Roma Tre University
Ph.D. in Computer Science and Engineering

State Channels for Blockchain Scalability: Capabilities, Limitations, Perspectives

Federico Spini

State Channels for Blockchain Scalability: Capabilities, Limitations, Perspectives

A thesis presented by
Federico Spini
in partial fulfillment of the requirements for the degree of
Doctor of Philosophy
in Computer Science and Engineering

Roma Tre University
Dept. of Informatics and Automation

October 2018

Committee:
Prof. Alberto Paoluzzi

Reviewers:
Prof. Dr. Thomas Bocek
Dr. Arthur Gervais

Nothing is immutable
except the need for change.
—Heraclitus

State Channels for Blockchain Scalability: Capabilities, Limitations, Perspectives

Abstract: Proof-of-work blockchain, as originally conceived by the pseudonymous creator of the Bitcoin protocol Satoshi Nakamoto, does not scale. Three different approaches are been explored to make the blockchain scale: 1. data sharding; 2. alternative consensus mechanisms; 3. off-chain solutions. This work contributes to the last approach by focusing on state channels. A state channel is a two-party ledger, unanimously updated, that resorts to the blockchain to resolve possible disputes (lack of unanimous consensus) that may arise during the off-chain interaction between channel endpoints. Transactions stored in the two-party ledger are unloaded from the underling public blockchain where only a succinct summary of the off-chain interaction is finally stored. This work focuses on state channels defined by means of smart contracts supported by a Turing-complete language (as offered for example by the Ethereum platform) and presents five main contributions: 1. the definition of the *propose/accept* scheme as a formalization of the off-chain interaction between channel endpoints; 2. the introduction of *Inextinguishable Payment Channels*, a protocol to allow *hot-refill* of and *hot-withdrawal* from a running payment channel thus preventing a skewed channel to be closed if involved parties still need to use it; 3. the introduction of *Fulgur*, a hybrid trustless wallet supported by a *FulguHub*, a centralized trustless payment hub; 4. the definition of μ and $\bar{\mu}$ agreement classes, a categorization that reflects the inherent characteristic of an agreement (implemented as a smart contract or as a state channel) to be resolved at any time with or without satisfaction for involved parties; 5. the introduction of *Smart Channel*, a protocol that equalizes performance (in terms of number of required transactions) between $\bar{\mu}$-agreements implemented as state channels and standard on-chain interaction (*i.e.*, directly intermediated by a smart contract), in case of irrational attacks, and maintains privacy for involved parties in case of perfect cooperation.

Keywords: blockchain, off-chain solutions, state channels, micropayment channels, inextinguishable payment channels, smart channels, fulgur

Contents

Chapter 1

Introduction

The blockchain technology realizes a distributed ledger firstly introduced to support Bitcoin, a peer-to-peer digital payment system. The blockchain stores value transactions among participants of the network and leverages economic friction and incentives to ensure security and correct system operation.

Due to its strongly multidisciplinary character it attracted the attention of many research fields: networking and distributed systems, cryptography, game theory. The way its deployment can influence daily life of people in the mid/long term involves economical, legal, political and sociological aspects.

Despite the increasing interest, blockchain remains a technology in its early stage of development. Nevertheless, it proved itself to work in practice, being able to overcome, at least until now, bans by national governments and central banks: the use of Bitcoin has been officially prohibited in several countries (*e.g.* China, Russia, Thailand, Vietnam, Taiwan, Colombia, Ecuador, Bolivia, Bangladesh, Kyrgyzstan) since its permissionless and distributed nature makes it difficult to impose regulatory compounds as a government may want to do on a currency that circulates on the territory of its jurisdiction.

Micropayment channels have been introduced in the Bitcoin ecosystem to tackle with the limited scalability offered by the blockchain technology. Replication of distributed ledger is supported by a synchronization mechanism that limits the maximum rate of transaction the network can process, making difficult to scale up to the hundreds of thousands of transactions per second required to support the peak of world-wide fiat money transactions.

Micropayment channels allow the two involved parties, the endpoints of the channel, to move funds back and forth, up to the amount blocked at the beginning of the interaction. Transactions that occur in the channel remain private to the participants and are not broadcast to the blockchain. Only a final settlement is required to reconcile the off-chain transaction history with the blockchain. The blockchain is therefore unloaded from the burden of all the transactions that take place on a channel.

Smart contracts leverage the blockchain technology to allow for the execution of arbitrary code that conveniently can manipulate account balances according to easily definable custom logic. The Ethereum platform has been the first implementation of such a technology. It broaden the possibilities offered by the blockchain and open the door for new applications that 1) require value exchange subjected to specific conditions, 2) rely on the trustless execution of critical fragment of code where

correctness of the output is guaranteed, 3) need the integrity protected storage and/or notarization of (small amount of) data, 4) a mix of the previous.

While Bitcoin's payment channels only support transactions that alter balance of the involved parties, enhanced expressivity of smart contracts enables state channels to handle an arbitrary data structure. Payment channels defined on top of smart contracts as a state channels (payment state channels) are more flexible and easy to handle than original micropayment channels introduced for the Bitcoin blockchain. They enable new possibilities to be explored and are the focus of part of this work.

Question 1 *How to prevent a skewed payment channel from being closed and reopened if the off-chain interaction between parties has still reason to exists?*

When payments are sent mainly in one direction, the party that pays more frequently may run out of balance and therefore being obliged to close the channel and reopen it. Channel open and close procedures require time and involve transaction fee costs. To this end, in Chapter 4 it is introduced the detach/attach interaction scheme that allows for moving funds to and from a payment state channel with no alteration for the regular channel activity. By avoiding the unnecessary closing of a channel, the underling blockchain is further relieved from closing/reopening on-chain transactions.

Question 2 *How to maximize utility of funds locked in a channel?*

Involved parties of a payment channel actually pays the opportunity cost of blocked collateral funds. This cost is reduced if funds locked in a channel can be used to several purposes instead of only support payments to and from the counterparty of the channel.

Payment networks enable this possibility by defining a mechanism for multi-hop payments. Once a payment route has been identified with enough capacity on each intermediate hop, the mechanism allows to atomically enforce payment execution. However this solution, to be actually viable, requires a network of payment channels to be deployed. Furthermore at least one route in between the payer and the payee with enough capacity to accommodate the payment amount must exist, condition ensured by a well connected network. Beside the development stage of the software needed to support payment network solutions, the creation of a well connected network that ensure connectivity among every source and destination point may take long time.

In Chapter 5 an extend version of the detach/attach scheme is presented to support a fast deployable solution: a hub-and-spokes architecture for hybrid payments. Off-chain payments toward hub-connected nodes are routed through the central hub while off-chain balances can be used as source and destination of funds for outbound and inbound payments where the counterparty is a standard on-chain account of the network.

Question 3 *Can state channels support any kind of agreement?*

To enter in a channel with a counterparty implies to take some commitments and therefore the existence of a mutual agreement between parties. The agreement can be explicit and simple to elicit and understand, like in the case of payments, or implicit.

In Chapter 6 is presented an analysis of state channels in relation to the type of agreement they require or imply. The analysis results in a partition of the space of agreements based on the existence of a mapping function between any intermediary state and a satisfying closing one. Furthermore, it turned out that under an irrational passive aggressive attack, a specific class of "channelized" agreement underperforms with respect to the standard on-chain interaction, nullifying the benefit of the off-chain approach. Smart channels are an extension to the state channel construction proposed to level on-chain and off-chain performances in such a case.

The remainder of this work is organized as follows. Chapter 2 provides the necessary background on blockchain and smart contracts. Chapter 3 introduces the state channel construction. Chapter 4, Chapter 5 and Chapter 6 answer research questions 1, 2 and 3, respectively. Finally, Chapter 7 contains conclusive remarks.

Chapter 2

Background on blockchain and smart contracts

Contents

This chapter provides the background on blockchain and smart contract execution platforms required to understand the reminder of this work.

Background on blockchain is provided according to a historical perspective. The reader that wanted to access low level details of the protocol and the ecosystem developed around the blockchain technology, is referred to documentation available on the Bitcoin website [1] and to extensive and detailed works such as [2, 3, 4, 5, 6].

2.1 Principles of operation

An overview of the mechanisms supporting the distributed permissionless ledger is presented with reference to the first blockchain introduced, the Bitcoin one, whose purpose was to support a peer-to-peer electronic cash system.

The double spending problem The critical problem an electronic cash system has to addresses is the avoidance of the "double spending" of digital coins. Since electronic cash is represented as informative units (string of bytes) they can be easily copied, as it can be done for any digital asset, *e.g.* digital pictures or videos.

The blockchain solution The solution offered by the blockchain protocol consists in providing every node of the peer-to-peer system with a copy of the whole transaction history. The recipient of a payment can therefore check the ledger to verify that received coins have not been sent to any other, or double spent. The blockchain solution is acknowledged to expose a peculiar characteristic: permissionlessness. A node is free to join and participate in the network, neither authorization nor identification process is required.

2.1.1 Bitcoin and the UTXOs model

Bitcoin's blockchain (alike any other introduced afterwards) strongly relies on asymmetric cryptography. The pseudo-identity of a node is associated with his/her pair of private/public keys. A payments encompasses an unlock/lock procedure. Payer unlocks his/her own coins and re-locks them so that only the receiver is subsequently able to unlock them. Although several complicated and powerful payment structures have been introduced over time, for simple payments, the locking part involve a derivation of the public key of the receiver called "address", while the unlocking part involves a digital signature that attests the ownership of the private key related to the public one used to lock the coin.

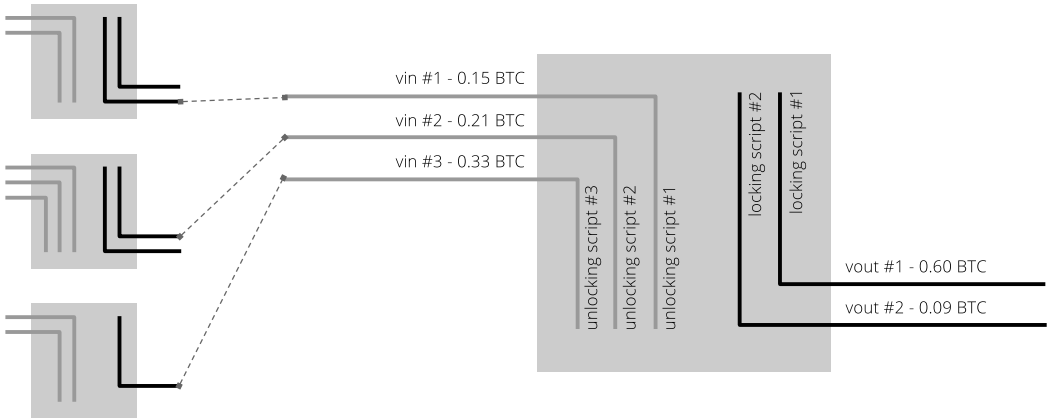

Figure 2.1: Bitcoin transaction.

Transaction structure Figure 2.1 show the anatomy of a very simple Bitcoin transaction. It allows for many inputs and many outputs. Locking and unlocking script are expressed in a language called *Script*, which is a Forth-like reverse-polish notation stack-based language. Every input corresponds to the locked output of a previous transaction. Every node can independently verify that an input is rightfully used by the transaction by concatenating the unlocking script with the locking one relative to the previous output that has to be unlocked. The resulting code string has then to be executed. A transaction is deemed valid if all the unlocking scripts correctly unlock the output they depend on. Once an input is unlocked the freed

value is available to be assigned to one or more outputs of the transaction. Each output is finally locked in such a way that only the receiver of the payment can unlock it. The sum of the value of unlocked inputs is usually greater then the outputs locked one. The difference constitutes a transaction fee.

According to this model, coins are represented by unspent transaction outputs (UTXOs). The balance of an account correspond to the sum of all the unspent transaction outputs that can be successfully unlocked and therefore spent. A client may (and is suggested to) own several private keys to which the owned UTXOs are locked to (possibly derived from a single master one according to a scheme called Hierarchical Deterministic Wallet[1]). A Bitcoin wallet is the software that stores and manages all the private keys owned by client.

Transaction replication Once a transaction is ready, *i.e.,* it has been provided with the needed signatures to unlock the inputs and each outputs has been secured to be unlocked only by the receiver, it is broadcast to the peer-to-peer network. Peers receive the transaction and check its validity. If the transaction results patently invalid it is ignored. Instead, if it is well-formed and not manifestly invalid, it is temporarily stored in a special structure called "mempool", waiting to be confirmed, or "ordered".

Transactions order Although every node owns a whole copy of all transactions ever made, the double spending issue is not resolved. The missing piece is the imposition of a order on transactions. It is a required condition to resolve the case when an entity create two different transactions that consume that very same unspent output but lock with different locking script.

The imposition of a total order on transactions resolve the situation dictating that only the first one can rightfully consume the output. The second one is therefore rejected as invalid.

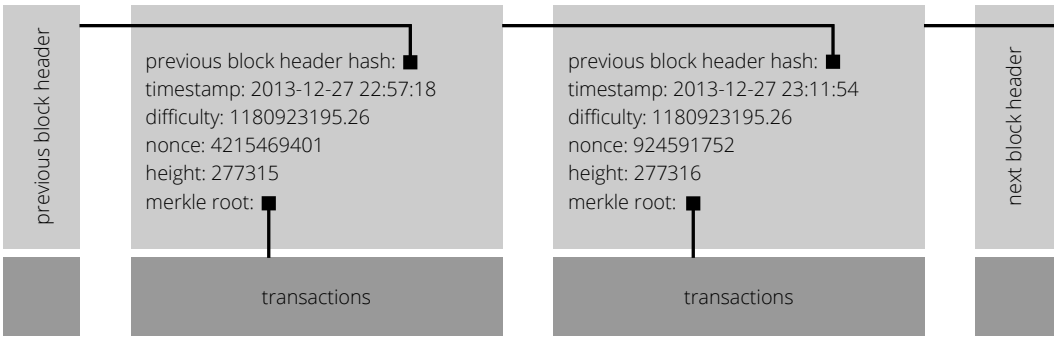

Figure 2.2: Bitcoin blockchain.

[1] https://en.bitcoin.it/wiki/Deterministic_wallet

The Bitcoin's blockchain achieves he imposition of this order in a permissionless way. Miners are the special players that accomplishes this task. Miners collect and order transactions in a data structure called *block*. Each block is unambiguously linked to the previous one, realizing a chain of blocks. The system is self-regulating in such a way that at least one miner is statistically able to produce a block every ten minutes.

Figure 2.2 exemplifies the blockchain data structure. A block is constituted by the transactions it contains and by a block header.

Every miner can autonomously decide which transactions to include in a block, and in which order. A block, to be accepted by all the other nodes, must show the solution of cryptographic puzzle. When a miner "close" a block, it can collect the transaction fees of all the transaction it decides to include in the block and it is also credited with a block reward, *i.e.,* a fixed amount of coins. Therefore, it is strongly preferable for a miner to not include invalid transactions inside a block since it would be rejected by other nodes for this reason, and the miner would lose block reward and transaction fees it could have earned instead.

Blocks confirmation The solution of the cryptographic puzzle that seals the block is found as result of a useless computation: the application of the SHA256 hashing function to the block header. The `nonce` field of the block header can be altered by the miner to the purpose of obtaining a hash value below a fixed threshold[2] (which the protocol adapts dynamically to maintain the block emission rate as constant as possible: one every ten minutes). The output of the hashing function is not predictable, the mechanism resembles a lottery drowning. A miner, to take advantage over other miners can only buy more lottery tickets, which means, out of metaphor, to increase the its hashing rate.

Forks A miner that resolves the puzzle, broadcast the block along with the found solution. A fork of the chain may happen when two competing miners found a solution for a block of the same height (a block that links back to the same previous one). The protocol states that the longest chain is the correct one, therefore the actual transaction order depends on which competing block the next successful miners decide to link back to new ones.

Permissionlessness The right to order transaction is granted to a pseudonym identity that is able to show to have enough computation power to resolve the puzzle before others. This mechanism is called "Proof of Work" (PoW). The system self-adjust the complexity of the puzzle in such a way that the total hashing power of the network is able to find a puzzle solution about once in ten minute. Nowadays difficulty is so high that a miner has to rely on special high electric consumption hardware to have reasonable probability to succeed in the generation of a proof of

[2]The nonce field is only 32 bytes long, therefore miners usually exploit also other (parts of) fields that are free to be altered since they do not contain significant information

work. Nevertheless, the production of a proof of work is the only action required to be entitled to confirm a block. No further authorizations are needed. In this sense the system is considered to be permissionless: an entity that wants to participate in the mining process, and therefore earns the reward and collect fees, only has to succeed in the generation of proof of work before competing miners.

Incentive scheme Security of the system is provided by the economic incentive scheme behind the construction. To participate in the mining process requires an investment of fiat money: the want-to-be miner has to buy the ad hoc hardware and pay the electric bill to run it. The return of investment consist of block reward and transaction fees it collects when it confirms a block. To attack the network would imply a deprecation of the coin value (with respect to fiat money) that would make the investment unprofitable.

Emerging consensus Having enough computational power the whole history can be rewritten. Immutability of past transaction order is guaranteed by the difficulty of the puzzle. To alter the history would imply for a miner to found several solutions to the puzzle to achieve the longest chain, starting from the forking block. Honest miners, however, keep doing their job, being incentivized by the profit the could earn and therefore the chain of blocks keep growing. As long as the hashing power is enough equally distributed among several independent miners, the probability that a fork successfully become the longest chain decreases as the distance of the forking point to the tip of the chain increases. It means that old blocks are more immutable than emerging ones, that may possibly change. Blocks, and therefore transactions order, "crystallize" with the appending of new blocks to the tip of the chain. This mechanism defines a type of consensus that emerges with the passing of time.

2.1.2 Ethereum and smart contract advent

Nowadays there are hundreds of blockchains deployed. Many of them are very similar to the Bitcoin's one (even sharing the same codebase). Others, instead, introduced interesting innovations. One remarkable breakthrough in the field has been brought by Ethereum [7]. It realizes a platform to the self-enforcing execution of smart contracts. A smart contract, in this context, is a piece of source code written in a Turing complete language[3]. Several high level languages has been introduced to specify contract code. One of the most used is called Solidity[4]. Its syntax is similar to that of JavaScript, and probably from this derives its widely adoption.

[3] Although the instruction set is actually Turing-complete, the gas-based execution model adopted by the Ethereum platform imposes tight constraints both on storage and computation capabilities. Those constraints establish a practical obstruction to the execution of heavy programs leaving open the question on the effective Turing-completeness of smart contracts.

[4] http://solidity.readthedocs.io/en/develop/index.html

Architecture The overall architecture is similar to the Bitcoin one: a peer-to-peer network supports transaction broadcast and an incentive-driven mining mechanism secures the system[5]. Although analogously stored in a blockchain, transactions have, instead, a completely different structure and purpose.

Account model As opposed to the UTXO model of Bitcoin, Ethereum is based on the so-called account model. A balance tracks the amount of coins owned by an account, which is represented by an address related to a private key used to sign transactions. A value transaction, whose execution is atomically enforced, has the effect of decrease the balance of the sender and increase the balance of the receiver.

Smart contract lifecycle Smart contract are the code unit whose correct execution is guaranteed by the platform. If written using a high level language like Solidity, a smart contract offers a constructor and several methods. In a parallel with oject oriented programming, a smart contract is a class that has to and can be instantiated several times. The instantiation process consists in deploying contract code on the blockchain, such that every peer knows about its existence. As for an instantiated object, a deployed contract gains the possibility to store a state described by a data structure defined inside the contract itself. Merkle Patricia Tree[6] is the fully deterministic cryptographically authenticated data structure that supports the data storage in the form of `(key, value)` pairs.

Methods of a deployed contract can be invoked. Methods invocation allows for arguments to be passed, according to method signature. An Ethereum transaction serves the purpose of deploying a contract or invoking one of its methods, passing in invocation parameters, and possibly taking some coins to the contract. Value transactions for account model can be thought of as the invocation of method of a predefined contract that only moves coins from one account to another, updating relative balances.

Execution model Solidity code is required to be compiled into a low level bytecode that the Ethereum Virtual Machine (EVM) is able to execute. The virtual machine runs in each node of the network and processes transactions (contract deployments or method invocations) to verify their correctness. As for Bitcoin, a fee is associated to each transaction. Each opcode deriving from the compilation of the high level language has a "gas" cost associated. This is also true for opcodes that permanently store data in blockchain. The total gas cost can be multiplcated by a factor, called "gas price", to establish the actual fee to send along with the transaction, which will be colleted by the miner that first includes it inside a block. Gas cost, hence, transaction fee, and block reward are expressed in Ether (ETH), the cryptocurrency natively supported by the platform. This supports economic in-

[5]Even though Proof of Work is planned to be replaced in future stages of the project with the less expensive and more scalable Proof of Stake.

[6]https://github.com/ethereum/wiki/wiki/Patricia-Tree

centive scheme consistency: miners mint new Ethers that sell to transaction issuers to pay for transaction fees.

Redundant self-enforcing execution Although the fee is collected only by the miner that includes the transaction in a block, every node can verify the correctness of a transaction by executing the invoked contract code parameterized with arguments passed through the transaction. This process is analogous to what in the Bitcoin's UTXOs model is the verification of the correct unlocking of an unspent output consumed by a transaction.

Once deployed, a smart contract is uniquely associated with an address (similar to the one provided to an account) and therefore can receive coin and invoke methods of other contracts. Code of the deployed instance of a contract is not modifiable, which might also constitutes a drawback from an applicative point of view. If not explicitly forbidden by the coded logic, anyone can indiscriminately invoke a method. Standing the congruency between transaction and method invocation, the economic incentive scheme that in Bitcoin guarantees value transactions, here ensures contract methods execution.

Smart contract execution platforms Ethereum has been the first project to introduce self-enforcing execution of code. Similar projects are being deployed. Tezos [8], for example, aims to provide a solution to the governance issues (briefly discussed below) commonly suffered by blockchain related projects. EOS [9], on the other hand, focus on vertical scaling of decentralized applications (the name commonly used to refer to those applications that exploit storage and computation power offered by the decentralized execution platform of smart contracts). EOS development team claims it can process thousands of transactions per second, as opposed to Ethereum that in its current development stage can only process up to hundreds of transactions per second.

The remainder of this work strongly relies on a distributed architecture for self-enforced execution of code. In what follows it is referred to as "Ethereum-like smart contract execution platform".

2.2 Origins

The concept of blockchain, a distributed replicated ledger of transactions, organized in blocks that are chained one another, has been first introduced in 2008 to support the Bitcoin [10] system. In this work Satoshi Nakamoto, a pseudonym that hides an unknown person or group of people, outlines the architecture of a system that allows for a "purely peer-to-peer version of electronic cash". A community of developers gathered around the proposed solution, started to develop and actively maintain the code base of the Bitcoin client. Nowadays, several hundreds of blockchains has been deployed. The software ecosystem grew enormously also driven by the enormous amount of capitals that are channeling into the crypto-economy.

2.2.1 Distributed consensus meets e-cash systems

Blockchain technology emerges from the intersection of at least two research lines: challenges of e-cash systems and distributed consensus are addressed simultaneously resorting to cryptography and game theory principles. In fact, blockchain arises when the central trusted entity deputed to prevent double spending of coins in early electronic cash systems was replaced with a special kind of distributed consensus.

E-cash systems In 1983 David Chaum in [11] introduced a blind signature scheme suitable for common public key signing scheme (*e.g.* RSA [12]) to support exchange of digital money. The blind signature scheme is exploited to provide unlinkability between spend and withdrawal transactions. A trusted central entity, a bank, intermediates payments to ensure coins are not double spent. Coins fungibility is also pursued by fixing coins denomination (or a predefined set of fixed denominations, say 0.5, 1, 5, 10, 50, 100 Dollars for example) to avoid payment traceability thought the payment amount.

This blind signature supported scheme was the trailblazer of a branch of research in the direction of identifying a viable digital electronic money scheme. The need for a trusted central intermediary to prevent the double spending of coins, however, has only been overcome with the advent of the blockchain technology. The significant contribution of the Chaum's work is the replacement of the real world identity with a non-repudiable digital signature in economic transactions, an innovation directly borrowed by the blockchain technology.

The quest for distributed consensus Research on distributed consensus is mainly due to the necessity of providing a reliable log replication mechanism among multiple instances of the same database. Data redundancy, in fact, is the obvious way to achieve resilience, a crucial characteristic for a database management system. Since machine failures can't be excluded from the discussion because they do actually happen, fault-tolerant approaches have to be addressed. State machine replication is the commonly adopted approach. One fault can be easily detected is at least three copies can be compared, and one (the faulty one) results different from the other two. Consensus come into play when the order of inputs has to be established, so that, by processing same inputs in the same order, each machine reaches the very same state.

Although in 1985 it has been demonstrated [13] the impossibility to have deterministic fault-tolerant consensus protocol that can guarantee progress in an asynchronous network (like Internet, where message rounds are not triggered by a central clock), due to possibility of nontermination of the protocol in presence of even one faulty process, several advancements have been achieved in the field.

In 1989 (even though only published long after, in 1998) Laslie Lamport introduced Paxos [14], proving that $n = 2f + 1$ processes suffices to detect f faulty ones. Paxos ensures consistency and the conditions that could prevent it from making progress are difficult to provoke. It only addresses technical faults, or fail-stops,

as opposed to Byzantine failures [15], which includes arbitrary failures of the participants (lying, fabrication of messages, collusion with other participants, selective non-participation, *etc.*).

In 1999 it has been shown a protocol for "Practical Byzantine Fault Tolerance" [16] that, provided at most $\lfloor \frac{n-1}{3} \rfloor$ replicas are faulty out of n, exhibits liveness under certain constraints.

Several Byzantine fault tolerance (BFT) approaches emerges afterwards, among them Fast Paxos [17] by Lamport again, and Tangaroa [18], a BFT version of Raft [19], which, in turn, is a consensus algorithm born as an answer to concerns about the complexity of Paxos, and therefore designed to be easily understandable.

In 2008, the "Nakamoto consensus", introduced in [10], completely changed the rules of the game. As opposed to previous approaches, where replicas are known and limited in number, the Bitcoin's permissionlessness nature requires BFT solution to operate in a public network, where new nodes can join any moment with no authorization. Therefore it is mandatory for the system to be resistant to sibyl attacks [20]. In such an environment in fact it would be impossible to count faulty and properly operating replicas. Proof of Work serves this purpose: establish a (quasi-)continuous metric space for faulty replicas that allows the system to correctly reach consensus as long as the majority of the hashing power of the network is owned by entities that abide by the protocol. As reported before, this consensus mechanism has a probabilistic foundation, also because leverages economic friction and incentives. The block emission rate (one every about ten minutes) defines a quasi-synchronous system where, as shown in [21] in 2016, performances decreases (more that the simple majority of hashing power is required to converge to consensus) as network synchronization worsens.

Over time different signature rights grant mechanisms than PoW appeared. Although PoW proved itself practically viable, the energy consumption [22] of the PoW-based mining process has attracted many criticisms and stimulated the research of analogous mechanisms which do not involve the resolution of cryptographic puzzle. Proof of Stake (PoS) [23, 24, 25, 26], which actually represents a whole panorama of different approaches, is one of the most interesting solution. Signature rights are gained in proportion to the amount of coin owned by the signing entity that as in PoW, also defines which transactions are included into the block. This constitutes an incentive to not behave maliciously by including invalid transactions into the block since it would imply a decrease of the market value of the coin (owned in quantity) by the signer.

2.2.2 Anatomy of a blockchain

Six fundamental aspects characterize a full-fledged blockchain.

Replication Any node who is allowed to operate transactions can also read the whole ledger. Relevant for fault tolerance and accountability.

Integrity Nodes can verify that data structure is integral and has not been tampered with. This is achieved with authenticated data structures.

Authenticity Cryptographic signatures provide authenticity and non-repudiation for any change committed to the ledger.

Publicity Network can be open for any party to join. If not, it is "permissioned" blockchain a registration/identification policy or process is required before accepting new nodes.

Consensus How nodes reach consensus on the ledger. Public open networks rely on PoW (or variants such as PoS). Private networks usually have multiple validators.

Governance Governance and authority over codebase and protocol. Usually trusted upon known individuals or companies.

2.3 Open issues

Blockchain is a young technology. Many issues have been identified over time. Some of them are critical, others can be conceived as development fronts to be explored in the near future. Scalability and interoperability are two examples that belong to the latter category. Scalability issue [27] has to deal with the limited number of transactions that the blockchain can process and store in the unit of time due to synchronization mechanism it relies upon. Interoperability addresses the possibility of moving coins between different blockchains in a secure and trustless fashion.

In what follows, instead, are collected some of the most critical open issues for the blockchain technology along with the currently known answers or counteracting practices, when available. Most of them are specifically related to the Bitcoin system which is the first and most studied blockchain.

Despite soundness and severity of presented issues, blockchain proved itself to be practically resilient until now: it always succeeded in recover from attacks mounted in the past, even though the solution required awkward maneuvers from the community that, to the purpose of quickly resume correct operation, in some cases did not care to violate basic principles of the technology.

2.3.1 Distribution and permisionlessness

Beside the trustlessness, achieved via ex-post verification of transactions, the most important feature of the blockchain is its permissionlessness nature. The permissionless extent of the specific blockchain ultimately depends on the absence of central authorities and therefore on the distribution of the system. In general, however, distribution and permissionlessness can't be considered binary properties or absolute features: they depend on several aspects. The blockchain in fact is a technology that succeeds in containing centralization and in this sense it exhibits a high degree of permissionlessness.

In [28] it has been shown that they do exist centralization forces in the Bitcoin blockchain ecosystem mainly concerning the governance of the protocol (decision making, mining, incident resolution processes, etc.) and it has also been demonstrated that third-party entities can unilaterally decide to devalue any specific set of Bitcoin addresses (a process called "coin tainting") therefore altering the fungibility of those coins. The fungibility issue can be tackled via intermediary services that mix coins from several transactions making them unlinkable (*e.g.* [29] proposes a trustless scheme named TumbleBit) or turning toward blockchains with greater privacy-preserving gurantees (*e.g.* Monero[7] or Zcash[8]). The governance issue is discussed in the following paragraph.

2.3.2 Governance

The highlighted Bitcoin's governance issues are also reported in [30] where the resulting highly technocratic power structure is stated to derive from the dichotomy between the governance by the infrastructure (achieved via the Bitcoin protocol) and the governance of the infrastructure (managed by the community of developers and other stakeholders). The governance is integral part of the protocol and a centralized governance process, to some extent, implies a centralization of the protocol itself and therefore a reduction of the permissionlessness of the system as a whole.

The Bitcoin blockchain, with respect to other blockchains, has the unique characteristic to have been established by a pseudonym subject, Satoshi Nakamoto, that disappeared short after (with the exception of sporadic contributions in the mailing lists), leaving the governance exclusively in the hands of the community. To not have a authoritative reference figure is beneficial for governance decentralization.

On the other hand a central reference figure can be crucial in the case a rapid authoritative decision is required. An emblematic case was the fork happened to the Ethereum blockchain as a consequence of theDAO incident [31]. Vitalik Buterin, the reference figure for the Ethereum community, rapidly made community converge on the fork decision and the potentially fatal incident has subsided quickly.

However, systemic solutions to the governance issues begin to appear. It is the case for example of the Tezos blockchain [8], where developers are incentivized to propose bug fixes and protocol enhancements in the form of deployable code patches that can be voted to be accepted and integrated. If so the proposing developer earns a reward. It is an open question whether this approach is definitive or partial, it clearly represents a further barrier to centralization.

Difficulty to evolve In [32] is pointed out that governance issues arising from the duality of a technology that needs to evolve but has immutability at its heart, creates a friction for the evolution of the technology itself. Hence, application-level innovation is deem as the most suitable type of innovation. The proposed solution makes use side chains where it would be possible to introduce enhancements to the

[7] https://getmonero.org/
[8] https://z.cash/

protocol of any sort, being guaranteed that possibly bugs would not afflict the main chain. Unfortunately a modification of the Bitcoin protocol is needed to have such a system not relies on trusted third parties.

2.3.3 Incentive scheme

Incompletness As pointed out by Emin Gün Sirer in [33], the incentive scheme underling the Bitcoin blockchain doesn't cover all aspects. He highlighted fourteen cases where the altruism of a node is crucial to ensure correct protocol operation.

Instability The incentive scheme at the base of the Nakamoto's protocol prescribes that miners are rewarded for their work with a fix amount of coins for each mined block. They can also collect the fee from every transaction included into the block. In [34] it is questioned that this situation will lead to instability once, because of the halving process (the block reward amount halves every 210.000 mined blocks to control coins issuance), the amount of collected fee will be greater than the block reward. In fact a miner could be incentivized to not include part of the transactions into the block it is currently mining with the purpose to push other miners to mine on top of his own block (therefore ensuring its earning) being they able to collect fee from left out transactions. This open issue is addressed by assuming that the value of the Bitcoin will constantly grow in time and since the protocol allows to increase the decimal precision halving process might never stop, so that the block reward will be always greater than the transaction fees collectable in one block.

Selfish mining This is the name of an attack described in the first instance in [35], whit which miners owning a strict minority of the computing power each may, by colluding, earn a revenue larger than their fair share. Rational miners will prefer to join the malicious group until the computational power owned by the attacking group becomes a majority, therefore decreeing the Bitcoin system as decentralized currency.

Several solutions has been proposed to mitigate this attack, in the form of brand new blockchains based on different protocols (*e.g.* [36]) but also as slightly modification to the original Nakamoto's one. Unfortunately due to the aforementioned governance issues it is very unlikely that those protocol improvements might seen the light anytime soon.

2.3.4 Network layer

Being strongly dependent on the peer-to-peer communication, network attacks can be very harmful for the blockchain. The one known as "Eclipse attack" [37] in particular allows an adversary controlling a sufficient number of IP addresses to monopolize all connections to and from a victim node. The attacker can then exploit the victim for attacks on bitcoin's mining and consensus system, including N-confirmation double spending, selfish mining, and adversarial forks in the

blockchain. Although proposed countermeasures can make more difficult to mount such an attack, they do not provide total shielding.

2.3.5 Immaturity of smart contract languages

According to the distributed execution model introduced by Ethereum smart contract, the low level bytecode is executed by the Ethereum Virtual Machine (EVM) that runs on each node of the peer-to-peer network. Despite the hype around this technology several issues has been exploited over time to mount dangerous attacks [38] and only recently a rigorous formalization of the EVM executable semantics has been proposed [39].

High level languages have been proposed to save developers to directly facing the bytecode. The most adopted, Solidity[9], owes its success to its easy of use. Behind the familiar syntax hides a different programming model and dangerous pitfalls with it. The syntax may result ambiguous in some circumstances [40]. Bugs have been and continue to be reported and fixed [41]. It is a strict design requirement that it must not be possible to alter the execution code of smart contract. Therefore the only known approach to remedy to a contract that contains a bug in it, is to stop to use it and, if possible, replace it with an amended version.

Despite the multitude of high level contract languages, most of them seem to address syntax, in an effort to keep it simple, or similar to other well-known languages, rather than to ensure the required security guarantees.

Formal verification tools have been introduced (*e.g.* Securify[10] for Ethereum smart contract) to cope with this insidious language drawbacks.

[9] https://github.com/ethereum/solidity
[10] https://securify.ch/

Chapter 3

State channels

Contents

This chapter introduces the state channel construction retracing the important developing steps that have brought the technology to its current state and provides some practical examples.

3.1 Principles of operation

State channels are one of the proposed solution to the blockchain scalability issue. State channels are only implemented at application level and therefore they do not require any change of the underling blockchain protocol and are immediately deployable. They achieve scalability by moving some transactions off-chain. The blockchain results unloaded from the burden of processing and storing off-chain transactions. A comparison between the on-chain and off-chain interaction model is shown in Figure 3.1.

A private association The off-chain interaction model dictates that (at least) two entities enter in a private association and, beside an initial and conclusive interaction with the blockchain, they directly exchange messages between them. Those direct messages correspond to transactions that are not sent to and not processed by the distributed network of peers, and consequently not stored on the distributed replicated ledger.

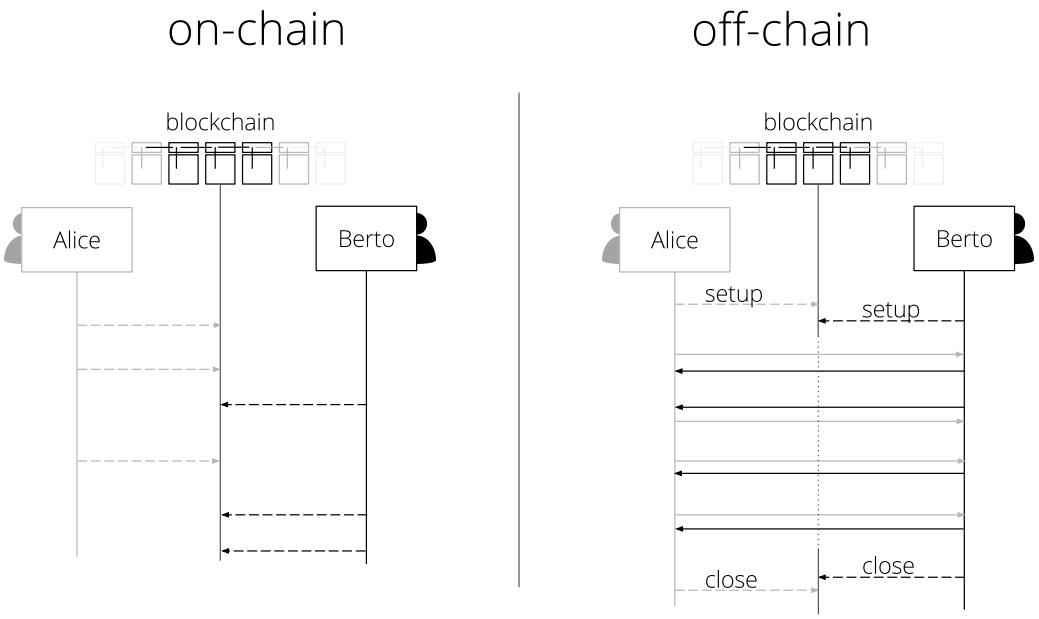

Figure 3.1: Differences between the on-chain and off-chain interaction model.

Consensus mechanism In its most simple form the off-chain association involves two parties and from here derives the name *channel* commonly adopted to refer to these constructions. Since only two parties are involved the most simple consensus mechanism can be applied: everyone agrees on everything. Both parties are in fact required to explicitly agree on the exact sequence and validity of the transactions issued on the channel. Mutual agreement is expressed by means of a digital signature. As long as asymmetric cryptography assumptions hold, a signature work as a not disownable mark of acceptance for the signed data.

Succinct summary To actually pursue scalability, the conclusive interaction with the blockchain has to take to the blockchain only a "succinct summary" of what happened during the off-chain interaction between parties. It must be succinct to not burden the blockchain, while at the same time contain enough information to reconcile the off-chain history with the on-chain one. Therefore the mutual agreement is actually expressed on the outcomes of a transaction rather than on the transaction itself. In the payment use case, for example, parties mutually agree on their balances after each payment transaction. Each party signs and locally stores a data structure containing the current balance of both parties. This data structure actually describes the current *state* of the channel.

Channel opening At opening time, parties interact with the blockchain to record the starting point from which the off-chain interaction branches. This operation often (always in case of a payment channel) involves the locking of funds. In this

case it can be considered as a collateralization of the channel. At closing time those funds will be reallocated according the off-chain interaction.

Channel closure The underling blockchain knows nothing about the off-chain interaction. At closing time, the off-chain history has to be reconciled with the on-chain one. The last off-chain state, signed by both parties, therefore not repudiable by none of them, can be presented to the blockchain to settle respective positions. To allow the underling blockchain to deterministically recognize the correct last state agreed upon during the off-chain interaction, even in case one party decide to cheat presenting an old and possibly more favorable state, both parties must have the opportunity to independently declare which state they consider the last one. Provided that, the blockchain makes the verdict. To minimize the on-chain transactions number, under the optimistic assumption that parties want to collaborate and save on-chain transaction costs, a simple scheme that involve a grace period is adopted. This scheme allows to further unload the blockchain, by saving one on-chain transaction in case of both parties behave honestly. The scheme is articulated as follows: one party makes its assertion, the other has a time window to possibly rebut with its own declaration. An honest party that agree with the first assertion have nothing to do but wait until the grace period expires. Nevertheless the second statement can also be malicious and it is up to the blockchain to deterministically decide the conclusive state.

State order The blockchain decision on which state is the rightful one is based on a total order relation which has to be established and agreed on the sequence of off-chain states. This fundamental feature is achieved differently according to the storage model and language expressivity of the underling blockchain. For the UTXO model, it is achieved exploiting timelocks, *i.e.,* the capability to "susped" a transaction by making it valid only after a certain time (or block height) in the future. Considering two transaction that consume the same UTXOs, the order between them is established, by assigning to the newer one a lock time lower than the old one. Being valid before the old one, the newer one consumes the UTXO that will be not available later on when the old one becomes valid.

For blockchains with Turing-complete scripting languages, a sequence number can be conveniently inserted in the agreed upon data structure along with the information actually describing the channel state.

Disincentive to cheat Similarly to the underlying blockchain, whose security highly relies on an economic incentive scheme, state channels may benefit from the introduction of such a structure. In fact, knowing that a punishment may result from broadcasting a wrong state, a rational attacker abstains from cheating. This disincentive scheme allow to save additional transactions to be broadcast since, in absence of malicious behaviors, arguing on-chain transactions are not required.

A possible, and often adopted, punishment consists in assigning all the funds blocked in the channel to the honest party depriving the malicious one of all its funds, regardless the assignment stated by the last state.

Again, the mechanism adopted to ascribe blame depends on the specific blockchain model: it relies on explicit and easy-understandable data structures when the underling blockchain is supported by a Turing-complete scripting language, contrarily it requires a complex scheme based on trees of pre-signed transactions in case of UTXOs non-Turing complete blockchains.

A two party ledger A state channel can be, therefore, thought of as a two-party ledger where each party stores the current state and possibly previous double signed ones (for the UTXO non-Turing-complete blockchains) to counteract malicious behaviors. Signatures of both parties are essentials to prove not repudiable mutual agreement on a specific state.

Advantages of state channels The off-chain simple consensus mechanism does not involve miners: no transaction fee are needed and therefore no confirmation time has to be waited. Furthermore, the off-chain interaction is only known to the involved parties until the closing of the channel. Hence, if the specific use case is suited for a point to point and continuous interaction, the adoption of a state channel instead of directly relying on the blockchain results in cheaper, faster, and more private transactions.

Security model These advantages come at the cost of a radically different security model with respect to the blockchain one. In particular, while for the blockchain to securely store private keys associated with the account suffices in abiding by the security requirements to guarantee no one can lose funds, for state channels a continuous monitoring activity must be endured along the whole life of the channel. A party, in fact, has to constantly observe the blockchain to rapidly detect the counterparty maliciously closing of the channel. Once detected, the honest party has also to make a move: it has to broadcast the correct state within the closing grace period. To fail in this monitor-and-react activity may imply a loss of funds since in the absence of different information, the blockchain takes for good the last state assertion of the malicious party.

3.2 System model

State channels are designed within the context of a decentralized blockchain that supports the trusted execution of smart contracts according to the Ethereum-like smart contract execution model.

Blockchain The blockchain is considered as an integrity protected and immutable root of trust. It is the decentralized database that contains a global view of account

and their balances as result of their transactions. Each account in the ledger is controlled by its own private key. Only the owner of the account knows the relative private key that uses to authorize transactions from its own account. Without the knowledge of the private key, no authorization can be granted on transactions that originate from an account. Authorized transactions modify the distributed ledger and are available to other accounts after a block is generated, on average every predetermined blocktime T. Finally it is assumed that a transaction inserted into a block b is permanently stored in the ledger after R blocks are generate on top of b, *i.e.,* the probability of a chain reorg that comprises more than R blocks is assumed to be 0.

Smart Contracts State channels also require a smart contract execution environment in addition to primitive ledger transactions that transfer balance from one account to another. Ethereum-like smart contract are allowed to hold a balance in the ledger, and control it according to their code. The code of a deployed smart contract is assumed to be not modifiable. Furthermore, a result obtained in explicit violation of the contract code is not accepted on the global ledger.

Communication Network The availability of reliable, integrity protected communication network (*e.g.* TCP connection) is assumed to support off-chain end-to-end interaction of the parties involved in a channel.

3.3 Usability concerns

There are two fundamental concerns about the usability of state and payment channels.

The first one regards the opportunity cost of funds blocked in the channel. The opportunity cost is paid since an amount of capital is locked in the channel instead of being invested somewhere else, where it can have more chances to return an interest. If a channel is prematurely closed due to the uncooperative behavior of one party, the opportunity cost of locked funds during the closing grace period is completely wasted. Although negligible in value for one unsuccessfully interacting (or for a few of them) the economic loss related to several repeated abortive interaction can become considerable. For this reason it is advisable to open channels with counterparts whose reliability is reputable.

The second critical aspect addresses the necessity to constantly monitor the underlying blockchain to detect a malicious or fraudulent attempt to close a channel. It calls for a radically change of the behavior of a node with respect to the blockchain security model. A node involved in a channel has to actively and continuously monitor the blockchain, even if it has no operation to perform. This establish an obstruction in usability since, for example determines a limitation in the development of mobile trustless solutions. Mechanisms to support the trustless delegation of

this monitor-and-react activity have to be studied in the next future, possibly driven by some form of economic incentive or friction to cheat for the delegate entity.

3.4 Micropayment channels for UTXO-blockchain

Micropayment channels are the first example of off-chain architectures to have been introduced in an attempt to have cheaper and faster transaction on the Bitcoin blockchain, while at the same time increase its transaction throughput.

The very fist appearance of payment channel was by Jeremy Spilman [42], that propose a viable scheme to back them in a discussion on "Anti DoS for tx replacement" in April 2013. As detailed in the resulting article on BitcoinWiki[1], the use case is about a client of a cafe that wants to use the untrusted WiFi access point (AP) to connect to the Internet. In the proposed example the client is willing to pay 0.001 BTC per 10 kilobytes of usage, without opening an account with the cafe.

Provided that transactions can be "paused" until a certain time in the future by means of the *nLockTime* field, the aforementioned use case can be easily supported by this interaction: as the client keep using more and more kilobytes from the AP, it signs and broadcasts a transaction that consumes the same UTXO (or set of UTXOs) but with decreasing *nLockTime*. Each transactions of this series, with respect to the previous one, credits more coins to the AP and less to client as change. All the broadcast transactions remain in mempool (the replicated data structure holding valid but still not confirmed transactions) until the first *nLockTime* expires. Last broadcast transaction has the shortest *nLockTime* and therefore is the only one to be confirmed. Following transactions are invalid since the output has already been spent by the last one and are therefore invalid.

Unfortunately, to keep time-locked transaction in mempool makes the network vulnerable to DoS attack and therefore in 2013 this behavior has been changed. Spilman introduced the very first micropayment channel in an attempt to support anyway this specific use case even after the modification of the *nLockTime* behavior.

Short time later, in June 2013, Mike Hearn announced [43] to have implemented the Spilman's approach in the BitcoinJ[2] library. Described channels are unidirectional: a payer can make server payments toward a payee up to the initially locked amount, but not receive payments. Full duplex micropayment channels have been introduced almost simultaneously in [44] and [45].

In [44] the construction is supported by a pair of unidirectional channels, one for each direction. Each party sees one inbound and one outbound channel, to receive and send payment, respectively. When the outbound channel is depleted and the payer is unable to perform any further payment even though its balance on the inbound channel is higher than zero, the structure can be reset. The reset procedure involves only an off-chain interaction and makes available the inbound

[1] https://en.bitcoin.it/wiki/Contract, Example 7: Rapidly adjusted micro payments to a pre-determined party.

[2] https://bitcoinj.github.io

amount to the outbound one, so that further payments are again enabled. This operation is supported by a data structure called invalidation tree, that ensures that only one path from the root of the tree is first valid by means of timelocks.

In [45], instead, a different approach is adopted to enable duplex channels. In this case the structure relies on the possibility of revoking commitment gave on a previous transaction. Transactions are deemed invalid when other signed transactions exist to "steel" inputs from maliciously broadcast transactions, actually punishing the misbehaving party.

The complex structures required to support duplex micropayment channels are a direct consequence of the UTXO model.

3.4.1 Network of micropayment channels

Both full duplex channel solutions [44, 45] are defined in the perspective of realizing a network of micropayment channels where multihop payments enable a node to pay a not directly connected one, as long as a path with enough capacity can be found from the source to the destination.

Atomicity of the payment is ensured by Hash Time Locked Contracts (HTLC). An HTLC bind a payment to the knowledge of the preimage of a hash function. The payment path is established from the destination back to the source. Only then the source reveals the preimage to the destination that pull funds from the path. Each node pulls from the previous one revealing the preimage. This atomically enforce the payment. In fact, each pair of nodes involved in the payment path can either cooperatively update the channel by invalidating the HTLC based transaction and replace it with a commitment one of the same amount, or, in case the counterparty should not cooperate, closing it. In the latter case, the knowledge of the preimage ensures the enforceability of the HTLC and therefore that no party loses funds.

3.5 Channels for arbitrary state

The innovation introduced by the Ethereum platform on the blockchain ecosystem has also affected the channel construction. With respect to the UTXO model, the account model and the enhanced expressivity of smart contracts to control accounts' balances lead to the definition of channels easier to deal with and that borrow some properties of the underling technology. As opposed to the UTXO model, where unspent outputs have to be singularly unlocked, the capability of smart contracts to execute any custom logic allows for the off-chain handling of an arbitrary data structure, or a "state", and from this the common name adopted for this construction: *state channels*.

3.5.1 Propose/accept scheme

The need for signatures of both channel participants on each state update results in an off-chain interaction scheme that in this work has been named *propose/accept*.

The scheme is rather simple: one party propose an update of the arbitrary data structure, sending the signed updated state to the other. The recipient, after a validity and correctness check, sends back its signature to testify non-disownable acceptance of the proposed update. At the end of the process both parties holds the signature of the counterparty on the last state.

The software client that supports off-chain interaction implements the propose/accept scheme by offering two endpoints. They can be conveniently named `propose` and `accept` (possibly with a suffix or a prefix). With the former one the client listen to new state update proposals from the counterparty, the latter, instead, is to receive its acceptance on a previously proposed update.

3.5.2 Challenge/reply scheme

Security of the channel construction is ensured by the fact that the ultimate root of trust, the underling blockchain, can correctly determine which is the last state parties agreed upon. This is supported by a grace period at the end of the off-chain interaction during which each party can independently state from its own perspective which state it considers the last one. While this ensures security for the payment use case, when an arbitrary data structure is supported, this scheme may not suffice. Arbitrary state implies arbitrary behavior of the channel (actually arbitrary agreement between parties, as shown in Chapter 6). To this end, a tool to enforce the counterparty to accept or propose a state is required.

The challenge/reply scheme serves this purpose. It is triggered when one party issue a challenge by broadcasting an on-chain transaction along with some data useful to instruct the practice. A reply period is also associated with the issuance of a challenge: if the challenged party fails to reply on-chain within this period, it can be punished and the channel closed.

Since the smart contract knows nothing about the off-chain interaction, the challenge can also be issued maliciously. In this case the issuer that can be punished if the honest challenged party presents a proof to allow the blockchain to ascribe blame to the issuer within the reply period. A couple of practical examples are provided underneath to clarify the scheme in greater detail.

In-depth analysis of the informational asymmetry among involved parties, *i.e.,* channel endpoints and supporting smart contract, may benefit from a rigorous formalization. A staring point has been identified in [46], where a model for interactive unawareness is proposed, but further investigation on these aspects are certainly needed.

3.5.3 Example 1: state channels for payment

In what follows is outlined how a payment channel can be implemented on a state channel backed by a smart contract executed by an account based blockchain with Turing-complete scripting language.

The channel is opened between Alice A and Berto B. The state is represented by a tuple of three elements: $< i, \beta_i^A, \beta_i^B >$, where the first element is the sequence number of the state, the second element is the current balance of A and the third one is the current balance of B. Signatures (σ) are represented as subscripts of the signed data structure. Superscripts specify the endpoint (A or B) and subscripts indicate which state update number a symbol refers to.

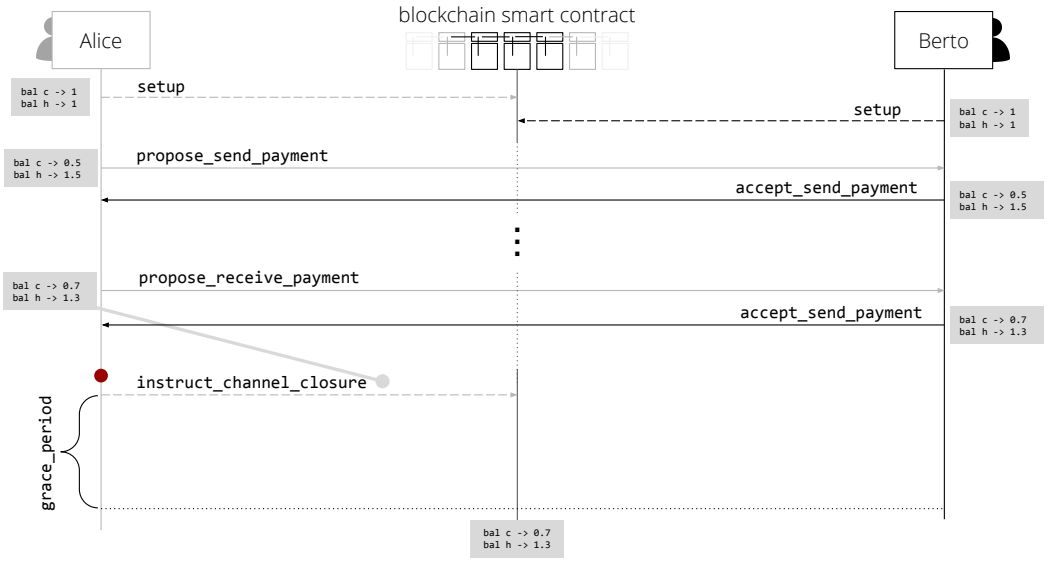

Figure 3.2: Example of cooperative behavior in a payment state channel.

Figure 3.2 shows how the protocol proceeds. After the two parties join the channel sending the amount they want to lock to the supporting smart contract, a first off-chain interaction must occur and the current state of the channel become $< 1, 1, 1 >_{(\sigma^A, \sigma^B)}$. If one party refuses to accept this first state update proposal, a smart contract method `closeNoOffChainInteraction` allows to collect injected funds, possibly applying punishment to the unresponsive party.

Then two payments are performed according to the propose/accept scheme. Going into deep for the first one, with the `propose_send_payment` A sends $< 2, 0.5, 1.5 >_{(\sigma^A)}$ to B, which accept the payment by answering with his signature of the proposed update σ_2^B. At the end of this interaction both A and B own $< 1, 0.5, 1.5 >_{(\sigma^A, \sigma^B)}$.

The second payment in the example, which is also the last one, possibly after a series of n, is triggered by A to receive an amount of 0.2. Hence, it is actually a request of payment, a possibility enabled by the propose/accept scheme. Interaction proceeds similarly to the first payment and once accomplished both parties own $< n, 0.7, 1.3 >_{(\sigma^A, \sigma^B)}$. Note that the total balance of the channel is a state invariant and can be regarded as a integrity check.

With `instruct_channel_closure` A triggers channel closing. She takes the correct last agreed upon n-th state represented by $< n, 0.7, 1.3 >_{(\sigma^A, \sigma^B)}$. The grace period starts in this moment and since B has nothing to argue can simply wait for the grace period to expire. After that the smart contract allows A to collect 0.7 and B to collect 1.3.

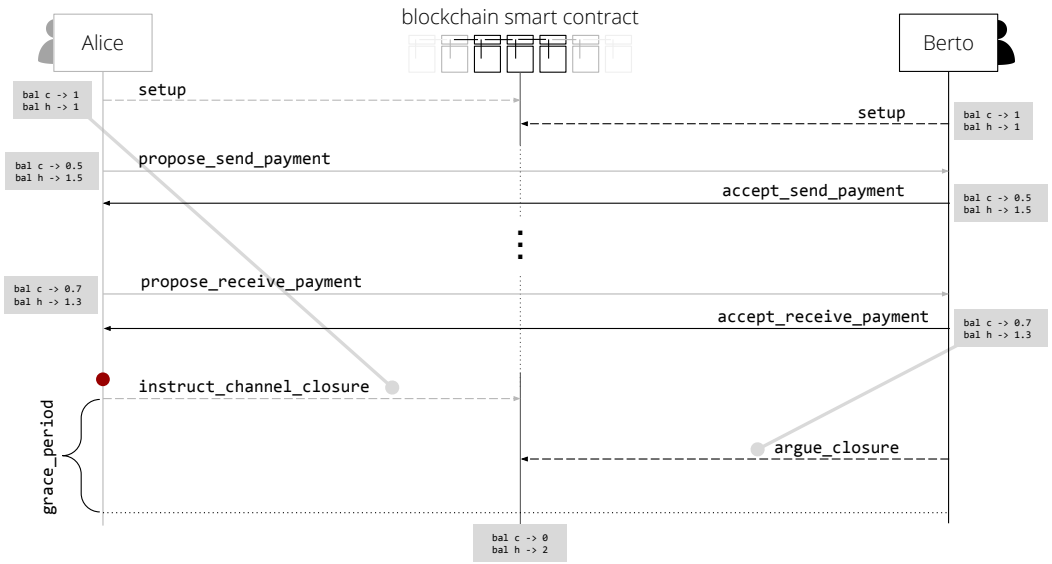

Figure 3.3: Example of malicious behavior in a payment state channel.

Figure 3.3 shows the same interaction just described but A try to close the channel bringing an old state more favorable for her: $< 1, 0.5, 1.5 >_{(\sigma^A, \sigma^B)}$. Therefore, within the grace period, B has to inform the smart contract about the existence of the more recent state $< n, 0.7, 1.3 >_{(\sigma^A, \sigma^B)}$. Since A signed the n-th state update, she can not disown she agreed on that state. By comparing the sequence numbers of the two states, the smart contract punishes A and assigns all the channel balance to B immediately.

IOUs flow in channels It is worth noting that during the off-chain communication parties do not exchange coins. They actually exchange promises of coins, fulfilled after the closure of the channel. Those promises are often called IOU (from the sound of the sentence "I owe you"). Security guarantees of the channel construction ensure that an involved actor that diligently abides by the protocol is sure to see those promises fulfilled at the closure of the channel.

3.5.4 Example 2: state channel with arbitrary state

The off-chain state shared between parties involved in a state channel can be any arbitrary data structure. There are some peculiar differences with respect to the

payment channel example described above that is worth to highlight. In what follows is shown how to exploit a general purpose state channel backed by a smart contract that runs on account based blockchain with Turing-complete scripting language to play a tic-tac-toe game [3]. Two players put their symbols (∘) and (×) on a three by three grid representing the game board. The game proceeds in alternate turns. The first player who aligns three symbols of his own wins the game. The ∘ always starts with the first move. In the example the two players (∘) and (×) bet coins on the outcome of the game. They inject the same amount of funds in the contract, play the game interacting off-chain and, once the game is ended, the contract assign the whole amount to the winner, or redistribute it in case of draw.

Although some of the contract methods described below can be merged together, they are left separated for the sake of clarity. The objective here is not to optimize transaction cost or minimize on-chain transaction number but provide an example to understand operation and pitfalls of off-chain constructions. Furthermore, for the sake of clarity the off-chain version of the tic-tac-toe game is introduced starting from a completely on-chain version of the same game.

The on-chain version The supporting smart contract, in this case must provide the following methods: `join`, to join the game and bring the amount to bet; `move`, to make a move, the contract only receive the move if it is a valid one, *i.e.,* the player, in its own turn, put its own symbol in a empty board slot; `check_victory` to let the contract verify the victory of one party and consequently assign funds; `withdrawval` to withdraw funds from the contract; `check_opponent_dropout` to check is a player has abandoned the game once a move timeout has expired. Furthermore, a data structure that store a serialized version of the game board and some variables to track the state of the game are involved.

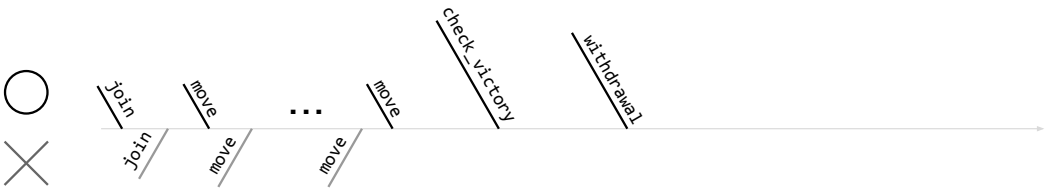

Figure 3.4: Tic-tac-toe example: on-chain cooperative behavior.

Figure 3.4 shows how a cooperative on-chain interaction might proceed, from the joining of the parties until the withdrawal of the winning one. After both parties have joined, they make moves until a player wins. The winner makes the contract acknowledge the victory and withdraws the won amount (player ∘ in the example).

[3]The reader not aware of the rules of the game is refered to the relative Wikipedia article: https://en.wikipedia.org/wiki/Tic-tac-toe.

Every interaction correspond to the invocation of a contract method by means of one on-chain transaction.

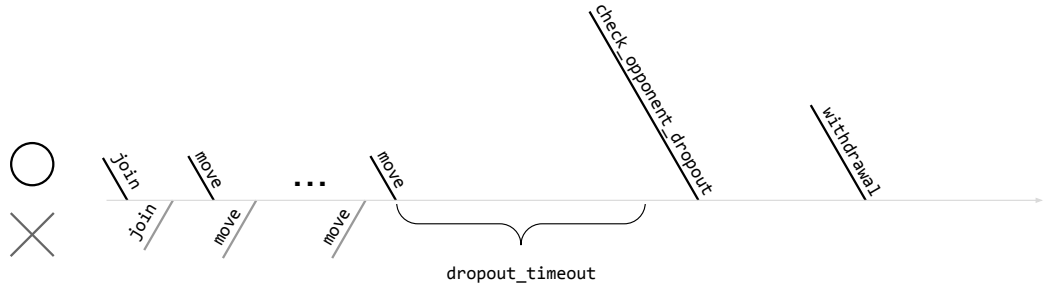

Figure 3.5: Tic-tac-toe example: on-chain uncooperative behavior.

Figure 3.5 outlines instead how the interaction proceeds when one party ceases to collaborate, player $\times$ in the example. Every move has a timestamp from which a `dropout_timeout` start counting. Since Ethereum-like smart contract are only reactive, the check on the timeout expiration is actually done ex-post, when the active player invokes the `check_opponent_dropout` method. The abandoning player is acknowledge as the loser by the contract and the active one is finally allowed to withdraw funds.

The off-chain version The client that handles the off-chain version of the tic-tac-toe game offers the following endpoints to support direct communication between players: `propose_move`, to propose a move to the counterparty; `accept_move`, to provide the signature to the counterparty and therefore accept the proposed move. Those endpoints could have been implemented as one single endpoint, exploiting the response to a proposal to reply with the signature. However, to clearly show the interaction, it has been preferred to separate responsibilities.

The off-chain state is represented by the tuple $< n, s1, \ldots, s9 >_{(\sigma_n^{\circ}, \sigma_n^{\times},)}$ which contains the sequence number n, a serialization of the game board slots, $s1, \ldots, s9$, and signatures of both players.

The supporting smart contract is more complicate than the one that backs the on-chain version. The on-chain state of the game is described by the following variables: `uri_o` and `uri_x`, to store endpoint physical address and therefore allow for a direct communication[4]; `bet_amount`, to specify the amount parties are required to bet on the specific instance of the contract; `challenging_grace_period` and `closure_grace_period`, to store grace periods duration; `game_state`, to track the game state (`Init`, `Unmatched`, `Running`, `Challenging`, `Ended`, `Aborted`); `withdrawer`, to store who is entitled to withdraw funds, the winner or the at-

[4]It is obvious that a production version of this contract should also provide management methods, for example to modify an endpoint physical address, conversely overlooked in what presented

tacked honest party, if punishments have been applied to the malicious one; `challenge_data`, to store details about the last challenge issued and possibly non resolved yet;

The off-chain game is backed by the following methods: `join`, to join the game bringing the amount to bet; `challenge_start`, to challenge an inactive player to make the first move; `reply_start`, to cooperatively reply to a `challenge_start`; `error_propose`, to report counterparty proposed an invalid move; `challenge_propose`, to challenge the counterparty to propose its next move; `reply_propose`, to cooperatively reply to a `challenge_propose`, providing the next move; `report_challenge_propose`, to report a maliciously issued `challenge_propose`; `challenge_accept`, to challenge the counterparty to accept the last proposed move; `reply_accept`, to cooperatively reply to a `challenge_accept`, providing the signature on the last proposed move; `report_challenge_accept`, to report a maliciously issued `challenge_propose`; `abort`, to let the contract evaluate the effective expiration of a grace period and, if so, abort the game; `end`, to let the contract evaluate a conclusive state, either a victory or a draw one; `withdraw`, to withdraw fund from the contract;

Since the smart contract completely ignores off-chain interaction, proofs that accompany a challenge method call generally include the last state on which both parties agreed upon and information on the subsequent pending state which is the actual reason of the challenge. The state checkpoint is needed to allow the chain to evaluate the correctness of the pending state. However, one malicious party might try to checkpoint an old state in an attempt to fork the off-chain history. The attacked party that detects such an attack must report this behavior to the contract (through `report_challenge_propose` or `report_challenge_accept` method) showing the correct most recent state on which both parties agreed upon. As usual, agreement is testified by a digital and un-disownable digital signature on the state data structure.

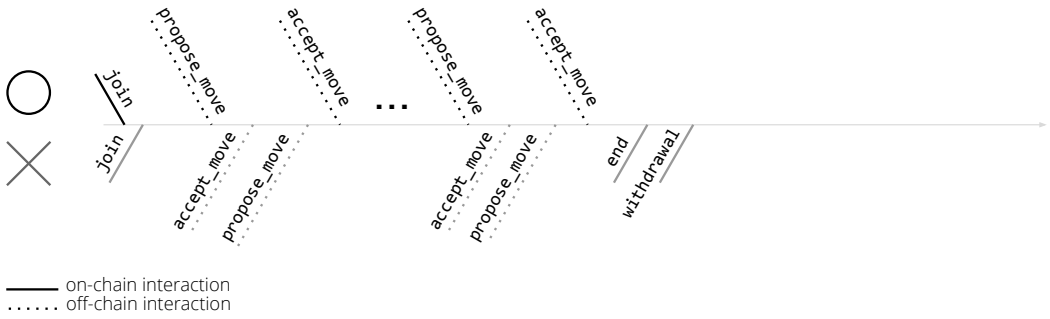

Figure 3.6: Tic-tac-toe example: off-chain cooperative behavior.

Figure 3.6 shows how a cooperative game should proceed. Parties join the game, they interact off-chain through the propose/accept interaction scheme. One party wins, informs the contract about its victory and withdraws the won amount.

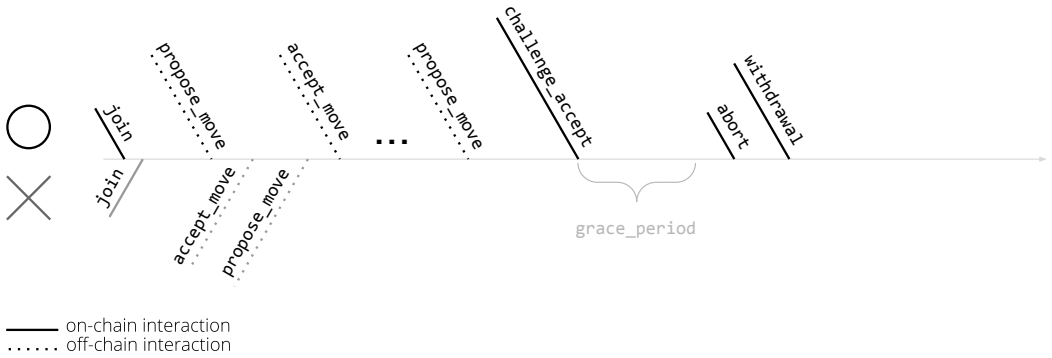

Figure 3.7: Tic-tac-toe example: off-chain uncooperative behavior.

Figure 3.7 shows what happens when a party definitively abandon the game. In the example player × drops out after receiving a proposal of a new move from player ∘. As opposed to what happens in the on-chain version, the timeout can't start autonomously and therefore an on-chain interaction is needed to mark an instant from which start counting. The `challenge_accepts` issued by player ∘ serves exactly this purpose. Once the grace period has expired, player ∘ can conclude the game and withdraw the amount won by opponent abandonment.

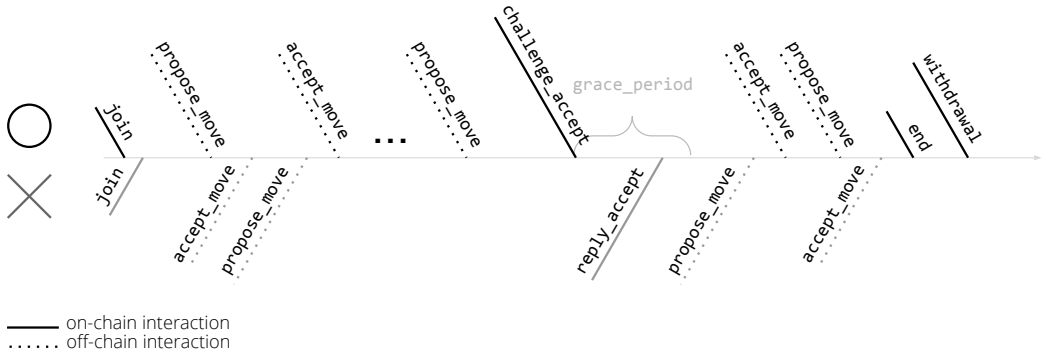

Figure 3.8: Tic-tac-toe example: off-chain challenge reply.

Should the opponent have not definitively abandoned the game but, for example, only suffered from a technical fail that it managed to resolve within the challenge grace period, it replies to the challenge via the `reply_accept` contract methods and the interaction can continue off-chain until the end of the game, as shown by Figure 3.8.

3.5.4.1 Security pitfalls

According to the presented off-chain scheme, a rational adversary is not able to cause damage to an honest party that diligently abides by the protocol. The intrinsic transaction costs associated with the issuance of a challenge along with the awareness that the challenge scheme, when correctly followed by the honest party, will certainly punish a malicious challenger, deter such a behavior. Conversely, if an irrational adversary is assumed, security guarantees have to be relaxed. In fact, an irrational adversary is willing to lose its money to the purpose of seeing the honest party economically damaged. An example of irrational active behavior is obtained when one party, after having proposed a new state update and also received the acceptance from its honest counterparty, proceeds anyway to challenge it on-chain. The attacker pay the transaction cost but the honest party is obliged to pay transaction cost as well to take the signature it had already conveyed to its malicious counterparty on-chain.

It has to be mentioned that also the underling PoW blockchain will fail to withstand an attack mounted by a hypothetical irrational adversary that controls enough computational power. This situation is extremely expensive to be realized nowadays, at least on the most capitalized blockchains (Bitcoin, Ethereum) therefore the economic loss of the irrational attacker would be huge and, hence, it constitutes a strong disincentive. Unfortunately, for state channel, this quantitative argument doesn't hold. If an endpoint of a state channel behaves irrationally, it may cause an economical damage to the honest one. Nevertheless, the honest party may be paid back by the allocation of the whole channel balance, if punishments are applied.

Irrational passive aggressive behavior A special and extremely harmful case of irrational behavior is the one defined passive-aggressive. According to [47] in fact, while an aggressive behavior leaves some proofs on the ground and, therefore, simple to detect and punish and consequently to be discouraged, the passive behavior is the most difficult to detect and disincentive.

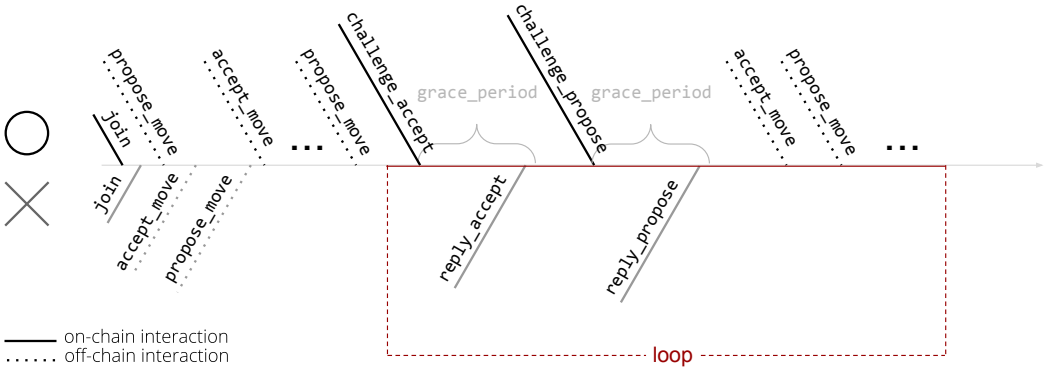

Figure 3.9: Tic-tac-toe example: off-chain passive aggressive behavior.

With reference to the tic-tac-toe game example provided before, Figure 3.9 depicts how player × should act to exhibit an irrational passive aggressive behavior. The irrational attacker, player ×, remains passive and forces the honest party, player ∘, to challenge him on-chain and player × wait until the very last moment before to issue the on-chain reply. Then the attacker returns passive and wait until player ∘ issues an on-chain challenge to stimulate the state update proposal by player ×, which remain passive until the very last moment before to reply on-chain. This sequence of events may be repeated in loop, therefore determining the most harmful situation for state channel. Player ∘ may not prefer to close the channel since, as in many games, she anticipates the outcome of the game and estimates herself in an advantageous position. Hence, the attacker is forcing the honest party to pay more and wait longer than he was not mounting the attack. Even worst, due to the propose/accept protocol that requires two interactions to progress of one move, the honest party is being penalized in terms of costs and time also with respect to the plain on-chain interaction.

The *smart channel* protocol introduced in Chapter 6 of this work tackles with this drawback of state channels by enabling the honest party to require a seamlessly migration of the off-chain interaction toward the on-chain interaction model, which, being mediated by a smart contract, is immune to the irrational attack.

3.5.5 Deterrent code

It is worth noting that a very peculiar kind of source code emerges from the implementation of smart contracts that support state channels. In this work it has been defined *deterrent code*. It is a code that best fulfills its function when it is never executed. To not have it in the code base means a tremendous security breach, but once it is there, it is never used. Method `report_challenge_propose` and `report_challenge_accept` introduced before are two practical example of deterrent code. They are useful to report a malicious challenger that tries to fork the off-chain history by issuing a challenge from a past off-chain state. If those methods were not implemented, those attempts would remain unpunished and a rational attacker would not be discouraged to mount such an attack. But since they exist, the attacker is aware that the diligent honest party will argue her reasons and the contract will punish him. Hence deterrent code, enhances the security level by promoting the threat model from rational to irrational: only by assuming an irrational adversary, deterrent code is called into question and may be executed. In case of a rational adversary assumption, instead, the deterrent code is never executed and its purpose is nothing but its existence.

Chapter 4

Inextinguishable payment channels

Contents

This chapter describes a practical construction to achieve payment channels that can be hot-refilled and from which funds can be hot-withdrew. With the hot-refill procedure one endpoint can arbitrarily decide to inject more funds into his side of the channel. Hot-withdrawal, on the other hand, is the dual procedure of hot-refill that allows one party to independently pull out funds from a channel. These procedures do not introduce any halt or delay with respect to a standard payment channel workflow in the cooperative scenario. The proposed architecture prevents a skewed

channel to be closed and reopened, therefore saving all the on-chain transactions needed for those purposes.

Inextinguishable channels rely on the *detach/attach* scheme which allows to detach (or isolate) part of the balance of an account into a token. The token can be later on attached (or merged) to the account's balance. The scheme can be used to move funds from an off-chain balance to an on-chain one. The hot-refill and the hot-withdrawal of funds is realized by selecting the source balance other than he destination balances, *i.e.,* on-chain and off-chain or vice-versa.

The solution is defined at application level and is suited for blockchains with by Turing-complete scripting language, since it requires enough expressivity to support the interactive challenge schemes that settle possible off-chain disputes.

The remainder of this chapter is articulated as follows. Section 4.1 introduces the motivations behind inextinguishable payment channels. Section 4.2 reports about related work. Section 4.3 provides details about the proposed interaction scheme. A security analysis is presented in Section 4.4 and details on system usability are discussed in Section 4.6. An overview of the proof of concept implementation is provided in Section 4.5. Finally, Section 4.7 outlines future developments.

4.1 Motivations

A payment channel can become skewed preventing one party to make any further payment. As pointed out in [44], this situation is especially likely to happen at the edge of a payment network where, due to the characterization of the players, that can be merchant or customer, the majority of payments flows in one direction. In this scenario, inextinguishable channels pursue blockchain scalability and payment channel usage optimization. In fact, if a frequent payer is able to refill a channel and similarly a party that often receives payments can pull out some funds from it any moment, the necessity to close and reopen the channel vanishes. In this way several on-chain transaction can be saved, namely: i. old channel closing (1 tx); ii. residual funds withdrawal (1 or 2 txs, depending on the particular implementation); iii.channel opening (1 tx); iv. new channel funding (1 or 2 txs, depending on the particular implementation). Furthermore, a skewed channel that needs to be closed and reopened also entail a the waiting of a closing grace period before the parties can regain possess over their locked funds. This implies one of the two followings: 1) the opening of the new channel has to be postponed until the end of the closing grace period of the old channel; or 2) parties have to sustain capital advance to fund new channel with funds different from those locked into the old channel. These drawbacks are avoided by using an inextinguishable payment channel, since parties continue to use the very same channel and refill and withdrawal operations seamlessly interleave with payments.

4.2 Related work

Inextinguishable payment channels are in line with previous advancements in the fields. Since their introduction, solutions have been proposed to avoid the unnecessary closing of a channel, provided that the relation between the endpoints is to continue. Preventing the superfluous closing results in an extension of the channel lifetime. As reported in Section 3.4, originally channels were exclusively unidirectional. [44, 45] introduced the possibility to have bidirectional payment channels, while contextually a way to maximize the usefulness of locked funds, *i.e.,* payment networks, that relying on multi-hop payments allow to send payments to recipients not directly connected. In [48] is proposed a solution to find closed loops in payment networks to be rebalanced. If perfect cooperation holds among rebalancing parties, the process does not require any on-chain transaction and ensures further off-chain interaction for rebalanced channels. [49] introduces virtual channels, *i.e.,* channels whose collateral is locked from off-chain balance already locked in a channel and not from on-chain balance.

4.3 The detach/attach scheme

The attach/detach scheme offers a smooth way for an endpoint of a payment state channel to move funds from his on-chain balance toward his off-chain one and vice-versa. The smoothness is ensured as long parties are collaborative and no disputes arise off-chain. Should parties not converge to an agreement on the off-chain operations, an interactive challenge scheme, that involves the underling blockchain, guarantees that no one is afflicted by any loss of funds.

Being defined as an extension for state channels, inextinguishable payment channels share the same system model, introduced in Section 3.2.

For the following explanation it is assumed that Alice (A) and Berto (B) have established a payment state channel between them.

4.3.1 Principles

The "detach/attach" name synthesizes how the scheme works. Part of the balance is detached from one kind of balance (the on-chain or off-chain one, depending on the direction of the transfer) and segregated into a token. A token of this sort, along with the relative signature, represents a proof of detachment and serves two purposes: 1) to be attached back into the other kind of balance of the token creator with respect to the one it was detached from, realizing the seamless transfer of funds; 2) to be used as evidence during the on-chain interactive challenge scheme that ensures no honest party can be defraud of owned funds. Hot-refill is realized when a token is detached from the on-chain balance of an endpoint and attached back to the off-chain balance of the same party. The effect of a hot-refill is to move funds from the on-chain balance toward the off-chain balance of a party, refilling party's channel balance. On the other hand, hot-withdrawal is characterized by an

off-chain token detachment, followed by an on-chain token attachment. The effect of a hot-withdrawal is to move fund from the off-chain balance toward the on-chain balance of a party, pulling-out funds from party's channel balance.

4.3.2 Data Structures

Here follows the description of the data structures required by the detach/attach scheme in order to realize the hot-refill and the hot-withdrawal to and from an inextinguishable payment channel.

Token A token is a tuple $< j, \alpha, \text{ID}(\texttt{owner}) >$, where j is a sequence number, α corresponds to the amount of funds represented by the token and ID(`owner`) indicates the owner of the token (*e.g.* the account address in the Ethereum realm).

Off-chain Channel State With respect to the off-chain state tuple described in Section 3.5.3, the detach/attach scheme requires two optional extra fields. The off-chain state is then described by the tuple $< i, \beta_i^A, \beta_i^B, \mathcal{H}(\texttt{tkn}_j), [\mathbb{D}|\mathbb{A}] >$, where the fourth optional element is the hash[1] of the token j and the fifth optional element is one of the two symbols $\mathbb{D}$ or $\mathbb{A}$. When $\mathbb{D}$ appears, it indicates the detachment of the token j; when it is the case of $\mathbb{A}$, it symbolizes the attachment of the token j.

Proofs If parties involved in an inextinguishable payment channel do not agree on the order or on the amount of detached and attached tokens, the party that believes to be cheated can resort to the underling smart contract presenting or asking for a proof to reveal and punish the malicious behavior of the counterparty. Therefore it is required to generate a proof for each step of the detach/attach scheme, namely a *Proof of Detachment* (PoD) and a *Proof of Attachment* (PoA). PoDs and PoAs can be generated both on-chain and off-chain as the detachment and the attachment of a token can take place both on-chain and off-chain. Here follows a description of each proof required by the scheme.

On-chain PoD The smart contract permanently stores on the blockchain the hash $\mathcal{H}(\texttt{tkn}_j)$ of each token j successfully detached from the on-chain balance of each party. The party that intents to demonstrate she has successfully detached a token have to submit to the smart contract the token tuple $< j, \alpha, \text{ID}(\texttt{owner}) >$ so that the contract can compute again the hash and execute a membership test on the set of on-chain detached tokens it holds.

Off-chain PoD It is generated through an off-chain state update. The detachment of the token j is wired into the new state and accepted by means

[1] Generate with a collision resistant hash function applied on a standard serialized form of the token tuple. A valid procedure would be to apply the SHA3-256 algorithm on a marshaled version of the tuple, where each field is prefixed with its byte length. Although this implies an upper bound for the number of allowed refills, this number can be chosen arbitrarily large.

of a signature by the counterparty. For example, being the state i described by the tuple $< i, \beta_i^A, \beta_i^B, \dots >_{(\sigma^A, \sigma^B)}$ (dots means that it does not matter if a token has been detached or attached at this point), the detachment of a token $< j, \alpha, \mathrm{ID}(\mathtt{A}) >$ entails a state update of the form $< i+1, \beta_i^A - \alpha, \beta_i^B, \mathcal{H}(\mathtt{tkn}_j), \mathbb{D} >_{(\sigma^A, \sigma^B)}$, which also represents the PoD. The signature of B on the state update testifies to the smart contact his undeniable acceptance of the token j off-chain detachment operated by A.

On-chain PoA The smart contract permanently stores on the blockchain the hash $\mathcal{H}(\mathtt{tkn}_j)$ of each token j successfully attached to the on-chain balance of each party. The smart contract can therefore autonomously verify if a token has already been attached on-chain or not, thus avoiding any reply attack on token attachment.

Off-chain PoA It is generated through an off-chain state update. The attachment of the token j is wired into the new state and accepted by means of a signature by the counterparty. For example, being the state i described by the tuple $< i, \beta_i^A, \beta_i^B, \dots >_{(\sigma^A, \sigma^B)}$, the attachment of a token $< j, \alpha, \mathrm{ID}(\mathtt{A}) >$ entails a state update of the form $< i+1, \beta_i^A + \alpha, \beta_i^B, \mathcal{H}(\mathtt{tkn}_j), \mathbb{A} >_{(\sigma^A, \sigma^B)}$, which also represents the PoA. The signature of B on the state update testifies to the smart contact his undeniable acceptance of the token j off-chain attachment operated by A.

4.3.3 Hot-refill

Say that A is the channel party that is performing the hot-refill. The operation is then achieved by 1) detaching the refilling amount from the on-chain balance of A and 2) attaching it back to the off-chain balance of A. The output of these two operations is a couple of proofs: an on-chain generated PoD and an off-chain generated PoA, respectively. Those proofs have to be used in case a dispute arises. In particular, PoD is needed by A to show that a certain amount of its on-chain balance has been token-segregated in the case B should refuse to attach it back to A's off-chain balance. PoA, instead, is useful for B to show that he has already provided his consent to the off-chain attachment of the specific token, should A maliciously challenge B on-chain. The process is summarized in Figure 4.1 and detailed in the next paragraphs.

On-chain Proof of Detachment generation This step is labeled as 1 in Figure 4.1. It consists of an action required from A toward the smart contract to generate a token j and to store into the blockchain its hash $\mathcal{H}(\mathtt{tkn}_j)$.

To thwart any malicious attack from B, A must store the token tuple until B has signed a state updated where the token is attached, as detailed in the next paragraph.

It is worth noting that an identifier of the channel is omitted, for the sake of readability, from the above description and from all the constructions of Figure 4.1:

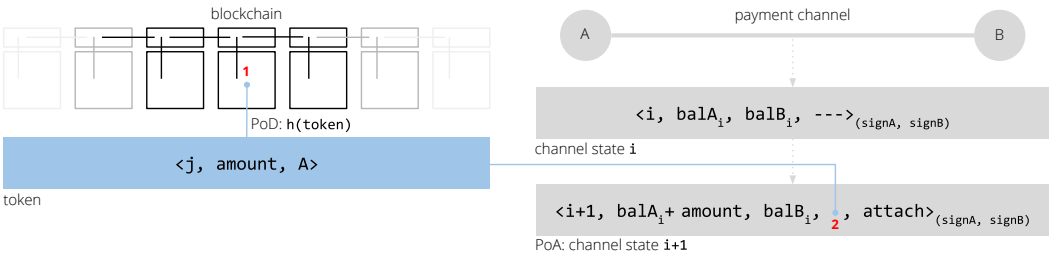

Figure 4.1: Hot-refill procedure for inextinguishable payment channel.

the state updates `i`, `i+1` and the token. However, it is of pivotal importance to avoid that a PoD or a PoA could be used in a different channel joined by same parties with the same pseudo-identities (addresses). The same holds for the next paragraph and for Figure 4.2.

Off-chain Proof of Attachment generation This second step, labeled as 2 in the Figure 4.1 produces an off-chain PoA. It can be used afterwards by B to answer a challenge maliciously issued by A pretending to have never seen its token attached back to ts off-chain balance. Since this reply attack can be mounted by A any moment, B must store the PoA until the channel is closed.

To attach the token back to the off-chain balance, *i.e.,* to generate the PoA means, for the attaching party, to get her off-chain balance incremented by the token mount α. The attaching party is therefore incentivized to propose the state update where the PoA is generated.

4.3.4 Hot-withdrawal

The hot-withdrawal can be conceived as the hot-refill dual process. Figure 4.2 depicts how it works. Say that A wants to withdraw part of her balance locked in her side of the payment channel with B. The withdrawal is achieved by 1) detaching the withdrawing amount from the off-chain balance of A and by segregating this amount into a token and 2) attaching it back to the on-chain balance of A. The output of these two operations is a couple of proofs: an off-chain generated PoD and an on-chain generated PoA, respectively. The PoD is used by A to certify to the blockchain that B agreed on the withdrawal. The PoA, permanently stored by the blockchain, makes it impossible to A to redeem the same token more than once.

Off-chain Proof of Detachment generation A generates a token of a certain amount α out of the channel state by proposing a state update where its balance is decreased by the same amount it is reported that this particular token has been detached (operation labeled as 1 in Figure 4.2). This particular state updated signed by B represents an off-chain generated PoD.

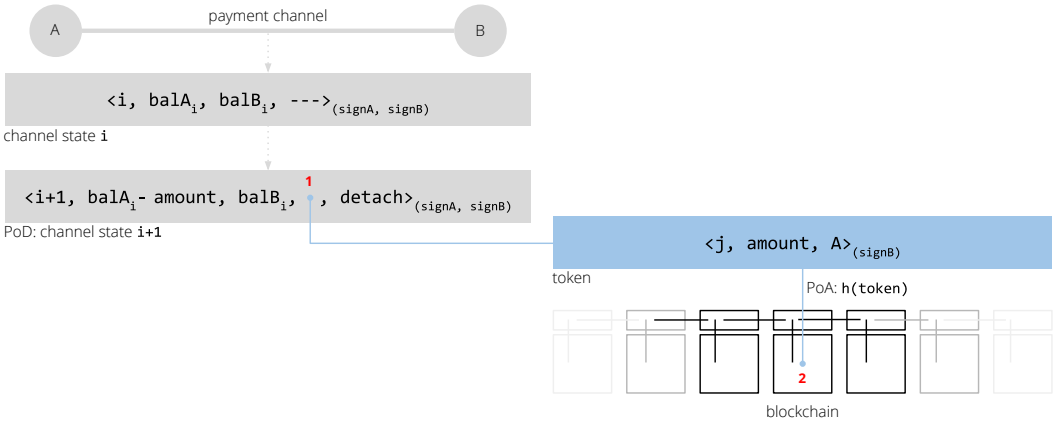

Figure 4.2: Hot-withdrawal procedure for inextinguishable payment channel.

There is no need for A to store this off-chain generated PoD after it has been replaced by a newer state update. In fact, the hot-withdrawal last step, namely, the on-chain token attachment, described in the next paragraph, only requires an interaction with the blockchain and can't be by tampered by B.

On-chain Proof of Attachment generation This second step, labeled as 2 in the Figure 4.2 produces an on-chain PoA. A presents the PoD to the smart contract that permanently store the hash $\mathcal{H}(\texttt{tkn}_j)$ of the token j only if it has not been already redeemed. This prevents any further token attachment attempts by A. Since the proof is stored by the blockchain, there is no need for A and B to store it off-chain.

4.3.5 Continuous operation

As long as the parties are collaborative, the proposed approach does not require any halt of the channel operativity during hot-refills and hot-withdrawals. Once separated from the original balance (either the on-chain or the off-chain one) and segregated into a token, the amount that has to be refilled or withdrew not interfere with the payment channel anymore and payments can be issued without concerns for the pending token.

In particular, the on-chain attachment operation required by the hot-refill procedure is perfectly compliant with a standard payment operated on the channel according the propose/accept scheme. It comprises a proposal of a state update from the refiller and the acceptance of the counterparty (A and B respectively in the previous example).

A tricky condition is configured when there are several pending tokens, *i.e.,* tokens that have been detached but not attached yet, and one party requires to close the channels. This situation is detailed in the next Section.

4.4 Security analysis

The detach/attach scheme is designed as an extension to be applied to a viable implementation of a payment state channel. The smart contract that supports the state channel is addressed to resolve disputes that may arise off-chain.

In what follows it is shown that additional parts with respect to state channel implementation, namely the hot-refill and the hot-withdrawal procedures, do not change the security model of a state channel: an honest party that diligently follows the protocol is able to protect itself from losing its funds.

Since resorting to the blockchain to resolve off-chain originated disputes implies transaction costs that depend on executed instructions and stored data, the challenge scheme is designed to minimize those costs and only requires the very minimal amount of information to be taken on-chain in case of disputes.

4.4.1 Threat model

The adversary is considered to occupy one endpoint of the channel and its objective is to damage the honest counterparty whose private keys are however correctly and securely stored and not accessible to the adversary. The adversary is considered irrational in the sense that to pursue the objective of damaging the honest party it is willing to lose its funds, partially or even totally. The kind of damage such an adversary can cause is to make the honest party to lose its funds, partially or totally.

4.4.2 Guarantees for honest parties

Under the aforementioned adversarial assumptions, an hones party that carefully follows the protocol does not incur any loss of funds. As for a standard payment channel, in case of an attack, what is lost is the opportunity cost of the locked capital. In fact, malicious counterparty behavior compels the honest party to close the channel waiting for the closing grace period to expire. However, the capital opportunity cost is often considered negligible also by virtue of the small amount of funds that is suggested to lock in channel with an unknown, and possibly not trusted, counterparty.

4.4.2.1 Balance conservation

This property holds if every rightfully detached token is attached back exactly once. This main property is ensured by the following ones.

4.4.2.2 Token attaching enforceability

The enforcement of the token attachment is only required in the hot-refill procedure, since the attachment procedure involves an interaction with the counterparty that may refuse to sign the state updated where the token is reported as attached. In the hot-withdrawal procedure, on the contrary, the attachment is performed by

the blockchain itself, and, as long as the withdrawer owns a valid PoD, it can autonomously proceeds and an adversary can not interfere in any way.

Focusing on the hot-refill procedure, if the counterparty refuse to collaborate by signing the state update that attaches back the token, theoretically the refiller may enforce the adversary to attach the token by challenge it on-chain. In practice, however, the refiller that detects non-collaborative behavior from the counterpart may prefer to close the channel despite still having even if it still has a pending token to attach. The scheme guarantees that even in this case the honest party does not suffer any fund loss. Details on this specific situation are provided underneath.

As for standard payment channel, to lose a challenge may imply that all the channel balance is awarded to the winning party as a form of punishment for the loser and simultaneously as a disincentive to issue wrong or invalid challenges that can be most likely lost.

Pending tokens at channel closure Once the channel closing procedure is triggered by any one of the two parties, a grace period starts in order to allow possibly pending tokens to be redeemed directly on-chain. A transaction has to be issued to take a pending token on-chain. This clearly involves transaction fees to be paid and determines an intrinsic economic incentive to contain the number of pending tokens. Ideally hot-refill and hot-withdrawal operations should be purely sequential: a new one should be triggered once the previous is completed, resulting in only one pending token at a time.

Channel closure attack The granted possibility to redeem pending token on-chain during the channel closing grace period may be used by an adversary to mount an irrational attack. It could exploit this time window to present on-chain a number of already off-chain attached tokens (*i.e.,* token generated to fulfill an hot-refill) pretending to not having redeemed them yet. Without any precaution, the honest party may be obliged to reply to each challenge, wasting the relative transaction costs. To tackle with this attack after the first lost challenge of this kind the attacker is punished losing all the balance in the channel, that is automatically closed.

4.4.2.3 Token re-attaching immunity

Hot-refill case Should the adversary propose to attach back a token which has been already attached previously, the honest party must refuse to sign the proposed state update. If the attacker try to challenge the honest party on-chain by exploiting the token attaching enforceability property described above, then the honest party must reply to the challenge producing the PoA it owns.

It implies that all the PoA produced during an hot-refill procedure must be saved by the counterparty and kept ready to be presented in case of a malicious on-chain challenge. An analysis of the required storing spaces is provided in Section 4.6.1.

Hot-withdrawal case In this case the adversary tries to withdraw the same token multiple time, by presenting it several times to the blockchain. Since at the first successful withdrawal attempt the blockchain permanently stors the token hash, and for each withdrawal request a test is performed to verify if the token has been already redeem or not, all the subsequent malicious attempts beside the first one are rejected.

4.4.2.4 Malformed token immunity

A malformed token is one that contains an amount different from the one subtracted from the balance at creation time. To attach a malformed token would obviously result in a violation of the balance conservation property. Malformed token are not considered a threat in this context. In fact the token generation is a supervised process either in the case of a hot-refill and hot-withdrawal. In the first case the token is generated out of the off-chain balance and has to be signed by the counterparty that must only consent to the detachment after a validity check of the proposed state update and the related detached token amount. In the second case, it is the blockchain itself that generates the token and, assuming that the smart contract does not contain errors, a malformed token is impossible to be generated.

4.5 Proof of concept implementation

Proof of concept implementation is based on the Ethereum platform. The smart contract that support the inextinguishable payment channel construction is written in Solidity language exploiting the Truffle[2] framework. Clients for channel endpoints are implemented in Node.js.

4.5.1 Overview of contract methods

Listing 4.1 shows the data structures and the signatures of the methods adopted for the smart contract that implements the inextinguishable payment channel.

```
contract InextiguishablePaymentChannel {

  enum MEMTEST {
    ABSENT, PRESENT
  }

  enum TokenPurpose {
    DETACH, ATTACH
  }

  enum State {
    INIT, RUNNING, CHALLENGING, CLOSING, CLOSED
  }
```

[2] http://truffleframework.com

```
struct Endpoint {
  address addr;
  string uri;
  uint balance;
}

struct StateUpdate {
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose, tokenPurpose
}

struct StateCheckpoint {
  uint timestamp;
  StateUpdate stateUpdate;
}

struct ChallengeData {
  uint timestamp;
  StateUpdate stateUpdate;
}

/***********/
/* channel */
/***********/

bytes32 channelId; // =: contract address

Endpoint A;
Endpoint B;

mapping (address => mapping (bytes32 => MEMTEST)) PoDs;
mapping (address => mapping (bytes32 => MEMTEST)) PoAs;

uint refillNonceCounter;

StateCheckpoint stateCheckpoint;
ChallengeData challenge;

uint challengeGracePeriod;
uint closingGracePeriod;
uint tokenArguingGracePeriod;

mapping (address => uint) drainableAmounts;

/***************/
/* constructor */
/***************/

function InextiguishablePaymentChannel (
  uint challengeGracePeriod,
  uint closingGracePeriod,
```

```
  uint tokenArguingGracePeriod
) {}

/*******************/
/* payment channel */
/*******************/

function join (string uri)
  payable isInit {}

/**************/
/* hot-refill */
/**************/

function hotRefill ()
  payable onlyEndpoints isRunning {}

function challengeAttach (
   uint seqNum,
   uint balanceA,
   uint balanceB,
   bytes32 tokenHash,
   TokenPurpose tokenPurpose,
   bytes signature,
   uint tokenNonce,
   uint tokenAmount
) onlyEndpoints isRunning {}

function replyAttach (
  bytes signature
) onlyEndpoints isChallenging {}

function replyAttachOldState (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature
) onlyEndpoints isChallenging {}

function replyAttachReplay (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature
) onlyEndpoints isChallenging {}

/******************/
/* hot-withdrawal */
/******************/
```

```
function hotWithdraw (
  uint tokenNonce,
  uint tokenAmount,
  bytes tokenSignature
) onlyEndpoints isRunning {}

function challengeDetach (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature,
  uint tokenNonce,
  uint tokenAmount
) onlyEndpoints isRunning {}

function replyDetach (
  bytes signature
  bytes tokenSignature
) onlyEndpoints isChallenging {}

function replyDetachOldState (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature
) onlyEndpoints isChallenging {}

function replyDetachReplay (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature
) onlyEndpoints isChallenging {}

/***********/
/* closing */
/***********/

function close (
  uint seqNum,
  uint balanceA,
  uint balanceB,
  bytes32 tokenHash,
  TokenPurpose tokenPurpose,
  bytes signature
) onlyEndpoints isRunning {}

function closeNoOffChainInteraction () onlyEndpoints isRunning {}
```

```
  function replyClosure (
    uint seqNum,
    uint balanceA,
    uint balanceB,
    bytes32 tokenHash,
    TokenPurpose tokenPurpose,
    bytes signature
  ) onlyEndpoints isClosing {}

  function redeemPendingRefillToken (
    uint tokenNonce,
    uint tokenAmount,
  ) onlyEndpoints isClosing {}

  function argueRedeemPendingRefillToken (
    uint seqNum,
    uint balanceA,
    uint balanceB,
    bytes32 tokenHash,
    TokenPurpose tokenPurpose,
    bytes signature
  ) onlyEndpoints isClosing {}

  function redeemPendingWithdrawalToken (
    uint tokenNonce,
    uint tokenAmount,
    bytes tokensignature
  ) onlyEndpoints isClosing {}

  function drain () onlyEndpoints isInitOrClosed {}

  /************/
  /* fallback */
  /************/

  function () { throw; }

  /*********/
  /* utils */
  /*********/

  function getTokenHash () internal {}

  function verifySignature () internal {}
}
```

Listing 4.1: Solidity contract interface for inextinguishable payment channel.

One instance of the contract supports one inextinguishable payment channel. Hence, the address of the contract can be adopted as `channelId` and used both in state and token tuple to avoid the discussed reply attack.

Funds are injected into the channel in the first place by the payable[3] `join` method. As it is implemented to work with native Ethereum currency, ETH, the amount injected is the transaction value. Future implementations that require to deal with ECR-20 compliant tokens must introduce an explicit parameter to indicate the amount of funds injection. The `hotRefill` method is also payable and represent the only additional way to inject funds into the channel.

Both `hotWithdrawal` and `drain` allow to extract funds from the channel, the first as the result of a hot-withdrawal process, the second after the channel is closed or in the case the counterparty never joins.

To pass a state update to a method the following parameters are required `seqNum`, `balanceA`, `balanceB`, `tokenHash`, `tokenPurpose`, `signature`. `signature` is the counterparty signature. It is the only one required to be explicitly provided, since the issuer signature is provided along with transaction itself. A token is passed using `tokenNonce` and `tokenAmount` parameters. The nonce for refill tokens is generated by the chain as an incremental number through the `refillNonceCounter`. Conversely, it is responsibility of the withdrawer to avoid nonce collision for withdrawal token as it is the withdrawer itself that generates the nonce off-chain.

The methods named `hotRefill` and `hotWithdraw` trigger the hot-refill and the hot-withdrawal of the channel. The refill procedure generate a PoD, while the withdrawal generates a PoA. They are stored on-chain in the data structures called `PoDs` and `PoAs` respectively, to be available afterwards in case the smart contract is called to resolve off-chain disputes. The gas model impose an efficient implementation for the lookup to check if a token relative to a PoA has already been redeemed on-chain, that in absence of precaution may be extremely expensive in terms of gas and, therefore, of transaction fee. For this reason a `mapping` type is chosen. It is a dynamically sized array, that can be thought of as a hash table and therefore allows for an efficient lookup and insertion of a not-predefined number of new elements. Only the `key` field of the `key` $\rightarrow$ `value` mapping is used for the purpose.

Most of the aforementioned security guarantees are obtained through interactive challenge schemes. Two different schemes are implemented: hot-refill off-chain token attachment and hot-withdrawal off-chain token detachment. Those schemes are triggered by `challengeAttach` and `challngeDetach` respectively. To lose a challenge results in the loss of the whole channel locked amount for the loser. These challenge scheme is an example of deterrent code. In fact the party that does not see his off-chain intent respected by the counterparty would most likely close the channel. However, a rational player, may result disincentivized to misbehave by the certainty of the punishment ensured by the challenge mechanism. This not applies in case of an irrational attacker that remains passive off-chain while still replies to on-chain challenges regardless the transaction cost they imply, therefore, obliging the honest one to take the burden of the challenge transaction costs. It is worth noting that `challengeDetach` is only accepted if the token is not already redeemed.

[3] A payable contract method is allowed to receive coins. Received coins are subsequently controlled by the contract logic.

`replyDetach` and `replyAttach` represent positive and collaborative answer to the challenge. Challenged party provide his signature or signatures to accept the proposed state update and detached token (only in the case of detaching procedure).

`replyDetachOldState, replyAttachOldState, replyDetachReplay, replyDetachReplay` serve to report malicious challenges and are good examples of deterrent code (see Section 3.5.5).

`replyDetachOldState, replyAttachOldState` are to be used to report a malicious challenge from an attacker that tries to fork the state update sequence by challenging the counterparty to attach or detach a token from an old state. The reply must show the signature of the attacker on a more recent state update (higher `seqNum`).

`replyAttachReplay` has to be invoked with a PoA testifying the token argument of the challenge has already been attached previously. `replyDetachReply` is to be used in the very particular case when the malicious challenger has detached a token but never withdrew it and challenge the honest party to detach that token again. The relative PoD presented to the chain resolves the dispute.

The closing procedure, triggered by the `close` method, create a checkpoint of the last state taken by the closer. The `closingGracePeriod` allows the counterparty to reply with a possibly newer state update. If the counterparty has a pending withdrawal token once the closing procedure has begun, it can be redeemed through `redeemPendingWithdrawalToken` method. `redeemPendingRefillToken` method, to redeem a refill token, must admit a `tokenArguingGracePeriod`, shorter than the `closingGracePeriod`, within which the `argueRedeemPendingTokenRefillToken` must be called by the counterparty to show a PoA of the very same token. The additional balance relative to the pending token is directly added to the state checkpoint balance.

Although not mandatory, for the sake of simplicity, the proof of concept implementation only allows one pending token at time (expressed by the `hasRedeemedAfterClosing` field of the endpoint structure).

Finally, the `drain` method allows counterparty to drain residual funds from the channel and `closeNoOffChainInteraction` allows one party to close the channel when the other one never signed a single off-chain state. `replyClosure` also counteracts malicious invocations of `closeNoOffChainInteraction`.

4.5.2 Overview of the client APIs

Inextinguishable payment channel construction is implemented as an extension of a payment state channel supported by an Ethereum smart contract. According to the propose/accept scheme (*cfr.* Section 3.5.1), client APIs is composed by a `propose` and an `accept` endpoint.

Although the proof of concept implementation only offers a minimal set of functionalities, the client is of pivotal importance for the adoption of this approach in the real world. The client must implement the execution of predefined policies that help in preserving the selected level of trust the user wants to adopt toward the

counterparty and therefore preserving the opportunity cost of locked funds. For example, the client may automatically triggers the closure of the channel when it observe the counterparty asks for a withdrawal of more than a threshold percentage of the total balance.

4.6 Usability

Off-chain solutions always require a certain degree of trust between involved parties. Although negligible if channels are collateralized with modest amounts, the cost opportunity of locked funds are always paid when an unresponsive counterpart obliges the honest one to close the channel and wait for the expiration of the closing grace period. For this reason, leaving untouched the security guarantees of the presented solution that prevents any involved honest party to lose its funds, it is however advisable to open an inextinguishable payment channel with a counterparty whose reliability is reputable.

Inextinguishable payment channels allows an interesting use case where a channel can be opened with an unknown and untrusted counterparty blocking a very small amount of funds. Collateral can be subsequently increased as the off-chain relation between party proceeds correctly and the mutual trust between parties increase as well.

4.6.1 Storage requirements

Whenever A performs an hot-refill of the channel, B must keep a copy of the related PoA until the definitive closure of the channel to possibly present it should A issue a malicious on-chain challenge or try to mount a channel closure attack (see Section 4.4.2.2). With reference to the proof of concept implementation presented in Section 4.5, the amount of space required to store a PoA is computed by the following expression. A PoA is composed by a state update tuple $< i, \beta_i^A, \beta_i^B, \mathcal{H}(\mathtt{tkn}_j), [\mathbb{A}|\mathbb{ID}] >$ along with the counterparty signature. Considering a signature of 65 bytes, unsigned integers of 32 bytes to store the first three fields of the token tuple, 32 bytes to store the token hash (the fourth element of the tuple) and overlooking the last field since only PoAs have to be stored, the size of a PoA sums up to 213 bytes[4]. 100MB of off-chain storage are therefore enough to secure more than 460K PoAs. This storage requirement is negligible if compared with the dimension of the whole Bitcoin blockchain (more than 127GB on October 2017).

4.6.2 Performance analysis

Performance analysis of inextinguishable payment channels are to be measured both from a systemic perspective and from the point of view of the endpoint of a channel.

[4]This size also contains 20 bytes for the channel id (the contract address) not shown in previous examples for the sake of clarity.

In a cooperative scenario, if a skewed channel has to be unnecessarily closed and reopened, a plain implementation requires a total of five on-chain transactions: one to instruct the closing, two for the withdrawal of funds (one for each party) once the grace period expires and finally two to open and fund a new channel. Sophisticated implementations may reduce this number by collapsing the withdrawals to only one on-chain transaction. Inextinguishable payment channel ensures the channel can continue to operate with only one on-chain transaction to perform the hot-refill for the depleted side. A second on-chain transaction could also be performed to withdraw funds realizing a configuration similar to the one obtained after a closing-and-reopening process. In this case three transactions can be saved to be processed by the underling blockchain.

An endpoint that, due to the depletion of its side of the channel, results unable to make further payments also benefits from the possibility of the hot-refill. It can save up to two transactions. In fact, in the closing-and-reopening process, it would be required to issue three on-chain transactions, one to instruct the closing of the channel, one to withdraw residual funds and a last one to join a new channel. On the contrary, an hot refill only require one on-chain transaction.

On the other hand, the hot-withdrawer saves one on-chain transaction, since the closing instruction is most likely in charge of the payer.

The on-chain transactions relative to the hot-refill and to the hot-withdrawal procedure cost slightly more than standard on-chain payment transaction but comparable to on-chain channel closing transaction. This is due to the gas model, where fee is paid for executed instructions and stored data, in conjunction with: 1) the more complex logic, which is required to verify the correctness of the counterparty signature on closing state, and 2) the storing of the hash of the token for on-chain generated PoA.

4.7 Future directions

Some enhancements for inextinguishable payment channels are planned for the next future. They are related to 1) the possibility of minimizing the storage requirements for PoA and 2) to the introduction of different type of collateral for the channel.

The mandatory requirement for the counterparty of an hot-refiller of storing the PoA can be overcome in the next future turning to a mechanism that provides membership test in constant time and space. This mechanism should be integrated in the state update as an additional field which is entitled to track PoAs produced during the life of the channel. Such a mechanism has been identified in the one-way accumulators [50, 51, 52, 53]. In their original formulation one-way accumulators are based on RSA's modular exponentiation scheme and therefore require the factorization of the modulus to be unknown to the party that produce the proof of membership. Therefore the application of this approach require two accumulators to be integrated in the state update tuple of the channel: each party initialize an accumulator and keep factorization private to itself. This enhancement would allow,

once implemented, to get rid of the necessity to permanently retain PoAs and only store the last state of the channel as for a standard payment channel.

For the sake of clarity, the detach/attach scheme has been presented taking for granted that parties collateralize the channel with the same currency, *i.e.,* sent and received payments are denominated in the same way. The scheme however allows party to collateralize the channel with any currency, even more than one. This possibility is especially interesting for the Ethereum ecosystem where multitude of different tokens are relentlessly been issued.

Chapter 5

Fulgur: hybrid trustless wallet

Contents

In this chapter the detach/attach scheme (see Section 4.3) is extended to support a hybrid trustless wallet. Wallet architecture encompasses a central hub that intermediates off-chain payments among its clients.

Trustlessness, conceived as the guarantee that no party can lose its own funds as long as it diligently follows the protocol, is backed by the extended detach/attach scheme that ensures atomicity for two-hops off-chain payments through the hub.

Hybrid payments allows new payment scenarios. Off-chain funds locked in a channel can be exploited to pay an on-chain recipients, either connected to the same hub or not (off-hub). Also the vice-versa is possible: the recipient of an on-chain originated payment can use it as a hot-refill for a payment channel already established.

This solution, as for payment networks, enhances the usefulness of locked funds decreasing their opportunity cost. If broadly adopted, this approach avoids several on-chain transactions, resulting in transaction fee saving for clients, and, from a systemic point of view, in an increase of the blockchain scalability.

The system design and the proof of concept implementation addressed the Ethereum platform. The presented solution, although discussed without any specific reference to the underling blockchain, is intended for Ethereum-like smart contract execution platform (*cfr.* 2.1.2).

This chapter is articulated as follows. Section 5.1 introduces the hybrid trustless wallet scheme from a general point of view, discussing project goals, motivations and related work. Section 5.2 goes into detail of the extended detach/attach scheme. Section 5.3 provides a security analysis of the system. Section 5.4 describes protocol usability. Finally, Section 5.5 reports about open issues and future directions.

5.1 Hybrid trustless wallet

Although a "wallet" is formally a software that manages private keys of a user, here the word is used as a real-world metaphor and refers to the software construction that helps in managing funds and making payments. The proposed solution does not handle keys of the user and assumes a software API is available to sign and broadcast transactions to the underling blockchain. Under these assumptions, the hybrid trustelss wallet solution allows to perform different kinds of (or hybrid) payments without the need to trust entities that intermediate the process.

Overall architecture Figure 5.1 shows the hub-and-spoke architecture of the system. The whole construction is backed by a smart contract that manages on-chain clients' balances and supports payment channels. Clients are represented by peripheral light gray point at the end of the spokes. Two clients, client A and client B are finer detailed: the figure shows their active payment channels with the hub and on-chain balances, the latter ones represented as gray circles encompassed in the smart contract light gray area.

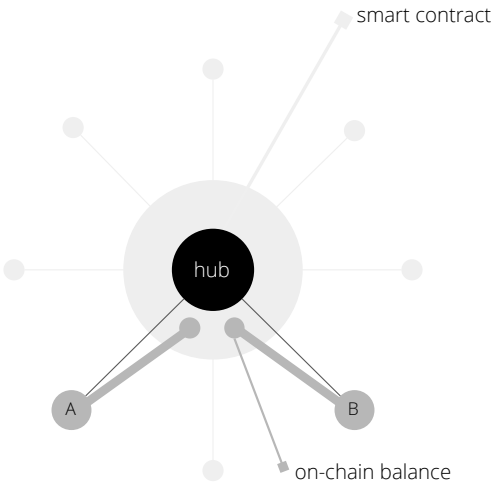

Figure 5.1: Overall architecture of the system.

Hub The hub is supported by a trustworthy software that interacts with the underling blockchain and smart contract. Since the central hub defines a single point of failure, it should be redundantly deployed to the purpose of supporting reliability.

An entity that can fund a large enough number of clients' channels, can decide to run a Fulgur hub, incentivized by fees it earns for the offered brokerage service. Entities with such availability of tokens are rapidly appearing in the cryptocurrency ecosystem due to the ICO (Initial Coin Offering) phenomenon[1].

Section 5.4.4 introduces the fee model. Fees are due to the hub by its clients on a per-transaction basis and also to repay the capital advance the hub has to sustain to fund channels. This second type of fee can be conceived as a subscription to the hub brokerage services. It is collected according to a reactive protocol each time a client perform an on-chain operation on the supporting smart contract (or through an ad hoc contract method if no on-chain operations are performed by the client). If a client has not settled its position since a predefined amount of time, the hub can ask to terminate the relationship with that particular client. This is the only case where the hub is entitled to trigger a channel closing. If the client is regular in its payments or simply exploits on-chain hybrid payments, the hub can't deliberately close the channel.

Conversely the hub can decide to stop operating as a hub. In this case it notifies its willing and clients has a predetermined, long enough (*e.g.* weeks) amount of time

[1]It is not uncommon nowadays to witness players that after a successful ICO collect huge amount of funds from the offering of self-minted new kind of tokens. Since the initial offering is often only partial with respect to the total coinbase generated, a great quantity of tokens remains available to project owners. Those may be ideal candidates to run a Fulgur hub, since they can earn from being in the position to sustain the required capital advance, as detailed in the next sections.

to close their channels. During this time, the hub must ensure regular support to client operations.

Clients The relationship between a client and the hub can be thought of as a "trustless subscription" of the client to the brokerage services offered by the hub. It is supported by a software that interacts with the underling blockchain and smart contract. Such a subscription involves a special inextinguishable payment channel (as presented in the previous chapter) and allows a client to exploit hybrid payments. As detailed in Section 5.2 hybrid payments offer chap and fast off-chain hub-intermediated payments toward other clients of the same hub and the possibility to use channel locked funds to make and accept payments to and from entities which have not a subscription with the hub.

A client can end the trustless subscription to the hub services any time. In particular it must be closed, along with the channel, as soon as a client detects unresponsiveness from the hub, which can be read as a signal of uncooperative behavior.

Smart contract The smart contract that supports the Fulgur architecture guarantees the trustless relationship between a client and the hub. In particular, it has to be resorted to if something goes wrong in the off-chain relation between parties. Furthermore, it has several accessory though important responsibilities, it handles: 1) the opening and closing of a client subscription, 2) hybrid payments that involve on-chain and off-hub endpoints, 3) hub fee collection and 4) the end of activity of the hub.

5.1.1 Design goals

In what follows are introduced the main design goals.

Trustless payments The essential requisite for a wallet is the certainty, ensured by the protocol, that honest parties can not lose any funds. This guarantee is often considered by the crypto-community to suffice in define the system as "trustless" (it is adopted for example by the Lightning Network construction [45]). In this sense, Fulgur does offer trustless payments, and diligent players that stick to the protocol described underneath are never defraud of their funds.

Hybrid payments Hybrid payment is a concept that derives from Fulgur architecture. A central hub acts as a payment gateway. All the clients connect to this hub through a payment channel. Every client is provided with two kind of balances, one on-chain, managed and secured by the supporting smart contract, and one off-chain, handled as a payment state channel. Figure 5.2 depicts hybrid payments capabilities. Payments can originate from the off-chain balance or the on-chain balance of a client of the hub and from an off-hub entity which is able to interact with the

supporting smart contract. These three origin points can be also the destination of a payment, determining the following eight types of hybrid payment:

1. on-chain ⇸ on-chain payments,
2. off-chain ⇸ off-chain payments,
3. off-chain ⇸ on-chain payments,
4. on-chain ⇸ off-chain payments,
5. on-chain ⇸ off-hub payments,
6. off-chain ⇸ off-hub payments,
7. off-hub ⇸ on-chain payments,
8. off-hub ⇸ off-chain payments.

Hot channel refill and withdrawal Provided that the interaction between a client and the hub has to continue, the channel must not be closed only because it is skewed and the depleted side can not make any further payment. An extended version of the detach/attach scheme supports hot refills of and hot withdrawals from the channel.

Reactive hub A strict client/server architecture is imposed to the system to the purpose of preserving clients' privacy. This choice implies that no information about user location (client's node IP address or URL) has to be shared with the hub. The hub is then not able to directly contact any client but it can only reply to a request of a client. A client node is not required to be always on-line but only contact the hub when needed. Furthermore, if the secrecy of physical address is a major concern, the node can avoid its disclosure by changing it at every interaction (relying for example on proxy services or onion routing).

Application level It has to be possible to implement the system as an application on top of existing technologies. The blockchain governance issues would make impossible to include any modification of the underlying blockchain protocol. To keep the design only at application level provides guarantees of the feasibility of the proposed approach.

5.1.2 Motivations

Tackle with centralization at design time As detailed in Section 3.4.1, one of the most promising solution to the blockchain scalability issue is commonly considered to be the deployment of an overlay network of (micro-)payment channels. However, several concerns arose [54, 55, 56, 57] mainly about the unfeasibility of

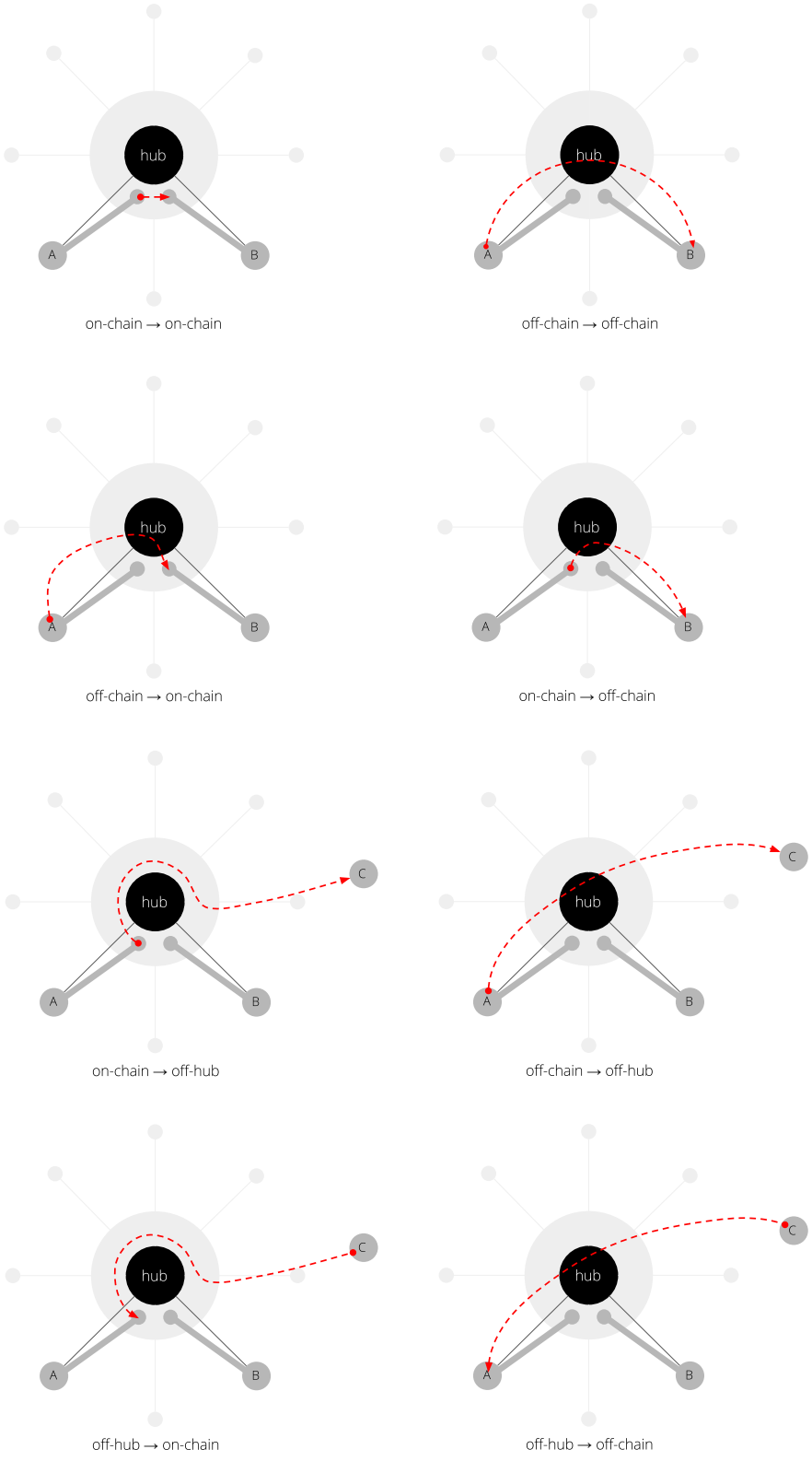

Figure 5.2: Capabilities of Fulgur hybrid payments.

economic routing of multi-hop payments. Regardless of whether these positions will be definitively proved as valid or not, what is certainly undeniable is the intrinsic centralization power of capital forces. Since to open a channel requires to immobilize funds the opportunity cost, of this action can be optimized by means of economies of scale. In the next future it will be likely to witness a first phase for payment networks where an initial hub-and-spoke topology will emerge anyway, precisely by virtue of the aforementioned capital optimization forces. In the first place for a connectivity argument: at the beginning the network will be represented by a disconnected graph and a well connected hub will encourage the establishing of a payment route. In the second instance, once the connectivity problem will be hopefully overcome, a payment will result cheaper if only routed through a well connected hub (two hops required, fee paid for each crossed node) instead of possibly pass trough several hops of a payment route resulting from a network with not predefined topology. To have centralization in mind at design time allows to tackle with the unwanted aspects of centralization and at the same time to take advantage of the simpler architecture.

Hybrid payments to decrease opportunity cost of locked funds The extended detach/attach scheme plays the same role of Hash TimeLock Contracts (HTLC) in multi-hop payments over payment networks: it ensures payment atomicity. With respect to HTCL, the proposed solution sacrifices some privacy to the purpose of achieving hybrid payments which help to maximize the usefulness of funds locked in a channel. In HTLC, in fact, payment enforceability is subject to the knowledge of the preimage of the application of a hash function, which guarantees payment untraceability. On the contrary, a detached token generated according to the detach/attach scheme, clearly reports the recipient address.

5.1.3 Related work

The analysis of related work is conducted taking into account six dimensions: centralization, trustlessness, privacy preserving, versatility of locked funds and efficient funding. They represent the most crucially aspects for a system that aims to support electronic money with efficient structures like payment channels.

E-cash system by David Chaum [11], one of the very first attempt to define electronic money, was a centralized trusted solution supported by a blind digital signature scheme that makes payments untraceable by the central server (which most likely had to be an institution like a bank for example).

In [58] a solution is proposed to conjugate benefits of the Chaum's approach with payment channels, exploiting blind signatures and efficient zero-knowledge proofs. The resulting construction is trustless and privacy preserving. Multi-hop payments are also allowed to enhance versatility of funds locked in channels. Unfortunately, the channels securing mechanism adopted, inhomogeneous with respect to the underlying blockchain, does not permit hot-refills and hot-withdrawals. Therefore a channel has to be entirely funded at opening time and a skewed channel has to be closed and reopened.

Tumblebit [29] is an anonymous payment hub thought for the Bitcoin blockchain. The centralized proposed approach offers anonymity properties similar to the E-cash system with enhanced trustlessness. Users perform off-chain payment through the hub without the need to trust it. A fair exchange protocol ensures atomicity of each payment. The particular payment channel structure adopted is unidirectional and have predefined lifetime since the protocols run in epochs.

CoinBlesk [59] is a bitcoin wallet that uses a central server to realize virtual payments: realtime payments of small amount that originate from a balance the client deposits at the server, which is also entitled to handle it in behalf of the client. Albeit very useful, the adopted approach for virtual payments can not be considered trustless.

An interesting economic incentive-driven centralized solution is been introduced by Plasma [60]. It aims to scale transactions by creating side chains that only interact with the main chain every once in a while. Centralization is tackled with economic bond and secured ex-post by "fraud proofs".

A solution for scalable funding of Bitcoin payment channels has been proposed in [61]. Many parties can jointly block funds at the cost of only one on-chain transaction. These blocked funds can be used to collateralize all the possible channels between a couple of blocking parties. Furthermore, as long as perfect cooperation holds among all the involved parties, rebalancing of skewed channels can happen. Although the solution produces great scalability, perfect agreement between parties, especially if many of them are involved to support maximum scalability, may be a not trivial condition to realize. Furthermore, no external funds can be injected once the funding took place.

5.2 Extended detach/attach scheme

It is an extension of the approach presented in Section 4.3 where there are three players involved. With reference to the overall architecture depicted in Figure 5.1, they are the two clients, A (Alice), the issuer of the payment, B (Berto), the receiver of the payment, and the hub H. The payment channel (A, H) is established between A and H. Similarly (B, H) is the payment channel between B and H. As for the plain version of the scheme, a Proof of Detachment (PoD) enforces that a payment is fulfilled and a Proof of Attachment (PoA) guarantees that token attachment procedure is executed only once. Balance conservation and payment atomicity, that guarantee hub and clients against the loss of funds, are supported by PoD and PoA: an interactive protocol that involves the supporting smart contract ensures that exactly one PoA is produced for each PoD. The system model is same assumed for state channels and described in Section 3.2.

On-chain data structures For each generic client C, and therefore for each channel (C, H) of the hub, the supporting smart contract has to handle some specific data structures: the on-chain balance $\boldsymbol{\beta}^C$; the current capacity of the channel $\boldsymbol{C}^{(C,H)}$,

which, in a non-transitory state, is the sum of the off-chain endpoints' balances β^C and β^H; a registry $\boldsymbol{R}^{\text{off}\to\text{on}}$ of tokens originated off-chain and attached on-chain; a registry $\boldsymbol{R}^{\text{on}\to\text{off}}$ of tokens originated on-chain and attached off-chain.

Symbolism The following symbolism is adopted in the text. A state update is described by a tuple of five elements enclosed in angle brackets, *e.g.* $< i, \beta_i^A, \beta_i^H, \mathcal{H}(\texttt{tkn}_j), \mathbb{ID} >$. The first one represents the sequence number of the state; the second and the third elements are the balances of the endpoints, possibly referring a previous value as in the example β_A^{i-1} which represents the balance A had at the state $i-1$; the fourth element is the hash of a token which, as indicated by the fifth element can be detached ($\mathbb{ID}$) or attached ($\mathbb{A}\backslash$) in the current state.

A token is represented by a tuple of four elements, *e.g.* $< j, \alpha_j, \text{ID}(H), t_{\text{exp}}, \mathbb{OFF} >_{(\sigma^A, \sigma^H)}$. The first one is the token nonce, which can be easily thought of as a sequence number; the second is the amount α isolated inside the token; the third element is a verifiable id of the payment receiver (an Ethereum address for example); the fourth element is the hash of a token which, as indicated by the fifth element can be detached ($\mathbb{ID}$) or attached ($\mathbb{A}\backslash$) in the current state. The fourth element indicates the token expiration time; the fifth element states the token type: whether it is redeemable off-chain ($\mathbb{OFF}$), on-chain ($\mathbb{ON}$) or to be sent off-hub ($\mathbb{EX}$, $\widehat{\mathbb{OFF}}$, $\widehat{\mathbb{ON}}$). The fourth and the fifth elements of the token tuple, the id of the receiver and the token type respectively, ensure that the supporting smart contract can correctly evaluate possibly pending token at channel closing time.

Signatures are represented as subscript of the structure they seal. The expression $< i, \beta_A^i, \beta_H^i, \dots >_{(\sigma_A, \sigma_H)}$ indicates that state update is signed by both A and H. Not needed or obvious fields are possibly omitted.

For the sake of clarity, the hub id is permanently omitted, both from state update and token tuple. However it is essential to avoid collision in PoA and PoD generation, that in turn may affect the trustlessness of the whole system. As an additional simplification of the notation, the fee that remunerates the hub for its brokerage service is not mentioned. Section 5.4.4 provides details on the fee model.

5.2.1 Hybrid payments

Hybrid payments supported by the Fulgur wallet can be divided into three categories: homogeneous payments, mixed payments and external payments. The categorization is based on the combination of origin and destination balance types (on-chain, off-chain or off-hub).

5.2.1.1 Homogeneous payments

Homogeneous payments originate from and are directed toward the same kind of balance. Possible configurations are on-chain ⇸ on-chain and off-chain ⇸ off-chain.

On-chain ⇾ on-chain payments This is the most simple configuration to handle, since it is totally delegated to the supporting smart contract. Assuming that the j-th payment originated by A is of an amount of α_j, then the output of the procedure is the update of the on-chain balances of both payer A and payee B. In particular, $\boldsymbol{\beta}_j^A = \boldsymbol{\beta}_{j-1}^A - \alpha_j$ and $\boldsymbol{\beta}_j^B = \boldsymbol{\beta}_{j-1}^B + \alpha_j$

Off-chain ⇾ off-chain payments This configuration implies the generation of a pair of PoD and PoA for the channel (A, H) and one for the channel (B, H). PoDs and PoAs used in this case are slightly modified versions of the ones introduced in the previous Chapter (see Section 4.3). In this version, it is H that actually pays B. A, in turn, precommits to refund Has soon as a proof is shown to certify that the payment from H to B has been accomplished. The steps of the payment protocol are detailed underneath.

Payer off-chain token detachment The first PoD is produced on the (A, H) channel. Let assume that $< i-1, \beta_{i-1}^A, \beta_{i-1}^H, \dots >_{(\sigma^A, \sigma^H)}$ is the last valid state of the channel (A, H) before the new payment begins. A detaches the amount she wants to pay to B out of her off-chain balance on the channel (A, H). This action results in an update of the state of the channel (A, H) and in the generation of a PoD. The new state is described by the tuple $< i, \beta_{i-1}^A - \alpha_j, \beta_{i-1}^H, \mathcal{H}(\mathtt{tkn}_j), \mathbb{ID} >_{(\sigma^A, \sigma^H)}$, while the PoD is represented by the j-th token $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OIFIF} >_{(\sigma^A, \sigma^H)}$ signed by both A and H. The detaching procedure is triggered by A that contacts the hub which is listening for new incoming connections. It is worth noting that A's balance is decremented by the token amount while the hub's one is not incremented yet: part of the balance is detached from this channel and isolated in a token that can be sent to the receiver.

PoD conveyance A's PoD is sent by A itself to B, the receiver of the payment. PoD conveyance has to take place over an out-of-band communication channel. This means that the bytes describing the PoD can be post via any medium, from an SMS to a handed off piece of paper on which bytes have been hand written. Although relying on a RESTful API is the most convenient approach, once the sender has generated and does own a PoD, no Internet connection is required for him to fulfill this step.

Payee off-chain token attachment Once the payee has received the PoD $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OIFIF} >_{(\sigma^A, \sigma^H)}$ generated by A, he has to check validity of signatures and attach the token amount to his off-chain balance of the channel (B, H). To accomplish this task, B interacts with H proposing a new state update. Being receiving a payment, B is economically incentivized to trigger this phase of the protocol. Assuming the last agreed upon state on the channel (B, H) was $< k-1, \beta_{k-1}^B, \beta_{k-1}^H, \dots >_{(\sigma^B, \sigma^H)}$, the new state becomes $< k, \beta_{k-1}^B + \alpha_j, \beta_{k-1}^H - \alpha_j, \mathcal{H}(\mathtt{tkn}_j), \mathbb{A} >_{(\sigma^B, \sigma^H)}$. This also constitutes a PoA of the

token to B's off-chain balance. The hub at this point is economically exposed but owns a PoA from B that proves payment fulfillment to the supporting smart contract (if a dispute arises). On the other hand, H needs also to show to A that the payment toward B has been completed. Although PoA could serve this purpose as well, this would entail to reveal the state on the channel (B, H) to A. To avoid this information leak, a payment receipt (PR henceforth) is sent by B to H along with the proposition of the state update that attaches the token. A PR is constituted by a signature of B on the attached token and represents an undisownable proof that the payment has been accepted by B. H can show a PR to A without disclosing any information about the state of the channel (B, H).

It is worth noting that the payment is deemed final in the moment H receives from B the PR along with the proposition of a state update where the token is attached. The hub, at this point, owns a PR but has not provided its signature on the updated state yet. A possibly uncooperative behavior of the hub configures a security threat analyzed in Section 5.3.

Payment Receipt conveyance (optional) If B is completely cooperative and a communication channel exists between B and A, B may decide to send back to A a payment receipt (henceforth PR).

If a PR is returned back to the payer, the subsequent phases of the protocol result simplified. This step, however, is not mandatory and the protocol succeeds regardless the collaboration or the actual ability for the receiver of sending back a PR, since also the hub owns the same receipt.

Hub off-chain token attachment The last phase of the protocol allows the hub to rebalance its exposed position toward A. It can happen according two different paths, depending on whether B has returned or not a PR to A. Regardless of which path is actually chosen, the conclusive state for the channel (A, H) is going to be $< i+1, \beta^B_{k-1} - \alpha_j, \beta^B_{k-1} + \alpha_j, \mathcal{H}(\mathtt{tkn}_j), \mathbb{A} >_{(\sigma^A, \sigma^H)}$, which also contains a PoA of the j-th token useful for A to defend himself on-chain against a malicious behavior of H.

Let assume the first scenario, the one where B has actually sent a PR to A. In this case it is A itself that contacts the hub proposing the aforementioned state update to conclude the payment process. A is not economically incentivized in an explicit way to propose the conclusive state update to H. The hub is only reactive to clients' requests and therefore unable to trigger this phase by itself. However, the hub, being in a exposed position, must refuse to fulfill any further request from A until its exposed position is finally settled. If A decides to not settle despite owning a PR by B, she is blocked and is not allowed to use her off-chain funds in any other way. The only thing she can do is to close the channel, which already reports her correct balance. Conversely, the balance of H is still missing the token amount, but it owns a PoA from B which can present during the closing grace period of the channel, thus avoiding to lose funds. Hence, it is not irrational for A to trigger this

phase and settle the payment process with H and therefore being ready to perform a new payment as soon as the need arises.

Assuming the second scenario, the one where B has not sent a PR to A, she can behave in two ways. Assuming a collaborative behavior, she asks every now and then to H whether B attached the token or not. She proposes to settle as soon as the answer is positive and a PR is presented by H. Should the answer be negative once the expiration time t_{exp} has passed, A can cancel the payment, as detailed in the next paragraph.

Alternatively, A ignores the exposed position of the hub until the next interaction between them, *i.e.,* a payment has to be received or sent. To proceed with collaboration with the hub, she has to settle the previous pending payment with H. This not implies that payments are sequential: the payer can have several detached (also called pending) tokens even if she is obliged to attach them back as soon as the hub answers to a request presenting a PR, otherwise she can't count anymore on hub's collabouration in making any further payment.

The hub is only virtually exposed: funds are however locked in the channel(s) and can not be stolen. To settle its position as soon as the payee attaches the token or when the payer needs to send or receive a further payment is economically indifferent for the hub, which is nevertheless guaranteed to not be defrauded of these locked funds.

Payment cancellation The payer can cancel the payment and withdraw the pending token. This require an interaction with H which only grants this possibility to A if B has not already redeemed the token, working as a synchronization point in between the payer and the payee. If B has already redeemed the token with the hub, it owns the PR that presents to A and proceeds with the conclusive settle of the payment. Furthermore, H must refuse a cancellation request also if the token expiration time t_{exp} has not passed yet (as detailed in Section 5.2.4.2).

To cancel a payment the relative token must be attached back to the channel from where it has been generated. Say for example that A decides to cancel the payment associated with her j-th token. She proposes an update of the channel (A, H) to H represented by the tuple $< i+1, \beta^B_{k-1}, \beta^H_{k-1}, \mathcal{H}(\texttt{tkn}_j), \mathbb{A} >$. Once mutually signed it constitutes a PoA useful to counteract malicious on-chain challenges. The amount of the canceled token results added back to the balance of A in the last state update of the channel.

5.2.1.2 Mixed payments

Every clients C owns two kinds of balance: an off-chain (β^C) and an on-chain one ($\boldsymbol{\beta}^C$). The mixed payment scheme enables two possibilities. It allows a client to 1) send a payment from its on-chain balance to the off-chain balance of the receiver ($\boldsymbol{\beta}^s \twoheadrightarrow \beta^r$) and 2) to have a payment originated from the off-chain balance of the sender, conveyed to the on-chain balance of the receiver ($\beta^s \twoheadrightarrow \boldsymbol{\beta}^r$). The off-chain balance is locked in the channel every client has opened with the hub. In order to

move funds from and to this kind of balance a client has to interact with the hub, which in turn has to check and agree on every clients' off-chain operation in order to guarantee its own security. The on-chain balance, on the other hand, is under the exclusively control of the client, which can manage it by interacting with the smart contract that supports the whole architecture.

Off-chain ⇸ on-chain payments For this kind of payments the protocol requires an off-chain detached token to be attached on-chain. This two-phase procedure is detailed in what follows.

Payer off-chain token detachment This step is similar the homonym one from the off-chain ⇸ off-chain payment case. At the end of this step the channel (A, H) is in the state $< i, \beta^A_{i-1} - \alpha_j, \beta^H_{i-1}, \mathcal{H}(\mathtt{tkn}_j), \mathbb{ID} >_{(\sigma^A, \sigma^H)}$ and the j-th token $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OIN} >_{(\sigma^A, \sigma^H)}$ has been detached, which also constitutes the PoD to convey to the receiver. As remarked by the last element of the tuple, in this case the detached token is only redeemable on-chain.

On-chain token attachment A presents the token to the smart contract that, after some validity and correctness checks, updates the interested data structures as follows. Capacity of the channel (A, H) is decremented by the payed amount α_j, $\boldsymbol{C}^{(A,H)}_j = \boldsymbol{C}^{(A,H)}_{j-1} - \alpha_j$. The on-chain balance of B is incremented by α_j, $\boldsymbol{\beta}^B_j = \boldsymbol{\beta}^B_{j-1} + \alpha_j$. The hash of the token is stored in the relative registry, $\mathcal{H}(\widehat{\mathtt{tkn}_j}) \hookrightarrow \boldsymbol{R}^{\mathrm{off}\to\mathrm{on}}$. Stored entry also represents the PoA. In fact, the smart contract, before to attach the token, checks that $\boldsymbol{R}^{\mathrm{on}\to\mathrm{off}}$ does not already store the token hash, aborting the payment otherwise[2].

It is wort noting that the state of the channel (A, H), does not need any further update. The last state on which both A and H agreed upon already report correct balances for both parties: $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OIFF} >_{(\sigma^A, \sigma^H)}$.

On-chain ⇸ off-chain payments To realize on-chain ⇸ off-chain payments, a token is detached on-chain and attached off-chain.

Payer on-chain token detachment Since the payment originate from the on-chain balance of A, the detachment procedure is performed on-chain. A provides the smart contract with information required to generate the j-th token: the amount α_j and the id of the payee $\mathrm{ID}(B)$. The contract: 1) virtually assembles the token $\mathtt{tkn}_j =< j, \alpha_j, \mathrm{ID}(B), \perp, \mathbb{OIFF} >$, where no expiration time is set; 2) stores the token hash in the registry: $\mathcal{H}(\mathtt{tkn}_j) \hookrightarrow \boldsymbol{R}^{\mathrm{on}\to\mathrm{off}}$, which also constitutes the PoD; 3) updates the on-chain balance of A such that $\boldsymbol{\beta}^A_j = \boldsymbol{\beta}^A_{j-1} - \alpha_j$; 4) updates the payee's channel capacity: $\boldsymbol{C}^{(B,H)}_j = \boldsymbol{C}^{(B,H)}_{j-1} - \alpha_j$.

[2]Efficient lookup for on-chain stored PoAs is implemented via a `mapping` type of the Solidity language, analogously to what described for in Section 4.5.

In this case, along with the hash of the token, also the payment amount and the payee's id are stored by the registry so that the payee learns about the payment observing the chain (which he is always monitoring according to the off-chain architectures security model). No PoD conveyance is therefore required.

Payee off-chain token attachment This step proceeds similarly to the corresponding off-chain $\twoheadrightarrow$ off-chain case but the balance of the hub is not touched by the update. Assuming the channel (B, H) was in the state $< k-1, \beta^B_{k-1}, \beta^H_{k-1}, \dots >_{(\sigma^B,\sigma^H)}$, after the attachment procedure has been completed, it becomes $< k, \beta^B_{k-1} + \alpha_j, \beta^B_{k-1}, \mathcal{H}(\mathtt{tkn}_j), \mathbb{A} >_{(\sigma^B,\sigma^H)}$.

5.2.1.3 External payments

This kind of payments allows for a client to send funds to and to receive funds from outside the hub. A crucial features, granted by Ethereum-like smart contract execution platform, is the possibilities for the smart contract to securely move funds on-chain. For the sake clarity, as shown in the bottom part of Figure 5.2, the off-hub sender or receiver of the payment is considered to be Carlo (C).

On-chain $\twoheadrightarrow$ off-hub payments A (a client of the hub) wants to pay C (which is not a client of the hub) from its on-chain balance. A informs the smart contract about the amount of the j-th payment α_j and address of the receiver ID(C). The smart contract, atomically updates the on-chain balance of the payer such that $\boldsymbol{\beta}^A_j = \boldsymbol{\beta}^A_{j-1} - \alpha_j$ and sends the amount to the payee. Relaying completely on the smart contract, no additional security concerns arise in this situation.

Off-chain $\twoheadrightarrow$ off-hub payments A (a client of the hub) wants to pay C (which is not a client of the hub) from its off-chain balance. H is involved in the interaction to initially detach the token. Details on the phases of the protocol are provided underneath.

Payer off-chain token detachment This step is similar the homonym one from the off-chain $\twoheadrightarrow$ off-chain payment case. At the end of this step the channel (A, H) is in the state $< i, \beta^A_{i-1} - \alpha_j, \beta^H_{i-1}, \mathcal{H}(\widehat{\mathtt{tkn}_j}), \mathbb{ID} >_{(\sigma^A,\sigma^H)}$ and the j-th off-hub token $\widehat{\mathtt{tkn}_j} = < j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \widehat{\mathbb{OIN}} >$ has been detached. The token $\widehat{\mathtt{tkn}}_{j,(\sigma^A,\sigma^H)}$ signed by the both channel endpoints constitutes the PoD to be taken on-chain to execute the payment.

On-chain payment execution Payment execution starts when A presents the PoD to the smart contract which: 1) creates a PoA by storing the token in a registry, $\mathcal{H}(\widehat{\mathtt{tkn}_j}) \hookrightarrow \boldsymbol{R}^{\mathrm{off}\to\mathrm{on}}$; 2) decreases the channel capacity $\boldsymbol{C}^{(A,H)}_j = \boldsymbol{C}^{(A,H)}_{j-1} - \alpha_j$; 3) sends the amount to C. The presence of the token hash inside the registry is checked at insertion time to guarantee that a token can not be used more than once.

Off-hub ⇸ on-chain payments The smart contract completely backs this case. The external payer C send an amount of α_j and the on-chain balance of the receiver B is updated consequently: $\boldsymbol{\beta}_j^B = \boldsymbol{\beta}_{j-1}^B + \alpha_j$.

Off-hub ⇸ off-chain payments Similarly to the on-chain ⇸ off-chain payment, the smart contract virtually creates a token which is later on attached off-chain by the receiver of the payment.

On-chain token creation C sends the amount α_j to the smart contract that in turn: 1) virtually assembles the token $\widehat{\texttt{tkn}}_j =< j, \alpha_j, \mathrm{ID}(B), \perp, \widehat{\mathbb{OFF}} >$, where no expiration time is set; 2) stores the token hash in the registry: $\mathcal{H}(\widehat{\texttt{tkn}}_j) \hookrightarrow \boldsymbol{R}^{\text{on}\to\text{off}}$; 3) updates the payee's channel capacity: $\boldsymbol{C}_j^{(B,H)} = \boldsymbol{C}_{j-i}^{(B,H)} - \alpha_j$.

Also in this case the payee learns about the incoming payment monitoring the blockchain.

Payee off-chain token attachment This step proceeds similarly to the corresponding off-chain ⇸ off-chain case but the balance of the hub is not touched by the update. Assuming the channel (B, H) was in the state $< k-1, \beta_{k-1}^B, \beta_{k-1}^H, \dots >_{(\sigma^B, \sigma^H)}$, after the attachment procedure has been completed, it is $< k, \beta_{k-1}^B + \alpha_j, \beta_{k-1}^B, \mathcal{H}(\texttt{tkn}_j), \mathbb{A} >_{(\sigma^B, \sigma^H)}$.

5.2.2 Channel closing conditions

Channel closing is the most critical aspect that needs to be carefully handled. In fact, most of the security of the system depends on the correct conclusion of the interaction between a client and the hub. A channel can be closed either cooperatively or not.

For the cooperative case, two reasons may trigger the closing: 1) the client does not longer need to use the channel; 2) the hub has signaled is going to shutdown, and clients are moving away from the hub, which cooperates in closing all the channels.

Two reasons as well entail the closing of a channel with no cooperation between a client and the hub: 1) a client has noticed unresponsiveness from the hub; 2) the hub detects a client has not settled its subscription for an amount of time that exceeds the agreed terms. This situation may occur if the client has neither performed any on-chain operation in a while nor she has paid the due fees by explicitly invoking the specific methods offered by the smart contract. Not cooperative channel closing is clearly the most delicate case, that involves the checkpoint of the channel state on the blockchain, with the possibility to rebut it, and the handling of possibly pending tokens.

Pending tokens Once the closing of a channel has been triggered, it may be the case that a client has some pending payments and the hub has some pending positions. Client pending tokens can be of two kinds:

payer pending tokens tokens that have been detached by the client herself but not used yet;

payee pending tokens tokens received by the payee as an off-chain headed payment but not attached yet to his balance[3].

A hub pending position derives from an off-chain ⇀ off-chain payment for which the hub has already paid the receiver, and thus owns the PR, but it is still exposed toward the sender.

5.2.3 Smart contract requirements

The smart contract underling the Fulgur architecture has to implement the data structures and methods previously introduced that support on-chain operations required by the extended detach-attach scheme. Operations such as on-chain balance handling, on-chain token detachment and attachment are analogous to the related ones presented in the previous chapter. Two critical interaction with the smart contract are discussed underneath: state checkpointing and on-chain redeem procedure for pending tokens.

5.2.3.1 Channel state checkpoint and rebuttal

As for a standard payment channels, a snapshot of the last state on which parties agreed upon off-chain has to be checkpointed to the smart contract at closing time to reconcile the off-chain with the on-chain history. If a client tries to cheat presenting an old state more favorable to her, a closing grace period allows the hub to rebut with the actual last state. The same holds also for a cheating hub that tries to close the channel with a cheating state after the client's subscription expires and the hub can rightfully close the channel.

Contextually with the rebuttal that follows a malicious closing of a channel, the smart contract assumes elements to ascribe blame to the cheating client or hub. On the bases of those evidences it can punish the malicious party by allocating the whole amount of funds locked in the channel by both parties to the honest attacked one.

As a form of additional punishment, hence as a further deterrent mechanism, also the on-chain balance of the client can be assigned to the hub in case of a malicious behavior. The hub has not an on-chain balance and therefore the vice versa does not apply[4]. Although it is true that a rational adversarial client would certainly withdraw all the on-chain balance from the account before to mount the attack, it would be the signal for the hub to put such a client under "special observation".

[3] Inbound payments directed toward either an on-chain balance or off-hub are not an issue since the smart contract (assumed secure and bug free) is entitled to handle them.

[4] This asymmetry is consistent with a client being the only one that can deliberately close a channel. The hub, in fact, can only remove a client as a result of her lateness in paying the due subscription.

State checkpoint as single-party claim Once a party checkpoints a channel state on the smart contract, she is implicitly stating that she is satisfied to withdraw the balance that the checkpointing state assigns to her. In addition to this amount, she can take to the smart contract some possibly pending tokens, as described by the following paragraph. If no dispute arises on state checkpoint and pending tokens, the smart contract recognizes to the other party the difference between the channel capacity stated on-chain and the total amount notified by the closing party.

This approach is similar to a two-phase accounting system and can be therefore implemented as a single signed integer that reports the balance of only one party, while the other one can be computed for difference from the channel capacity reported (and keep updated) on the smart contract.

5.2.3.2 On-chain pending tokens redemption

The smart contract has to support the *on-chain redeem procedure for pending tokens* to resolve the regrettable situation where a channel closing is initiated when one or more pending tokens still exist. Since the contract knows nothing about the off-chain interaction, an interactive scheme supports the procedure that may happen only during the channel closing grace period.

Once a pending token has been presented to the smart contract, the counterparty has an arguing period, that ends with the closing grace period, to possibly produce a PoA for that token. If the PoA is not presented within the arguing period, the token amount is considered withdrawable by the party who presented it. Conversely, if a valid PoA is presented by the counterparty within the time constraint, the party who maliciously tried to redeem the token is punished and all its funds blocked in the channel are assigned to the other one.

While a payer pending token can be redeemed on-chain even after its expiration, this is not true for a payer pending token. The smart contract only accepts a token from the payee for on-chain redemption if it is not already expired.

Implementation remarks On-chain redeemed tokens are stored in an easy-to-retrieve data structure as soon as they are presented to the smart contract. In this way there is no need for a client to issue any further transaction to conclude the on-chain redeem procedure. One final transaction is eventually required to drain out client's funds from the contract (*i.e.,* off-chain and on-chain residual balance and the amount due to on-chain redeemed tokens).

The implementation of this logic in a smart contract requires trivial operations: the counterparty signature verification, required when the last state is checkpointed or a pending token is presented to be redeemed; the sequence number comparison between two state, in case of a dispute arises on the actual last state of the channel; the generation of the hash of a token, the counterparty signature verification on PoA and the correspondence check in between the generated hash and the one reported in the PoA presented to argue a pending token redeem process.

On-chain pending token redemption invalidation via PR Also PRs can be presented on-chain to testify that a payment has been correctly accomplished and therefore to invalidate the on-chain redemption of a token. This is only allowed for the hub and only in two specific cases.

In the first one, the payer conveyed a token to the payee and afterwards tried to cancel the payment, but the hub did not collaborate. The payer closes the channel and tries to redeem the token on-chain. If the hub actually paid the payee, it can present the PR to halt the redeem procedure.

Alternatively, it can be the payer that despite the hub being collaborative, arbitrarily decides to close the channel and redeem the token on-chain.

Since the smart contract is not in the position to ascribe blame to the payer or the hub for this behavior, when a PR is presented by the hub, to nullify the on-chain redemption of a token, no punishment can be applied.

5.2.4 Discussion

In what follows are discussed two peculiar aspects of the extended detach/attach scheme, namely concurrent payments and token expiration timeout.

5.2.4.1 Concurrent payments

On-chain transaction fee has to be paid to trigger the on-chain redeem procedure for pending tokens. This establishes a direct proportionality between the number of pending token a client owns at channel closure time and the on-chin transaction fees she has to pay to redeem all them. Since a token represents a payment in this model, it emerges a relationship between concurrent payments and channel closing costs in case the hub is not cooperative anymore. The more concurrent payments a client wants to operate, the more she is exposed to the risk of paying a high on-chain transaction fee to redeem all of them.

Fortunately, this argument only applies for outbound payments: the receiver of the payment, in fact, only considers the payment final once he presents to the hub the PoA and the PR. Until then he can ignore an inbound payment (it will be responsibility of the payer to apply the cancel token procedure once the token will expire). In this sense, off-chain $\multimap$ off-chain payments are inherently sequential from the payee's point of view.

Concurrency for on-chain originated payments On the other hand, both on-chain $\multimap$ off-chain payments and off-hub $\multimap$ off-chain payments configure the dangerous situation where the payment can't be cancelled, since the token is detached on-chain. If the hub becomes uncooperative before the payee attaches the token to his off-chain balance, he may have to pay the on-chain fee transaction to receive it. In this case the inherent sequential character of inbound payments does not hold anymore. If several payments of this kind are received and the hub becomes uncooperative before the receiver attaches them to his off-chain balance, he has to

close the channel trigger the on-chain redeem procedure for each one of them. However, it is to expect that such payments, which implies on-chain transaction costs also for the payer, are not used for micro-payments. They are more likely exploited for substantial payments and, in the eventuality that the hub becomes uncooperative before the payment is completed, it is worth to pay on-chain transaction fees to redeem them. Nevertheless is it advisable for the payer to have some form of nullaosta from the payee before to trigger the payment, since it could be expensive for a receiver to close a channel with several pending tokens to redeem on-chain.

A major concern about concurrency of on-chain originated payments regards the effect of a chain reorganization. The token sequence number of an on-chain detached token is an input of the contract method that produces it on per-client basis as a result of its invocation (only if all the parameters pass a validity check, including the sequence number). The couple `(transaction-nonce, tkn-seq-number)` defines a total order over the on-chain generated per-user token set. Once the transaction is issued, if it has been deemed correct and included in a block by a miner once, and if a chain reorganization occurs and the previous block is orphaned, it will be included in a new block sooner o later. Nevertheless, it would be advisable to use (spend) an on-chain generated token only after a reasonable number of blocks has been mined after the one in which it has been generated, and the probability of an on-chain reorganization is considered negligible.

Inherent sequential payments Ideally, however, payments have to occur sequentially: payer detaches a new token, thus initiating a new payment, only after the previous one has been completed. To complete an off-chain payment only requires order of milliseconds in a cooperative scenario. In fact, only five off-chain interactions, *i.e.,* five http requests/response, are required: 1) `[payer` $\leftrightarrow$ `hub]` token detachment, 2) `[payer` $\leftrightarrow$ `payee]` token conveyance, 3) `[payee` $\leftrightarrow$ `hub]` token attachment, 4) `[payee` $\leftrightarrow$ `hub]` PR conveyance, 5) `[payer` $\leftrightarrow$ `hub]` token attachment. Concurrent payments are only required when, for any reason, the previous payment slows down to such an extent that the payer can not delay any longer the subsequent one. Therefore concurrency is somehow dependent on the resistance that a payment may encounter, which has to be interpreted as a signal that something wrong or dangerous is happening and therefore it would be advisable to not undertake any further payment, at least not toward the same recipient, or even to cancel the pending one.

This sequential behavior is consistent with the inherent sequential nature of transactions on the Ethereum platform, where a sequence number (called nonce) given to each transaction issued from an account imposes per-account transaction confirmation order.

5.2.4.2 Token expiration time

Token expiration time describes a moment in the future (or a future block number) until which the payer can't present the token to the smart contract to cancel it and

contextually trigger the on-chain redeem procedure. It provides a form of synchronization for the payment procedure defining the moment within which a successful payment must end. A payee only has to redeem a token that gives him enough time to evaluate hub collaboration and, if not so, redeem the token on-chain before its expiration. With respect to the HTLC approach, it is the equivalent of the locktime of the contract. The existence and the respect of the constraint imposed by the token expiration time ensure the security guarantees of the system, analyzed in the next section.

5.3 Security analisys

The proposed solution is designed to prevent any honest participant from losing its funds despite an irrational attacker is assumed.

Critical conditions The analysis focus on those payments which in at least one side involve an off-chain endpoint. When the payment is not generated off-chain or not sent toward an off-chain balance, the security guarantees are backed by the smart contract that supports the whole architecture, which is assumed bug-free and vulnerability-free.

Directions for honest clients Every honest client is called to close the channel as soon as she detects unresponsiveness from the hub. Furthermore, she has to immediately trigger the on-chain redeem procedure for possibly pending tokens.

The accomplishment of an off-chain headed payment is subject to some activities to be performed by the receiver side. Since the receiver might refrain to perform these actions, no guarantee can be provided to the payer on the accomplishment of such a payment. However, if the receiver is not collaborative in receiving an off-chain headed payment within a bounded amount of time, the sender is granted the possibility to cancel it.

Directions for a honest hub The hub has to constantly monitor the blockchain to detect malicious behavior of its clients. It can have the form of: 1) an on-chain checkpoint of a wrong state determined by a malicious attempt to close a channel; 2) a malicious attempt of a client to redeem a pending token on-chain. In the first case the hub has to produce the correct final state of the channel; in the second one, it has to present to the smart contract the PoA or the PR for the questioned payment.

Fundamental properties The *atomicity of payments* and the *absence of pending tokens after channel closing* ensure the conservation of balance for the hub and correct execution of payments for clients. In what follows are analyzed the threat configurations that may arises and shown that those two proprieties hold for honest participants that diligently follows the protocol.

5.3.1 Threat model

An irrational attacker is assumed, that is willing to lose its funds to the purpose of seeing other parties economically damaged. Depending on the attack scenario, the irrational attacker can control: 1) the payer, 2) the payee, 3) the hub or a coalition of two of the previous, namely, 4) the payer colludes with the hub to attack the payee, 5) the payee colludes with the hub to attack the payer and 6) payer and payee collude to attack the hub. In each one of the previous six deployment configurations the adversary can be mount rational or irrational, aggressive or passive attacks. The system guarantees that the honest party that diligently abides by the protocol is immune to all of the mentioned attack scenarios and can not be deprived of its funds.

5.3.2 Analysis of threat configurations for off-chain payments

In what follows is shown that the fundamental properties hold for honest parties despite the irrational adversary takes control of one of the six aforementioned configurations in the most critical scenario of homogeneous off-chain (off-chain $\twoheadrightarrow$ off-chain) payments. For each configuration are enumerated plausible threats and how the proposed scheme counteracts or prevents them.

The adversary controls the payer

Payment attempt via expired token The payer tries to make a payment conveying an invalid token to the payee. The token can be expired, cancelled or the expiration time can be imminent, so that the payer can cancel it immediately after the payee accepts it.

Solution. Token expiration time is reported in the token tuple. A honest payee must check the token expiration time and refuse each payment made through token which is about to expire. Obviously, any expired, invalid or cancelled token must not be considered as vehicle of a valid payment.

Hub settlement prevention Once the hub has correctly forwarded a payment toward a payee and therefore it owns the PR for that payment, the malicious payer refuses to settle the off-chain hub position by attaching the token on its side of the channel between them.

Solution. The hub must stop collaborating with the malicious payer. As soon as the channel will be closed, the amount of the unsettled payment will be recognized to the hub thanks to the single-party claim mechanism for channel state checkpoint. An irrational behavior for the payer would be to not immediately close the channel. On the contrary, she can indefinitely pay the subscription to the hub services. She can only exploiting those services that do not require cooperation from the hub (*e.g.* on-chain $\twoheadrightarrow$ off-hub payments). As long as the hub earns the subscription, it keeps the client even if no collaboration is provided for off-chain interaction.

Myriad of tokens generation The payer asks to detach a huge number of tokens, then close the channel and try to redeem all of them on-chain.

Solution. Since the payer is legitimate to do such a thing, the hub as nothing to argue on-chain and does not spend anything for on-chain transaction fees. After the channel closing grace period has expired the payer will withdraw all the amount. The hub has earned from this activity the off-chain transaction fee which is due at token detaching time.

Discussion. This adversary behavior, if the number of token detaching requests is beyond the capabilities of the hub, is more dangerous as a denial of service attack, and has to be counteracted by the hub adopting common precautions.

Malicious pending token redemption attempt During the channel closing grace period the payer triggers the on-chain token redeem procedure for a token which has already been correctly exploited to accomplish a payment.

Solution. The hub own a PoA for this token that has to present to the smart contract that in turn will punish the malicious payer.

Repeated malicious pending token redemption attempts The payer, after having correctly accomplished several payments, close the channel and tries to redeem on-chain a large number of already off-chain attached tokens.

Solution. The hub has to correctly reply only to the first malicious on-chain redeem attempt. After that, the malicious payer is punished and all the other still undecided on-chain redeem procedures are settled in favor of the honest hub.

The adversary controls the payee

Non-cooperation in payment reception The payee receives a valid token that he deliberately decides to not redeem.

Solution. The payer can cancel the payment and withdraw off-chain the relative token after it has been expired (as described in Section 5.2.1.1).

Malicious pending token redemption attempt During the channel closing grace period the payee triggers the on-chain redeem procedure for a token which has already been correctly attached to his off-chain balance.

Solution. This threat is similar to the homonym one from the malicious payer configuration. Also in this case the hub owns a PoA relative to the attachment of this token and must present it to the smart contract which punishes the malicious payee.

Repeated malicious pending token redemption attempts The payee, after having correctly accomplished several payments, close the channel and tries to redeem on-chain a large number of already off-chain attached tokens.

Solution. This threat is similar to the homonym one from the malicious payer configuration and is analogously addressed.

The adversary controls the hub

Non-cooperation in token detachment The hub does not allow the payer to detach a token to make a payment.

Solution. The payer closes the channel and redeems on-chain all the possibly pending tokens she owns.

Non-cooperation in token attachment The hub prevents the payee to receive a payment by not cooperating in the off-chain attachment of the relative token.

Solution. The payee closes the channel and redeems on-chain all the possible pending token he owns.

Ianus Bifrons attack In this attack the hub, that has received a PR along with the proposal of a state update from the payee, becomes passive and does not send back his signature on the proposed update. On the other side, it asks to the payer to settle his position presenting the PR. The hub has therefore two different behaviors (as the two faces of the Roman god Ianus Bifrons after which the attack has been named): it is passive with the payee and aggressive with the payer.

Solution. The payee that detects unresponsiveness from the hub after he has proposed a state update and has sent it the relative PR, must close the channel and trigger the on-chain pending token redeem procedure before the token expires. The payment is considered accomplished.

Non-cooperation in payment conclusion The payer performed a payment and the payee correctly attached the token but did not send back the PR to the payer. The hub owns the PR but has become passive toward A. When token expires, A tries to cancel it interacting with the hub.

Solution. The payer closes the channel presenting the correct last state to the smart contract. This state reports the correct balance for the payer, but not for the hub which has not already attached the token amount to his off-chain balance. However, according to the single-party claim for state checkpoint, the hub receive the difference between the channel capacity and the balance claimed by the payer, which is the fair treatment from the point of view of the hub.

The adversary controls the payer and the hub

No additional threats arise from the coordination of payer and payee with respect to the configurations where the adversary controls only the payer or only the hub.

The adversary controls the payee and the hub

Also in this case no more dangerous threats are introduced by the coordinated versions of those presented in the configurations where the adversary controls only the payee or only the hub.

The adversary controls the payer and the payee

Uncooperative payer channel closing with many hub unsettled PRs The payer detaches several tokens, passes them to the payee that attaches all of them. Then the payer, before the hub settles its exposed position on the multitude of executed payments, closes the channel presenting the last valid state of the channel to the smart contract. This state reports the correct balance for the malicious payer but a wrong one for the hub that due to the non cooperation of the payer, has several pending PRs to settle.

Solution. The single-party claim for state checkpoint ensures that, after the malicious payer closes the channel, the hub gets the right balance computed as difference between the amount claimed by the payer and the channel capacity stored on-chain.

Uncooperative payer token cancellation with many hub unsettled PRs This is a variant of the previous threat scenario where the malicious payer, in addition to close the channel with several unsettled PRs for the hub, also tries to redeem on-chain all the related tokens. To counteract on-chain redemption of a payer pending token, the hub has to present the relative PR to the smart contract which can not ascribe blame and therefore halt the attack. If the irrational payer triggered the redemption of a multitude of tokens, the hub has to pay the on-chain transaction fee for each PR it presents.

Solution. The hub must contain the number of concurrent pending tokens allowed for a payer. This number has to be agreed upon at the beginning of the relationship between the hub and its clients and influences the subscription fee: the more pending tokens a client wants to have granted, the more a client has to pay for subscription, since the hub has to take into account the possibility of this irrational threat scenario. To have a small number of pending tokens, however, is also in the interest of an honest payer, which in this way reduces the on-chain transaction fee she has to pay to redeem them if the hub becomes uncooperative preventing her to cancel them on-chain (if necessary).

5.3.3 Remarks for mixed and external payments

Mixed and external payments that involve an off-chain endpoint can be divided into two groups: 1) off-chain originated payment (off-chain $\twoheadrightarrow$ on-chain and off-chain $\twoheadrightarrow$ off-hub payments); 2) off-chain headed payment (on-chain $\twoheadrightarrow$ off-chain and off-hub $\twoheadrightarrow$ off-chain).

Mixed and external off-chain originated payments Since the smart contract verifies the validity of the token used for the payment, threats for this configuration are analogous to those reported in the previous analysis where the adversary controls the payer.

Mixed and external off-chain headed payments In this case token detachment is supervised by the smart contract and the payee learns about an inbound payment by continuosly monitoring the blockchain. Once on-chain token detachment has been accomplished, the payment is considered final and is up to the payee to attach the token to his off-chain balance. Possible threats in this configuration are analogous to those previously presented where the adversary control the payee.

Mixed and external on-chain payments For those payments that are completely supported by the smart contract, namely on-chain $\rightarrow\!\!\triangleright$on-chain payments, on-chain $\rightarrow\!\!\triangleright$ off-hub payments and off-hub $\rightarrow\!\!\triangleright$ on-chain payments, security relies on correctness of the smart contract and system model assumptions.

5.4 Usability

In this section are discussed the conditions under which Fulgur would be suitable for use and which precautions are advisable to adopt.

A client has to instruct the cease of her subscription with the hub and consequently the closing of the related payment channel as soon as she detects a not cooperative behavior by the hub. Not cooperation of the hub consists in its refusal to answer to client's requests or in providing wrong answers.

The hub protects itself by constantly monitoring the blockchain to detect possibly malicious channel closings and pending token on-chain redeem requests it has to reply withing the expiration of the grace period.

5.4.1 Token acceptability

A payee has to refuse a payment conveyed through a token with an imminent expiration time. This is because, if the hub becomes not collaborative, the client has to redeem the token presenting it on-chain before its expiration, otherwise it is refused by the smart contract. There are two conflicting interests. The payer prefers a short expiration time. In this way, if the receiver does not redeem the payment, she can rapidly cancel it, thus minimizing the number of her pending tokens. The payee prefers a long expiration time that ensures he has enough time to redeem it on-chain should the hub become uncooperative. However, since it is the payee that decides whether accept a payment conveyed by a token with a specific expiration time or not, a payer that tends to adopt expiration times considered too short, sees all (or most) of her payments ignored by payees. On those payments she pays anyway the fees to the hub. She is therefore incentivized to set a correct expiration time,

avoiding in this way to to get her payments rejected and her funds wasted in fees paid to the hub for for unsuccessful payments.

Minimization of pending token number To the purpose of minimizing on-chain transaction costs to redeem pending tokens at channel closing time, every actor of the system has the interest of minimizing the number of pending tokens in which he is involved. This interest is shared among rational actors: any clear contravention of this principle has to be interpreted as a possible threat. The hub act as a synchronization point for payments it intermediates. From the point of view of a client, under normal and cooperative operation, payments have to be quickly completed and sequential. The necessity of concurrent payments, and therefore, of several pending tokens, must be read a warning signal.

5.4.2 Payment finality

A payment, according to the extended detach/attach scheme, is considered final when one party owns the necessary cryptographic proof to oblige the other to behave as expected, possibly closing the channel and resorting to the pending token on-chain redeem procedure. All the steps of the scheme one party accomplished before the reaching of this critical point can be undone.

The step in which a payment can be deemed final depends on the payment type. For those types that involve an on-chain interaction, a payment is considered final once the on-chain interaction has been correctly and successfully concluded.

For off-chain $\twoheadrightarrow$ off-chain payments, payer deems the payment final as soon as she receives the PR from the payee, or, if the payee does not convey the receipt, as soon as the hub provides it. From the point of view of the payee an inbound payment is considered final as soon as he sends the PR to the hub along with the state update proposal to attach the token.

5.4.3 Payment anonymity

As long as no involved player reveal its off-chain interaction, the system inherits the privacy preserving properties of state channels, thus privacy is guaranteed among clients: each client only knows about transactions in which it is actually involved. However, the hub in the position to know every transaction made by its clients. Payment recipient pseudo-identity, namely the recipient address, is contained in each detached token. Potential clients of the hub must keep in mind this privacy concern, which determines an intermediate situation between the totally public approach of the blockchain and the private solution offered by state channels, where off-chain interaction is only known to the two channel endpoints.

The privacy sacrifice made in this first Fulgur proposal serves the purpose of allowing the smart contract to easily resolve potential on-chain disputes that may arise on pending tokens. The recipient address reported in a token can be replaced with an interactive scheme supported by information asymmetry as for example

the knowledge of the preimage resulting from the application of a hashing function. Such an approach would slightly complicate the on-chain resolution of a dispute but would preserve clients privacy in the case of perfect cooperation. Despite the knowledge of the recipient identity, the current architecture, that strongly relies on state channels, requires for the hub to intermediate a payment the precise knowledge of the payment amount. Payment amount, therefore, allows the hub to link payer and payee. Chaum's e-cash system [11] addresses this problem by having payments made by electronic coins each worth the same amount (or chosen among a fixed set of predefined amounts). The exploration of this approach is left to future investigators.

5.4.4 Fee model

The entity of capital advance required for the hub to run its business is related to the sum of its own funds blocked in channels opened with clients. Funds locked in channels are to be thought as an operative cots for the hub, and may even be higher that the cost the hub has to support in order to actually update the channel balance, which involves the deployment and the maintenance of the technology infrastructure. The hub is actually paying the opportunity cost for its locked funds.

For this reason the fee model distinguishes two different kinds of fee a client can be charged by the hub: a *financial fee*, due to the capital advance the hub has to sustain to operate the channel, and an *informative fee*, due to the actual technological operative cost.

The existence of a financial fee introduces a cost related to the length of the channel lifetime: the longer a channel is operative, the longer the funds are locked, the longer the hub has to sustain costs induced by capital advance.

While technological operative costs can be estimated quite well given the average and peak loads, the capital advance required to profitably run a hub is direct consequence of behavior and needs of the clients.

Hence, those are the concerns of the stakeholders: for the hub manager, to establish an upper bound to the required capital advance; for clients, to lengthen the life of channels and therefore avoid to close and re-open a skewed channel and save on-chain transaction costs. The clients' concern is congruent with the pursuing of blockchain scalability by unload it using off-chain transactions as much as possible.

The financial fee is therefore to be paid by a client in relation to the extent of hub's locked funds required to ensure off-chain operation. The hub maximum exposition corresponds to the total amount of incoming payments a client is able to receive along the life of the channel. It can be integrally collected by the hub at the channel opening time, being known and agreed upon the maximal amount of the funds the hub has to block and for how long. Alternatively, it can be the client that blocks some funds, from which the hub is allowed to collect financial fee as time passes, according to an agreed fee collection rate. As long as the client wants to keep the channel open, it has to ensure that funds to collect financial fee from are available. This approach is comparable to a "subscription" to the hub services.

Furthermore the client may state a maximum amount, and actually require the hub to block only portion of this amount, executing a channel refill when the necessity arises (further discussed in the next Section 5.4.5). The actual fee collection strategy put in place for a specific hub is up to the hub manager.

The concept of financial fee is radically different from the fee model of blockchain: fee for on-chain standard payments are due in relation to the transaction "weight" (expressed, for example, in terms of bytes for Bitcoin and gas usage for Ethereum). Off-chain transactions re-introduce a relation typical of the fiat money world where (often) the transaction fee is due proportionally to the transaction value.

On the other hand, informative fee is similar to the blockchain fee model and is charged by the hub for each state update required by a client.

It is to expect that once deployed in production, the informative fee will be negligible with respect to the financial one.

5.4.4.1 Financial fee collection

The financial fee due by a client is collected on the base of a reactive protocol. Every time a client instructs an on-chain operation, her financial fee is settled until the invocation time. If no client funds are available to settle her position, the operation aborts; the client is required to fund the smart contract and re-invoke the on-chain method. Since the channel closure is triggered via an on-chain method invocation, this guarantees the settlement of a client position before she can close the subscription with the hub.

This approach prevents the hub to pay on-chain transaction fees to collect financial ones from its clients. It would be arguable if the collected fee is to be immediately assigned to the hub or set aside until the closure of the channel to be part of the funds the hub has to be punished on if caught misbehaving. The first option has been preferred since it contributes to mitigate the economic pressure the hub is exposed to. Furthermore, since the mechanism to handle skewed channels (detailed in Section 5.4.5) put the client in the position to control the amount of funds the hub has blocked on its side of the channel, the client herself can supervise this amount suffices in discouraging the hub to cheat, or take action on it by initiating a channel rebalancing.

5.4.4.2 Usage costs

A client that connects to a hub has to pay a third kind of fee: the one due to the miner of the underling blockchain to see her on-chain transaction inserted into a block and confirmed.

Figure 5.3 summarizes for each type of payment supported, who has to pay which kind of fee to whom. Since financial fee is collected in bulk, it is not reported in the scheme, which focuses on per-transaction costs.

Light gray dots represents the informative fee possibly due by the payee to the hub for the attachment of a token, *i.e.,* to complete an inbound payment. It is to

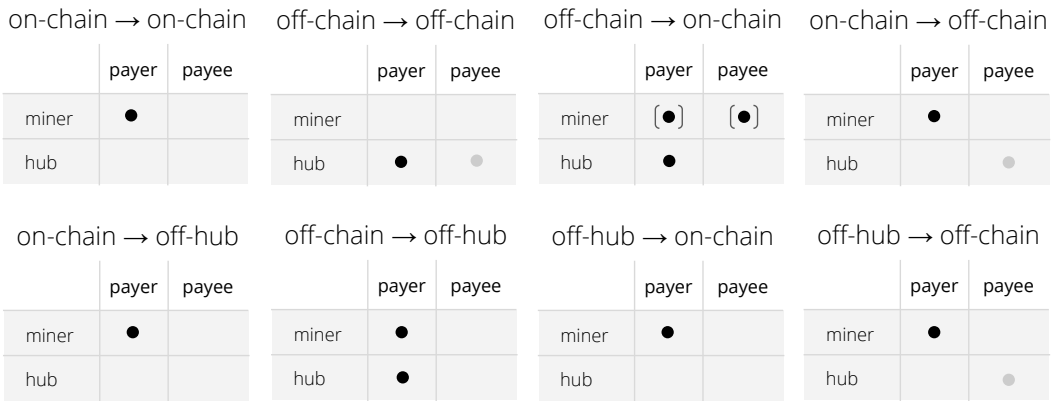

Figure 5.3: Per-transaction fee model.

expect that this fee is cheaper that the one applied to the payer, (represented with black dots) or even (most likely) completely absent. Although the system supports collection of payee fees, it will be responsibility of the hub manager to elaborate the particular model that best suites his business needs.

Black dots in between parentheses for the off-chain ⇸ on-chain payment represent the fee related to the on-chain transaction required to attach the token on-chain. It is paid by the payer if the payment has been initially conceived to be attached on-chain. Alternatively, the payee pays it if he triggered the alter token procedure for an original off-chain ⇸ off-chain payment.

5.4.5 Handling skewed channels

When a channel in between a client and the hub is skewed, it may be impossible to handle a payment of an amount higher than the residual balance. In particular, depending on which side the channel is skewed, it might be impossible to send a payment (channel skewed toward the hub) or receive it (channel skewed toward the receiving client). This condition has to be avoided since it goes against blockchain scalability. To deal with this situation two mechanisms are defined.

5.4.5.1 Channel constant ratio rebalancing

At channel opening time, the client and the hub agree on the *channel contribution ratio*. It is the ratio between the amount blocked by the client and the one blocked by the hub. An high ratio is preferable for a frequent payer while a low ratio may be suitable for a merchant, which is likely to receive more payments than it sends.

Following the scheme of an on-chain ⇸ off-chain or off-hub ⇸ off-chain payment a client can trigger a channel refill. Similarly, using the scheme introduced for off-chain ⇸ on-chain or off-chain ⇸ off-hub payments, a client can withdraw funds from his side of the channel. Once the procedure is invoked by the client (the only

entitled to trigger it), the channel is rebalanced according to the constant ratio. If the client is refilling, some funds are freed for the hub and vice versa.

If an upper bound has been agreed upon for the economic exposition of the hub, the procedure only succeeds if the procedure respects the limit.

Since a ratio is involved and smart contract execution platforms (Ethereum in particular) do not support floating point arithmetic, the implementation introduces a rounding that can lead to a negligible loss for one of the parties.

This procedure, along with the imposition of a maximum economic exposure of the hub, allows to unburden the hub of the economic management of the channel and therefore saving the on-chain related transactions.

5.4.5.2 Alter token procedure

If a client receive an off-chain payment but the channel is skewed and a refill is not possible due to the reaching of the hub exposition ceiling, the alter token procedure allow to change the off-chain token into a token redeemable on-chain. The alter token procedure implies an on-chain transaction for the receiver of the payment. It can be thought of as an off-chain $\twoheadrightarrow$ on-chain payment where the detached token was not meant for an on-chain redemption, and therefore it is the receiver that sustain on-chain transaction cost. The alter token scheme proceeds akin an off-chain $\twoheadrightarrow$ off-chain payment for the first part, while the last one resembles an off-chain $\twoheadrightarrow$ on-chain one. Details on the four steps required to accomplish the alter token procedure are provided in what follows.

Payer off-chain token detachment The channel (A, H) is updated to the state $< i, \beta^A_{i-1} - \alpha_j, \beta^H_{i-1}, \mathcal{H}(\mathtt{tkn}_j), \mathbb{ID} >_{(\sigma^A, \sigma^H)}$ and the j-th token $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OFF} >_{(\sigma^A, \sigma^H)}$ is detached, which also constitutes the PoD to convey to the receiver.

PoD conveyance At the end of this step B owns the signed token $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{OFF} >_{(\sigma^A, \sigma^H)}$.

Token alteration As reported by the last element of the tuple, the token at this point can be only attached off-chain, and the smart contract would refuse it, if presented. This is to prevent the race condition that would arise if B would try to attach the token back both on-chain and off-chain. Therefore the hub has to agree on making the token only redeemable on-chain. This procedure has been named *token alteration* and is performed through an off-chain interaction between the payee and the hub. The signed altered token $\widehat{\mathtt{tkn}_j}$ has the form $< j, \alpha_j, \mathrm{ID}(B), t_{\mathrm{exp}}, \mathbb{ON} >_{(\sigma^B, \sigma^H)}$. Once altered, the standard token can not be redeemed off-chain anymore: the hub would refuse to attach it off-chain. Furthermore, if B issues an on-chain malicious challenge to force the hub to attach the token off-chain, the hub must present the altered token signed by B to testify the previous intention of B to redeem it on-chain and, consequently, win the challenge.

Payee on-chain token attachment B presents the token to the smart contract that, after some validity and correctness checks, updates the interested data structures as follows. Capacity of the channel (A, H) is decremented by the payed amount α_j, $\boldsymbol{C}_j^{(A,H)} = \boldsymbol{C}_{j-1}^{(A,H)} - \alpha_j$. The on-chain balance of B is incremented by α_j, $\boldsymbol{\beta}_j^B = \boldsymbol{\beta}_{j-1}^B + \alpha_j$. The hash of the token is stored in the relative registry, $\mathcal{H}(\widehat{\texttt{tkn}_j}) \hookrightarrow \boldsymbol{R}^{\text{off}\to\text{on}}$, which represents the PoA.

If the alter token procedure is put into effect by B, the channel (A, H) does not need any further update to be settled. In fact, it already reports correct balances for both parties: the payment amount has been subtracted from the off-chain balance of A and the channel capacity has been updated consequently.

Notes on security The alter token procedure introduces an attack scenario that has not been previously analyzed. The hub and the payee may cooperate to the end of defraud A of her funds. With the cooperation of H, B may attach the token both on-chain and off-chain. Owning the off-chain payment receipt, H may also proceed to settle its alleged off-chain exposition on the channel (A, H). This aggressive behavior of H, however, is easily detectable since its signature is present both on the altered token $\widehat{\texttt{tkn}_j}$ presented on-chain by B and on the last state update of the channel (A, H), where the plain (not altered) version has been attached. This constitutes a proof of misbehaving (PoM) that can be presented by A to the smart contract to punish the hub by crediting the whole channel balance to her. It is worth noting that no funds can be stolen by the hub since it is not able to autonomously withdraw funds or trigger the channel closing.

5.4.6 Storage requirements

To counteract a malicious issued on-chain challenge both the hub and a generic client have to store PoAs relative to tokens attached by the channel counterparty. A PoA is composed by a channel state tuple along with a counterparty signature. Considering an implementation based on Ethereum through the Solidity smart contract language, the state tuple part of a PoA (*e.g.* $< i, \beta_i^A, \beta_i^H, \mathcal{H}(\texttt{tkn}_j), \mathbb{A} >$) can be implemented exploiting a data structure with the following fields:

1. i (sequence number) - type: `uint`, 32 bytes length
2. β_i^A (A's balance) - type `uint`, 32 bytes length
3. β_i^H (H's balance) - type `uint`, 32 bytes length
4. $\mathcal{H}(\texttt{tkn}_j$ (token hash) - type `byte32`, 32 bytes length
5. $\mathbb{A}$ (state function) - type `custom enum`, 8 bytes length

Assuming a 65 bytes length signature, the sizes of individual fields sum up to 201 bytes. Assuming no data compressing strategies are applied, these numbers allows to secure one million payments with less than 200MB of local storage. Once the

channel is closed and settled on-chain, relative stored data can be disposed with no flaws for security.

The choice of 32 bytes length data structure for sequence number and balances implies an upper bound for all of them. A variable length data structure with infix size could have also been chosen. This, at the cost of a slightly more complex implementation, would have introduced an optimization in storage usage, since only strictly needed bytes are occupied.

This choice for sequence number and balances storing data structure implies that: 1) after 2^{32} state updates a channel has to be closed; 2) the maximum allowed balance is 2^{32}. Those limited values are large enough to do not represent a practical obstruction for the protocol.

5.5 Open issues and future directions

Being based on an extended version of the detach/attach scheme, Fulgur would benefit from the same enhancements already discussed for inextinguishable payment channels (see Chapter 4 and in particular Section 4.7). The introduction of a solution to accumulate PoAs in a fixed length new element of the state tuple would limit storage requirements for clients and hub. Furthermore, payment channels may be collateralized with different denominations. Once the hub has fixed an exchange rate, this solution would enable a client to use a coin denomination of its choice to perform payments through the hub.

One big next step is the definition of a protocol to allow an off-chain payment to be intermediated by more than one hub, therefore realizing a network of Fulgur hubs. The extended detach/attach model already provides the guarantees required for prevent loss of funds in such a scenario: two hubs may open a payment channel in between them and behave as the first was a client of the second (or vice versa). However, in absence of a solution to mitigate the entity of capital advance required to the hub, it is not a viable solution. In fact, it might be the case that all the clients of one hub require to pay the clients of the other hub. Therefore the channel between hubs should guarantee enough capacity to accommodate all the payments. It might imply a demand of a not affordable capital advance for hubs. A solution for this issue is to be further investigated.

Chapter 6

Smart channels

Contents

A state channels is a two-party ledger updated off-chain that allows parties to share an arbitrary state resorting to a smart contract to resolve possible disputes. As soon as one party begins to believe the trust relationship has been violated by the other one, it has to close the channel, presenting to the smart contract the last off-chain state both parties agreed upon. In this chapter it is show that, for specific cases and under particular attacks, the state channel solution underperforms with respect to the on-chain interaction mediated by a smart contract, thus vanishing typical advantages of the off-chain approach. This is imputable to the nature of the agreement between parties, that in a standard on-chain interaction is wired into the smart contract code, but remains implicit when parties are involved in a state channel construction.

In this chapter it is shown that opening a state channel always implies an agreement between parties. A classification of these off-chain agreements is therefore introduced. The Smart Channel protocol is defined to tackle with the most complex of those categories, the one including an irrational adversary that can mount an harmful attack. The objective of a smart channel is to perform comparably to on-chain smart contract interaction in case of attack in terms of transaction costs and execution time. Furthermore, an operative correspondence is established between state channels, and the pioneering Szabo's definition of smart contract.

This chapter is organized as follows. Section 6.1 recalls the required background. In Section 6.2 is elicited the implicit agreement that stand behind a state channel and a classification is provided for different types of agreements. Section 6.3 presents the smart channel protocol. In Section 6.4 provides the security analysis of the protocol. Finally, Section 6.5 reports about usability.

6.1 Backgound

In what follows is recalled the irrational passive aggressive attack that can be mounted by a malicious party on a state channel. Furthermore, it is summarized the pioneering work of Nick Szabo, that in the '90 introduced the concept of smart contract from a theoretical point of view. His definition, being free of the constraints imposed by the technological concretization, comprised some features that are not present in state channels but are exhibit by smart channels.

6.1.1 Irrational passive aggressive attack

Section 3.5.4.1 describes an example of irrational passive aggressive attack. Such an attack can be successfully mounted by a channel adversarial endpoint that, to the purpose of causing a loss of funds for its counterparty, does not care to suffer an economic damage. The passive connotation implies an interactive scheme where the attacker voluntarily suspend its interaction. The combination of irrational and passive behavior produces a dangerous attack scenario: irrationality makes the attacker indifferent to economic incentives, while passiveness makes the attack difficult to be detected. As reported in Section 3.5.4.1, if the attack is successfully mounted, the off-chain interaction results more expensive in terms of on-chain transaction costs and execution time than the standard on-chain interaction.

6.1.2 Szabo's view on smart contracts

In 1997 Nick Szabo in his work entitled "Formalizing and Securing Relationship on Public Networks" [62] extended the concept of smart contract he himself had introduced three years before in [63]. The multitude of bright ideas introduced have been (partially) materialized only fifteen years later thanks to the spurring that the blockchain technology gave to the field.

Szabo defined smart contract as a digital version of the real world "control protocols". Control protocols, to use Szabo words, *"allow a quarrelsome species ill-suited to organizations larger than small tribes to work together on vast projects like manufacturing jumbo jets and running hospitals"*.

A real world control protocol is composed by the formalization of a protocol, *e.g.* a flow of forms, along with checks and procedures called "controls", that serve the same purpose of cryptographic protocol: integrity and authorization verification.

This smart digital version of standard contracts must exhibit three main features. 1) *Observability* by principals: complete access must be granted for involved parties since security guarantees derives from auditability of the contract and related information. 2) *Verifiability* by third parties: involved parties must be able to prove to an adjudicator that a contract has be performed or breached; reactive measures strongly rely on verifiability. 3) *Privity*[1] to contain information leak and reduce the surface exposed to attacks: while observability dictates that involved parties must have complete access to contract information, this possibility should not be granted to external entities that may exploit them in some malicious ways.

The Ethereum platform renewed the concept of smart contract providing a working implementation. The reborn concept has an operative connotation, we know what they are as a consequence of their operation. No theoretic definition or formalization has been provided. With respect to this recent, diffuse and accepted concept, the Szabo's definition presents significant differences. For the sake of clarity, the Szabo's formulation of smart contract is hereafter called "Szabo contracts"; the operative version based on distributed ledgers, instead, is henceforth referred as "Ethereum contract". The most evident difference is the focus on privity for Szabo contracts while Ethereum contracts rely on redundant contract code execution and public accesses to contracts code, execution inputs and outputs, as a pivotal source of information to ensure security guarantees.

6.2 Implicit agreements behind state channels

Every off-chain construction implies an agreement between parties, be it a payment channel or a state channel that supports a complex turn-based game. A payment channel, for example, implies a pretty simple agreement:

- a payment is sent from one party to the other;
- the amount of a payment is less or equal to the balance that the issuer owns on her side of the channel at the moment of the payment;
- parties can use the channel as long as one party close it.

The construction guarantees that no party can lose its funds. Although channel closing can be triggered for no reason, a rational endpoint is likely to close it only

[1] Szabo conducted a short etymological analysis on the unfamiliar term "privity". Although some differences stand, it clearly recalls the assonant and more common concept of privacy.

for two reasons: 1) the channel is not needed any longer, there is no longer reason to make frequent payments between parties or the channel is skewed and no longer exploitable; 2) the trust bond between parties is not standing any longer, one party is unresponsive or acted maliciously.

A payment channel allows to make off-chain payments. A state channel allows to handle off-chain an arbitrary data structure[2]. Those extended capabilities are a direct consequence of the enhancement of the blockchain capabilities introduced by the Ethereum platform for the execution of smart contract.

However, the agreement implicitly accepted by the parties involved in the channel is more important than the kind of data structure that can be shared and managed by the channel. Data structure is just one part of the agreement, or even less: one of the several possible instantiations. The "close-at-will" approach well suited for payment channels, might not apply in case of a more complex off-chain agreement.

A clear, human readable description of the agreement, be it implicit or explicit, should always go along with every self-enforcing piece of code (of which Ethereum contracts are an example). Ricardian contracts [64] address this problem and represent one of the first experience in this area of intersection between computer science and law which is attracting more and more interest.

6.2.1 (Un)satisfactory closing states

When backed by a smart contract defined through a Turing-complete language as in Ethereum, a state channel may implement any control logic. The missing ring for a state channel to provide to two parties what a smart contract (or better an Ethereum contract) provides to n, is the ability to overcome the "close-at-will" characterizing feature. In fact, it may be the case that parties want to agree to not close the channel until the channel state is "satisfactory" for both of them. The example of state channel that implements tic-tac-toe game provided in Section 3.5.4 offers also a good example of off-chain agreement where the parties may not be willing to conclude the interaction until the game state is clearly identifiable and therefore satisfactory to conclude the interaction: one winner and one loser or a draw.

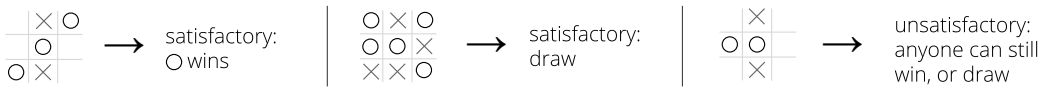

Figure 6.1: Examples of satisfactory and unsatisfactory closing states.

Figure 6.1 shows examples of satisfactory and unsatisfactory closing states for a tic-tac-toe game played on a state channel. The existence of off-chain states that involved parties deem as not satisfactory to conclude the off-chain interaction further

[2] As long as the arbitrary data structure is small enough to be processed and/or stored on the underling blockchain

exposes the construction to the irrational passive aggressive attack. In fact, being in an unsatisfactory state implies that some additional steps have to be taken before to close the channel. If the malicious party remains passive, the propose/accept scheme (*cfr.* Section 3.5.1) obliges the honest one to issue one on-chain challenge to propose a state update and one on-chain challenge to accept a state update, thus resulting in two on-chain interactions per step. Furthermore, the attacker can wait for the whole grace period before to provide its on-chain reply (and therefore avoid to see dropout punishment applied). This results in the honest party to wait for (almost) a whole grace period in each step.

6.2.2 μ function

The μ function is defined as a generalization of the previous example of unsatisfactory closing state on tic-tac-toe game.

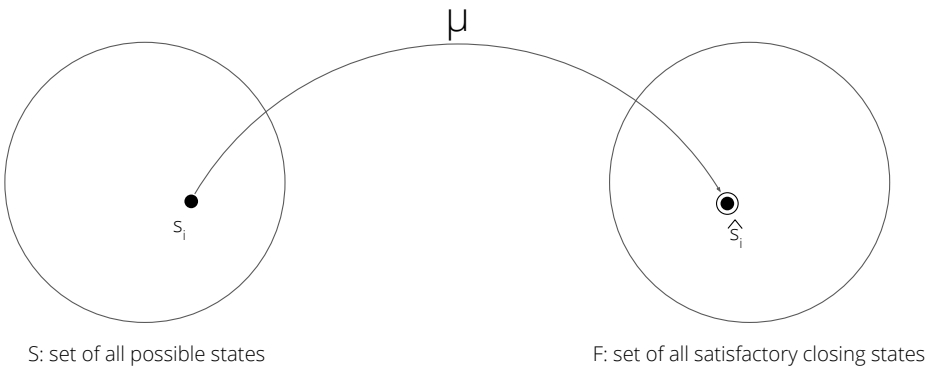

Figure 6.2: μ function domain and codomain.

The μ function, as shown in Figure 6.2, is defined as follows $\mu : S \to F$ where S is the set of all possible valid off-chain states and F is the set of all satisfactory ones. The function maps an element s_i in S to one element $\hat{s_i}$ of F. It associates an off-chain state, be it satisfactory or not, to a certainly satisfactory one.

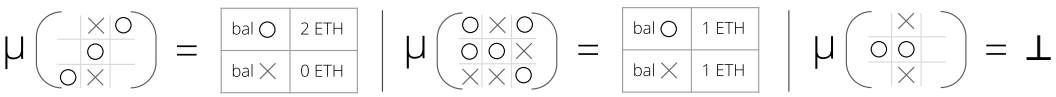

Figure 6.3: Examples of μ function applications.

Figure 6.3 shows the results of three applications of a plausible μ function for a tic-tac-toe game. Reasonable fund allocations are produced by the application of the function for satisfactory closing states from Figure 6.1, while no result is produced when the function is applied to an unsatisfactory closing state (last example in figure). A further example for a chess game could be to have a μ function

that reallocates bet funds in proportion to the weighted sum (with a priori defined weights) of the remaining pieces on the board. Payment channels implicitly admit the identity function as μ function.

6.2.3 μ-agreements and $\bar{\mu}$-agreements

The particular μ function adopted for an specific instance of a state channel only depends on the agreement that the involved parties implicitly accept when they join the channel. Two different instances of a state channel that serves the same purpose may admit different μ functions. It is a clause of the agreement.

Depending on the nature of the μ function on which parties agreed upon, it is possible to have μ-agreements and $\bar{\mu}$-agreements:

$$\mu\text{-agreement} \coloneqq \exists \mu_i \forall i \in S$$
$$\bar{\mu}\text{-agreement} \coloneqq \exists i \in S \text{ s.t. } \nexists \mu_i$$

For μ-agreements it holds that each off-chain state can be mapped to a satisfactory closing state by applying the agreed μ function. For $\bar{\mu}$-agreements it exists at least one off-chain state j which, as agreed by the parties, does not admit a valid μ function, *i.e.* $\mu(j) = \perp$.

6.2.4 Szabo contracts and state channels

Since an agreement, be it explicit or implicit, underlies a state channel, it makes sense to investigate the relationship between state channels and Szabo contracts. The latter ones, in fact, seems to be much more similar to state channels than to Ethereum contracts. Szabo contracts share more defining characteristic with state channels than they do with Ethereum contracts: while both state channels and Ethereum contracts exhibit observability and verifiability, the former is clearly more privacy preserving then the latter, being the off-chain interaction completely private and, as long as parties maintain a collaborative behavior, known only to the involved parties.

Smart channels, presented underneath, impose constraints on channel closing due to the cease of trust bond of involved parties and therefore constitute a further step toward the realization of Szabo's vision of smart contracts.

6.3 Privacy preserving channels for $\bar{\mu}$-agreements

Smart channels are introduced to allow off-chain handling of $\bar{\mu}$-agreements with performance that, in case of an irrational passive aggressive attack, are not worst than on-chain interaction with a smart contract (Ethereum contract). To achieve this, a seamlessly migration from off-chain to on-chain interaction can be triggered any time by any party. This approach allows parties to benefit of the off-chain

advantages as long as they mutually trust each other. Although to fall back to on-chain interaction entails a loss of privacy, this ensures the reaching of a satisfactory closing state and therefore provide guarantees against an irrational passive aggressive attack.

Smart channels are defined assuming the same system model adopted for state channels (*cfr.* Section 3.2). In particular, the data structure describing the shared state between endpoints has to be of a size compatible with the underling blockchain processing/storing capabilities.

6.3.1 Protocol

The protocol steps in Figures 6.5, 6.6,6.7, 6.8, 6.9, 6.10, outline how the a smart channel is expected to proceed. An involved node, in a perfect cooperative scenario, is required to interact with the blockchain only to setup and close the channel.

To handle the $\bar{\mu}$-agreements where the current state is possibly not a closure one and an irrational passive aggressive attack could be mounted by a malicious adversary, a transition scheme allows to gracefully move the interaction from off-chain to on-chain. The scheme allows for the issuance of a blind challenge, a particular challenge where no additional information is leaked. In fact, in case of positive reply to the challenge, *i.e* the counterparty is not malicious and, for example, experienced a technical fault from which it recovered before the grace period expires, the interaction can proceed off-chain with no information leak.

In the following description is assumed that Alice (A) and Berto (B) are the endpoints of a smart channel.

6.3.1.1 On-chain involved entities

In compliance with state channels' system model, the blockchain is considered as the root of trust and the output of the execution of the smart contract code as not questionable auto enforcing statements.

The smart contract code needed to support the smart channel protocol can be divided into functional units whose behaviors, adopting a metaphor of the real world, resembles the responsibilities of a *safe deposit box*, a *registrar* and a *judge*. While the safe deposit box and the registrar can be implemented in the same contract, to maximize privacy of involved parties as described below, the protocol requires that the judge is wrapped in a separated unit of code.

Safe deposit box This functional units locks funds of the parties that enter in a private association by opening a channel. This special safe deposit box grants the possibilities to be opened and therefore to recover locked funds only if a mutual agreement exists between parties on the allocation of the funds. Deposit box accepts opening instructions on how and when release funds only from the registrar and from the judge.

Registrar The registrar is the actual interaction point for involved parties. It accepts proofs of agreement between them (*e.g.* an off-chain state signed by both of them) or proof of misbehaving (PoM) that, in turn, it presents to the judge for the emission of a verdict. Furthermore, if trust between parties falls, the registrar is also responsible to enforce the migration from off-chain toward on-chain interaction.

Judge The judge evaluates proof of misbehaving presented by the registrar and emits the final verdict. Moreover, once instructed by the registrar, it becomes the intermediary of the on-chain interaction between parties that no longer trust each other to persevere in an off-chain agreement.

6.3.1.2 Smart channel lifecycle

Figure 6.4 summarizes the lifecycle of a smart channel. Details about each step and conditional branches of the lifecycle are provided underneath.

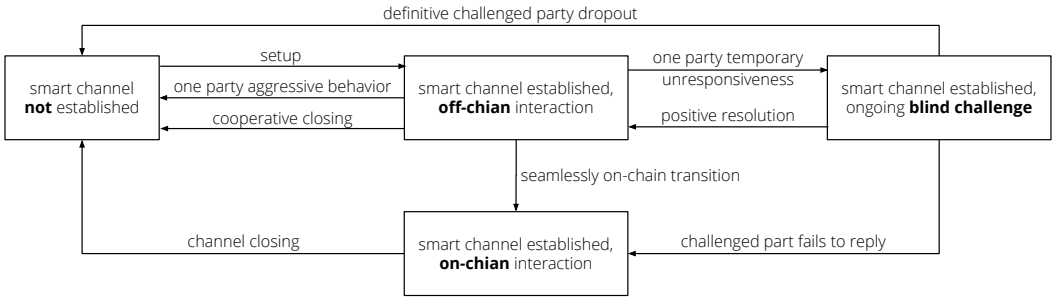

Figure 6.4: Smart channel lifecycle.

Setup (ex ante confidential judge selection) The first phase is the setup of the smart channel. Either the safe deposit box and the registrar are implemented as single or separated contract, they must have been deployed and ready to be invoked.

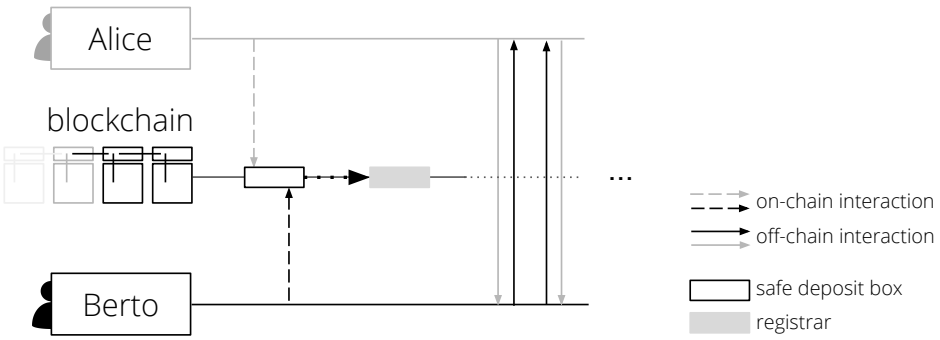

Figure 6.5: Smart channel setup interaction.

Figure 6.5 shows how the interaction proceeds. Funds are locked by both parties and, if implemented as separated contracts, an association is established between specific safe deposit box and the registrar entitled to instruct the release of the box.

The critical part of this stage is the judge selection. The judge is entitled to resolve disputes and/or intermediate the on-chain interaction once the parties no longer trust each other to remain in an off-chain agreement. To pursue the privacy concern, an indirection must be introduced between the setup of the registrar and the judge selection. In fact, the code the realizes the judge contains the details about the nature and the purpose of the agreement between parties. Therefore, hiding which judge rules on which smart channel dramatically enhance the privacy of the construction. It has to be mentioned, however, that parties pseudo-identities (*e.g.* their addresses on the Ethereum platform) involved in a smart channel, are known to the world, even though, at this point, it is not known the reason they have made an agreement.

Ex ante confidential judge selection can be achieved on Ethereum-like smart contract execution platforms by having the judge contract deployed in advance by a third entity not linkable to the smart channel endpoints. Once deployed the address a of the judge is known and can be hashed with salt s obtaining an obfuscated version of the contract address $\tilde{a}$ by computing $\mathcal{H}(a \odot s)$ (where $\odot$ is for example a binary infix operator that returns the concatenation of its arguments). s, a and $\tilde{a}$ are known to both parties. $\tilde{a}$ is actually conveyed to the registrar that stores it. Should arise the need to call into question the judge, a party reveals the s and a to the register which verifies that the hash correspond to the stored value by independently computing $\tilde{a}$. This simple scheme allows to inform the registrar to the actual judge contract address once the obfuscated version is conveyed at setup time.

This approach, however, requires the judge contract to be deployed even though in a cooperative setup it can be never invoked, not even once. Nevertheless, deployment costs must be payed and some precautions have to be take to avoid easy linkage of the judge contract deploying address to addresses of parties involved in the channel. On the Ethereum platform, for example, an ad hoc account should be used to deploy the judge contract. Furthermore, to maximize privacy, no direct transaction paths should link smart channel endpoints and deployment address to maximize privacy.

It is worth to mention a second possible way to obtain ex ante confidential judge selection, called "counterfactual contract deployment" [65], that allows to deploy the judge contract when and only if needed, therefore saving deployment transaction costs and completely avoiding any privacy leaks. This approach exploits peculiarity of the Solidity smart contract language, its scope of application is limited to the Ethereum realm and is only doable until EIP86 will be deployed.

Cooperative closing In the optimistic and rational case where parties maintain perfect collaboration until the closing of the channel, the interaction only occurs off-chain and remain private to the parties.

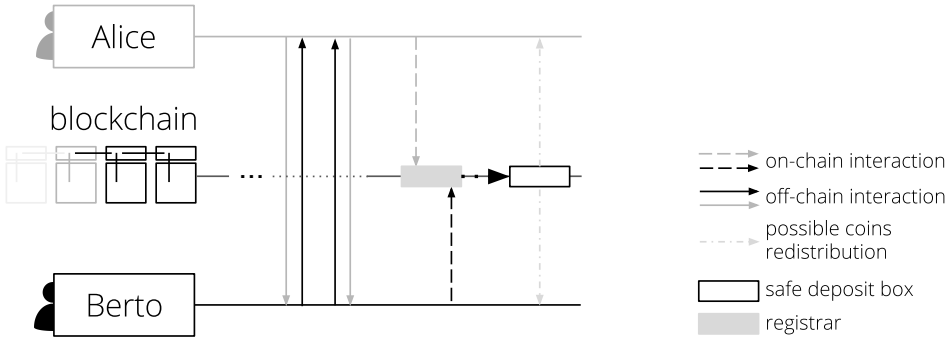

Figure 6.6: Smart channel cooperative closing interaction.

Figure 6.6 shows the interaction between smart channel endpoints and on-chain entities to cooperatively close the channel. A final state signed by both participants is sent to the registrar. After some validity checks the registrar instructs the safe deposit box to release locked funds according to the provided signed state.

As long as parties succeed in maintain unlikability in between them and the judge contract, especially for the contract deployment phase, this cooperative path does not require any detail about the agreement to be publicly revealed, thus supporting privacy of endpoints.

Aggressive counterparty behavior Should one party owns an off-chain generated proof of misbehaving testifying the aggressive malicious behavior of the counterparty, she can present the proof to the blockchain.

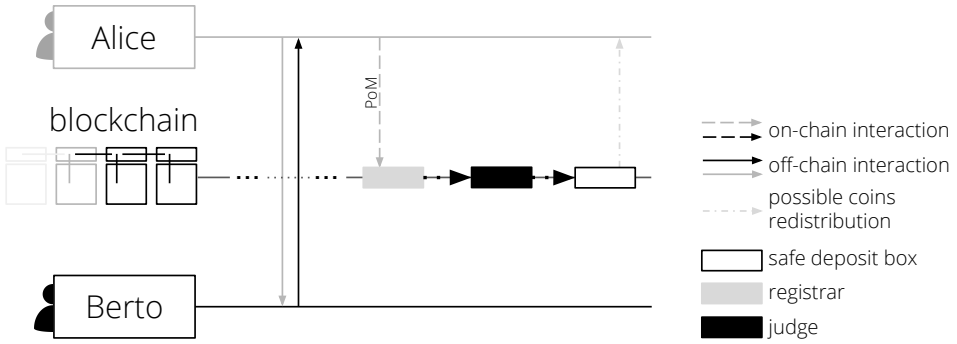

Figure 6.7: Reporting an aggressive behavior.

Figure 6.7 outlines the interaction to report PoM to the judge. The suing party interacts with the registrar, discloses the judge address and present the PoM. Since the PoM is not arguable, the judge, after a correctness check of the proof, ascribes blame to the dishonest party and decrees the punishment by instructing the safe deposit box to allow the reporting party to collect all the locked funds.

As in the real world, when one party sues the other one, details of the breached agreement becomes the public acts of the process and are revealed to the world.

Assuming the smart channel has complete coverage of the malicious behaviors that can originate off-chain and therefore each generated PoM is certainly punished if reported to the chain, then a PoM can be only left on the ground by an irrational party and the interaction to report aggressive counterparty behavior can be categorized as deterrent code (*cfr.* Section 3.5.5).

Temporary counterparty unresponsiveness At any moment a party that detects unresponsiveness from its counterparty may decide to issue a challenge. In the optimistic assumption that the unresponsive part is experiencing a technical fault and is not acting maliciously and therefore trusting in the possibility to recover an off-chain interaction, the challenge scheme must preserve privacy. No information has to be leaked about the judge contract until the malicious behavior of the unresponsive party is confirmed. To this end, a special challenge scheme is introduced. It is called "blind challenge".

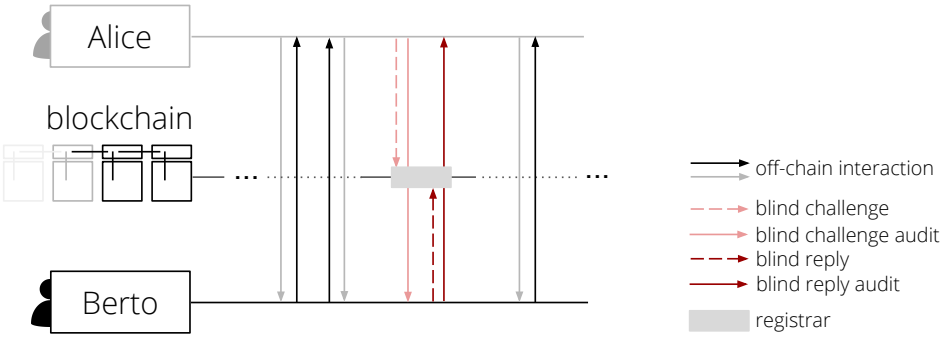

Figure 6.8: Cooperative interaction for a blind challenge.

Figure 6.8 show the interaction for a positively resolved blind challenge. Say that A is the party that issues the blind challenge. She prepares data d that the judge will be able to evaluate, but only sends a hash $\mathcal{H}(d)$) of the data to the registrar. Afterwards, she tries to inform B about the issued challenge by sending him an off-chain audit containing challenge plain data d. If B is effectively cooperative, he prepares data r for the reply and in turn only send an hashed version $\mathcal{H}(r)$ to the registrar and send r to A. A verifies correctness of received data, by also matching the hash publicly sent by B to reply to the registrar. If A is satisfied with the reply of B, interaction can continue off-chain, still benefits of the off-chain advantages. No information is therefore leaked to the public blockchain and privacy remains preserved for this cooperative interaction.

Definitive counterparty dropout If B fails to provide the registry with $\mathcal{H}(r)$ within the challenge grace period, he is definitively considered unresponsive and A can proceeds to withdraw locked funds.

Figure 6.9 show this kind of interaction. It is worth noting that also in this case no details about the nature of the agreement is leaked: the timeout can easily be

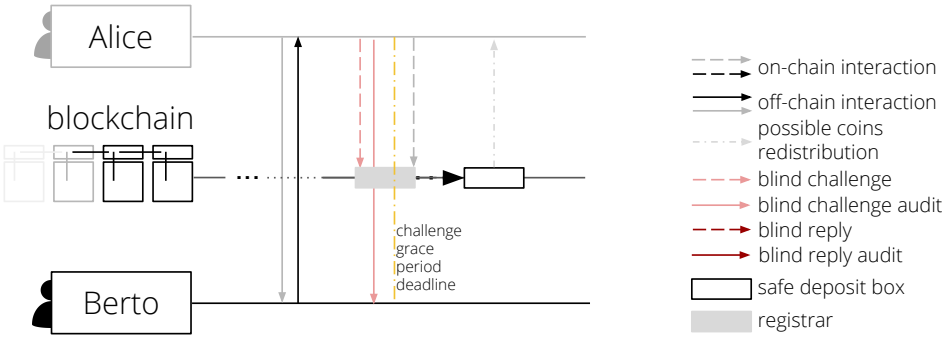

Figure 6.9: Party dropout after blind challenge.

handled exclusively by the registrar that takes note of the time at which the blind challenge has been issued and verifies the grace period expiration at the second interaction of A.

On-chain seamlessly transition Should B not send the off-chain blind challenge audit, or should the audit not pass correctness tests, A instructs the registrar to trigger the on-chain transition. The judge is therefore disclosed and the interaction between parties continue on-chain, intermediated by the judge that instantly verify correctness of each transaction, as for a typical on-chain interaction of an Ethereum contract.

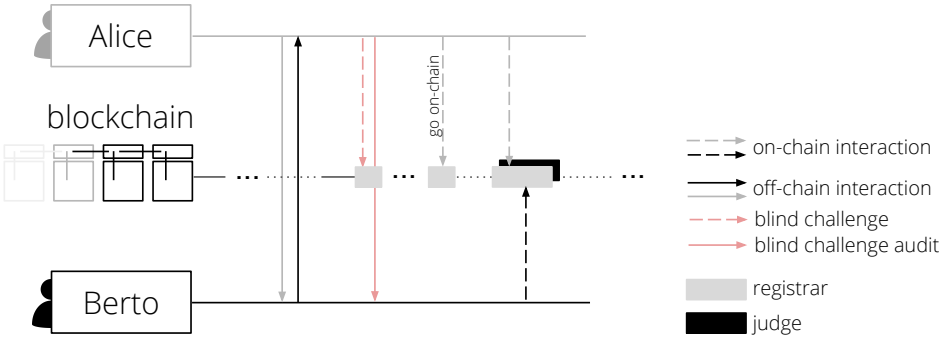

Figure 6.10: Uncooperative interaction for a blind challenge.

Figure 6.10 depicts this situation. In the case a malicious blind challenge is issued, the trust bond between parties needed for the off-chain interaction is cut, and the interaction has to carry on on-chain as well. In this case, it is the challenged party that instruct the migration. A detailed analysis of the scenarios that may arise is reported in Section 6.4.2.

6.4 Security Analysis

The smart channel protocol is designed to prevent any honest participant from losing any funds despite strong adversarial assumptions. Security guarantees are however function of the implementation of the specific agreement. The judge contract, that applies punishments and manage the possibly on-chain interaction, and the client used by the parties for the off-chain interaction, are considered secure for the purpose of this analysis. The funds unlock process preformed by the safe deposit box is a trivial verification of a digital signature. The join process and the grace period expiration, supervised by the registrar, are analogous to a standard state channel. The blind challenge process, that originates several possible paths, is the critical part which is examined in detail in what follows.

6.4.1 Treat model

One of the reason for the introduction of smart channels is to enhance state channels so that they can tackle with an irrational adversary, which is willing to waste its funds to damage the attacked party. In particular, a smart channel is expected to perform the same a smart contract do in terms of time and cost in handling the case of an irrational passive aggressive attack (situation where a state channel underperforms in case of a $\bar{\mu}$-agreement). A seamlessly migration from an off-chain to an on-chain interaction equalizes performances of the two approaches, when such an attack is mounted by an irrational adversary.

6.4.2 Blind challenge analysis

Once a blind challenge is issued, different scenarios may realize.

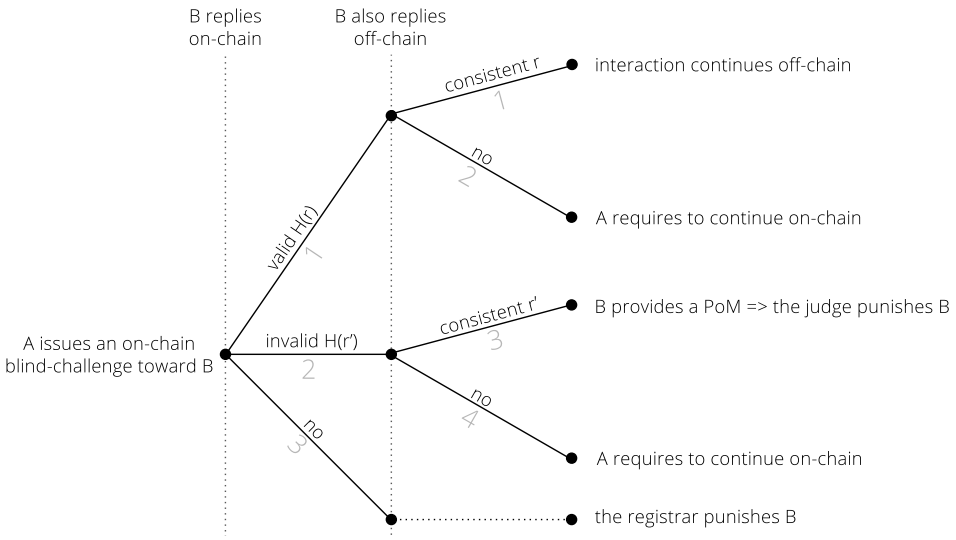

Figure 6.11: Analysis of the blind challenge scenarios.

In the scheme of Figure 6.11 it is assumed that A has issued the challenge. The first turning point is the behavior of B, the challenged party. He can either reply within the grace period or not. If he does not reply, than the evidence is collected by the registrar that punishes him by bestowing his funds to A. Conversely, if B replies, he can reply either with valid or invalid data. Since the on-chain conveyed data is blinded by the application of a hashing function, nor the chain neither A can check the validity until B also proceeds with the off-chain audit, sending non-hashed data directly to A.

First it is analyzed the case when B answers. He can send either consistent or inconsistent data. In the first case A is satisfied and the interaction continues off-chain. Alternatively, if the hash of off-chain sent data does not match the on-chain stored reply, A compels the migration toward on-chain interaction mediated by the judge contract. It is worth noting that the on-chain migration can be triggered any moment by any party that considers the bond trust with the counterparty consumed. In fact, there is no proof required to activate the migration. It would be squandered, since the path $[1, 2]$ of the scheme is always available to a malicious party. Choosing to stay in an off-chain interaction is up to the parties: it is the analogous of the closing procedure triggered in a standard payment (or in a state channel implementing a μ-agreement) when the trusted bond is severed.

In the case B have maliciously answered on-chain with the hash of wrong reply data, he again can opt to either conveyed or not the off-chain audit. If data is not sent off-chain but the on-chain timeout has been stopped by the wrong reply, A requires to continue the interaction on-chain, where the malicious behavior of B is going to be detected. Conversely, if B conveyed wrong although consistent data with the on-chain stored hash, this suffices in determining a proof of misbehaving that A presents to the judge to punish the aggressive conduct of B.

6.4.2.1 Maliciously issued blind challenge

Blind challenge scheme, as for a standard challenge, can be used either honestly or maliciously. It can be the issuer which in the first place triggers a non-needed or wrong and malicious blind challenge. An honest party that detects such a behavior has to trigger the on-chain migration since the trust bond between him and his counterparty does not subsist anymore.

6.5 Usability

Although smart channels even performances of on-chain and off-chain interaction in case of irrational passive aggressive attack, it is advisable to not incur in such an eventuality by opening a channel with a party whose reliability is not reputable.

6.5.1 Performance analysis

In what follows is presented a comparison of the expected cost and execution time for three cases: case C_1, a smart contract; case C_2, a state channel; case C_3, a smart channel. It is assumed that:

1. all three cases implement the same $\bar{\mu}$-agreement involving two parties, Alice A and Berto B;
2. the interaction is composed by n alternate steps, ($\frac{n}{2}$ propose steps and $\frac{n}{2}$ accept steps for each party), as for a turn-base game. The first h steps are honestly performed by both parties, in the remaining $a = n - h$, B mounts an irrational passive aggressive attack;
3. despite the attack, B does not dropout, he always reply within the grace period;
4. all the n steps are required to reach the first satisfactory closing state;
5. all the smart contracts run on the same Ethereum-like platform for smart contract execution (*cfr.* Section 2.1.2);
6. honest interaction takes no time to the purpose of this analysis;
7. challenge grace period lasts t_{grace} and the attacker always successfully replies to on-chain challenges in $t_{\text{reply}} = t_{\text{grace}} - \varepsilon$.

Case A smart contract Since in this case the interaction between parties is mediated by a smart contract, it vanishes by design the possibility for B to mount an irrational passive aggressive attack. This case is considered as a reference point to evaluate performance of the other two. The grace period within which parties must act to avoid dropout is $\check{t}_{\text{grace}} \simeq t_{\text{grace}}$, and, during the attack phase, the attacker always mange to reply in $\check{t}_{reply} \simeq t_{\text{reply}}$.

On-chain transaction counting: **join**: $A \rightarrowtriangle 1$ tx, $B \rightarrowtriangle 1$ tx; **honest phase**: $A \rightarrowtriangle \frac{h}{2}$ tx, $B \rightarrowtriangle \frac{h}{2}$ tx; **attack phase**: $A \rightarrowtriangle \frac{a}{2}$ tx, $B \rightarrowtriangle \frac{a}{2}$ tx; **closing**: A or $B \rightarrowtriangle 1$ tx; **withdrawal**: $A \rightarrowtriangle 1$ tx, $B \rightarrowtriangle 1$ tx. Total transactions: $h + a + 5$. Total time: $\frac{a}{2}\check{t}_{\text{reply}}$ (due to the $\frac{a}{2}$ attacking step from B).

Case B state channel The $\bar{\mu}$-agreement is implemented on a state channel accordig to the following logic: the smart contract that backs the state channel can recognize satisfactory and unsatisfactory closing state and refuses to trigger channel closing if instructed with an unsatisfactory one. Party are therefore obliged to reach a satisfactory state before to trigger channel closing. Hence, during an irrational passive aggressive attack, the honest party can only rely on on-chain challenges to force the attacker to both accept and propose state updates until a satisfactory one is reached. During the attack, one challenge is require to progress of one step. It is worth noting that the expected time estimation has a certain level of uncertainty

Table 6.1: Performance comparison for different implmentation of $\bar{\mu}$-agreements. h is the number of steps honestly performed by both parties. a is the number of attacking steps. t_{reply} is the time the attacker takes to reply to a challenge.

	smart contract	state channel	smart channel
number of txs	$h + a + 5$	$2a + 5$	$a + 6$
required time	$\frac{a}{2} t_{\text{reply}}$	$a t_{\text{reply}}$	$\frac{a}{2} t_{\text{reply}}$

because the actual reply time depends on how well the attacker can answer to the challenge, balancing two opposed interests: the maximization of the t_{reply} and the maximization of the probability to see his transaction mined before grace period expires. The latter one is not completely under attacker's control and one way to maximize the related probability is to decrease t_{reply}. The real t_{reply} will be always less then t_{grace} which constitutes an upper bound.

On-chain transaction counting: **join**: $A \rightarrowtriangle$ 1 tx, $B \rightarrowtriangle$ 1 tx; **honest phase**: only off-chain interaction $\rightarrowtriangle$ 0 tx; **attack phase**: $A \rightarrowtriangle \frac{a}{2}$ tx (challenge propose) + $\frac{a}{2}$ tx (challenge accept), $B \rightarrowtriangle \frac{a}{2}$ tx (reply propose) + $\frac{a}{2}$ tx (reply accept); **closing**: A or $B \rightarrowtriangle$ 1 tx; **withdrawal**: $A \rightarrowtriangle$ 1 tx, $B \rightarrowtriangle$ 1 tx. Total transactions: $2a + 5$. Time time: $a t_{\text{reply}}$ (due to the a challenges required).

Case C smart channel On-chain transaction counting: **join**: $A \rightarrowtriangle$ 1 tx, $B \rightarrowtriangle$ 1 tx; **honest phase**: only off-chain interaction $\rightarrowtriangle$ 0 tx; **attack phase - detection**: $A \rightarrowtriangle$ 1 tx (blind challenge propose), $\rightarrowtriangle$ 1 tx (blind challenge accept); **attack phse - on-chain transition**: $A \rightarrowtriangle$ 1 tx; **attack phse - remaining interaction**: $A \rightarrowtriangle \frac{a}{2} - 1$ tx, $B \rightarrowtriangle \frac{a}{2} - 1$ tx; **closing**: A or $B \rightarrowtriangle$ 1 tx; **withdrawal**: $A \rightarrowtriangle$ 1 tx, $B \rightarrowtriangle$ 1 tx. Total transactions: $a + 6$. Total time: $t_{\text{reply}} + (\frac{a}{2} - 1 \simeq \frac{a}{2} \check{t}_{\text{reply}}$ (due to the blind off-chain challenge and the $\frac{a}{2} - 1$ on-chain attacking steps from B).

Final considerations Table 6.1 summarizes the results of the previous analysis. It is worth noting that the advantage to use a smart channel over a state channel scale linearly with the length of the attack. Smart channel performs not worst than an on-chain smart contract when under attack, while saves the honest on-chain interaction.

6.5.2 Payment channels for economic μ-agreements

The definition of μ-agreements has an interesting implication. Since a μ function exists for each possible intermediate state and parties mutually agreed on its definition, a payment channel suffices in supporting μ-agreements that have an economic outcome, *i.e.,* a reallocation of the locked funds of involved parties.

Rectangles in Figure 6.12 are composed by two parts. The upper one is the actual payment channel state. Bytes in the lower one represent, instead, an arbitrary data structure (whose definition in not needed to the purpose of this discussion). The

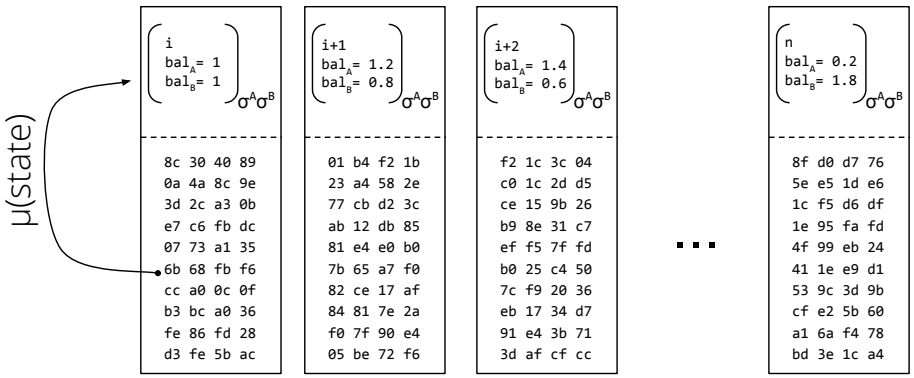

Figure 6.12: Payment channel for μ-agreements.

state definition reveals that the example makes use of a smart contract to support this payment channel. Nevertheless the discussion perfectly applies also to Bitcoin's payment channels.

The underlying blockchain has to be informed only about the existence of a payment channel. The off-chain interaction to propose the i-th state update, for example, by A to B, encompasses the exchange of both the signed state $s_{i,\sigma^A} =< i, \beta_i^A, \beta_i^B >_{\sigma^A}$, relative to the payment channel (the upper part of the rectangles in figure), and the updated data structure d_i that describes the state of the μ-agreement (the lower part of the rectangles in figure). The party that receives the proposal verifies that the payment channel has been rebalanced as established by the application of the μ function to the updated state of the agreement: $(\beta_i^A, \beta_i^B) = \mu(d_i)$. Only in this case B accepts the update and sends his signature back to A.

If the proposed update does not comply with the result of the μ function, *i.e.,* $(\beta_i^A, \beta_i^B) \neq \mu(d_i)$, than the trust bond between parties is not holding any longer and B closes the channel by broadcasting the last valid agreed upon payment state s_{i-1}.

The μ function can be complex at will since there is no need to code its behavior in a smart contract and it is only applied off-chain.

Chapter 7

Conclusion

State channels are a valuable tool to support blockchain scalability. Understanding their capabilities and limitations is a fundamental requirement to see their broadly deployment, without which significant blockchain scalability remains only a theoretic possibility.

In this work two substantial aspects are explored that support the adoption of payment state channels on large scale: 1) the extension of their lifetime when only a technical hitch (namely the funds displacement) requires for closing; 2) the increase in the number of ways for using funds blocked in a channel through hybrid payment enabled by a central hub. Furthermore an analysis of the implicit contractual consequences of the arbitrary state that can be agreed upon off-chain is provided, along with a practical solution to counteract underperforming (with respect to purely on-chain solutions) situations that may originate from attacks by irrational adversaries.

Research field on state channels is, however, wide open and at the very beginning of its path. No rigorous formalization or even a commonly accepted notation have been defined yet. Proposed contributions only scratched the surface of a vast area that is to be further investigated. In fact, despite being defined to address frequent micropayments in the blockchain context, it is a strong belief of the author that state channels (or their evolutions) will survive the blockchain technology. They are of their own interest and blockchain is just the best settlement layer that can be currently offered them. Future researches on this field should aim to decouple channel implementation from a specific blockchain. One possible solution consists in providing several interchangeable adapters and enabling one off-chain interaction to produce multiple proofs, viable for multiple underling blockchains. This would support the long searched blockchain interoperability, by means of different denomination for channel collateral, as it is already possible nowadays by virtue of the numerous tokens available on the Ethereum platform. Even blockchain itself can be substituted with a whole range of solutions spanning from trusted hardware (TEE) to traditional settlement system supported by standard legal/banking structures.

Furthermore, the totally private nature of the agreement between parties involved in a state channel paves the way for an essential piece for the deployment of a full-blown digital economy: the credit. A Fulgur hub, for example, can act as a bank and credit one of its client to pay through itself when no collateral is blocked on the client's side. The same applies for an intermediate node of a payment network. Credit, however, implies trust which is a sensible topic in the crypto-community. State channels are acting as trailblazers also on this aspect. In fact, a certain

amount of trust is required, to open a channel with a counterparty, where funds can be blocked for a certain amount of time resulting in the loss of opportunity cost of the collateral. Despite that, they are widely accepted, also by the most trust-adverse figures of the community.

Acknowledgments

The author acknowledges Dr. Fabio Castaldo and Sebastiano Scròfina for continuous exchanges and intense collaboration on topics presented and discussed in this work.

Bibliography

[1] B. community. (). Bitcoin developer documentation, [Online]. Available: https://bitcoin.org/en/developer-documentation (visited on 10/11/2017) (*cited on p. 5*).

[2] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014 (*cited on p. 5*).

[3] M. Swan, *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015 (*cited on p. 5*).

[4] D. Tapscott and A. Tapscott, *Blockchain Revolution: How the technology behind Bitcoin is changing money, business, and the world*. Penguin, 2016 (*cited on p. 5*).

[5] R. Wattenhofer, *The science of the blockchain*. CreateSpace Independent Publishing Platform, 2016 (*cited on p. 5*).

[6] P. Vigna and M. Casey, *The age of crypto currency*, 2016 (*cited on p. 5*).

[7] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger", *Ethereum Project Yellow Paper*, vol. 151, 2014 (*cited on p. 9*).

[8] L. Goodman, *Tezos: A self-amending crypto-ledger position paper*, 2014. [Online]. Available: https://www.tezos.com/static/papers/white_paper.pdf (*cited pp. 11, 15*).

[9] block.one. (2017). Eos.io technical white paper, [Online]. Available: https://github.com/EOSIO/Documentation/blob/master/TechnicalWhitePaper.md (visited on 09/19/2017) (*cited on p. 11*).

[10] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system", 2008 (*cited pp. 11, 13*).

[11] D. Chaum, "Blind signatures for untraceable payments", in *Advances in cryptology*, Springer, 1983, pp. 199–203 (*cited pp. 12, 61, 81*).

[12] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", *Communications of the ACM*, vol. 21, no. 2, pp. 120–126, 1978 (*cited on p. 12*).

[13] M. J. Fischer, N. A. Lynch, and M. S. Paterson, "Impossibility of distributed consensus with one faulty process", *Journal of the ACM (JACM)*, vol. 32, no. 2, pp. 374–382, 1985 (*cited on p. 12*).

[14] L. Lamport, "The part-time parliament", *ACM Transactions on Computer Systems (TOCS)*, vol. 16, no. 2, pp. 133–169, 1998 (*cited on p. 12*).

[15] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem", *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982 (*cited on p. 13*).

[16] M. Castro, B. Liskov, *et al.*, "Practical byzantine fault tolerance", in *OSDI*, vol. 99, 1999, pp. 173–186 (*cited on p. 13*).

[17] L. Lamport, "Fast paxos", *Distributed Computing*, vol. 19, no. 2, pp. 79–103, 2006 (*cited on p. 13*).

[18] C. Copeland and H. Zhong, *Tangaroa: A byzantine fault tolerant raft*, 2016 (*cited on p. 13*).

[19] D. Ongaro and J. K. Ousterhout, "In search of an understandable consensus algorithm.", in *USENIX Annual Technical Conference*, 2014, pp. 305–319 (*cited on p. 13*).

[20] J. R. Douceur, "The sybil attack", in *International Workshop on Peer-to-Peer Systems*, Springer, 2002, pp. 251–260 (*cited on p. 13*).

[21] J. A. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications.", in *EUROCRYPT (2)*, 2015, pp. 281–310 (*cited on p. 13*).

[22] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint", 2014 (*cited on p. 13*).

[23] S. King and S. Nadal, "Ppcoin: Peer-to-peer crypto-currency with proof-of-stake", *self-published paper, August*, vol. 19, 2012 (*cited on p. 13*).

[24] I. Bentov, C. Lee, A. Mizrahi, and M. Rosenfeld, "Proof of activity: Extending bitcoin's proof of work via proof of stake", *ACM SIGMETRICS Performance Evaluation Review*, vol. 42, no. 3, pp. 34–37, 2014 (*cited on p. 13*).

[25] S. Micali, "Algorand: The efficient and democratic ledger", *arXiv preprint arXiv:1607.01341*, 2016 (*cited on p. 13*).

[26] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol", in *Annual International Cryptology Conference*, Springer, 2017, pp. 357–388 (*cited on p. 13*).

[27] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. Gün Sirer, D. Song, and R. Wattenhofer, "On scaling decentralized blockchains", in *Financial Cryptography and Data Security: FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, J. Clark, S. Meiklejohn, P. Y. Ryan, D. Wallach, M. Brenner, and K. Rohloff, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 106–125, ISBN: 978-3-662-53357-4. DOI: 10.1007/978-3-662-53357-4_8. [Online]. Available: https://doi.org/10.1007/978-3-662-53357-4_8 (*cited on p. 14*).

[28] A. Gervais, G. Karame, S. Capkun, and V. Capkun, "Is bitcoin a decentralized currency?", *IEEE security & privacy*, vol. 12, no. 3, pp. 54–60, 2014 (*cited on p. 15*).

[29] E. Heilman, F. Baldimtsi, L. Alshenibr, A. Scafuro, and S. Goldberg, "Tumblebit: An untrusted tumbler for bitcoin-compatible anonymous payments.", *IACR Cryptology ePrint Archive*, vol. 2016, p. 575, 2016 (*cited pp. 15, 62*).

[30] D. F. Primavera and B. LOVELUCK, "The invisible politics of bitcoin: Governance crisis of a decentralized infrastructure", *Internet Policy Review*, vol. 5, no. 4, 2016 (*cited on p. 15*).

[31] D. Siegel. (2016). Understanding the dao attack, [Online]. Available: https://www.coindesk.com/understanding-dao-hack-journalists/ (visited on 09/19/2017) (*cited on p. 15*).

[32] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, *Enabling blockchain innovations with pegged sidechains*, 2014 (*cited on p. 15*).

[33] E. G. Sirer. (2015). Bitcoin runs on altruism, [Online]. Available: http://hackingdistributed.com/2015/12/22/bitcoin-runs-on-altruism/ (visited on 09/18/2017) (*cited on p. 16*).

[34] M. Carlsten, H. Kalodner, S. M. Weinberg, and A. Narayanan, "On the instability of bitcoin without the block reward", in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2016, pp. 154–167 (*cited on p. 16*).

[35] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable", in *International conference on financial cryptography and data security*, Springer, 2014, pp. 436–454 (*cited on p. 16*).

[36] R. Pass and E. Shi, "Fruitchains: A fair blockchain", in *Proceedings of the ACM Symposium on Principles of Distributed Computing*, ACM, 2017, pp. 315–324 (*cited on p. 16*).

[37] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network.", in *USENIX Security*, 2015, pp. 129–144 (*cited on p. 16*).

[38] N. Atzei, M. Bartoletti, and T. Cimoli, "A survey of attacks on ethereum smart contracts (sok)", in *International Conference on Principles of Security and Trust*, Springer, 2017, pp. 164–186 (*cited on p. 17*).

[39] E. Hildenbrandt, M. Saxena, X. Zhu, N. Rodrigues, P. Daian, D. Guth, and G. Rosu, "Kevm: A complete semantics of the ethereum virtual machine", Tech. Rep., 2017 (*cited on p. 17*).

[40] @peoplewindow. (2017). Solidity design issues on hacker news, [Online]. Available: https://news.ycombinator.com/item?id=14691212 (visited on 09/19/2017) (*cited on p. 17*).

[41] Etherscan. (2017). Solidity bug info, [Online]. Available: https://etherscan.io/solcbuginfo (visited on 09/19/2017) (*cited on p. 17*).

[42] J. Spilman. (2013). Anti dos for tx replacement, [Online]. Available: https://lists.linuxfoundation.org/pipermail/bitcoin-dev/2013-April/002433.html (visited on 10/06/2017) (*cited on p. 24*).

[43] M. Hearn. (2013). Micro-payment channels implementation now in bitcoinj, [Online]. Available: https://bitcointalk.org/index.php?topic=244656.0 (visited on 10/06/2017) (*cited on p. 24*).

[44] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels", in *Symposium on Self-Stabilizing Systems*, Springer, 2015, pp. 3–18 (*cited pp. 24, 25, 36, 37*).

[45] J. Poon and T. Dryja, *The bitcoin lightning network: Scalable off-chain instant payments*, 2015. [Online]. Available: https://lightning.network (*cited pp. 24, 25, 37, 58*).

[46] A. Heifetz, M. Meier, and B. C. Schipper, "A canonical model for interactive unawareness", in *Proceedings of the 11th conference on Theoretical aspects of rationality and knowledge*, ACM, 2007, pp. 177–182 (*cited on p. 26*).

[47] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services", in *ACM SIGOPS operating systems review*, ACM, vol. 39, 2005, pp. 45–58 (*cited on p. 33*).

[48] R. Khalil and A. Gervais, *Revive: Rebalancing off-blockchain payment networks*, Cryptology ePrint Archive, Report 2017/823, http://eprint.iacr.org/2017/823, 2017 (*cited on p. 37*).

[49] S. Dziembowski, L. Eckey, S. Faust, and D. Malinowski, *Perun: Virtual payment channels over cryptographic currencies*, Cryptology ePrint Archive, Report 2017/635, http://eprint.iacr.org/2017/635, 2017 (*cited on p. 37*).

[50] J. Benaloh and M. De Mare, "One-way accumulators: A decentralized alternative to digital signatures", in *Workshop on the Theory and Application of of Cryptographic Techniques*, Springer, 1993, pp. 274–285 (*cited on p. 52*).

[51] N. Barić and B. Pfitzmann, "Collision-free accumulators and fail-stop signature schemes without trees", in *Advances in Cryptology—EUROCRYPT'97*, Springer, 1997, pp. 480–494 (*cited on p. 52*).

[52] K. Nyberg, "Fast accumulated hashing", in *Fast Software Encryption*, Springer, 1996, pp. 83–87 (*cited on p. 52*).

[53] N. Fazio and A. Nicolosi, "Cryptographic accumulators: Definitions, constructions and applications", *Paper written for course at New York University: www.cs.nyu.edu/nicolosi/papers/accumulators. pdf*, 2002 (*cited on p. 52*).

[54] DJC. (2016). Lightning network: Will it save bitcoin? or break it?, [Online]. Available: http://www.wallstreettechnologist.com/2016/10/03/lightning-network-will-it-save-bitcoin-or-break-it (visited on 10/11/2017) (*cited on p. 59*).

[55] J. Fyookball. (2017). Mathematical proof that the lightning network cannot be a decentralized bitcoin scaling solution, [Online]. Available: https://medium.com/@jonaldfyookball/mathematical-proof-that-the-lightning-network-cannot-be-a-decentralized-bitcoin-scaling-solution-1b8147650800 (visited on 10/11/2017) (*cited on p. 59*).

[56] ——, (2017). Continued discussion on why lightning network cannot scale, [Online]. Available: https://medium.com/@jonaldfyookball/continued-discussion-on-why-lightning-network-cannot-scale-883c17b2ef5b (visited on 10/11/2017) (*cited on p. 59*).

[57] J. Stolfi. (2017). Responding to murch's "responding to jonald fyookball's article" article, [Online]. Available: https://www.reddit.com/r/btc/comments/6jxem4/responding_to_murchs_responding_to_jonald (visited on 10/19/2017) (*cited on p. 59*).

[58] M. Green and I. Miers, "Bolt: Anonymous payment channels for decentralized currencies", Cryptology ePrint Archive, Report 2016/701, Tech. Rep., 2016 (*cited on p. 61*).

[59] T. Bocek, S. Rafati, B. Rodrigues, and B. Stiller, "Coinblesk–a real-time, bitcoin-based payment approach and app", *Blockchain Engineering*, p. 14, (*cited on p. 62*).

[60] J. Poon and B. Vitalik, *Plasma: Scalable autonomous smart contracts*, 2017 (*cited on p. 62*).

[61] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks", in *Stabilization, Safety, and Security of Distributed Systems: 19th International Symposium, SSS 2017, Boston, MA, USA, November 5–8, 2017, Proceedings*, P. Spirakis and P. Tsigas, Eds. Cham: Springer International Publishing, 2017, pp. 361–377, ISBN: 978-3-319-69084-1. DOI: 10.1007/978-3-319-69084-1_26. [Online]. Available: https://doi.org/10.1007/978-3-319-69084-1_26 (*cited on p. 62*).

[62] N. Szabo, "Formalizing and securing relationships on public networks", *First Monday*, vol. 2, no. 9, 1997 (*cited on p. 88*).

[63] ——, "Smart contracts", *Unpublished manuscript*, 1994 (*cited on p. 88*).

[64] I. Grigg, "The ricardian contract", in *Proceedings. First IEEE International Workshop on Electronic Contracting, 2004.*, Jul. 2004, pp. 25–31. DOI: 10.1109/WEC.2004.1319505 (*cited on p. 90*).

[65] J. Izquierdo. (2017). Advanced solidity code deployment techniques, [Online]. Available: https://blog.aragon.one/advanced-solidity-code-deployment-techniques-dc032665f434 (visited on 10/06/2017) (*cited on p. 95*).