design results in isomorphic channels that are faster than any previously implemented ones and boast performance near physical limits.

2.1.5. Multi-party virtual state channels

The authors of this work offered two significant contributions to state channels existing so far at the time of publication.

Introduced in this work are multi-party virtual state channels, an expansion on 2-party virtual channels that had already been implemented. Virtual channels can be opened and closed without ever referring to the blockchain. Multi-party virtual state channels can execute contracts that concern over two parties. They stand and are built recursively on 2-party ledger channel networks, through which all participants must be connected, and no creation of channels of intrinsic greater length is supported. For every multi-party virtual channel created, participants must install in every sub-channel contract instance that guarantees that the intermediary’s funds are safe and that the outcome of the virtual channel will be updated on said sub-channels.

The second major contribution is the introduction of direct dispute state channels. Those introduce functionality that redesigns the dispute process so that participants refer to the ledger as soon as possible after an honest party identifies possibly malicious behaviour, instead of contacting the intermediary. The dispute board is an on-chain component

where states are uploaded in case of dispute to be compared and the valid one is finalised. This approach makes the worst-case time complexity of the channel independent of its length and allows dispute outcomes to be seen and used by other contracts. However, it can cause a serious transaction load on the blockchain depending on the scenario, since all parties can be required to upload their state view for a dispute. Channels that implement direct and indirect dispute processes can cooperate seamlessly in whatever proportions serve each application best.

The authors' approach to the overall design is based on previous work and results rather than conducting an implementation from scratch.

2.1.6. Sprites and state channels: payment networks that go faster than lightning

Even though the main focus of the Sprites proposal is payment channels and networks, they base their design on a modular approach leaning on a general state channel construction. State channels are mainly used in the dispute process, the sequence of which has been adopted by most following state channel designs: a party raises a dispute after malicious behaviour occurs, a time period follows during which evidence is submitted, and finally, depending on the situation, the resolve is either cleared off-chain or resolved on it.

2.1.7. Two-party state channels with assertions

The work [8] presents a design that is innovative in how it aims to reduce the cost of the process of bringing a state on-chain, as well as to unburden the honest party from the costs of raising a dispute. The design concerns strictly two-party applications of a turn-based nature. The authors proposed that initially only a hash of the state is submitted to the contract during a dispute in an attempt to make the size of the data going on-chain constant and independent of the size of the state.

2.2. On cryptographic accumulators

The latest insights into the implementation of accumulators resolve around Bitcoin and specifically the scalability problems of the UTXO protocol. To the authors' best knowledge, there is no prior work affiliating accumulators to state channels. Nevertheless, some interesting approaches are presented for further understanding the utility of accumulators.

2.2.1. Batching techniques for accumulators with applications to IOPs and stateless blockchains

In this work, Boneh et al. [9] used new accumulator and vector commitment constructions to design a stateless blockchain where the nodes may participate without storing the entire state of the ledger but alternatively short commitment to it. The benefits of smaller membership proofs, the constant size proofs and the ability to aggregate those for a batch of transactions are promising aspects for future research and practical applications.

2.2.2. Zerocoin: anonymous distributed E-cash from Bitcoin

Zerocoin [10] is a distributed e-cash scheme constructed as an extension to the Bitcoin system, relying on the cryptographic properties of RSA accumulators and non-interactive zero-knowledge signatures to allow anonymous currency transactions.

2.2.3. EPBC: efficient public blockchain client for lightweight users

EPBC [11] aims at providing a mechanism for checking the validity of a given block and the transactions contained in it to lightweight users of blockchain-based applications without storing the entire blockchain. The purpose is to give blockchain transaction history a compact form and reduce storage requirements using the RSA accumulator.

2.2.4. Utreexo: a dynamic hash-based accumulator optimised for the Bitcoin UTXO set

Another applied methodology for the efficient management of the UTXOs set is described [12] based on a dynamic hash-based accumulator, designed as a forest of perfect binary hash trees.

2.2.5. The state of statechains: exploring statechain improvements

In alignment with state channels, statechains [13] are a second-layer solution for off-chain transactions of an on-chain UTXO. The notion of a state-accumulator is proposed, as they maintain a constant size independent of the number of transfers that are made within the lifetime of the channel and improve the storage requirements for the participants. However, this scheme requires a trusted setup, and on top of that, is based upon the existence of a trusted manager entity for the channel that, besides setting up the accumulator, is also required to preserve all previous states for the lifetime of the channel. This significantly limits the applicability of the protocol, as it cannot operate efficiently for an unbounded number of state updates.

2.3. Summary

Since there are no previous efforts to solve the problem of unbounded channel state size with any similar techniques, the works cited concern the technologies our design is motivated by: state channels and cryptographic accumulators. The single publication that approaches the issue of the size of the state [8] does so on significantly different terms (initial attempt to resolve disputes by only submitting a hash of the state) and with results only in terms of the data submitted on-chain and not the storage space occupied in participating nodes since they still have to maintain the information in its entirety.

3. Preliminary concepts

3.1. State channels

Blockchain technology turned out to have a much greater impact than what could have been predicted when it was initially popularised. That fact has brought to the surface major scalability issues that present blockchain designs suffer from. Three properties have been identified as the ones a blockchain system should prioritise: security, decentralisation, and scalability. Much effort has gone into keeping all three up to par, but with limited success, resulting in the necessary sacrifice of at least one of them in the vast majority of implementations [4]. Between a verification process that forces the slowest node as the design's bottleneck and a consensus mechanism as intensive as PoW is, scalability is the feature that most often suffers, both in terms of speed and costs.

Approaches to resolving this problem are mainly split into two categories:

Layer 1 involves changes in the fundamentals of a blockchain system such as a different consensus mechanism [14,15] or the implementation of sharding [16,17].

Layer 2 involves constructions on a different layer on top of existing blockchain designs that can rectify their weaknesses. Dominant approaches on this aspect include:

- **Sidechains** [18]: the method of having parallel, independent chains running next to the main blockchain to reduce its transaction load.
- **Plasma** [19]: a separate blockchain that operates beside Ethereum and makes periodic commits to it for security. Plasma does not support the execution of generic smart contracts.
- **Rollups** [20]: rollups enable execution to happen off-chain while data storage happens on-chain. They are further divided into Zero Knowledge and Optimistic Rollups depending on the transaction validation methods used.
- **Payment** [21] **and State Channels** [22]: both kinds of channels promote full transaction execution off-chain in the optimistic case. Payment channels can only accommodate transactions that express payments, while state channels can also enable off-chain execution of smart contracts.

Payment and state channels are often grouped since one is heavily influenced by the other. Payment channels are the predecessors of state channels and initially introduced the concept of performing transactions off-chain whenever possible. However, they limit the application of this concept to payments, and therefore can only benefit cases that involve strictly withdrawals and deposits. State channels came later to extend this functionality to include the alteration of state and the execution of code.

The root of the scalability issues blockchain systems are facing is mainly the necessity that no matter how small the user pool a transaction is relevant to, it must be executed by every blockchain node. State channels allow communication to remain off-chain and between this limited subset of users until an update on the blockchain becomes absolutely necessary. They also manage to maintain the security guarantees and transaction finality that on-chain interaction can offer.

To be able to provide these guarantees, a channel must rely on the liveness of the underlying blockchain system since in the pessimistic case, a channel transition must be able to be executed on-chain. To that end, channel participants must lock collateral in the channel contract that can later be used to punish malicious activity if necessary. In every case, those assets will be redistributed to users at the closing of the channel according to its outcome.

There are four main phases a state channel can be in while it operates:

Opening/Funding Phase: This phase involves the establishment of the state channel between its participants. This process most often includes instantiating a state channel contract on-chain, funding it by providing collateral assets and funds to be used in the application if applicable, and generating the starting state of the channel.

Update Phase: This phase encapsulates the main intended functionality of a channel. Users communicate with each other off-chain through cryptographically signed messages to proceed from each state to the next one. Total consensus between channel participants is necessary for a state to be considered valid and of equal finality to an on-state update. This phase is only exited in the case of malicious activity or the need to close the channel.

Dispute Phase: User behaviour that diverges from the protocol makes the state channel enter this phase. Such behaviour can include proposing an invalid state or remaining inactive when action is required. In such cases, obtaining full consensus on a state is infeasible, and the channel must be able to rely on the blockchain system for a fair resolution; therefore, this is when on-chain interaction is necessary.

Closing Phase: This phase can occur in one of two ways: (a) optimistically, users decide to consensually close the channel and bring an end to their interaction, so they provide a valid state for the channel to finalise on, or (b) pessimistically, after the occurrence of malicious behaviour and the failure to resolve it on-chain in a way that allows the channel to continue operating, assets are distributed according to the last valid state that is brought to the contract. That is necessary to protect the assets of honest parties being trapped in the contract of a channel that cannot close optimistically.

A functional state channel must be able to boast the three following properties:

- Be trustless, in the sense that a participant can be assured of the relevant safety of their assets regardless of the behaviour of the other participants. Therefore, no trust is required between users.
- Guarantee the finality of the state at the same level, which can be guaranteed on the blockchain.
- Be efficient in serving its main purpose, which is to reduce the transaction load that the blockchain system would suffer.

3.2. Cryptographic accumulators

An accumulator is a cryptographic function that enables the generation of inclusion proofs for sets of elements without disclosing any member in the subjacent set. Various accumulator schemes have been proposed for diverse applications. All of those share the objective of determining whether an element is part of a set or not. In general terms, the set of values accumulated is represented by a compact data structure in such a way that for every value of the set, a witness proof can be produced, and thereupon determine whether the value is incorporated in the accumulator or not [23].

Relying on fundamental cryptographic principles, accumulators have raised increasing interest due to their space and time efficiency, especially when an arbitrarily large set of values is examined. Since the first proposed cryptographic accumulator by Benaloh et al. [24], which used a one-way RSA function to address document time-stamping and membership purposes, many studies have expanded the accumulator usage and have delivered numerous derivations. Merkle trees and bilinear accumulators present various characteristics that can be useful in a vast variety of anonymous authentication applications, while RSA accumulators have the advantage of constant-size proofs.

Depending on the cryptographic primitives that they are built upon, accumulators can be categorised as symmetric or asymmetric. Furthermore, accumulators can present a variety of features regarding the size of the initial set, the type of membership proof, the existence of a trusted coordinator, known as an accumulator manager (AM), and the required update frequency (communication) for the participants [25]. Combining different features can provide many design choices, subsequently affecting the implementation complexity and the overall performance of the accumulator.

3.2.1. Main characteristics of accumulator schemes

The main characteristics of accumulator schemes along with relevant cryptographic assumptions are presented below.

3.2.1.1. Dynamic accumulator. A dynamic accumulator can be updated efficiently as elements are added or removed from the set, at a unit cost independent of the number of accumulated elements [9].

3.2.1.2. Universal accumulator. Universal accumulators extend dynamic accumulators [26] with the support of non-membership witnesses.

3.2.1.3. Trapdoorless accumulator. Accumulators conventionally use a trusted accumulator manager allowing efficient deletion of elements from the accumulator using “trapdoors”. Trapdoors in cryptography consist of information needed to perform the inverse cryptographic operation [25]. Since the first RSA accumulator by Benaloh and de Mare [24] trapdoors were considered a disadvantage, anyone who has access to such information may forge membership proofs. In recent works [9, 27–29], efficient schemes have been proposed without requiring a trusted setup.

3.2.1.4. Strong RSA assumption. The strong RSA assumption is the basis upon which a variety of cryptographic procedures have been built stating that given a random generator of unknown order $g \in G$, it is infeasible to find any root of it, i.e., an integer $l \in Z$ and an element $u \in G$ such that $g^{1/l} = u$. The strong RSA assumption generalises and implies the RSA assumption.

3.2.1.5. Adaptive Root Assumption. This assumption was introduced by Wesolowski [28], who initially presented it as the “root finding game”. These two assumptions are incomparable, as the latter states the difficulty of finding a random root of a chosen group element, whereas the former upholds the difficulty of finding a chosen root of a given random group element [9].

3.2.2. A basic accumulator scheme

Below, we describe the main functionality of an accumulator considering the existence of a trusted manager, a user responsible for an element and the corresponding membership witness acting as the prover,

and a user given the relevant proofs acting as the verifier at a specific verification round [30].

- **Generation/Set up algorithm** is performed by the manager and generates the initial set of the accumulator A_0.
- **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 w_x. Additionally, an update message is produced $upmsg_{(t+1)}$ that enables the accumulators' users and proof holders to keep the witnesses of their elements up to date.
- **Del algorithm** respectively deletes an element x from the accumulator, produces the updated accumulator value A_{t+1} and the non-membership witness proof for element x, labelled as u_x and produces 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.
- **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.
- **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. A non-membership witness can also be verified.

4. Design of state channels with compact states

In the present section, we present a revised state channel design that supports applications, the state of which can be partially expressed by one or more dynamic sets of unordered elements. The transition between different states of the application includes the addition or removal of an element in the set. The design enables the secure and efficient operation of such applications through the use of state channels and cryptographic accumulators. The goal of the proposed scheme is to support the aforementioned applications irrespective of the size of the elements set by combining on-chain security guarantees and off-chain efficiency.

The presentation of the scheme is divided into two parts. In the first part, the functionality of cryptographic accumulators is described in the form and capacity that is used within the scope of this work. In the second part, the integration of this functionality into the processes of state channels is analysed.

4.1. Accumulator functionality

The main accumulation procedures adopted for the scope of this work are based upon the trapdoorless universal accumulator proposed by Boneh et al. [9]. This design follows the definitions and conventions of Baldimtsi et al. [30] and was built upon a basic RSA accumulator with the addition of batching and aggregation techniques. Assuming that the strong RSA assumption holds in the generated group of unknown order and that all accumulated values are odd primes, the accumulator consists of the following basic procedures, according to those presented in Section 3.2. These basic procedures are described in three subsections related to basic accumulator handling, membership/non-membership proofs management and proofs verification. It has to be noted that the present subsection describes all operations available for an RSA accumulator. Our design adapts to the requirements of each application and may accordingly make use of the optimal subset of the operations described.

4.1.1. Basic accumulator handling

- **Setup:** Generation of a group of unknown order and initialisation of the group with a generator $g \in \mathbb{G}$, where the strong RSA assumption holds. Let g be the generator of every element $x \in \mathsf{S}$, $\mathsf{S} = \{x_1, x_2, \ldots, x_{(n)}\}$, and H_{prime} be a hash function that maps any element x_i to a unique odd prime number e_i:

$$e_i = H_{prime}(x_i) \tag{1}$$

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

$$A = acc(S) = g^{\Pi_{i \in [n]} e_i} \tag{2}$$

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

$$A_{t+1} = (A_t)^{e_i} \tag{3}$$

- **Del:** Removing an element from an accumulator without the use of a trapdoor requires the reconstruction of the whole set. In this case, the root factor algorithm (adaptive root assumption) can be used for time efficiency. In the proposed scheme, 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. Based on this assumption the process of updating the accumulator is presented in Eq. (4).

$$A_{t+1} = w_{i_t} \tag{4}$$

4.1.2. Membership/non-membership

- **MemWitCreate:** The membership witness for an element with regards to a specific set is equal to the accumulator value for the same set excluding the specific element. The membership witness w_j for an element e_j can be computed as

$$w_j = (A_t)^{-e_j} \tag{5}$$

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, this limitation can be bypassed given the assumption that the user that adds an element to an accumulator maintains its previous value A_{t-1}. In that way, an approach to verify membership witness in constant time with a single exponentiation is feasible (as explained below).

- **NonMemWitCreate:** The non-membership witness u_i of an element e_i that is not included in an accumulator A can be calculated as long as the product of all the accumulated elements is known. The non-membership witness is equal to the pair (a, g^b), where a, b are the Bezout coefficients between e_i and the product of the accumulated elements, relying on the fact that for any element $x \notin \mathsf{S}, gcd(x, \prod_{s \in S} s) = 1$.

A user of an accumulator that maintains knowledge of the elements in the accumulated set and can easily compute the pair a, g^b that provides the necessary non-membership witness. Because this operation induces sub-optimal storage requirements, our design provides several alternative solutions (according to the application requirements) in Section 4.3 that enable the state channel operation without the need to maintain knowledge of all the elements in the accumulated set.

4.1.3. Witness management and verification

- **MemWitUpdateAdd:** 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}.

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

- **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. Through using the Shamir Trick, we can compute the Bezout coefficients a, b such that $a \times e_i + b \times e_j = 1$ for the pair of coprime integers e_i, e_j. After computing a, b, we can produce the updated membership witnesses according to equation

$$w_{it+1} = w_{i_t}^{b} A_{t+1}^{a} \tag{6}$$

- **VerMem:** Verification of a membership proof w_i for an element e_i given the current accumulator state A_t requires one exponentiation in $\mathbb{G}$. It is required to add element e_i to the set accumulated in w_i and check if the result equals A_t.

$$A_t = (w_i)^{e_i} \tag{7}$$

- **VerNonMem:** A given non-membership witness $u_i = (a, g^b)$ for an element e_i is verified if Eq. (8) holds.

$$A^a \times \left(g^b\right)^{e_i} = g \tag{8}$$

4.1.4. Accumulator setup requirements

It is important to understand the security implications of the requirement for a trusted setup. A lot of work has been focused on eliminating this requirement, as picking an RSA modulus $N = pq$ leads to knowledge of the order $\varphi(N) = (p-1)(q-1)$ [9].

One way to resolve this is the use of class groups of imaginary quadratic order [29,31]. Given the recent rise of interest in groups of unknown order fueled by recent applications such as delay functions [28] and zero-knowledge proofs [32], numerous research works have explored the use of class groups of imaginary quadratic fields such as Refs. [29,33,34]. As stated in Refs. [34,35], some of these constructions [28] rely on novel, nonstandard cryptographic assumptions, namely, Low Order (LO) or Adaptive Root (AR) assumptions in groups of unknown order, which lead to an emerging need for better understanding these new, non-standard cryptographic tools and explore their potential.

As already noted, the proposed scheme is built upon the accumulator design proposed in the work of Boneh et al. [9] and does not attempt to address the inherent disadvantages of a trusted setup. The scheme operates on the assumption that the RSA generator has been generated through a secure function. Based on Wesolowski's [28] Adaptive Root Assumption, Boneh et al. [9] showed that Wesolowski's proof is a succinct proof of knowledge of a discrete log in a group of unknown orders and therefore provides a secure basis for our approach.

4.2. State channel functionality

The goal of the proposed design is to minimise the required storage space along with the financial overhead when operating in a channel, the state of which can be fully or partially expressed through dynamic sets of elements. To this end, we have redesigned each phase of a state channel as described in Section 3.1 and propose an enhanced state channel design that efficiently integrates cryptographic accumulators into state representation. The presented scheme enables participants to maintain a compact state representation (off-chain) that, through the use of membership/non-membership proofs, can serve the need for managing sets of elements with the same level of security that an on-chain implementation would provide.

The design of the scheme prioritises the maximisation of efficiency and the reduction of costs that result from on-chain interactions with the channel smart contract. Some parts of accumulator functionality have been identified as quite computationally demanding; therefore, actions have been taken to avoid bringing them on-chain and wherever possible eliminate the need for their use. To include accumulators for different sets in the state of the state channel, a default accumulator setup is required. When the channel contract for the state channel is deployed, default values for g and N have to be set. During the operation of the state channel, each new accumulator must be initiated with the default g and N values, to maintain the same level of security. As discussed in Section 4.1, N must either be generated by a trusted node, which will not make use of p and q values during the operation of the channel, or through a secure distributed process based on class groups of imaginary quadratic order [31]. According to the context under which the state channel is deployed, channel operators have to choose one of the two aforementioned approaches.

The rest of the present subsection analyses how those suffice to cover functionality that might be required in a state channel environment. The notation used in the subsequent figures is analysed in Fig. 1.

4.2.1. Funding

Since this phase of the channel does not involve the creation or alteration of an accumulator, its functionality does not differ significantly from most state channel designs. The funding process and establishment of the channel happen concurrently.

The pattern followed is apparent in Fig. 2. Every participant X that joins the channel locks into a smart contract asset dep_X that is used in the context of the application served by the channel and also serves as collateral in the case of malicious behaviour. Participants also store any required data for operating the application, such as their public keys pk_X or any other application-specific data. The contract maps this information to every participant's address for future use.

4.2.2. State updates

This design aims to assimilate accumulator functionality into state channels. We assume that the state of the state channel includes (potentially among other data structures) an accumulator to represent an unbounded element set. In the present paragraph, an analysis of the operations of the accumulator in the context of updating the state of the channel is presented. The state in its entirety can include more structures besides the accumulator, but the details of managing those are out of the scope of the present paper.

4.2.2.1. Establishment of a set. The process depicted in Fig. 3 is followed for the creation of a new accumulator that will represent a new set of elements. The steps of this process are as follows:

- A channel participant wants to create an accumulator that may include one or more initial elements of the set.
- The participant copies the default accumulator parameters (existing in the channel contract) to the state of the channel. If it is required (by the application) to add one or more initial elements to the set, then the **Add** process is also followed.
- The generated accumulator is included in the state proposed by the participant to be signed by all other channel members.

4.2.2.2. Modification of a set.

- A participant can modify the accumulator through the addition or removal of an element according to the **Add** or **Del** processes, respectively, as described in Section 4.1. The process of adding an element to the accumulator is depicted in Fig. 4

- The participant must also update accordingly all the relevant witnesses they hold and explicitly declare the proposed action (e.g., add

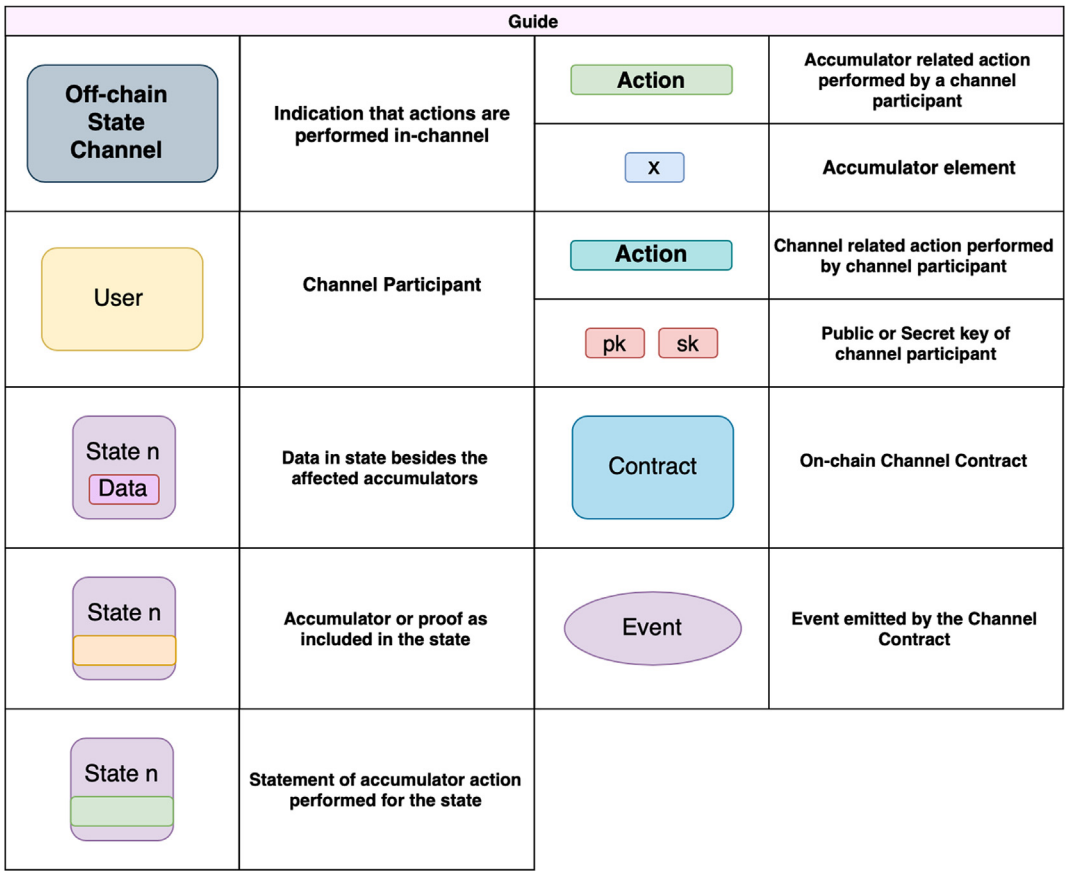

Fig. 1. Figure notation.

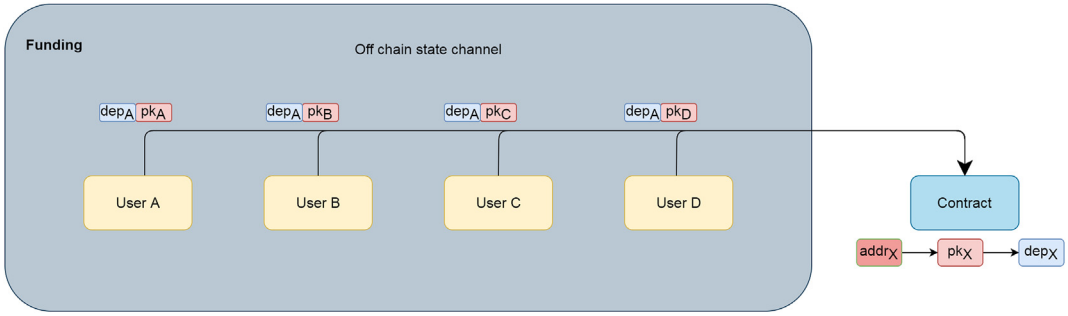

Fig. 2. Establishment and funding of the state channel.

element X to the accumulator), which facilitates the update of witnesses of all elements by the corresponding channel members. This ensures that the load of updating every witness does not fall on a single node. It is assumed that for every element in a set, there is at least one member with an interest in maintaining an updated membership witness for it.

- The updated accumulator and witness update info (proposed action) are included in the state proposed by the participant to be signed by all other channel members.
- After receiving the state but before signing it, channel participants must update and verify their witness proofs to ensure that the modification of the accumulator is exactly as stated by the state transition proposer.
- If the state proposal is valid, members sign it with their secret key. Once a state proposal receives all signatures, it is established as the latest valid state.
- The processes of adding or removing an element are identical in terms of in-channel interactions. The difference is apparent in the witness update process that is analysed in Section 4.1.

4.2.2.3. Proving that an element belongs to a set. In traditional state representation, where the element set is expressed as a readable list, the

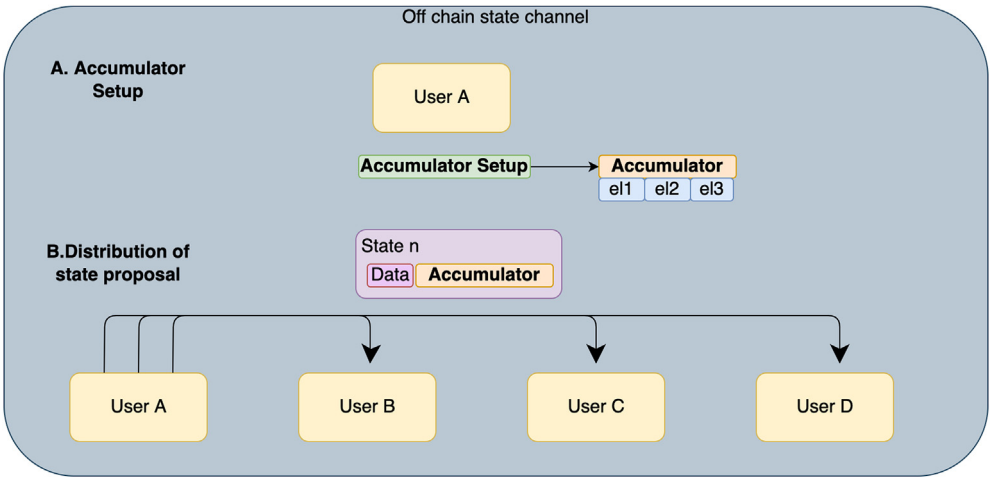

Fig. 3. Establishment of an accumulator with initial elements.

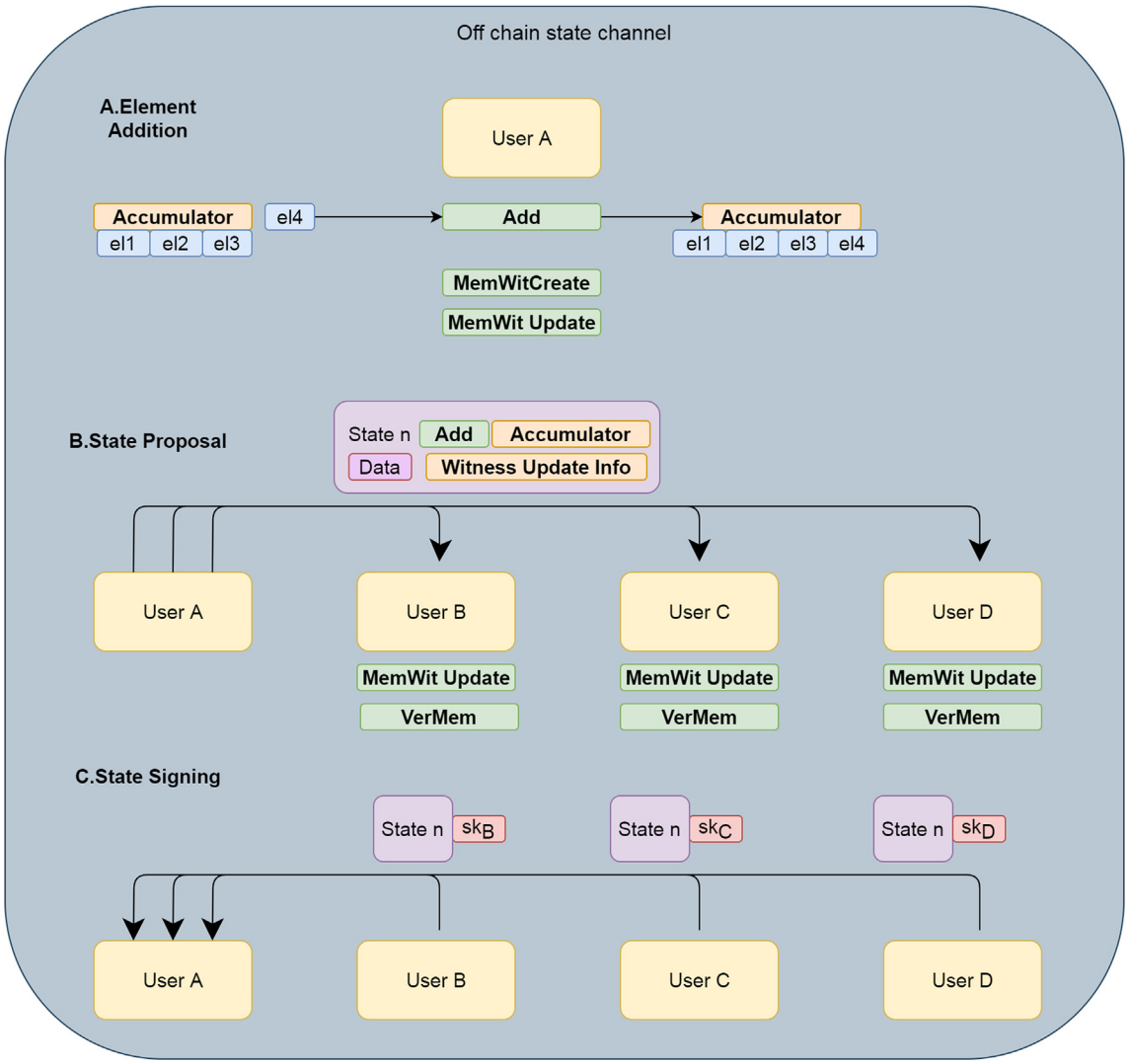

Fig. 4. Addition of elements to the accumulator process.

validity of the transition is transparent. In contrast, the use of accumulators means that the actual data have been replaced by data commitments that cannot support the straightforward state transitions' validation. To that end, proofs of membership and non-membership are proposed to support the validation process of the state transitions.

Proving whether an element is included in a set is synonymous with whether a membership or non-membership proof can be provided for that element. The generation of those proofs is done through the process

described in Section 4.1. Because of the complexity associated with processes related to non-membership proofs, alternative ways to achieve the same functionality are analysed in Section 4.3. For the verification of accumulator alterations, membership and non-membership proofs are not necessary. There may be cases where such proofs must be included in the state because of application-specific reasons.

4.2.3. Dispute

State channels' security guarantees stem from the fact that there is an on-chain contract that can function as an adjudicator during disputes. According to general practice, the said contract shall carry functionality that verifies whether a transition between two states is valid. This can be achieved by replicating the transition in the contract to check if it produces an identical result. Therefore, in this case, the state channel contract needs to be able to perform the required operations for the verification of state transitions and any membership or non-membership witnesses in relevance to a particular accumulator.

A fundamental principle of state channels is that they maintain the same level of transaction finality as the underlying blockchain. State channel designs achieve this by requiring full consensus for every state update, which translates to a necessity for every participant to sign every update. Because of this requirement, the most prevalent type of challenge is the inactivity dispute against a member being inactive while expected to act. In fact, most kinds of malicious behaviour can be dealt with through an inactivity dispute on the assumption that the channel treats an invalid response as a non-existent one. There is a meaningful distinction between different kinds of disputes in terms of who they were initiated by:

4.2.3.1. Dispute initiated by the state proposer. This dispute is used by the member who attempts to submit the next state update when one of the other channel members does not respond with a signed state update.

However, the proposer might have submitted an invalid update and been trying to capitalise on other members' refusal/inability to sign it in order to "force" the channel to a close. In this case, the dispute progresses as depicted in Fig. 5 and analysed herein:

- User A proposes an update from $State_{n-1}$ to $State_n$ that never gets signed by user B. Therefore, user A initiates an inactivity dispute (against user B) on the on-chain smart contract. This is done by submitting to the contract the new state proposal ($State_n$), the previous state ($State_{n-1}$), the identity of user B and the information the contract needs to verify there's been a valid transition between the two.
- The smart contract checks the state transition's validity, and the process continues according to the result of this check:
  - If the transition is invalid, then the dispute is resolved in favour of user B, and the state transition is rejected.
  - Otherwise, the contract emits an event informing members of the channels for the dispute and waits for a predefined time window for user B to respond. The dispute can be resolved according to the following possible outcomes:
    * User B can provide the contract with a version of $State_n$ they have signed.
    * Any member can submit to the contract a valid state more recent than $State_n$, therefore showing that user A is proposing a stale state and user B has no obligation to sign it.
    * If the contract does not receive either of those two valid responses within the time limit, the dispute is resolved in favour of user A, and the channel terminates in the most recent valid state provided to the contract, $State_{n-1}$.

4.2.3.2. A dispute initiated by a channel member that is not the proposer because the proposer is displaying inactive behaviour or is proposing invalid state updates. In that case, the dispute is handled by the contract differently.

- User B initiates through the contact an inactivity dispute against the current proposer. User B provides $State_{n-1}$ as the last valid state to serve as a reference point. Since the application is turn-based and the order is predefined, the party responsible for providing $State_n$ is known. Let us assume that the party is user A.
- The contract allows a time window for one of two valid responses to be submitted:

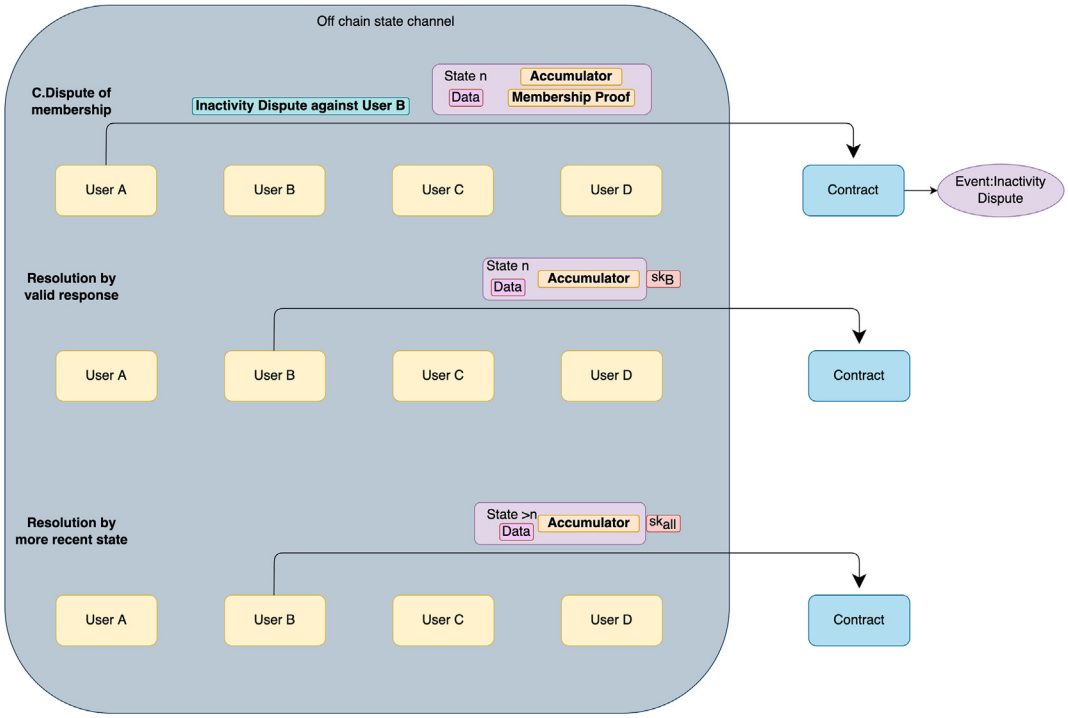

Fig. 5. Issuing and resolution of a proposer-initiated dispute.

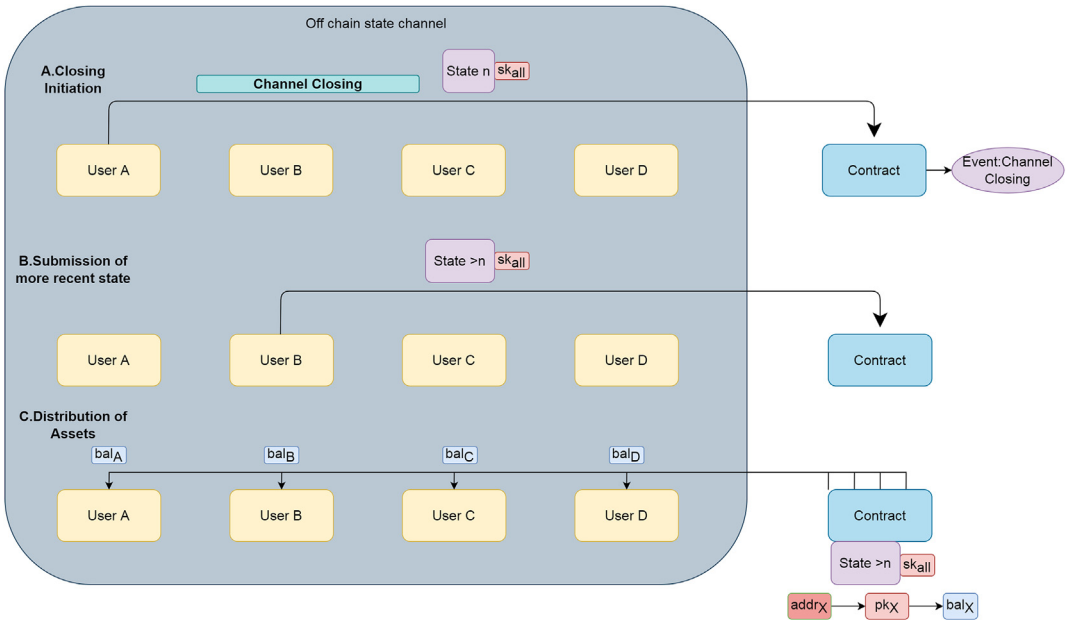

Fig. 6. Optimistic channel closing.

  - User A can provide the contract with a proposal for $State_n$ that the contract will manage to verify as valid.
  - Any member can provide the contract with a valid state more recent than $State_{n-1}$, therefore showing that user A is not the one obligated to provide the next update.
- If the contract does not receive either of those two valid responses within the time limit, the dispute resolves in favour of user B, and the channel terminates in the most recent valid state provided to the contract, $State_{n-1}$.

4.2.4. Closing

Apart from terminating the channel after failing to resolve a dispute, there is also the option of optimistic closing, depicted in Fig. 6.

- Any participant can bring a valid state to the contract and request the initiation of the closing process.
- The contract will then allow a time period for any other participant to submit a more recent valid state, if such a state exists.
- The channel will close, and the assets will be redistributed according to the most recent valid state that participants have provided the contract with during this time window.

This process is preferable over the pessimistic closing of an unresolved dispute because it is associated with reduced costs and delays. A participant could go inactive and force a lengthy dispute process that would remain unresolved and therefore again result in the closing of the channel. Through this process, this user is incentivised to opt for cooperatively closing instead, to escape the penalty fees of inactivity.

4.3. Handling of proofs

Due to the high complexity and therefore load that the algorithms associated with non-membership proofs bring both to the participants of the channel off-chain but also to the on-chain smart contract for necessary validation processes, we have opted to exclude their use from this design. Alternative ways to achieve the same functionality, when necessary, have been developed. According to the requirements of the application implemented in the channel, different approaches are proposed:

4.3.1. Non-membership proofs unnecessary

Applications that are by default functional with just the use of membership proofs are perfectly served by the scheme as described in Section 4.2.

4.3.2. Non-membership proofs necessary, finite set of elements involved

For applications that require non-membership proof functionality and are managing (through an accumulator) a finite set of elements that either belong to or do not belong to a given subset, we propose the following approach, also apparent in Fig. 7.

- For every accumulator A_t created to represent a given subset, another one is also created to represent the elements that do not belong to said subset $(A_t^{'})$. Every state must contain both accumulators of this pair.
- When a modification is made to A_t, the opposite modification must be made to $A_t^{'}$: An addition of element x to A_t requires the concurrent deletion of element x from $A_t^{'}$. A deletion of element x from A_t requires the concurrent addition of element x to $A_t^{'}$.
- Keeping in mind that the representation of the original set is through A_t, then a non-membership proof for element x in A_t can now be replaced by a membership proof for element x in $A_t^{'}$.

4.3.3. Non-membership proofs necessary, infinite set of elements involved

The applications that are in need of non-membership proof functionality and cannot know beforehand the elements that will be involved in their set are further divided into two cases:

- If it fits the nature of the application that the membership proof for an element x is maintained by more than a single channel member, then if a participant is falsely claiming non-membership of said element, another participant providing the element's membership witness is sufficient to prove the false claim. If a valid membership witness cannot be produced by any member, then the claim of non-membership is valid.
- If the application cannot accommodate the aforementioned functionality, then members are obliged to maintain the list of elements of the state so that they can produce a membership proof for any element to dispute a non-membership claim when necessary.

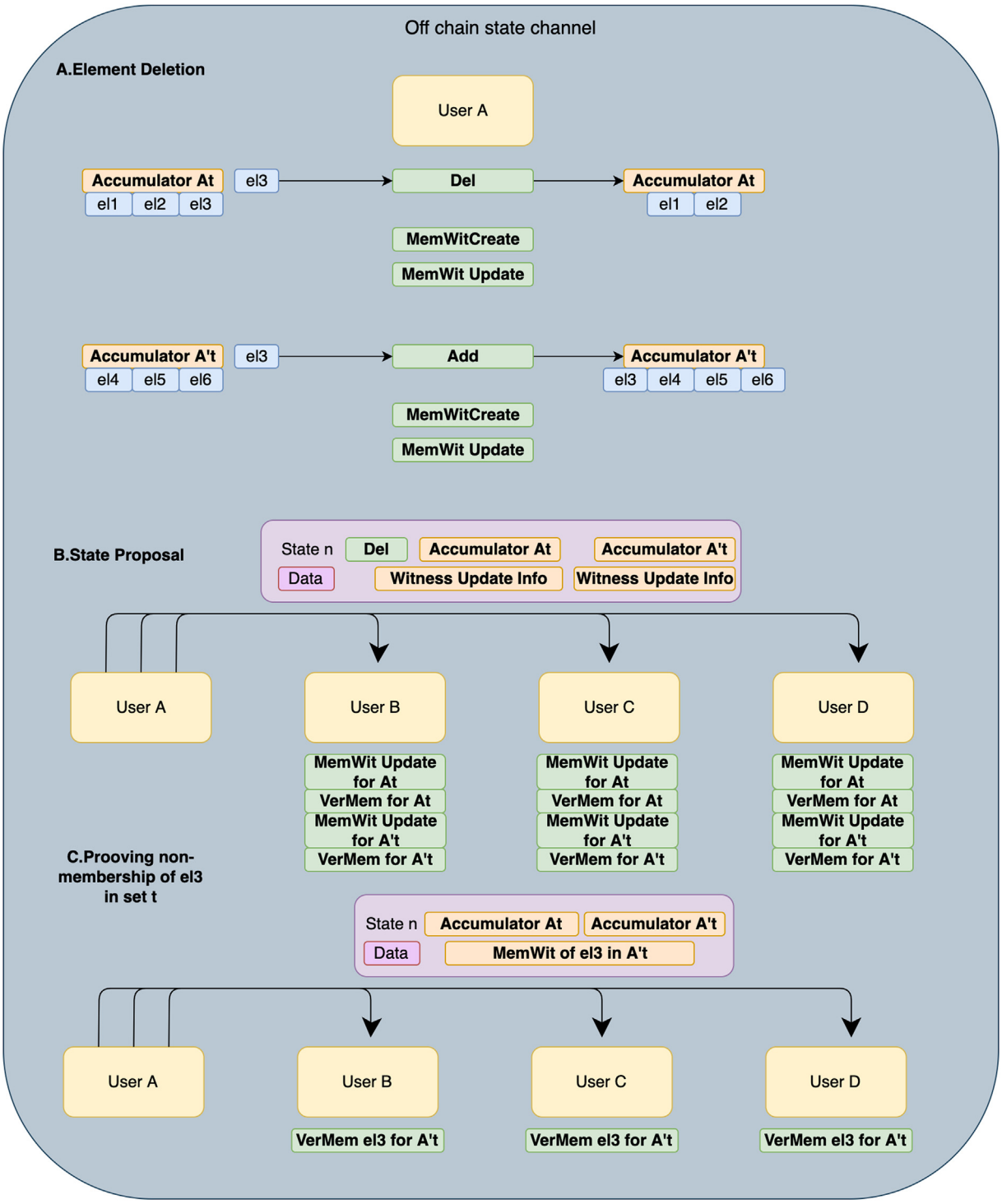

Fig. 7. Proving non-membership in an application with finite elements involved.

4.3.4. Use of non-membership proofs

The reason that this design tries to eliminate the use of non-membership proofs is because their calculation is a demanding process that also has as a prerequisite that the calculator of the proof is aware of all the elements added to the accumulator. This is counterproductive in most applications, considering the aim of the design is to minimise costs and storage load. However, if the nature of the application is such that a member is by default interested in the full set of elements in an accumulator, then the use of non-membership proofs becomes feasible. The user can easily produce a non-membership proof, and the process of verifying it on-chain is rather simple.

5. Security analysis

A state channel bases its security guarantees on the underlying blockchain network. Even if that means there is a strong dependency of the channel on the liveness of the network, this fact cannot be mitigated. However, the state channels also significantly rely on their members' availability, to the point where almost every major threat to the channel can be viewed as either a member displaying inactive behaviour on

purpose or a member exploiting another participant's situational unavailability. This is even more accurate when the fact that an invalid action is treated as a nonexistent one is taken into account.

At its core, securing in-channel interactions relies on ensuring that every malicious action can be identified by participants and mitigated through a dispute process. This is feasible, assuming that all parties are rational and want to protect their assets. An analysis of how this is achieved for each scenario while taking accumulator functionality into account is presented in the present section. The essentials are for the behaviour to be identified by channel members, reported through a dispute, and then verified by the smart contract.

1 **Invalid transition: Invalid initiation.** A user may attempt to hide an element in an accumulator before publishing it in the channel to use the corresponding membership proof at a later stage. In the proposed scheme, this is not feasible, as for a state transition that includes the initiation of an accumulator to be valid, the accumulator should adhere to the default g and N values specified in the channel contract. If a user attempts to change this value to include an element without stating that on the state update, this would initiate a dispute.
2 **Invalid transition: Invalid addition.** Let us assume that user A attempts to perform an invalid addition of elements to the accumulator (add element x, while stating the addition of element y). In practice, this would translate to user A sharing the new state A_{t+1} (which would correspond to all elements included in A_t along with element x) along with the proposition of the addition of x. Other participants would then update the membership witnesses for the elements, so they are responsible for but those would be invalid against the new accumulator A_{t+1}. This indicates that there have been further, unrevealed alterations to the set. Any member can identify this by validating their updated proof and checking if it is valid or not. In such a case, the channel may progress in one of two ways. Any of the members that realised this is an invalid transition can initiate a dispute in the contract, forcing user A to provide an update that the contract can verify as valid or be penalised. Alternatively, the channel participants can refuse to sign the state update. User A's choices now are either sending out a new, valid update or attempting an inactivity dispute through the contract against the channel's other members. The contract, however, has the necessary data to replicate the addition as described on the state update and check whether the transition has been valid. Based on the result of this check, it can then reject the dispute and penalise user A, thus making it pointless for him to initiate the process in the first place.
3 **Invalid transition: Invalid deletion.** The process of identifying and dealing with a user performing an undeclared deletion of an element while stating an addition or a deletion of another element is essentially identical to dealing with an invalid addition. However, because the process of deleting an element from the accumulator is computationally demanding, an alternate on-chain validation process is opted for. Assuming the proposal was for $State_n$, the contract performs an addition to the accumulator of that state and checks if the result matches the accumulator of $State_{n-1}$ to determine if the transition was valid.
4 **Invalid proof.** A membership proof can be easily verified against the accumulator it concerns as long as it is known which element it is for. This verification can also be performed by the smart contract and will be part of validating the transition if a proof is part of the proposed state.
5 **Inactive behaviour.** The process associated with dealing with inactivity either from the state proposer or one of the other members is extensively analysed in Section 4.2.3.

6. Use case analysis: asset trading state channel

In the present section, we extensively discuss the use of the proposed design to support a specific use case. We consider this use case a practical implementation of our structure that justifies the need for a compact representation of an extended collection.

Let us assume a state channel that is built to serve an application requiring asset exchanging. Every user can commit the list of their possessions to an accumulator and provide proof of ownership during the exchange of such assets. In this case, the user is burdened with the maintenance of their accumulator, which will be updated only when they participate in an exchange. It has to be noted that in the specific use case, the user is not required to maintain updated proofs when they are not involved in a transaction.

Based on the analysis of the state channel functionality in Section 4.2, in the following subsection, we discuss an indicative state channel implementation in the context of the asset trading use case.

6.1. Asset trading state channel design

A smart contract, which will be called a channel contract, supports the operation of the channel.

6.1.1. Funding

During the establishment of the channel, every participant commits to the channel contract the initial balance that will be used in the channel. Subsequently, the participants transfer their assets to the channel contract, and for every user, an accumulator A_0^x that will eventually contain the assets of the user is initiated according to g and N parameters in the channel contract, as described in Fig. 8.

6.1.2. State updates

There are three main actions regarding the transitions of the accumulators that need to be declared and validated through the state channel update:

- Adding an asset (Fig. 9): A participant can expand the collection of their assets by adding a new asset a_i to their accumulator. Primarily, the user has to transfer the asset to the channel contract. The asset is then owned by the contract, and the user has no way to transfer it to another account. In the channel contract, the asset is mapped to the address of the user. Consequently, the user has to calculate a non-membership witness proof of the element u_{a_i}, as discussed in Section 4.1, that proves that the element is not already part of their

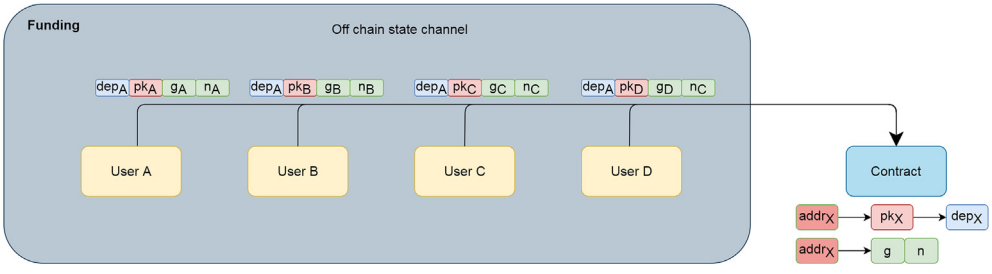

Fig. 8. Funding of an asset trading channel.

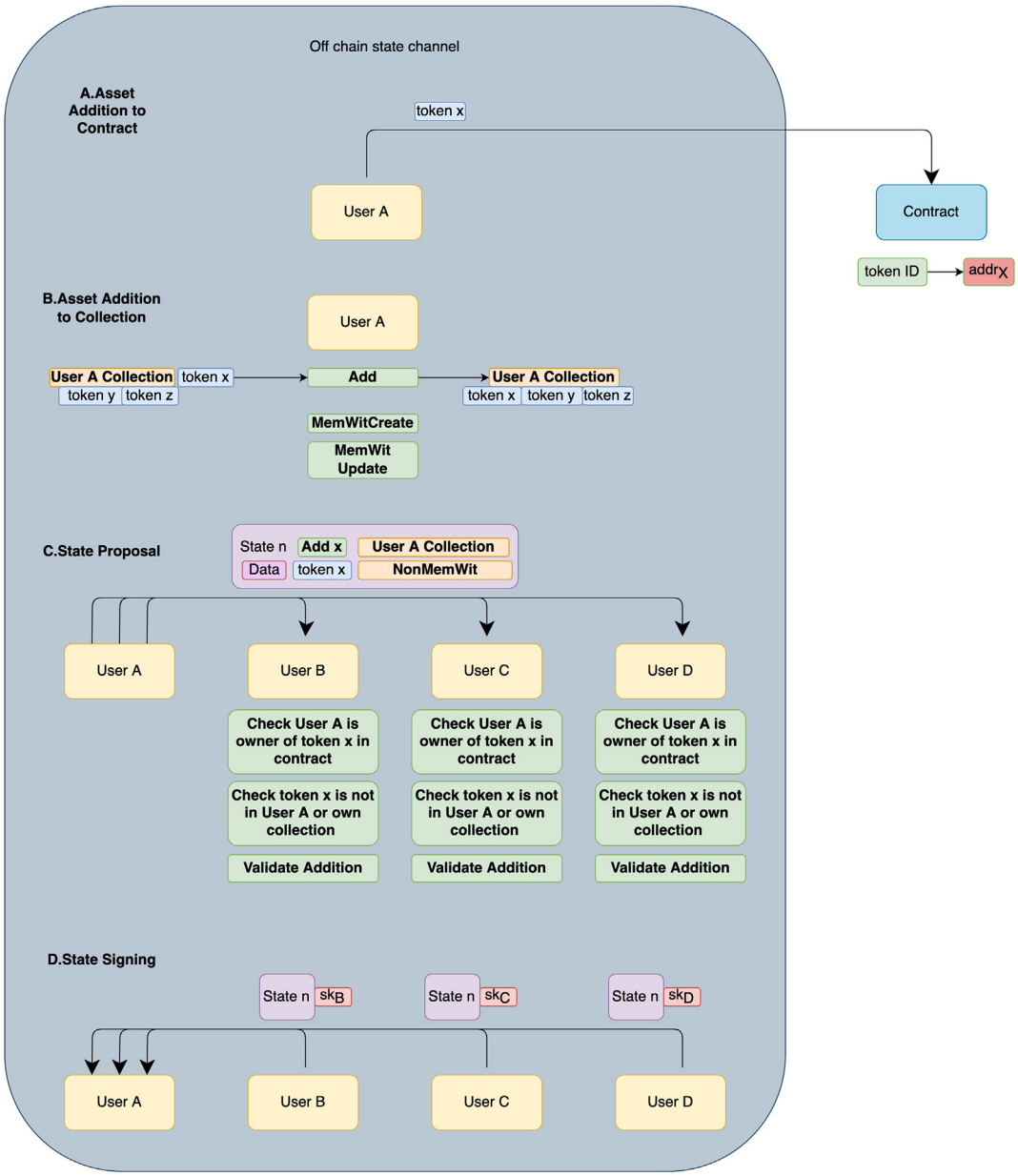

Fig. 9. Adding an asset to a user's collection.

accumulator. Finally, the user updates the accumulator with the new element. The state update proposal is defined as the add declaration, the updated accumulator, and the non-membership proof for the previous accumulator. This is distributed to the group to be signed by every participant.

For every such state update proposal, the participants have to do a number of checks. Specifically, they have to (i) check whether the asset is locked in the channel contract to the account of the user that proposes the state update, (ii) check that the asset is not already included in the accumulator of the user that proposes the state update (by validating the provided non-membership proof), (iii) validate the addition of the element to the accumulator of the state proposing participant (through repeating the addition of the element to the accumulator to check the resulting value), and (iv) check that the asset is not already included in their accumulator. According to the results of those checks, the state transition is validated or a dispute process is initiated.

- Transferring an asset (Fig. 10): A user can transfer an asset to another user through a state update in the state channel. Let us assume that user A wants to transfer asset a_i (that they own at time t) to another user B. The accumulator for user A (A_t^A) includes element a_i, while the accumulator for user B (A_t^B) does not. User A (the one that initially owns the asset) needs to propose a new state that includes the updated accumulator values for users A and B. A_{t+1}^A is calculated by removing a_i and A_{t+1}^B is calculated by adding a_i. The state proposal must also clearly declare which asset is being transferred to which participant. The definitions in Section 4.1 describe the required calculations that burden participant A. Other users, upon receiving the state proposal, have to confirm that A_t^A can be calculated by adding a_i to A_{t+1}^A and that A_{t+1}^B can be calculated by adding a_i to A_t^B. When the state update is validated by every participant, participants A and B can update the membership witnesses of their assets.

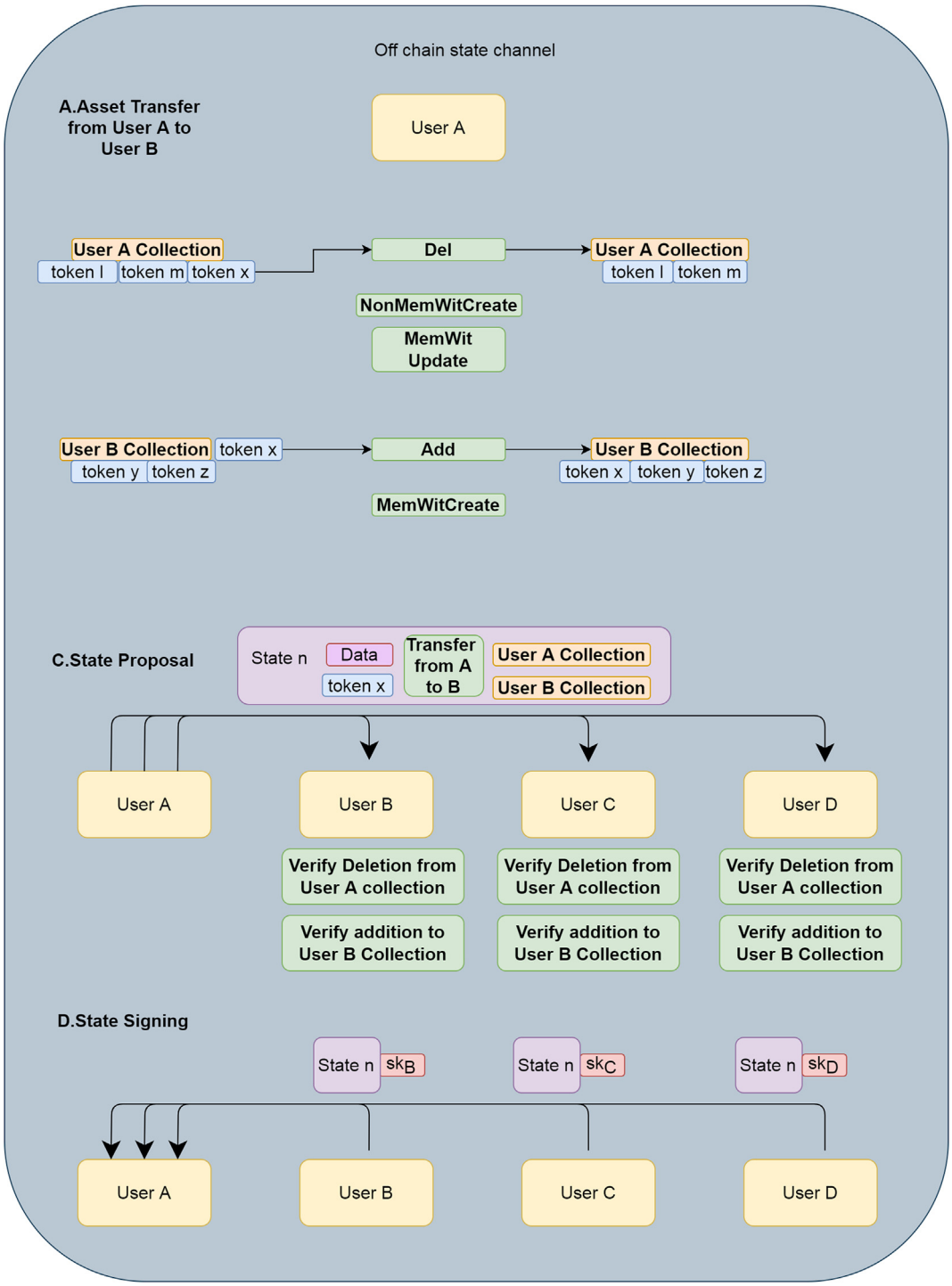

Fig. 10. Transferring an asset from user A to user B.

- Removing an asset (Fig. 11): Let us assume that user A who holds an asset a_i in the channel wants to transfer it out of the channel to exchange it with a user who does not take part in it. The action of removing an element from the state channel requires a state update ($State_{n+1}$) that is proposed by user A to the channel contract (in contrast to other cases where the state update is broadcast to other participants). The specific state update consists of the proposed action to remove a specific element a_i and the updated value of accumulator $A_{t+1_{(A)}}$ for user A. Along with this state update, user A has to send the previous state $State_n$. The channel contract checks that the accumulator value $A_{t+1_{(A)}}$ has been produced by removing element a_i from the accumulator value $A_{t_{(A)}}$. Because deleting an element is a demanding process, the contract verifies the transition by adding element a_i to $A_{t+1_{(A)}}$ and expecting to receive $A_{t_{(A)}}$. If that is the case, it initiates a timeout period during which other participants can submit a valid state more recent than $State_n$ if one exists and terminate the process. In the positive scenario, the element a_i is unlocked in the channel contract, by removing the corresponding record when the asset was initially imported to the channel. Consequently, the asset is transferred from the channel contract to the account of the user on-chain. Regarding the channel, the new valid state is $State_{n+1}$. This state is valid because it is successfully recorded in the channel contract, in

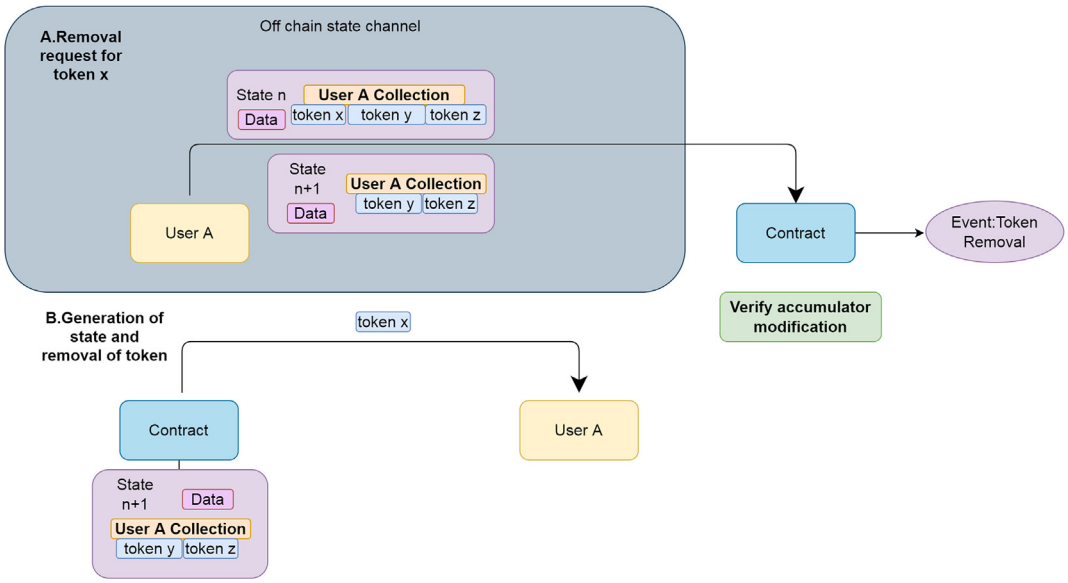

Fig. 11. Removing an asset from the channel.

contrast to all other state updates that become valid because they are signed by all participants.

6.1.3. Dispute

As described in Section 4.2.3 for the general scenario, an inactivity dispute is enough to deal with any kind of malicious behaviour that can be displayed in the channel. This claim, as well as the ways to deal with a dispute in every case, hold true for the specific use case without any necessary modifications. There is, however, one particularity that must be addressed.

The asset trading process involves the initial owner of the token creating a state that trades it to another user. It is not prudent to allow the channel's mechanism to force the second party to agree to that state. Therefore, the dispute process has been slightly modified to allow the option for the second party to not accept the transfer and at the same time be protected from the penalty of an inactivity dispute against them.

Assuming user A is attempting to transfer an asset to user B, and user B does not want to accept that asset or some other part of the transfer state, their only option is not to sign it. This leaves user B vulnerable to an inactivity dispute from user A.

Upon the initiation of the dispute process, user A must provide the contract with the signatures already gathered. If the missing signature is that of user B, who is clearly stated in the state as the receiver of the asset, then the contract does not accept the dispute, and it becomes clear to all channel participants that user A is obligated to provide a different state update. If user A refuses to do so, then any participant can proceed to an inactivity dispute against them.

6.1.4. Closing

The channel can transit to the closing phase either because a participant has optimistically requested that or because a dispute has remained unresolved. At the closing phase of a state channel, there exists a final state that for every participant includes an accumulator holding all assets belonging to them and an indisputably stated balance. The contract no longer accepts state updates and only allows for assets and balance withdrawal according to the final state.

The participants have to make a withdrawal request for each asset at their disposal to the contract that is accompanied by the corresponding membership proof. The channel contract checks the validity of the proof, and if the asset still belongs to the contract, it transfers it to the external account of the user. The participants can also make a withdrawal request for their balances to withdraw any funds they possess in the channel. Fig. 12 depicts the related functionality.

6.2. Use case threat model

The general principles as described in Section 5 do still apply to the use case design. However, the list of possible scenarios has expanded. The additional cases that have emerged need to be addressed, while fine-tuning is necessary for some of the existing ones in order to better serve the asset trading channel. Only the diversions from the general scenario will be addressed in this section.

1 **Invalid transition.** The obvious difference between the general scenario and this use case is that there is no longer cause for multiple members to maintain witnesses for the same element. Therefore, an invalid modification to the accumulator will not be spotted through the realisation that the updated witnesses are invalid.

However, additions in the asset trading channel are accompanied by the on-chain action of adding a token to the contract. Since this is a prerequisite and the validity of the transition can still be checked by replicating the addition, it is guaranteed that any misbehaviour will be spotted.

Deletions require that the state is produced on-chain, and therefore, the topic of an invalid proposal, in this case, does not need to be raised.

In the case of a transfer, there are two parties involved with conflicting motives, and therefore, since assumed rational, they will be checking each other's actions.

In all instances, the means to validate any alteration to the accumulators come through providing all channel members with the means to replicate the process, a function built into the design.

2 **Inactive behaviour.** Inactive behaviour is sufficiently dealt with through the extensively analysed dispute process that takes into account the particularities of the Asset Trading Channel.

3 **Access control violations.** Some restrictions apply in this use case regarding which actions can be performed by which participant. Specifically, each participant can only add an element to their collection, remove an element from their collection or transfer an element they own to another participant. Any state update that does not adhere to those rules (e.g., a participant removes an element from the collection of another participant) is invalid and leads to a dispute.

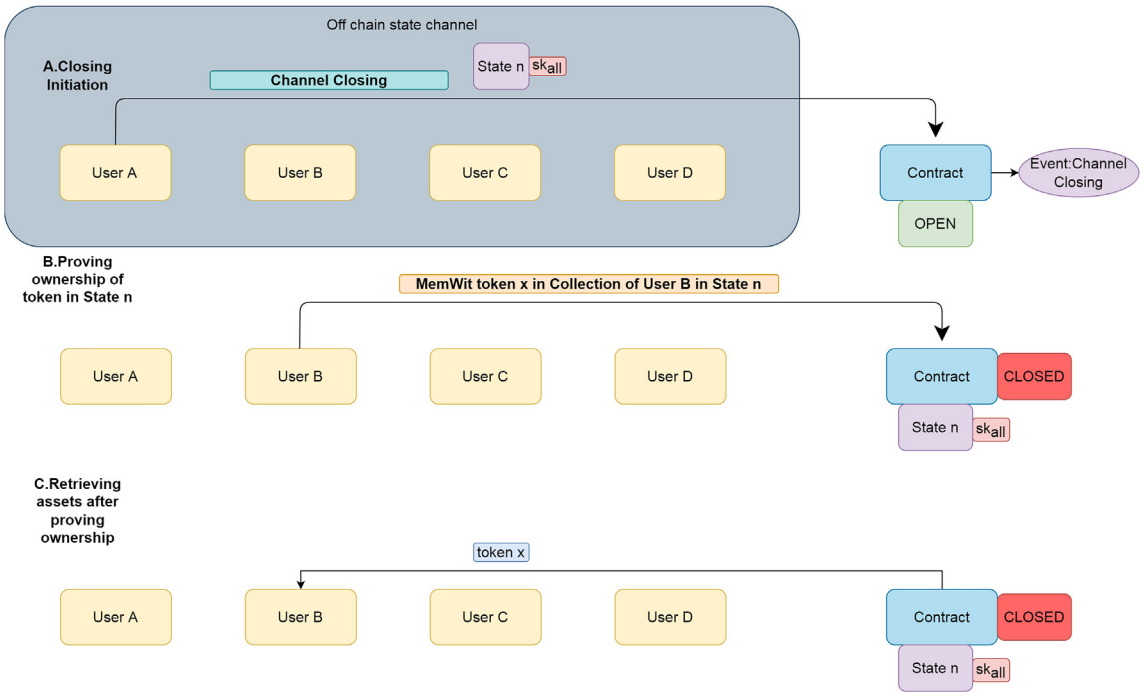

Fig. 12. Cooperative closing of the channel.

6.3. A practical use case example: NFTs exchange market

A concept especially well suited to the asset trading channel as described in Section 6 is that of Non-Fungible Tokens (NFTs).

NFTs have become increasingly popular in recent years and have ignited interest in the spread of crypto-art, the adoption of metaverse-based systems [36], and the support of novel approaches in the gaming industry [37].

NFTs are based on smart contracts that define their functionality and features. These contracts are otherwise referred to as standards, with the most popular and recognizable one being ERC-721 [38]. NFT standards provide the basic outline but also allow for a considerable degree of customisation per implementation.

A typical ERC-721 NFT is comprised of the following parts:

- Token ID: a token identifier unique within the ERC-721 smart contract.
- Metadata: media mapped to the NFT, usually linked through a URI.
- Owner Address: the account address that currently owns the NFT.

Out of those components, the ones relevant to the asset trading channel would be the Owner Address and the Token ID.

Ensuring that the account transferring the token to the channel also owns it is essential. The Owner Address is retained even after the contract becomes the owner of the NFT to regulate in-channel actions. It will also be updated as the token changes hands within the channel and eventually used if it is ever transferred out of it.

As for the Token ID, it is expected that one state channel will be handling several different types of NFTs, coming from several different contracts. Hence, the pair Token ID-contract address is used within the channel as a token's global identifier.

Therefore, an NFT collection can be easily represented by a cryptographic accumulator and an NFT by an element consisting of the two aforementioned components.

7. Comparison with existing schemes

As mentioned in Section 1, the main goal of the proposed protocol is to provide a more efficient approach for state channels that can support applications, the state of which includes boundless lists of elements. Our protocol is designed to enable the required off-chain and on-chain interactions with fixed storage and gas fee requirements regardless of the number of elements that the mentioned lists may hold.

7.1. Storage requirements

Regarding the off-chain management requirements for the participants of the state channel, let us assume that the state of the state channel includes a list of n elements e_i, $i = 1, \ldots, n$. In the traditional state channel design, the list has to be locally maintained by participants. The required size depends on the type of the elements e_i, but in general, it can be expressed as

$$S = n \times size(e_i) \tag{9}$$

Our approach expresses the list of elements with an RSA accumulator. This means that each participant shall locally store an RSA accumulator acc, along with membership proofs $proof$ for all elements in which they are interested (let us assume that they have an interest in m elements). The required storage is:

$$S^{'} = size(acc) + m \times size(proof) \tag{10}$$

The sizes of the accumulator and the proofs are dependent on the security parameters upon which the RSA accumulator is built. The main parameter that determines the storage requirements is the number of elements in which the participants are interested. In cases where m is significantly smaller than n, $m \ll n$ and n is large (the list can potentially grow in length), then $S' \ll S$.

As already explained in Section 3.2.2, constructing an RSA accumulator requires the selection of a modulus N from a group of unknown order that defines the size of the structure. In Fig. 13, the storage requirements of the proposed scheme (representing the collection through an RSA accumulator) are compared to the storage requirements of the baseline approach (holding the actual collection in the state). The comparison depicted has been conducted for a collection of $n = 1000$ elements, and it calculates the required storage for different accumulator

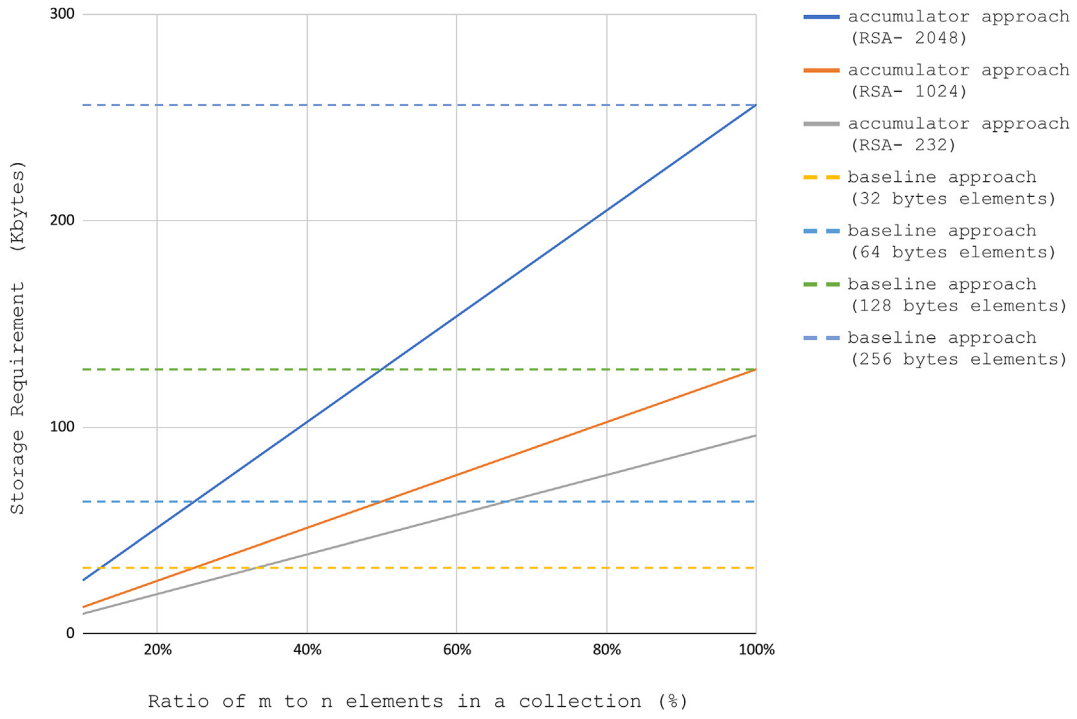

Fig. 13. Comparing the storage requirements for a collection of $n = 1000$ elements.

sizes and different element sizes. The continuous lines show the storage requirement of our approach (for different accumulator sizes), which is proportional to the ratio of the elements of the collection the user is interested in. The dotted lines show the storage requirement of the baseline approach (for different element sizes), which is constant and irrelevant to the ratio of the elements of the collection the user is interested in.

For a given application, parameters such as (a) the security requirements that define the size of the RSA accumulator, (b) the size of the elements, and (c) the ratio m/n have to be studied, in order to understand what is the relation of the storage requirement for the two compared approaches. For example, if an RSA-2048 accumulator is used and the size of each element is 128 bytes, then the storage required by our approach is less than what is required by the baseline approach given that the user is interested in less than approximately 50% of the elements. It is expected that this ratio is going to be much lower in real-world scenarios with multiple users.

7.2. Gas fees

Calculating gas fees for the general case with high precision is impossible. The cost of gas units fluctuates over time, and the number of gas units that a transaction costs depends greatly on its form. In this case, the type of the element, as well as the nature of the application supported by the channel, have a big influence on the total cost of the process.

In a state channel, the need to upload the state to the blockchain and therefore be subject to paying gas fees only arises in the case of a dispute. In such an incident, the contract must be provided with all the necessary information to verify both the validity of the state and the state transition, as described in Section 4.2.3.

Let us initially examine the scenario where the sets of elements are traditionally stored in dynamic lists, arrays, or mappings. To the best of the authors' knowledge, there is no way to provide a contract with a full array, mapping, or any kind of structure if it does not already internally exist in the contract. Therefore, the only way to provide the contract with the state is to recreate the structure it contains by individually supplying every element e_i. Also, since disputes are most often a comparison between two states, in order to check the validity of the transition, two structures would have to be created and compared, essentially doubling the cost of the process. The cost of this process can therefore be expressed as:

$$C = n \times gasfees(e_i) + n' \times gasfees(e_i) \tag{11}$$

For every reoccurring dispute, the contract must be presented with the latest instance of the structure as contained in the freshest state and cannot depend upon any previously-stored form. That is because there is no way to securely communicate to the contract the updates between two states, even if those are valid. Therefore, the structure must be reconstructed in every dispute by adding each individual element.

To produce a comparable value, the scenario has been defined as follows:

- A state is defined as a list of elements, and two consecutive states ($State_n$) and ($State_{n+1}$) are identical but for one less or one more element in ($State_{n+1}$).
- Elements are compared in pairs, with their position in the list as key. The result of this comparison must be **true** in order for the dispute to be valid and move on to the next pair. This process must be repeated for every position but the last of ($State_{n+1}$), therefore, as many times as there are elements in ($State_n$).

The above scenario was implemented in a Solidity smart contract, with each element being represented by a bytes32 variable. Executing the smart contract yields a cost of **43,990 gas units** for every pair of identical elements compared.

It has to be noted that submitting individual elements in this fashion means they are not verified, as typically channel participants sign the entirety of the state and that does not apply to its individual contents. This suggests that the actual cost of the procedure would be even higher as the validation of the submitted elements would have to be incorporated.

In the scenario where the proposed scheme is utilised and the elements are stored in an accumulator, the necessary information to provide to the contract, as described in Section 4.2.3, would be:

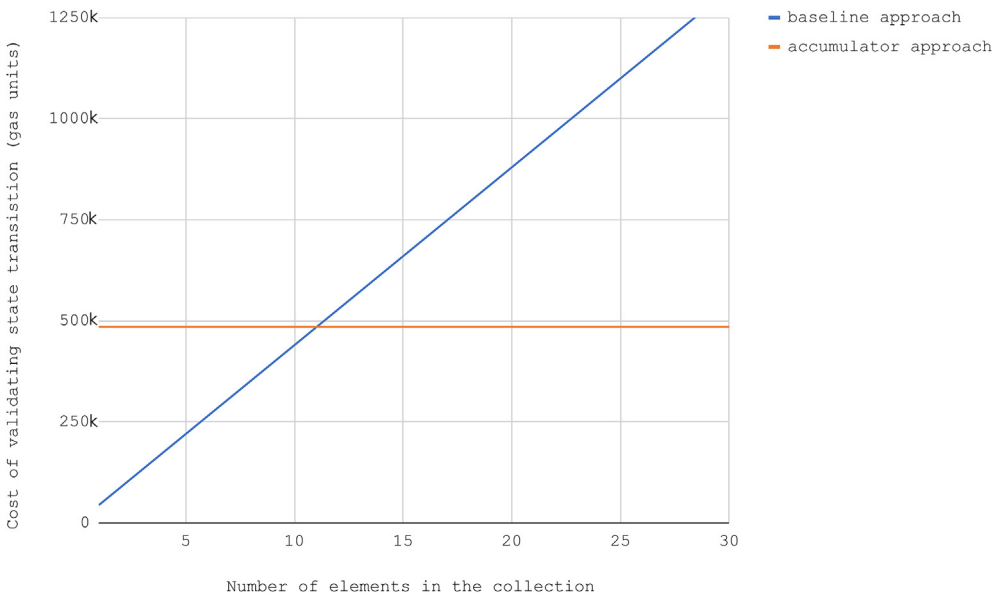

Fig. 14. Comparing the cost of validating state transitions when an unbounded list of elements is presented.

- The cryptographic accumulator of the previous state A_t
- The cryptographic accumulator of the proposed state $A_t^{'}$
- The element that expresses the change between the accumulators e

An additional cost would be the process the contract has to execute to verify the proof against the provided accumulators. Therefore, the total fees can be expressed as:

$$C^{'} = gasfees(A_t) + gasfees(A_t^{'}) + gasfees(e) + gasfees(verify) \quad (12)$$

While the cost accumulates multiple different gas fee values, it has to be noted that all such values are independent of the size of the set, and hence, the gas fees remain constant as the number of elements increases. It is obvious that given the fact that boundless lists have to be supported, the proposed scheme is more efficient than any traditional approach where the cost is strongly dependent on the size of the list.

To verify this expectation, a simulation of a dispute was realised for the proposed design based on the smart contract implementation provided by Ref. [39]. The cost of verifying the transition from the accumulator representing ($State_n$) to the one representing ($State_{n+1}$) comes up to **484,998 gas units**. This process has to be performed only once per dispute, and the cost remains static, regardless of the number of elements contained within the accumulators.

It becomes obvious through Fig. 14 that upon handling sets that surpass 11 elements, the proposed scheme is less expensive, and the difference in cost escalates as the size of the set grows.

8. Conclusions

Public blockchain networks are heavily criticized for low transaction rates, and one of the approaches to indirectly improve their capacity is state channels. In the present paper, a novel state channel design that can efficiently accommodate unbounded sets of elements in the state of the channels has been proposed. The design uses RSA accumulators to store commitments regarding elements' membership in specific sets.

We have refined the existing state channel approach to make all different operations in a state channel compatible with the modified state structure. We have discussed the security of the proposed design and elaborated on how this could be applied to a specific use case related to an asset exchange state channel.

We recognise the limitations of the design in its current form as follows:

- Due to the use of the RSA accumulator, some level of trust is required since a participant will be responsible for picking an RSA modulus $N = pq$. However, this can be avoided through the use of class groups of imaginary quadratic order [31] for the determination of N, and this does not affect the design's other functionalities.
- We have identified a type of scenario, described in Section 4.3, where the use of non-membership proofs cannot be avoided. In such cases, it falls upon the developers of the application to decide if it is more beneficial to use our design with non-membership proofs or to store their data in the traditional manner.
- The usefulness of our design applies to state channels where the form of state includes dynamic sets of unordered elements and is proportional to the size of the set, as described in Section 7.

While the main reason behind the fact that public blockchain platforms are inefficient is their low transaction processing rate, it is also evident that we tend to use such platforms in a way that is not sufficiently optimised. The main concept of deploying an application on a public blockchain system is that the integrity of each interaction of the users with the application is guaranteed. State channels' early research proposals have indicated that it is feasible to enjoy the same guarantees with a substantial decrease in the required on-chain activity by the users. The application is mainly run upon off-chain interaction between users. At any point in the lifetime of the application, users are in possession of the required cryptographic commitments that will enable them to go on-chain and achieve the same results as they would do if the application was completely run on-chain.

Through the present paper, we have taken a step forward with regard to state channels and proposed a scheme that can support states of large scale through a representation of set membership through RSA accumulators. It has been shown that for applications that adhere to the specific need (unbounded sets of elements), our approach can provide an efficient alternative that scales up well with regards to the number of elements in the set, as all on-chain and off-chain activity of the channel is irrelevant to that number.

We have provided a security analysis of the proposed protocol, but it would be beneficial to attempt to define the protocol's security goals formally through the use of the Universal Composability Framework [40, 41]. This will enable a thorough analysis of the proposed approach and

will better highlight its advantages and weaknesses. As part of future work for the present paper, we aim at providing such an analysis for the proposed protocol or for any revised version we come up with in a future paper.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] T. Clohessy, T. Acton, N. Rogers, Blockchain adoption: technological, organisational and environmental considerations, in: H. Treiblmaier, R. Beck (Eds.), Business Transformation through Blockchain, Palgrave Macmillan, Cham, 2019, pp. 47–76, https://doi.org/10.1007/978-3-319-98911-2_2.

[2] F.A. Alabdulwahhab, Web 3.0: the decentralized web blockchain networks and protocol innovation, in: Proceedings of the 2018 1st International Conference on Computer Applications Information Security (ICCAIS), IEEE, 2018, pp. 1–4, https://doi.org/10.1109/CAIS.2018.8441990.

[3] J. Xie, F.R. Yu, T. Huang, R. Xie, J. Liu, Y. Liu, A survey on the scalability of blockchain systems, IEEE Network 33 (5) (2019) 166–173, https://doi.org/10.1109/MNET.001.1800290.

[4] Q. Zhou, H. Huang, Z. Zheng, J. Bian, Solutions to scalability of blockchain: a survey, IEEE Access 8 (2020) 16440–16455, https://doi.org/10.1109/ACCESS.2020.2967218.

[5] L.D. Negka, G.P. Spathoulas, Blockchain state channels: a state of the art, IEEE Access 9 (2021) 160277–160298, https://doi.org/10.1109/ACCESS.2021.3131419.

[6] T. Close, A. Stewart, Forcemove: an N-Party State Channel Protocol, 2018. https://magmo.com/force-move-games.pdf. (Accessed 27 November 2021).

[7] J. Coleman, L. Horne, L. Xuanji, Counterfactual: Generalized State Channels, 2018 https://l4.ventures/papers/statechannels.pdf. (Accessed 4 November 2019).

[8] C. Buckland, P. McCorry, Two-party state channels with assertions, in: A. Bracciali, J. Clark, F. Pintore, P. Rønne, M. Sala (Eds.), Financial Cryptography Workshops, Springer, Cham, 2019, pp. 3–11, https://doi.org/10.1007/978-3-030-43725-1_1.

[9] D. Boneh, B. Bünz, B. Fisch, Batching techniques for accumulators with applications to iops and stateless blockchains, in: A. Boldyreva, D. Micciancio (Eds.), Advances in Cryptology – CRYPTO 2019, Springer, Cham, 2019, pp. 561–586, https://doi.org/10.1007/978-3-030-26948-7_20.

[10] I. Miers, C. Garman, M. Green, A.D. Rubin, Zerocoin: anonymous distributed e-cash from bitcoin, in: Proceedings of 2013 IEEE Symposium on Security and Privacy, IEEE, 2013, pp. 397–411, https://doi.org/10.1109/SP.2013.34.

[11] L. Xu, L. Chen, Z. Gao, S. Xu, W. Shi, Epbc: efficient public blockchain client for lightweight users, in: Proceedings of the 1st Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers, ACM, 2017, pp. 1–6, https://doi.org/10.1145/3152824.3152825.

[12] T. Dryja, Utreexo: a dynamic hash-based accumulator optimized for the bitcoin utxo set, Cryptology ePrint Archive, Paper 2019/611, https://eprint.iacr.org/2019/611, 2019.

[13] K. Wijburg, The state of statechains : exploring statechain improvements, February, http://essay.utwente.nl/80780/, 2020.

[14] S. Bouraga, A taxonomy of blockchain consensus protocols: a survey and classification framework, Expert Syst. Appl. 168 (2021) 114384, https://doi.org/10.1016/j.eswa.2020.114384.

[15] S.M.H. Bamakan, A. Motavali, A.B. Bondarti, A survey of blockchain consensus algorithms performance evaluation criteria, Expert Syst. Appl. 154 (2020) 113385, https://doi.org/10.1016/j.eswa.2020.113385.

[16] G. Wang, Z.J. Shi, M. Nixon, S. Han, Sok: sharding on blockchain, in: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, ACM, 2019, pp. 41–61, https://doi.org/10.1145/3318041.3355457.

[17] H. Dang, T.T.A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, B.C. Ooi, Towards scaling blockchain systems via sharding, in: Proceedings of the 2019 International Conference on Management of Data, ACM, 2019, pp. 123–140, https://doi.org/10.1145/3299869.3319889.

[18] A. Singh, K. Click, R.M. Parizi, Q. Zhang, A. Dehghantanha, K.-K.R. Choo, Sidechain technologies in blockchain networks: an examination and state-of-the-art review, J. Netw. Comput. Appl. 149 (2020) 102471, https://doi.org/10.1016/j.jnca.2019.102471.

[19] J. Poon, V. Buterin, Plasma: Scalable Autonomous Smart Contracts, 2017. White paper, https://www.plasma.io/plasma-deprecated.pdf. (Accessed 11 August 2017).

[20] Layer 2 rollups. https://ethereum.org/en/developers/docs/scaling/layer-2-rollups/, 2021. (Accessed 27 November 2021).

[21] Payment Channels, 2021. https://en.bitcoin.it/wiki/Payment_channels. (Accessed 27 November 2021).

[22] J. Coleman, State Channels, 2015. https://www.jeffcoleman.ca/state-channels/. (Accessed 27 November 2021).

[23] A. Kumar, P. Lafourcade, C. Lauradoux, Performances of cryptographic accumulators, in: Proceedings of the 39th Annual IEEE Conference on Local Computer Networks, IEEE, 2014, pp. 366–369, https://doi.org/10.1109/LCN.2014.6925793.

[24] J. Benaloh, M. de Mare, One-way accumulators: A decentralized alternative to digital sinatures (extended abstract), in: T. Helleseth (Ed.), EUROCRYPT, Springer, Berlin, Heidelberg, 1993, https://doi.org/10.1007/978-3-642-31284-7_14.

[25] I. Ozcelik, S. Medury, J. Broaddus, A. Skjellum, An Overview of Cryptographic Accumulators, 2021 arXiv preprint arXiv:2103.04330.

[26] J. Li, N. Li, R. Xue, Universal accumulators with efficient nonmembership proofs, in: Proceedings of the 5th International Conference on Applied Cryptography and Network Security, ACNS '07, Springer, Berlin, Heidelberg, 2007, pp. 253–269, https://doi.org/10.1007/978-3-540-72738-5_17.

[27] H. Lipmaa, Secure Accumulators from Euclidean Rings without Trusted Setup, in: F. Bao, P. Samarati, J. Zhou (Eds.), ACNS, Berlin, Heidelberg, 2012, pp. 224–240, https://doi.org/10.1007/978-3-642-31284-7_14.

[28] B. Wesolowski, Efficient verifiable delay functions 11478 (2019) 379–407, https://doi.org/10.1007/978-3-030-17659-4_13.

[29] S. Dobson, S. Galbraith, B. Smith, Trustless unknown-order groups, Mathematical Cryptology 1 (2) (2022) 25–39. https://journals.flvc.org/mathcryptology/article/view/130579.

[30] F. Baldimtsi, J. Camenisch, M. Dubovitskaya, A. Lysyanskaya, L. Reyzin, K. Samelin, S. Yakoubov, Accumulators with Applications to Anonymity-Preserving Revocation, 2017, https://doi.org/10.1109/EuroSP.2017.13.

[31] J. Buchmann, S. Hamdy, A survey on iq cryptography, in: Proceedings of Public Key Cryptography and Computational Number Theory, 2001, pp. 1–15, https://doi.org/10.1515/9783110881035.

[32] B. Bünz, B. Fisch, A. Szepieniec, Transparent snarks from dark compilers, in: A. Canteaut, Y. Ishai (Eds.), Advances in Cryptology – EUROCRYPT 2020, Springer, Cham, 2020, pp. 677–706, https://doi.org/10.1007/978-3-030-45721-1_24.

[33] K. Belabas, T. Kleinjung, A. Sanso, B. Wesolowski, A Note on the Low Order Assumption in Class Group of an Imaginary Quadratic Number Fields, Cryptology ePrint Archive, 2020. Paper 2020/1310, https://eprint.iacr.org/2020/1310.

[34] I.A. Seres, P. Burcsi, A Note on Low Order Assumptions in Rsa Groups, Cryptology ePrint Archive, 2020. Paper 2020/402, https://eprint.iacr.org/2020/402.

[35] K. Pietrzak, Simple verifiable delay functions, in: A. Blum (Ed.), 10th Innovations in Theoretical Computer Science Conference (ITCS 2019), Vol. 124 of Leibniz International Proceedings in Informatics (LIPIcs), Schloss Dagstuhl–Leibniz-Zentrum Fuer Informatik, 2018, pp. 60:1–60:15, https://doi.org/10.4230/LIPIcs.ITCS.2019.60. Dagstuhl, Germany.

[36] R. Belk, M. Humayun, M. Brouard, Money, possessions, and ownership in the metaverse: Nfts, cryptocurrencies, web3 and wild markets, J. Bus. Res. 153 (2022) 198–205, https://doi.org/10.1016/j.jbusres.2022.08.031.

[37] D. Vidal-Tomás, The new crypto niche: Nfts, play-to-earn, and metaverse tokens, Finance Res. Lett. 47 (2022) 102742, https://doi.org/10.1016/j.frl.2022.102742.

[38] J.E.N.S. William Entriken, Dieter Shirley, RSA-Accumulator on Solidity, 2018. https://eips.ethereum.org/EIPS/eip-721. (Accessed 27 September 2022).

[39] oleiba, RSA-Accumulator on solidity. https://github.com/oleiba/RSA-accumulator/blob/master/contracts/RSAAccumulator.sol, 2019. (Accessed 27 September 2022).

[40] R. Canetti, Universally Composable Security: A New Paradigm for Cryptographic Protocols, Cryptology ePrint Archive, 2000. Paper 2000/067, https://eprint.iacr.org/2000/067.

[41] R. Canetti, A. Stoughton, M. Varia, Easyuc: Using Easycrypt to Mechanize Proofs of Universally Composable Security, Cryptology ePrint Archive, 2019. Paper 2019/582, https://eprint.iacr.org/2019/582.