Blockchain: Research and Applications 7 (2026) 100339

Contents lists available at ScienceDirect

Blockchain: Research and Applications

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

Research Article

Smart listeners: A hybrid-optimistic inter-blockchain communication protocol

Alessandro Bigiotti [a,*], Leonardo Mostarda [b], Alfredo Navarra [b], Andrea Pinna [c], Roberto Tonelli [c], Matteo Vaccargiu [c]

[a] Division of Computer Science, University of Camerino, Via Madonna delle Carceri, 9, Camerino 62032, Italy
[b] Department of Mathematics and Computer Science, University of Perugia, Via Vanvitelli, 1, Perugia 06123, Italy
[c] Department of Mathematics and Computer Science, University of Cagliari, Via Ospedale, 72, Cagliari 09124, Italy

ARTICLE INFO

Keywords:
Blockchain
Consensus algorithm
Interoperability
Cross-chain
Inter-chain protocol

ABSTRACT

In recent years, blockchain technology has seen significant practical growth, yet it has not seen the same advancement from a theoretical perspective. This has led to the creation of numerous blockchains that are very different from each other and behave like isolated worlds. The research and development of theoretical frameworks, which define the fundamental properties of blockchains and define standards to follow for a more homogeneous implementation approach, have become extremely important. A theoretical model can help not only to design blockchains in the future but also to define a set of minimum requirements to be met for the creation of interoperability protocols between existing blockchains. In this work, we propose a theoretical model of blockchain that describes its most significant properties. Starting from our theoretical model, we present *Smart listeners*, a blockchain interoperability protocol that finalises an inter-chain transaction with just two transactions: one on the source blockchain and one on the destination blockchain. The protocol is optimistic since changes on the source blockchain occur as if inter-chain transactions were successful. It is a hybrid since it combines some properties of watchtowers and oracles in the off-chain components. We provide a benchmark on the performance of the proposed protocol in terms of latency and transactions per second. Finally, we define the minimum requirements that a blockchain should satisfy to allow the application of general-purpose interoperability protocols.

1. Introduction

Blockchain, at its core, is a peer-to-peer network of untrusted nodes. By relying on cryptography and distributed algorithms, the network participants are able to exchange transactions that are validated by the network itself. Cryptography has a dual function in blockchains; asymmetric digital signatures are used to authenticate participating nodes, whilst hash functions are used to ensure tamper-proof transaction data. The nodes of the network reach an agreement on the transactions that have occurred using distributed algorithms, which are called consensus algorithms. The transactions are kept in a distributed ledger as a linked list of signed blocks. At the dawn of this technology, we find Bitcoin, which offers a distributed payment system implementing cryptocurrency [1]. More recent blockchains, such as Ethereum [2] and Algorand [3], in addition to cryptocurrency exchanges, also allow the development of smart contracts, digital contracts that are executed within the blockchain [4]. Each blockchain implements its own cryptographic scheme, a dedicated consensus algorithm and allows the development of smart contracts with specific programming languages. These features make blockchains isolated and self-sufficient worlds. This means the blockchains are unable to communicate with each other. Other differences can be found in governance [5], and we can distinguish between permissionless blockchains, where anyone can participate, and permissioned blockchains, where only a predefined set of nodes can interact with the network.

* Corresponding author.
E-mail addresses: alessandro.bigiotti@unicam.it (A. Bigiotti), leonardo.mostarda@unipg.it (L. Mostarda), alfredo.navarra@unipg.it (A. Navarra), pinna.andrea@unica.it (A. Pinna), roberto.tonelli@unica.it (R. Tonelli), matteo.vaccargiu@unica.it (M. Vaccargiu).

https://doi.org/10.1016/j.bcra.2025.100339
Received 23 January 2024; Received in revised form 23 January 2025; Accepted 7 May 2025

Available online 16 July 2025

In recent years, interest in this technology has grown considerably, and dozens of blockchains with very different characteristics have emerged. Thus, the search for interoperability protocols that allow generic interactions has become essential for the applicability and survival of this technology.

Numerous solutions have been proposed in the literature to make blockchains communicate, mainly with the aim of carrying out cryptocurrency exchanges [6–11] or increasing scalability and extending features with sidechains [12–15]. Standards have also recently emerged that aim at leading interoperability between blockchains, such as inter-blockchain communication [16] and cross-chain message passing [17] adopted by Cosmos [18] and Polkadot [19], respectively. Also, interoperability based on off-chain Oracles has been proposed as the cross-chain interoperability protocol by Chainlink [20,21] and Gravity [22].

However, the proposed solutions lack solid theoretical foundations to guarantee the safety and liveness properties of the implemented protocols. Moreover, the above-mentioned solutions are designated for public blockchains, where the privacy requirement is not a central factor. Nowadays, the demand for the integration of blockchains in business contexts, such as Industry 4.0 [23,24], also requires that blockchain data is somehow preserved and not disclosed. The authors in Ref. [25] also highlighted how the problem of interoperability between different blockchains is a field to be explored. Despite the proposal of various techniques, there is still a gap in the development of secure and efficient strategies that avoid any bottlenecks in the so-called cross-chain transactions, especially for consortium-based blockchains.

This paper proposes a theoretical contribution, formalising the blockchain by emphasising the main characteristics that make interoperability very complicated. In particular, as outlined by the authors in Ref. [26], the main issues we need to consider concern the heterogeneous models implemented in different blockchains. The first aspect involves very different consensus protocols; thus, close attention must be paid to the finality of the consensus protocols (deterministic or non-deterministic), as well as the latency and block production time. Again, in accordance with Ref. [26], the second aspect involves the use of very different cryptographic primitives, and this could concern the lack of efficient protocols for interoperability between blockchains.

Keeping these properties in mind, we define an interoperability protocol between different blockchains that is of general purpose and that somehow preserves the privacy of the blockchains involved (i.e., only the data object of the inter-chain transaction must be accessible from the other blockchain). The protocol is mainly intended for private or consortium permissioned blockchains. In particular, in our theoretical framework, we want to define the technical interoperability [27] that the protocol must follow. Finally, we will answer the open questions highlighted by the authors in Ref. [27]: *Q.1*: What are the technical requirements that a blockchain should meet to enable general-purpose interoperability protocols? *Q.2*: What are the technical requirements an inter-chain protocol must provide to assure safety and liveness properties?

1.1. Paper structure

The paper is structured as follows: in Section 2, some works that include theoretical approaches to blockchains are provided and the motivations that led us to this work are given. In Section 3, our formalisation of blockchain is given, emphasising the differences in the cryptography and the consensus algorithms, and a formal definition of an inter-chain transaction is given. In Section 4, the properties of the third party responsible for implementing inter-chain transactions are defined. In Section 5, an overview of the operations of the protocol is presented, showing the behaviour in different failure cases, including Byzantine faults. In Section 6, the procedures performed by the inter-chain protocol during the various steps are detailed, and discussions about safety and liveness properties are provided. In Section 7, we provide a series of examples demonstrating the applicability of the proposed protocol, along with a performance benchmark measuring transaction latency and throughput. In Section 8, some considerations are made about the protocol and the answers to questions *Q.1* and *Q.2* are given. In Section 9, the results and open points that we wanted to address are highlighted, and a series of useful observations are provided for future work.

2. Background and motivations

Although blockchain technology has already found ample confirmation from an applied point of view, the same cannot be said from a theoretical perspective. There are not many works in the literature that try to formalise the blockchain with the aim of defining a series of properties or minimum requirements useful for guiding the design and development of future blockchains in order to simplify the interoperability processes between them. In this section, we show some works in the literature that have tried to formalise the blockchain.

One of the earliest theoretical works on formalising distributed ledgers was presented in Ref. [28], where the operations and features that a distributed ledger should support are provided, along with an appropriate formulation of a distributed ledger itself. The authors define the formal structure, ledgers, required properties and procedures that can be executed. Being one of the first works attempting to formalise blockchain, it still lacks many details and case histories. As the authors conclude, they do not address potential Byzantine behaviours and acknowledge that they have only just begun to explore the topic. They emphasise the need for further research to fully understand and properly formalise blockchain technology.

One of the first works that tries to formalise the interoperability between blockchains can be found in Ref. [29]. The authors provide a formal blockchain model on the basis of which they give a definition of interoperability between different blockchains. In their basic definition of blockchain, they demonstrate that interoperability is not viable. By relaxing some constraints on their assumptions, they conclude that interoperability between blockchains is possible, but the blockchains would degenerate into a single blockchain with two distinct ledgers, what they call a “2-in-1” blockchain.

The authors in Ref. [26] propose a formalisation of the blockchain focusing on the properties of the fair exchange problem. In their theoretical model, they summarise the general properties that every blockchain interoperability protocol should have. The first conclusion they draw about what they call a correct cross-chain communication (CCC) protocol is the recommended presence of a trusted third party who is responsible for managing communications. With this in mind, they create a framework for creating new CCC protocols and evaluating existing ones. The same authors propose a series of guidelines for the design of protocols for asset exchange and data migration. The authors themselves leave a series of open points that inspired the present work. In particular, in our model, we explore the properties that an inter-chain communication protocol should satisfy for blockchains that present profoundly different characteristics, both in the consensus algorithm adopted, which drastically affects transaction throughput and in the cryptographic primitives used to manage authentication and data integrity.

2.1. Interoperability among blockchains: fair exchange problem or distributed transactions?

Interoperability between blockchains is a problem that requires the transfer of information from one blockchain to another. In the literature, we can find numerous works that have described and addressed similar problems. In this section, we want to show the similarities between the problem of interoperability between blockchains and the fair exchange problem, as well as the problem of distributed transactions.

The first problem that presents similar characteristics is the fair exchange problem. In the fair exchange problem, we have two parties P and Q that have certain information i_P and i_Q, respectively. The aim is

to ensure that P and Q exchange information and the possible results can be: *success*, in which case both P and Q possess the reciprocal information; *abort*, in which case neither P nor Q possesses the reciprocal information. The two parties P and Q must reach an agreement on the information i_P and i_Q to be exchanged. Generally, the respective information is associated with a description d_{i_P} and d_{i_Q} used to guide the exchange and verify that the parties are behaving honestly. The content and definition of the specific information are strictly linked to the specific use case. For instance, users can exchange goods [30] for which they know the information in advance, e.g., cryptocurrencies between different blockchains, tokens, or the sale of a certain good in exchange for money. In other cases, users do not know the information about the goods they are exchanging, e.g., the purchase of a file [31] where the information is divided into smaller portions which are exchanged incrementally.

Any solution to the fair exchange problem must guarantee the following properties [32,33]:

Effectiveness: If both parties P and Q behave correctly and do not want to abandon the exchange, when the protocol has been completed P has i_Q and Q has i_P;

Timeliness: An honest party will eventually reach a termination state, either success or abort;

(*Strong*) *Fairness*: There is no outcome of the protocol in which P receives i_Q and Q does not receive i_P (and vice versa).

The authors in Ref. [32] provide a reduction of the consensus problem to the fair exchange problem, showing that the fair exchange problem is a particular case of the decision problem. In the decision, or consensus, problem, we have a set of participants $\mathcal{P} := \{p_1, \dots, p_n\}$ that fulfil a set of possible tasks $\Upsilon := \{v_1, v_2, \dots, v_k\}$. The tasks performed by the participants end in a set of possible valid states $T := \{t_1, t_2, \dots, t_p\}$. The participants can be honest or malicious, leading to Byzantine faults [34,35]. Two fundamental properties must be guaranteed in distributed protocols: the *safety* property guarantees the correctness of the protocol, and the *liveness* property guarantees the termination of the protocol [36–39]:

Agreement: is a safety property ensuring that all the honest parties p_i agree on the same value or decision $v(p_i)$. In other words, it cannot happen that two different honest parties reach a conflict:

$$v_i(p_i) = v_i(p_j),\ \forall v_i \in \Upsilon,\ \forall p_i, p_j \in \mathcal{P}$$

Validity: is a safety property ensuring that the value or decision provided by the system is a valid one. It guarantees that the system does not get stuck in an incorrect state:

$$v_i(p_j) \in T,\ \forall v_i \in \Upsilon, p_j \in \mathcal{P}$$

Termination: is a liveness property ensuring that the protocol eventually produces a result or decision, even in the presence of Byzantine faults. It prevents the system from entering an indefinite or non-terminating state:

$$v_i(p_j) \neq \emptyset,\ \forall v_i \in \Upsilon, p_j \in \mathcal{P}$$

The authors in Ref. [26], exploiting the characteristics of the fair exchange problem, propose a reduction of the interoperability problem to the fair exchange problem. Thus, they propose a CCC protocol that, satisfying the properties of *Effectiveness, Timeliness* and *(Strong) Fairness*, solves the interoperability problem among blockchains. In particular, these properties must be satisfied by a protocol that allows the effective exchange of information.

Blockchains are distributed ledgers that maintain data; hence, we also find similarities with the problem of distributed transactions [40]. The criteria and requirements adopted for the study of distributed transaction systems are inherited in interoperability systems between blockchains since they share analogous issues.

Transactions must comply with the following four (ACID) properties [41]:

Atomicity: The transaction is indivisible. It must ensure that all the operations of a transaction are completed (success) or none of them are executed (abort), it must not happen that some of them are completed and others not;

Consistency: The transaction does not violate system invariants. It allows the preservation of data integrity within the system;

Isolation: Concurrent transactions do not interfere with each other;

Durability: Once a transaction is finalised, the changes are permanent. More precisely, it ensures that the data alteration is irreversible and that data will not be lost even in the event of a system failure.

In order to ensure the ACID properties, several protocols have been proposed in the literature for distributed transaction management. Some methods, such as two-phase commit [42] and the try, confirm and cancel [43], prepare the transaction and then seek consensus among the network to commit or rollback. Other mechanisms involve messaging to perform a transaction: local messaging [44] and transactional messaging [45]. In these approaches, the initial notifier ensures that the message is sent and received at the receiving end. In other words, the reliability of the message is ensured by the notifier. Depending on the result of the messages exchanged, either a commit or a rollback takes place. Another approach is called best effort notification [46], in which parties can make queries about notifications. With this model, the initiating notifier does its best to notify the recipient of the business processing result, but the message may still not be received. As a result, the receiving side must actively call the initiator notifier interface to query the business processing result, which means that the reliability of the notification relies on the receiving side. Finally, another type of solution for managing distributed transactions is the one proposed by the authors in Ref. [47], in which they introduce the concept of "saga". This requires dividing a transaction into many smaller transactions, each of which must be executed atomically. If one of the transactions fails, a reverse-order execution is performed to restore data consistency.

Our approach was influenced by keeping in mind both the safety and liveness properties of the consensus problem, and also by keeping in mind the properties of distributed transactions. In particular, the properties of the protocols that solve the fair exchange problem and the consensus problem will be used to prove the safety and liveness of the interoperability protocol that we will present in the paper. Additionally, the properties of distributed transactions help verify the correctness of the operations carried out by the protocol and the state consistency of the involved blockchains.

2.2. Contribution

1. Present a theoretical model that emphasises the properties and constraints of the blockchain resulting from consensus algorithms and cryptographic primitives.
2. Define an inter-chain transaction and its properties. Present *Smart listener*, an interoperability protocol that aims at preserving the privacy of the involved blockchains. In the protocol, off-chain components do not communicate with each other, and only two transactions are required to finalise an inter-chain transaction. Therefore, we show a performance benchmark of the proposed solution.
3. Answer the questions: *Q.1*: What are the technical requirements that a blockchain should meet to enable general-purpose interoperability protocols? *Q.2*: What are the technical requirements an inter-chain protocol must provide to assure safety and liveness properties?

3. Notation and terminology

In this section, a formalisation of the blockchain is provided, and a series of terminologies will be introduced that will help us to present an interoperability protocol that is independent of the particular blockchain. To this end, this section introduces a formalisation of the blockchain that focuses mainly on the aspects of cryptography and distributed algorithms, which, in our opinion, are the cornerstones of the blockchain structure. Exploiting the properties of the theoretical model, we will define an interoperability protocol between blockchains.

In what follows, we provide a formalisation of the blockchain that manages to capture the greatest number of its characteristics and properties. To do so, we will define the basic elements of the blockchain, and then we will define a blockchain in general terms. The list of all symbols is reported in Tables A.1 and A.2 in Appendix A.

3.1. Digital signature and hash functions

One of the central aspects of blockchain concerns cryptography. Cryptography allows to guarantee authenticity between blockchain participants and the integrity of the data maintained in the distributed ledger. To guarantee authenticity, digital signatures are used [48]. A cryptographic digital signature scheme can be defined as, [49,50]:

Definition 1 *(Digital signature).* A Digital signature scheme is a tuple of algorithms $\Sigma := <G, S, V>$, where:

- $G(\kappa) \rightarrow (pk, sk)$ is the key generation algorithm that, given a security parameter κ, generates a pair of keys (pk, sk); pk is called public key and sk is called private key or secret key;
- $S(m, sk) \rightarrow \sigma(m)$ is the signing algorithm that, given a generic message m and a private key sk, generates a signature $\sigma(m)$ associated to the message m;
- $V(m, \sigma(m), pk) \rightarrow \{0, 1\}$ is the verification function that, given a message m, its signature $\sigma(m)$ and a public key pk, outputs 1 if the signature is valid, 0 otherwise.

The concept of a signature scheme can be extended to contexts in which multiple parties cooperate. The parties must act as a single entity. To this end, a scheme called threshold signature has been proposed, which is defined as follows, [51]:

Definition 2 *(Threshold signature).* A (t, n)-Threshold signature scheme is a tuple of algorithms $\Sigma^{(t,n)} := < G_n, S, V >$, where:

- $G_n(\kappa) \rightarrow < pk, vk, [sk_1, sk_2, \ldots, sk_n] >$ is the key generation algorithm that, given a security parameter κ, generates a public key pk, a verification key vk and a list of secret keys or private keys $[sk_1, \ldots, sk_n]$ owned by the single participants. All participants know the public key pk and the verification key vk;
- $S(m, vk, < \sigma_1, \sigma_2, \ldots, \sigma_t >) \rightarrow \sigma(m)$ is the signing algorithm that, given a generic message m, the verification key vk and the partial signatures σ_i for m, generates a signature $\sigma(m)$ associated to the message m. The verification key vk is used to verify the correctness of the individual signature σ_i. If the minimum number of signature t is reached, the signer will output the global signature $\sigma(m)$;
- $V(m, \sigma(m), pk) \rightarrow \{0, 1\}$ is the verification function that, given a message m, its signature $\sigma(m)$ and the public pk, outputs 1 if the signature is valid, 0 otherwise.

The (t, n)-threshold signature allows signing a message m only if at least a minimum number t of the total participants n signs the message. One of the main peculiarities of the threshold signature is that its verification is carried out as if it were a normal signature.

Hash functions are used to guarantee integrity [52,53]. A hash function, in its most general form, can be defined as:

Definition 3 *(Hash Function).* A hash function is a function defined as:

$$H := D \rightarrow R$$

where the domain $D := \{0, 1\}^*$ and the co-domain $R := \{0, 1\}^n$, for some fixed $n \geq 1$.

According to Ref. [54], a hash function H should satisfy some requirements in order to be applied for security purposes, i.e.,:

I. H can be applied to data of any bounded length;
II. Given any input m, it is polynomial to compute $H(m)$ (also referred to as the *digest* message);
III. Given H and $H(m)$, it is computationally hard to find m;
IV. Given H and m, it is computationally hard to find m' such that $H(m) = H(m')$;
V. Given H, it is computationally hard to find m and m' such that $H(m) = H(m')$.

The requirements I and II are for a practical perspective, making the hash functions applicable and efficient. The requirement III, also known as *pre-image resistant* or *one-way property*, states that given a message digest $H(m)$ it is hard to reconstruct the original message m. The requirement IV, also known as *second pre-image resistant*, guarantees that an alternative message hashing to the same code as a given message cannot be found. Finally, the requirement V, also known as *collision resistant*, guarantees that it should be computationally infeasible to find any two distinct messages with the same digest.

Hash functions are used within the blockchain as proof of the integrity of the blocks composing the blockchain itself. In the blockchain, each transaction and each block are marked with its hash.

Blockchains use different cryptographic primitives for the authentication of the nodes participating in the network and for the integrity of the data exchanged between the nodes. Just as an example, Bitcoin [1] uses the elliptic curve digital signature algorithm (ECDSA) based on asymmetric cryptography [55] to authenticate the nodes, while it uses Secure Hash Algorithm (SHA)-256 for data integrity. Ethereum [2] uses ECDSA like Bitcoin to authenticate the nodes, while it uses SHA-3, also known as keccak-256 [56], for data integrity. The use of different cryptographic schemes constitutes a barrier to building interoperability protocols. In fact, even if Bitcoin uses the same cryptographic scheme as Ethereum to authenticate the nodes (both use ECDSA and make use of the same elliptic curve, that is, the secp-256k1), interoperability remains difficult since each scheme has a different instance on the two blockchains and the participants cannot be shared between the two blockchains. The integrity interoperability is even more complicated, as the schemes are completely different; therefore, the generation and verification of hashes is not compatible between the two blockchains. A generic blockchain interoperability protocol should take these differences into consideration and should be able to manage them at their best.

3.2. Distributed protocols and consensus algorithm

Another cornerstone of the blockchain concerns the use of distributed protocols for maintaining events that have occurred between the participants of the blockchains themselves. In general, a distributed protocol meets the property of *correctness* and *termination* if and only if both the property of *safety* and *liveness* are verified. Mainly, the distributed protocols within the blockchain are needed to implement the so-called *consensus algorithm*, an algorithm that allows network participants to agree on what happens within a given period of time. Theoretically, this type of protocol finds its foundations in the problem of the Byzantine generals [35]. One of the first consensus protocols was the practical Byzantine fault-tolerant algorithm (PBFT) [34], a protocol for state machine replication. The protocol proceeds in rounds, each

round is composed of three steps, in which the parties must agree on the correctness of the computations and the correctness of the states. Nowadays, several variations of the PBFT protocols are proposed; one of the most recent is the Istanbul Byzantine fault-tolerant algorithm (IBFT) [57]. These kinds of protocols are sometimes referred to as proof of authority (PoA) ones [58], and are mostly used in private blockchains. A widely used consensus algorithm for public blockchains is the proof of work (PoW) [59] where computational power is required to resolve a complex cryptographic puzzle. Another well-known algorithm is the so-called proof of stake (PoS) [60], where a set of participants have to share a minimum amount of *staking* (i.e., some amount of cryptocurrencies). Each consensus protocol is profoundly different from the others, and the ways in which it meets the properties of *agreement, validity* and *termination* are quite different. Just to give an example, the agreement and validity in BFT are guaranteed by assuming a maximum number of tolerated Byzantine nodes in the network equal to 1/3 of the total nodes. In PoW, they are guaranteed, assuming that the number of honest nodes is greater than 50% of the total nodes. The termination property is also very different, and it drastically affects the time needed to produce the correct value or decision. For example, in the IBFT protocol, termination is guaranteed by the pre-established time for each round available to the network nodes, while in PoW, it requires finding the solution to a cryptographic puzzle. The difference between the consensus algorithms of the various blockchains constitutes another barrier in the development of interoperability protocols.

3.3. Formalisation of the blockchain

To interact with the blockchain, each participant must be registered in the blockchain and be authorised to operate on it. We define a participant as follows:

Definition 4 *(Participant).* A participant is a tuple $p := < a, v_a, pk_a, sk_a >$, where:

a : is the identifier of a participant, which we call "address";
v_a : it is the value that can be spent by the participant;
pk_a : is the participant's public key;
sk_a : is the participant's private key.

The private key sk_a is used by the participant identified by a to conduct some operations. Each pair (pk_a, sk_a) is generated with a precise cryptographic scheme Σ.

Modern blockchains also provide the possibility of developing what are called *smart contracts*, i.e., digital contracts that can be executed within the blockchain. We formalise smart contracts as follows:

Definition 5 *(Smart contract).* A smart contract is a tuple $C := <\mathcal{F}, \mathcal{E}, h, q, v>$ where:

$\mathcal{F}$: is a set of functions, can be empty;
$\mathcal{E}$: is a set of events, or log system, generated after the execution of the functions. It can be empty;
h : is a unique identifier (i.e., a hash);
q : is the state, it can be empty;
v : is a value that models the currency assigned to C for its execution.

The set of all smart contracts that can be represented on a blockchain is generated by a set of primitives Γ_C.

Let $\mathcal{F} = \{f_1, f_2, \dots, f_n\}$ be a set of functions within a smart contract C. A function $f_i \in \mathcal{F}$ is *independent* if it operates autonomously and does not rely on the execution, state or output of any other functions or contracts. Its operations depend on its internal logic, its storage and direct inputs. We denote the set of independent functions as $\mathcal{F}_{ind}$ and the set of dependent functions as $\mathcal{F}_{dep} = \mathcal{F} \backslash \mathcal{F}_{ind}$.

Each blockchain has a *global state* that tracks the presence of participants and smart contracts within the blockchain itself. The global state of a blockchain is mutable over time. We define the global state of a blockchain as follows:

Definition 6 *(Global state).* The global state Ω is a tuple ($\mathcal{P}$,C) where $\mathcal{P}$ is the set of all participants and C is the set of all smart contracts.

In the following, we define the concept of operations. These can change the global state of the blockchain.

We first define the operation that moves currency from a participant to another one.

Definition 7 *(P2P operation).* Let $p_1 := < a_1, v_{a_1}, pk_{a_1}, sk_{a_1} >$ and $p_2 := < a_2, v_{a_2}, pk_{a_2}, sk_{a_2} >$ be two participants in $\mathcal{P}$. A P2P operation is a function $P2P(p_1, p_2, v)$ that transfers the value v (with $v \leq v_{a_1}$) from p_1 to p_2. After the application of $P2P(p_1, p_2, v)$, v_{a_1} and v_{a_2} take the values $v_{a_1} - v$ and $v_{a_2} + v$.

In the following, we refer to p_1 and p_2 as the sender and the recipient of the operation. It is worth mentioning that a $P2P$ operation can be generalised to allow many recipients, that is, the operation $P2P(p_1, p_2, v_2, p_3, v_3, \dots, p_t, v_t)$ can be defined. This multiparty operation would transfer from v_{a_1} the values v_2, … v_t to v_{a_2}, …, v_{a_t}, respectively. For the sake of simplicity, we avoid using these multiparty operations by considering that, without loss of generality, a multiparty operation can be modelled as a sequence of $P2P$ one-to-one operations. In the following, we define the operation that models a participant that executes a smart contract function.

Definition 8 *(P2C operation).* Let $p := < a, v_a, pk_a, sk_a >$ and $C :=< \mathcal{F}, \mathcal{E}, h_c, q_c, v_c >$ be a participant and a smart contract, respectively. A P2C operation is a function $P2C(p, C, f, v)$ that models the execution of the function $f \in \mathcal{F}$ where v is a value that p assigns to v_c for the execution of f. After the operation, the value v_a and v_c are updated to $v_a - v$ and 0, respectively, whereas q_c is updated to q'_c. As a consequence of the function execution the set $\mathcal{E}$ could be updated in $\mathcal{E}'$.

In the following, we refer to p_1 and C as the sender and the recipient of the operation. A smart contract can call another smart contract. This is modelled by the following operation:

Definition 9 *(C2C operation).* Let $C_1 :=< \mathcal{F}_1, \mathcal{E}_1, h_1, q_1, v_1 >$ and $C_2 :=< \mathcal{F}_2, \mathcal{E}_2, h_2, q_2, v_2 >$ be two smart contracts. A C2C operation is a function $C2C(C_1, C_2, f, v)$ that models the execution of the function $f \in \mathcal{F}_2$ where v is the value that C_1 spends for the execution of f. After the operation, the value v_1 is updated to $v_1 - v$, q_1 is updated to q'_1, and q_2 is updated to q'_2. As a consequence of the function execution the sets $\mathcal{E}_1$ and $\mathcal{E}_2$ could be updated in $\mathcal{E}'_1$ and $\mathcal{E}'_2$.

It is worth mentioning that a C2C operation can be generalised to allow a smart contract to call many smart contracts. As we already discussed for the P2P operations we avoid using these multiparty operations by considering that, without loss of generality, a multiparty operation can be modelled as a sequence of C2C operations. We can finally define a smart contract to participant operations:

Definition 10 *(C2P operation).* Let $p := < a, v_a, pk_a, sk_a >$ and $C :=< \mathcal{F}, \mathcal{E}, h_c, q_c, v_c >$ be a participant and a smart contract, respectively. A function $C2P(p, C, f, v)$ models the transfer of a value v from C to p. After the operation, the value v_c and v_a are updated to $v_c - v$ and $v_a + v$, respectively, while q is updated to q'.

For the sake of simplicity, in the following, we can use the notation $o_{i,j}$ to denote an operation from a participant p_i (or smart contract

C_i) to a participant p_j (or a smart contract C_j). We can now define a transaction as a sequence of operations.

Definition 11 *(Transaction).* A transaction tx performed by a participant p is a sequence of operations $o_{p,1}, o_{2,3} \ldots, o_{i,i+1} \ldots, o_{t,t+1}$ (with $t > 0$) where:

(Starting call) $o_{p,1}$ is equal to $P2C(p, C_1, f, v)$ or $P2P(p, p_1, v)$;
(Cascade call) $o_{i,i+1}$ can be equal to $P2P(p_i, p_{i+1}, v)$; or equal to $C2C(C_i, C_{i+1}, f, v)$ or $C2P(C_i, p_{i+1}, f, v)$ with C_i being the recipient of a previous operation $o_{j,i}$ with $j < i$.

This definition implies that a transaction is always initiated by a participant (starting call) and can generate various operations (cascade call). Once a transaction is invoked, it is maintained in a waiting pool which we denote by $\mathcal{W}$. Transactions in the waiting list are pending and have to be validated. Transactions that are already validated are part of the ledger $\mathcal{L}$. This is a sequence of blocks $l_1, \ldots, l_t$ and each block l_i is a tuple $< Tx_i, h(l_{i-1}), \mathcal{I} >$ where Tx_i is a transaction set, $h(l_{i-1})$ is the hash of the previous block, and $\mathcal{I}$ represents any other information specific of the considered blockchain. For the sake of readability, we ignore $\mathcal{I}$ and we abuse the notation $\mathcal{W} \bigcup \mathcal{L}$ to indicate the set $\mathcal{W} \bigcup_{i=1}^{t} Tx_i$. As a consequence, $\mathcal{W} \bigcap \mathcal{L} = \mathcal{W} \bigcap_{i=1}^{t} Tx_i = \emptyset$, that means a transaction cannot stay in the waiting pool and in the ledger at the same time. We also define the operation $\mathcal{L}||l$ that appends a block l to a ledger $\mathcal{L} = l_1, \ldots, l_t$ that is $\mathcal{L}||l = l_1, \ldots, l_t, l$.

In different blockchains, the definitions given above can be specialised in different ways. For example, in the Ethereum [2] and Algorand [3] blockchains, which are account-based blockchains, each participant has an account associated with a balance that can change through the transactions. So, the global state of an account-based blockchain reflects the balance of an account at a certain time and determines the amount that can be spent at that time. In the unspent transaction output-based blockchains (UTXO), such as Cardano [61], there is no concept of account and the global state reflects UTXO ownership in a certain moment and who can spend a certain UTXO at that moment [62].

Transactions will remain in the waiting pool until they are verified and collected into a new block to be added to the ledger of validated transactions $\mathcal{L}$. The verification and validation process for transactions is done by a *consensus algorithm*. This is implemented by a set of distributed processes that have the task of validating transactions and creating blocks. Transactions are taken from the waiting pool, collected into a block that is appended to the blockchain. We formalise the *consensus algorithm* as follows:

Definition 12 *(Consensus algorithm).* Let $\mathcal{W}$, $\mathcal{L}$ and Ω be the waiting pool, the ledger $l_1, \ldots, l_t$ and the global state, respectively. A consensus algorithm $\mathcal{A}$ is a function $\mathcal{A}(\mathcal{W}, \mathcal{L}, \Omega)$ that extracts a set of transactions $Tx = \{tx_1, \ldots, tx_n\}$ from $\mathcal{W}$ (i.e., $\mathcal{W}' = \mathcal{W} \setminus Tx$), adds Tx in $\mathcal{L}$ (i.e., $\mathcal{L}' = \mathcal{L}|| < Tx, h(l_t) >$)), and updates Ω into Ω' with the application of all the transactions in Tx.

The last block appended to the blockchain determines the height of the blockchain (i.e., $t+1$ with respect to the above definition). The consensus algorithm is responsible for moving the global state Ω to a new global state Ω'. All the changes executed by the validated transactions must be consistent with respect to the global state Ω, and Ω' is obtained by sequencing the execution of transactions Tx. The process of moving transactions from $\mathcal{W}$ to $\mathcal{L}$ takes a finite number of steps λ. In our definition of a consensus algorithm, a unique block l is created and immediately added into the ledger $\mathcal{L}$ (i.e., the block selected is final). In practice, not all consensus algorithms have this immediate finality, that is, a consensus can go into states where various valid blocks (e.g., l_1 and l_2) are produced and added into the ledger $\mathcal{L}$. This will produce two valid ledger $\mathcal{L}_1 = \mathcal{L}||l_1$ and $\mathcal{L}_2 = \mathcal{L}||l_2$. For example, it is known that Proof of Work consensus algorithms do not satisfy immediate finality, as they are non-deterministic consensus protocols. In non-deterministic consensus protocols, it may happen that, during the validation of transactions, two valid blocks are produced that contain different transactions. This phenomenon is called *fork*. The fork problem, which arises from the probabilistic finality of blockchains like Bitcoin and Ethereum, significantly increases the likelihood of a fork during inter-chain transactions, as differences in block confirmation times and potential chain reorganizations can lead to temporary or conflicting states across networks, violating the durability property of distributed transactions. It is essential to account for the confirmation time on each chain to ensure the probability of a fork becomes negligible, as waiting for multiple confirmations reduces the risk of a fork.

In general, the presence of forks is not the only aspect that could influence interoperability between ledgers, but attention must also be paid to the transaction selection priorities by the consensus protocol. How long a transaction will remain in the waiting pool is an important factor for building inter-chain protocols. In general, consensus algorithms' transaction selection rules can vary, and different strategies can be employed. However, we have divided the transaction selection strategies of consensus algorithms into three general categories: (*i*) transaction and waiting pool dependent; (*ii*) transaction dependent; (*iii*) waiting pool dependant. Based on these three categories, we will try to associate each transaction with a certain *priority*. The priority of a transaction determines its right of preemption over other transactions during its selection by the consensus algorithm.

Definition 13 *(Transaction priority).* Let $tx \in \mathcal{W}$ be a transaction in the waiting pool and $\mathcal{A}$ be a consensus algorithm. A priority p is a function that can be defined in one of the following ways:

(i) Dependent on tx and $\mathcal{W}$:

$$p(\mathcal{A}, tx, \mathcal{W}) = r;$$

(ii) Dependent only on tx:

$$p(\mathcal{A}, tx) = r;$$

(iii) Dependent only on $\mathcal{W}$:

$$p(\mathcal{A}, \mathcal{W}) = r,$$

where $r \in \mathbb{R}^+$ determines the transaction priority.

Typically, in the model expressed by Definition 13. (i) predicting the validation time for transactions in the waiting pool is a challenge. This difficulty arises because, while the priority of transactions can be managed, the selection process also depends on the current state of the waiting pool. Usually, proof of work consensus algorithms fall under this selection strategy, such as Bitcoin [1], where transactions are prioritised based on the age of the UTXO that is being spent in their inputs. Some PoS consensus algorithms fall under Definition 13. (ii) such as Ethereum [2], where the selection strategy depends exclusively on the priority fees associated with a transaction. The PoA consensus algorithms generally fall under Definition 13. (iii) such as IBFT 2.0 [57], where the transactions are selected using a deterministic rule.

According to the previous definitions, we give a formalisation of the blockchain:

Definition 14 *(Blockchain).* A Blockchain is a tuple $\mathcal{B} := < \Sigma, h, \Gamma_C, \mathcal{W}, \mathcal{L}, \mathcal{A}, \Omega >$ where:

Σ : is a cryptographic scheme responsible for generating all key pairs (pk_a, sk_a) associated with the addresses of the participants $\mathcal{P}$;
h : is a hash function;

Γ_C : is a finite set of primitives that allow the generation of all smart contracts C that can be represented in the blockchain;
$\mathcal{W}$: is a set of pending transactions;
$\mathcal{L}$: is a set of validated transactions. The validated transactions are maintained as an append-only linked list of signed blocks;
$\mathcal{A}$: is the consensus algorithm;
Ω : is the global state of the blockchain.

Each blockchain has its own set of pending transactions $\mathcal{W}$ and validated transactions in $\mathcal{L}$ that are governed by a specific consensus algorithm $\mathcal{A}$; has a specific cryptographic scheme Σ responsible for generating all the valid public and private keys used by participants $\mathcal{P}$, a specific hash function h is responsible for guaranteeing the integrity of the data and a precise set of primitives Γ_C responsible for representing all smart contracts C. It also has its own global state Ω that determines the operations that can be carried out by individual participants and the smart contracts present. These characteristics make blockchains isolated, self-sufficient and self-secure environments. The research for interoperability protocols is very important to extend the functionality of blockchains and increase their applicability.

3.4. Formalisation of inter-chain transaction

In this section, we formalise an inter-chain transaction following the formalism we are adopting. An inter-chain transaction should respect the following:

Definition 15 *(Inter-chain transaction (v.1))*. Let $\mathcal{B}_A = <\Sigma_A, h_A, \Gamma_{C_A}, \mathcal{W}_A, \mathcal{L}_A, \mathcal{A}_A, \Omega_A>$ and $\mathcal{B}_B = <\Sigma_B, h_B, \Gamma_{C_B}, \mathcal{W}_B, \mathcal{L}_B, \mathcal{A}_B, \Omega_B>$ be two different blockchains. An inter-chain transaction $tx_{A\to B}$ is a pair (tx_A, tx_B) where tx_A is a transaction that has been validated in $\mathcal{L}_A$ and tx_B is a transaction that has been validated in $\mathcal{L}_B$.

If we consider the definitions listed so far, it remains difficult for us to believe that the blockchains defined in this way can interoperate with each other. In fact, we note that from Definition 14, each blockchain among $\mathcal{B}_A$ and $\mathcal{B}_B$ has its own set of transactions $\mathcal{W}_A \cup \mathcal{L}_A$ and $\mathcal{W}_B \cup \mathcal{L}_B$ which are managed through the consensus algorithms $\mathcal{A}_A$ and $\mathcal{A}_B$, respectively, as expressed by Definition 12. From Definition 15, then, there must exist a transaction $tx_A \in \mathcal{W}_A$ and there must exist a transaction $tx_B \in \mathcal{W}_B$. According to our Definition 14, since blockchains work on disjoint sets of transactions and adopt different consensus algorithms, creating an inter-chain transaction like the one expressed by Definition 15 is infeasible without enhancing the blockchain $\mathcal{B}_A$ with the ability to at least read the transactions from the blockchain $\mathcal{B}_B$. It is trivial to note that $\mathcal{B}_B$ has no way to verify the global state of $\mathcal{B}_A$ and therefore has no way of verifying the existence of the transaction tx_A. This is in line with the result obtained by the authors in Ref. [26].

Therefore, to guarantee interoperability as asserted in Definition 15, it is necessary to extend the definition of the blockchains and ensure that they can verify the presence of events that occur in the other blockchains.

3.5. Formalisation of off-chain Listener

To enable seamless interoperability between different blockchains, aiming to cover general-purpose applications, it is desired to introduce a trusted third party in charge of conducting inter-chain transactions.

Although there are numerous works in the literature that model the trusted third party on which to build protocols to enable communication between different blockchains, here we want to propose our idea of a trusted third party, formalising its functionalities and properties. The peculiarity we want to introduce is the absence of direct communication between the off-chain parties, with the aim of maximising the privacy of the blockchains involved. We call *Listener* the third off-chain part. This is formalised as follows:

Definition 16 *(Listener)*. Let (tx_A, tx_B) be an interchain transaction between two blockchains $\mathcal{B}_A$ and $\mathcal{B}_B$. A *listener* Λ_X tied to $\mathcal{B}_X$, $X \in \{A, B\}$, is a tuple $< P_X, \Sigma^{(t,n)}, Sender_X, Receiver_X >$, where P_X is a set of processes, $\Sigma^{(t,n)}$ is a (t,n)-threshold signature associated with P_X, whereas $Sender_X$ and $Receiver_X$ are two smart contracts in Γ_{C_X}, such that $Sender_A$ and $Receiver_B$ contain the transactions tx_A and tx_B, respectively.

The Listener Λ_A, tied to a blockchain $\mathcal{B}_A$, is not a participant on the target blockchain $\mathcal{B}_B$, has no knowledge about the entire ledger of validated transactions $\mathcal{L}_B$, but is able to verify a particular subset of transactions $Tx_B^{\Lambda_A} \in \mathcal{L}_B$ that are directed to the $Sender_B$ and $Receiver_B$ smart contracts (and vice versa for Λ_B through the blockchain $\mathcal{B}_A$). The subset of transactions $Tx_B^{\Lambda_A}$ (resp. $Tx_A^{\Lambda_B}$) must satisfy the following properties:

i. All transactions in $Tx_B^{\Lambda_A}$ must be directed to $Sender_B$ and $Receiver_B$;
ii. All $Sender_B$ and $Receiver_B$ smart contract executions must emit particular events or a log system $\mathcal{E}_{\Lambda_B}$ that can be read by Λ_A.

A participant p of $\mathcal{B}_A$ submits a transaction tx_A in the smart contract $Sender_A$. The listener Λ_B has read access to the event of $Sender_A$ and can complete the interchain transaction by submitting $\sigma(tx_B)$ on the smart contract $Receiver_B$. We recall that $\sigma(tx_B)$ is the transaction tx_B signed by t processes in P_B. When tx_B is successfully completed, no further steps are required. Depending on the use case, there may be a need to roll back to the source blockchain in case the inter-chain transaction fails. In such cases, the listener Λ_A rolls back tx_A, invoking a transaction tx_A^{-1}, when tx_B fails.

Therefore, listeners allow to partially read the contents of other blockchains. Now, we can finally extend the definition of blockchain by introducing a listener as part of the system:

Definition 17 *(Extended blockchain)*. An extended blockchain is a blockchain (cf. Definition 14) with a further parameter Λ being a Listener.

From now ongoing, we refer to a blockchain as an extended blockchain. The definition of an extended blockchain is slightly different than the definition of a basic one, as the Listener has the possibility of influencing the evolution of the blockchain to which it is linked by considering the events that happen in other blockchains.

Having changed the definition of blockchain, it is also necessary to change the definition of inter-chain transactions, as the blockchains now have their respective Listeners with which they are able to understand what has happened on the blockchains with which they want to communicate.

Definition 18 *(Inter-chain transaction (v.2))*. Given two extended blockchains $\mathcal{B}_A = <\Sigma_A, H_A, \Gamma_{C_A}, \mathcal{W}_A, \mathcal{L}_A, \mathcal{A}_A, \Omega_A, \Lambda_A>$ and $\mathcal{B}_B = <\Sigma_B, H_B, \Gamma_{C_B}, \mathcal{W}_B, \mathcal{L}_B, \mathcal{A}_B, \Omega_B, \Lambda_B>$, the Listener Λ_A can read a subset of transactions $Tx_B^{\Lambda_A} \in \mathcal{L}_B$ and the Listener Λ_B can read a subset of transactions $Tx_A^{\Lambda_B} \in \mathcal{L}_A$. An inter-chain transaction $tx_{A\to B}$ is a pair (tx_A, tx_B) such that the following property holds:

$$\exists tx_A \in \mathcal{L}_A \Rightarrow \exists tx_B \in \mathcal{L}_B \mid tx_A \in Tx_A^{\Lambda_B} \wedge tx_B \in Tx_B^{\Lambda_A}.$$

Theorem 1. *Let $\mathcal{B}_A$ and $\mathcal{B}_B$ be two extended blockchains. An inter-chain transaction $tx_{A\to B}$ is possible.*

Proof. From Definition 17, each blockchain among $\mathcal{B}_A$ and $\mathcal{B}_B$ has its own set of transactions $\mathcal{W}_A \cup \mathcal{L}_A$ and $\mathcal{W}_B \cup \mathcal{L}_B$ which are managed through the consensus algorithms $\mathcal{A}_A$ and $\mathcal{A}_B$, respectively, as expressed by Definition 12. Each blockchain $\mathcal{B}_A$ and $\mathcal{B}_B$ has its own

Listener Λ_A and Λ_B. Λ_A can read a subset of transactions validated on $\mathcal{B}_B$ (this is denoted with $Tx_B^{\Lambda_A}$). Λ_B can read a subset of transactions validated on $\mathcal{B}_A$ (this is denoted with $Tx_A^{\Lambda_B}$). If the blockchain $\mathcal{B}_A$ initiates an inter-chain transaction $tx_{A\to B}$ then the consensus algorithm $\mathcal{A}_A$ selects a subset $Tx_A \subseteq \mathcal{W}_A$ that contains a $tx_A \in Tx_A^{\Lambda_B}$, creates a block $l_i =< Tx_A, h(l_{i-1}) >$ and updates its ledger with the new block, i.e., $\mathcal{L}_A = \mathcal{L}_A || l_i$. It means that the ledger $\mathcal{L}_A$ contains the transaction tx_A. By assumption, the transaction tx_A belongs to the subset of transactions that can be read by the Listener Λ_B, i.e., $tx_A \in Tx_A^{\Lambda_B}$. Transactions in $Tx_A^{\Lambda_B}$ are directed towards $Sender_A$ and their functions $\mathcal{F}_{Sender_A}$ emit specific events $\mathcal{E}_{Sender_A}$ (see Definition 16). The events $\mathcal{E}_{Sender_A}$ can guide the Listener Λ_B to perform the corresponding transaction tx_B on the destination blockchain $\mathcal{B}_B$. Thanks to Listener Λ_B, the blockchain $\mathcal{B}_B$ is aware that an inter-chain transaction is started and, using the events in $\mathcal{E}_{Sender_A}$, can prepare a transaction $tx_B \in \mathcal{W}_B$. The consensus algorithm $\mathcal{A}_B$ selects a subset $Tx_B \subseteq \mathcal{W}_B$ that contains $tx_B \in Tx_A^{\Lambda_B}$, creates a block $l_i =< Tx_B, h(l_{i-1}) >$ and updates its ledger with the new block, i.e., $\mathcal{L}_B = \mathcal{L}_B || l_i$. Again, by Definition 18, the transaction tx_B belongs to the subset of the transactions that can be read by the Listener Λ_A, i.e., $tx_B \in Tx_B^{\Lambda_A}$. From Definition 16, transactions in $Tx_B^{\Lambda_A}$ are directed towards $Receiver_B$ and their functions $\mathcal{F}_{Receiver_B}$ emit precise events $\mathcal{E}_{Receiver_B}$. The events $\mathcal{E}_{Receiver_B}$ can be read by the Listener Λ_A, which can verify the effective completion of the inter-chain transaction. A similar argument can be provided for the interchain transaction $tx_{B\to A}$.

Consensus algorithms play a crucial role in ensuring the integrity and consistency of the underlying ledger. These algorithms can be broadly categorised into those that provide deterministic finality and those that provide probabilistic finality. The differences between these types of consensus mechanisms affect the feasibility and reliability of cross-chain transactions. Deterministic finality means that once a transaction is confirmed, it is final and cannot be reversed. In consensus algorithms with deterministic finality, such as IBFT [57], most widely adopted in private or permissioned blockchains, or Tendermint adopted by Cosmos [18], transactions are final as soon as they are confirmed. Probabilistic finality means that the finality of a transaction increases over time as more blocks are added to the blockchain. In other words, the likelihood of a transaction being reversed decreases as more subsequent blocks are confirmed. In consensus algorithms with probabilistic finality, such as proof of work adopted by Bitcoin [1], or proof of stake adopted by Ethereum [2], the probability of a transaction being removed from the ledger decreases as new blocks are validated.

Building interoperability protocols between blockchains implementing probabilistic consensus algorithms is more challenging, as care must be taken not to violate the durability property of distributed transactions.

Using our blockchain formalisation, we will define a generic inter-chain communication protocol that finalises an inter-chain transaction using only two transactions, one on the source blockchain and one on the target blockchain.

4. Inter-chain properties and components

Here, we want to present the ideas on which our interoperability protocol is inspired. The protocol is primarily designed for communication between private blockchains. Its purpose is to maintain security and privacy properties while making communication as efficient as possible. The interoperability solution we propose is at the application level (i.e., via smart contracts). The protocol we want to propose is based on a hybrid approach that wants to combine some properties of watchtowers [63–65] and oracles [20–22]. Our protocol allows the implementation of correct and secure inter-chain transactions. The peculiarity we introduce is the fact that we propose dedicated off-chain components for each blockchain and no direct communication takes place between these off-chain parties. Moreover, no dispute can occur during the off-chain computations. The reasons why we avoided direct communications between off-chain parties are multiple. First, it improves the security of the system, as all the processes of the off-chain component have access to the same information that is the output of another blockchain. Second, it improves efficiency, as it allows avoiding consensus protocols in the off-chain process, which is necessary when direct communications are used to avoid possible fraudulent behaviour in the processes of the communicating listeners. Finally, it also reduces latency, as the off-chain components can access the data as soon as it is available, avoiding any communications. Summarising, our protocol requires that each blockchain implements its own off-chain component that can write to it and can only read the inter-chain object portion of data from the communicating blockchains.

Watchtowers are basically third-party services introduced to monitor activities in off-chain payment channels, such as the Lightning Network [66], in order to mitigate disputes arising due to a party involved in the transaction going offline. In our approach, the watchtower does not monitor off-chain activities, but monitors on-chain activities. The idea is to use the watchtower to monitor particular events or log systems related to particular smart contracts deployed on both a source and a target blockchain.

Oracles are services initially designed to bring data from the outside world into the blockchain via smart contracts. Then they also started to be used for the construction of interoperability protocols between blockchains. However, oracles, in their basic definition, require read and write access to the blockchains being communicated. This is an inconvenience if we are in the context of private blockchains, where data would not want to be disclosed to third parties. Furthermore, having read and write access to both blockchains could open up the possibility of forcing the execution of an inter-chain transaction that benefits one of the two blockchains. Inspired by some properties of watchtowers and oracles, we propose a listener-based approach that will be presented in detail in the next subsection.

4.1. Listeners

Listeners are a distributed off-chain network of nodes tied to each blockchain, as expressed by Definition 16. The choice to make the Listener distributed (or at least decentralised) was made to avoid centralising communications, undermining the distributed or decentralised nature of the blockchain. These nodes can be the blockchain's validator nodes, or they can be on independent machines. To manage transactions, Listeners use a (t,n)-threshold signature scheme. Each node holds a signing key with which it produces its partial signature. Once at least t partial signatures have been generated, they are combined to generate the threshold signature. The threshold signature scheme aims at increasing privacy between participants in the signing process while ensuring that only authorised entities can send transactions. In fact, to verify the threshold signature, it is necessary to know only the public key, which keeps secret both the number of nodes making up the Listener and the minimum threshold required for the validity of the signature. Some advances in the research of threshold signature-based cryptographic schemes have made them efficient and versatile. In recent years, research into threshold signature schemes has been very lively, resulting in numerous variations being proposed. The most recent works are aimed at proposing threshold variants of schemes based on elliptic curve [67,68], or schemes based on Boneh–Lynn–Shacham (BLS) [69]. Applying an ECDSA-based threshold signature has also been explored by the authors of Ref. [70] to increase security and efficiency in supply chain finance, as well as by the authors of Ref. [71] for the management of inter-chain transactions. The main contributions in the research of threshold signature schemes concern greater efficiency and security in key distributions; improve the dynamism of the schemes, allowing to vary the number of nodes in the system and the minimum threshold

for signature generation. This last point is extremely important as it allows to extend and reduce the number of nodes making up the Listeners without having to reconfigure all keys.

The Listeners of each chain do not communicate directly with each other, but become aware of the events that are present on the target blockchain. To do so, they make use of a technique similar to the best-effort notification [46], i.e., the Listeners can query only particular events emitted by the target blockchains. This is a characteristic for which a Listener acts like a watchtower and differs from the basic oracle properties. Listeners must be able to access the events or log system related to specific smart contracts deployed in the target blockchain and write the data relating to the events on their own blockchain. In this way, a Listener acts like an oracle, making decisions based on the data read from other blockchains.

For the protocol to function properly, there must be at least t honest processes, where t is the threshold required to generate the signature. The minimum number of processes needed to configure the threshold is $n/2+1$, where n is the number of processes in the off-chain component.

Moreover, it is essential to establish a time limit on the validity of events. This enables Listeners to determine if the inter-chain transaction time has expired, necessitating a rollback of any state changes. To this end, there must exist a time interval Δ_A monitored by the Listener Λ_A and a time interval Δ_B monitored by the Listener Λ_B. In general, if we want to move a transaction from a blockchain $\mathcal{B}_A$ to a blockchain $\mathcal{B}_B$, we must also consider the block generation time of the destination chain, as it could affect the necessary validity time of the transaction. Suppose that the maximum block generation time on blockchain $\mathcal{B}_B$ is λ_B, then the following constraint must exist:

$$\Delta_A > \Delta_B + c\lambda_B \tag{1}$$

where:

- Δ_A : is the time that the Listener Λ_A of the source blockchain $\mathcal{B}_A$ is willing to wait for the generation of the event in the destination blockchain that completes the inter-chain transaction. If such an event is issued before the time Δ_A expires when the transaction is completed, otherwise it is rolled back;
- Δ_B : is the time that the Listener Λ_B of the target blockchain $\mathcal{B}_B$ must wait to verify that the transaction it sent to its blockchain is validated;
- $c\lambda_B$: is an extra time in which the Listener Λ_B of the target blockchain $\mathcal{B}_B$ is willing to wait before deleting the inter-chain transaction. In particular, c is a multiplication factor and λ_B is the maximum time needed to generate a new block on $\mathcal{B}_B$. If the transaction is not validated within $\Delta_B + c\lambda_B$ time, the transaction will be cancelled.

The constraint expressed by Eq. (1) guarantees the correctness in case of malfunctions during inter-chain transactions, which will be explained in detail in Section 5. The multiplication factor $c \geq 1$ must be carefully selected, as it plays a crucial role in the protocol. Further details on this aspect will be provided in Section 7.

The nodes composing the Listener work in rounds, i.e., each node is selected as leader in each turn and has an operability time of δ. Once δ has expired another leader will be selected.

4.2. Smart contracts: sender & receiver

Interoperability is exclusively based on smart contracts. The smart contracts must be capable of emitting events or log systems that are read by the Listeners to carry out some operations. The semantics of interoperability [27] must be defined in events or log systems related to these smart contracts. Our work has been inspired by the restful pattern [72,73] in which the events (or log system) are in well-structured formats and can be interpreted by the respective Listeners to correctly prepare the transactions to be submitted to their own blockchains. An example of the semantics that could be used within the protocol can be found in Ref. [74] where the authors show a feasible interoperability protocol between EVM-based blockchains.

The main features of these smart contracts are:

Sender: is the smart contract responsible for managing the initiation of inter-chain transactions on the source blockchain. The initiation of an inter-chain transaction occurs by invoking the functions present on this smart contract, which, once executed, emits the events that initiate the inter-chain transactions. The main tasks of this smart contract are: (i) performing the transaction on the source blockchain; (ii) emitting events that will enable the Listener on the destination blockchain to build the transaction on the destination blockchain; (iii) check whether or not the transaction was successful at the destination blockchain; (iv) roll back the transaction at the source blockchain when the destination transaction failed; (v) other functions could be provided that have the purpose of sending internal transactions directed to other smart contracts and/or other addresses;

Receiver: The receiver smart contract has the following tasks: (i) complete the inter-chain transaction on the destination blockchain. (this requires the Listener leader to invoke the function that finalises an inter-chain transaction); (ii) verify the threshold signature of the Listener; (iii) emit events that finalises an inter-chain transaction (this can be successful or to be aborted); (iv) possibly other functions could be provided that have the purpose of sending internal transactions directed to other smart contracts and/or other addresses.

The main subject for operating inter-chain transactions lies in the events $\mathcal{E}$ generated by smart contracts *Sender* and *Receiver*. The main events are as follows:

Inter-chain transaction request. This event is emitted by the *Sender* smart contract on a source blockchain $\mathcal{B}_A$ with the intent of initiating an inter-chain transaction. This event is read by the Listener Λ_B of the destination blockchain $\mathcal{B}_B$. The Listener Λ_B will use the data contained in the event to prepare a transaction directed to the *Receiver* smart contract of the destination chain $\mathcal{B}_B$;

Inter-chain transaction completed. This event is raised from the *Receiver* smart contract on the destination blockchain $\mathcal{B}_B$ if the inter-chain transaction is successful. This event is read by the Listener Λ_A of the source blockchain $\mathcal{B}_A$ that will be certain about the inter-chain transaction occurred correctly;

Inter-chain transaction rolled back. This event occurs when the Listener that starts the inter-chain transaction goes in time out. In this case, the Listener is responsible to roll back the state of the source blockchain. This event is present in the *Sender* smart contract and will be emitted after a roll back function is invoked.

Regarding rollbacks, here we assume that the *Sender* smart contract, under Definition 5, has for each function $f_i \in \mathcal{F}_{Sender}$ an inverse function $f_i^{-1} \in \mathcal{F}_{Sender}$ to restore the state of the blockchain. Referring to Definition 18, an inter-chain transaction that fails on the destination blockchain is characterised by a pair (tx_A, tx_A^{-1}). The transaction tx_A is characterised by a starting call of type $P2C(p, Sender, f_i, v)$ and tx_A^{-1} is characterised by a starting call of type $P2C(p, Sender, f_i^{-1}, v)$. To clarify this point, let's consider the use case of transferring fungible or non-fungible tokens [75–77]. The basic operations of the token transfer use case involve the execution of two functions, which are called *mint* and *burn*. The *mint* function is intended to generate a certain amount of tokens, while the *burn* function is intended to burn a certain amount of tokens. In case an inter-chain transaction for token transfer fails, then the pair (tx_A, tx_A^{-1}) is characterised by the operations $P2C(p, Sender, burn, v)$ and $P2C(p, Sender, mint, v)$. This concept can be extended to different use cases, assuming that the *Sender* smart contract, and other

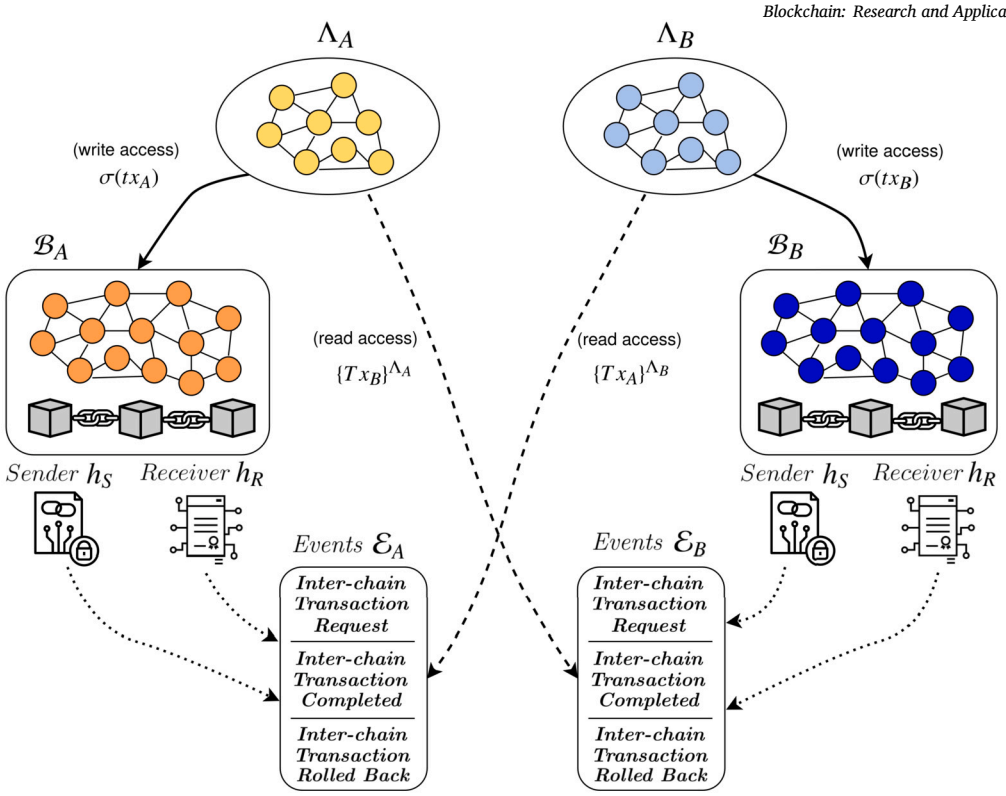

Fig. 1. The figure shows the various parts of the inter-chain protocol and how they are connected. Each blockchain ($\mathcal{B}_A$ and $\mathcal{B}_B$) has an associated Listener (Λ_A and Λ_B) that can send transactions ($\sigma(tx_A)$ and $\sigma(tx_B)$, where σ is a threshold signature) to the blockchain to which it is connected and read the events ($\mathcal{E}_A$ and $\mathcal{E}_B$) emitted by the transactions ($Tx_B^{\Lambda_A}$ and $Tx_A^{\Lambda_B}$) that reside on the smart contracts *Sender* (h_S) and *Receiver* (h_R) of the target blockchains.

smart contracts queried in the cascade call phase, admit inverse functions of the inter-chain object functions.

Fig. 1 shows the components provided by the inter-chain protocol and how they are interconnected with each other.

There must be a temporal relationship between the events *Inter-chain transaction request* and *Inter-chain transaction completed*. The operability interval is defined by Eq. (1).

Our approach is *optimistic* in the sense that changes on the source blockchain after an inter-chain transaction has been initiated are made as if the transaction were successful. This allows to perform an inter-chain transaction with just two transactions. At the same time, the Listener must be allowed to roll back the state of the source chain in case the transaction responsible for finalising the inter-chain transaction in the destination blockchain fails. In the next section, the hypotheses under which the protocol works will be provided, and the functioning of the protocol will be presented in detail.

5. Overview of the Smart listener protocol

This section presents the high-level steps of the protocol. During the discussion, we assume that an inter-chain transaction occurs from a source blockchain $\mathcal{B}_A$ to a destination blockchain $\mathcal{B}_B$. The steps of the protocol are provided in case of correct execution and in case of faults.

5.1. Behaviour in case of correct execution

In case of correct execution, the protocol involves the following steps:

1. **Start of inter-chain transaction on $\mathcal{B}_A$ (Listener Λ_A)**
   (a) A participant p_k that wants to perform an inter-chain transaction $tx^i_{A\to B} = (tx^i_A, tx^i_B)$ must request to the Listener Λ_A a proof that certifies the ability to operate the transaction, where $tx^i_A = P2C(p_k, Sender_A, f_i, v_i)$. The leader of the Listener will provide proof using the threshold signature. The proof is provided only if one of the following conditions holds:
   $c.1$: f_i is an independent function ($f_i \in \mathcal{F}_{ind}$) and there is no other pending inter-chain transaction involving f_i from the same participant p_k;
   $c.2$: f_i is a dependent function ($f_i \in \mathcal{F}_{dep}$) and there are no other pending inter-chain transactions involving f_i from any participants.
   The $c.1$ constraint allows for regulating the activity of individual participants, preventing them from repeatedly opening other inter-chain transactions that insist on the same independent function. This helps to maintain a more balanced load between connected blockchains, especially if they have very different block production time. At the same time, the constraint $c.1$ does not prevent other participants from operating on the function f_i. The $c.2$ constraint allows for keeping the global state of the interconnected blockchains consistent in the case the function f_i is dependent, preventing other participants from interacting with the function f_i if there are pending transactions on it.
   (b) The participant p_k, using the proof of Listener Λ_A, starts an inter-chain transaction through the $Sender_A$ smart contract on the blockchain $\mathcal{B}_A$, adding the transaction tx^i_A to the waiting pool. The starting call of tx^i_A has the form $P2C(p_k, Sender_A, f_i, v_i)$. The $Sender_A$ smart contract on the blockchain $\mathcal{B}_A$ may interact with other smart contracts or other participants on the source blockchain $\mathcal{B}_A$, as expressed by Definition 11;
   (c) The transaction tx^i_A is processed by the consensus algorithm $\mathcal{A}_A$ of the source blockchain, and, if it is valid, it will be added to the validated ledger $\mathcal{L}_A$ within λ_A, contributing to modify the global state Ω_A of the source blockchain $\mathcal{B}_A$, as expressed by Definition 11. If so, there exists an *Inter-chain Transaction*

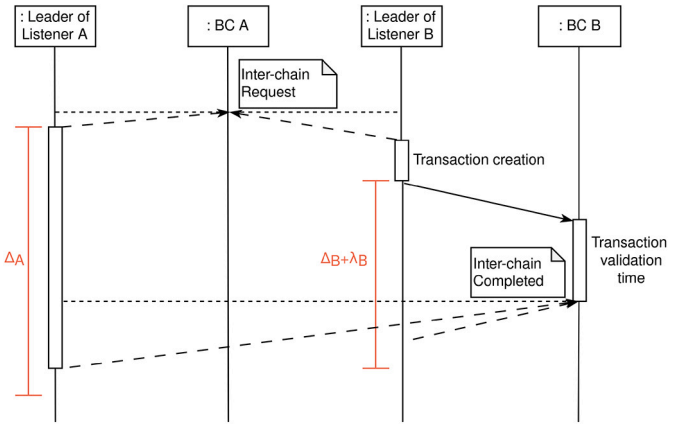

Fig. 2. The figure shows the timeline in case of correct execution of the protocol (Section 5.1).

Request event containing the data needed to the leader of the Listener Λ_B on blockchain $\mathcal{B}_B$, i.e., $tx_A^i \in Tx_A^{\Lambda_B}$;

(d) After the transaction is validated, the leader of the Listener Λ_A starts the counter Δ_A for the inter-chain transaction validity. At this point, the state on the source blockchain doesn't need further updates. After the emission of the event related to the inter-chain transaction we are in the following condition: $\exists tx_A^i \in \mathcal{L}_A \wedge \nexists tx_B^i \in \mathcal{W}_B$.

2. **Completion of an inter-chain transaction on $\mathcal{B}_B$ (Listener Λ_B)**

(a) The leader of the Listener Λ_B on blockchain $\mathcal{B}_B$ reads the event *Inter-chain transaction request* emitted by the blockchain $\mathcal{B}_A$, i.e., becomes aware about $tx_A^i \in Tx_A^{\Lambda_B}$;

(b) The leader of the Listener Λ_B on blockchain $\mathcal{B}_B$ prepares a transaction using the data in the event and signs it with the threshold signature $\sigma(tx_B^i)$, for which the starting call is of the form $P2C(p_{\Lambda_B}, Receiver_B, f_i, v_i)$. Then, the leader of Λ_B sends the transaction to the *Receiver* smart contract on blockchain $\mathcal{B}_B$, adding the transaction tx_B^i to the waiting pool. Then Λ_B starts a counter $\Delta_B + c\lambda_B$ for the inter-chain transaction validity. The leader Listener Λ_B has to verify that the transaction tx_B^i is executed within the interval $\Delta_B + c\lambda_B$.

(c) If the transaction is correct, and if the changes were made after the transaction execution are consistent with the global state Ω_B of the destination blockchain, then the transaction should be validated by the consensus algorithm $\mathcal{A}_B$ within $c\lambda_B$, i.e., $tx_B^i \in \mathcal{L}_B$. If so, there exists an event *Inter-chain transaction completed* needed by the leader of the Listener Λ_A to prevent the roll back on blockchain $\mathcal{B}_A$, i.e., $tx_B^i \in Tx_B^{\Lambda_A}$. After the emission of the event related to the completion of the transaction we are in the condition $\exists tx_A^i \in \mathcal{L}_A \wedge \exists tx_B^i \in \mathcal{L}_B$.

3. **Ack of completion on $\mathcal{B}_A$ (Listener Λ_A)**

(a) The leader of the Listener Λ_A reads the event *Inter-chain transaction completed* from the blockchain $\mathcal{B}_B$ and becomes aware of the completion of the inter-chain transaction, i.e., becomes aware of $tx_B^i \in Tx_B^{\Lambda_A}$. No other operations are needed since the global state Ω_A on blockchain $\mathcal{B}_A$ was already updated.

Fig. 2 shows the timeline diagram of the protocol in case of correct behaviour.

The meaning is as follows: the dotted arrows indicate listeners reading events generated by blockchains. The lines indicate the actions of listeners writing to their own blockchains. The red lines indicate the validity time of inter-chain transactions. The same goes for the next figures.

5.2. Behaviour in case of timeout

In the case of a timeout, the Listener will have to verify that the consistency of the related blockchain is not compromised. The leader of the Listener Λ_A will have to create a transaction that rolls back the changes on the source blockchain. The leader of the Listener Λ_B has to verify that the transaction of the destination blockchain will not be executed.

Given the constraint expressed by Eq. (1), the Listener Λ_B of the destination chain will always time out first. Furthermore, from how the protocol is defined, we can have two scenarios: the first is that the Listener Λ_B times out (Δ_B) without having sent the transaction $tx_B^i \notin \mathcal{W}_B$ (it happens if the step 2. (b) in the previous section takes too long), the second case is that the Listener Λ_B times out, and the transaction remains in the destination blockchain's waiting pool $tx_B^i \in \mathcal{W}_B$.

1. **Timeout of Listener A**: In this case the leader of the Listener Λ_A will create a rollback transaction that brings the blockchain $\mathcal{B}_\mathcal{A}$ to a consistent global state prior to the execution of the inter-chain transaction.
2. **Timeout of Listener B without having sent the transaction**: In this case, blockchain $\mathcal{B}_B$ has not undergone any changes since no transaction has been sent. In this case, the leader of the Listener Λ_B is aware that no event will be generated, and when the Δ_A interval expires, the leader of the Listener Λ_A will carry out a transaction that will roll back any changes on the blockchain $\mathcal{B}_A$. This will bring blockchain $\mathcal{B}_A$ to a consistent global state and blockchain $\mathcal{B}_B$ has not undergone any changes.
3. **Timeout of Listener B with the inter-chain transaction in the waiting pool**: In this case, the situation is slightly delicate, as we have to distinguish between two scenarios: the first is that the transaction is in the waiting pool but will be validated within some next blocks ($c\lambda_B$) and the second is that the transaction is in the waiting pool but will not be validated within some next blocks ($c\lambda_B$).

case 1: validated within some next blocks. If the time Δ_B has expired on Listener Λ_B, then the latter must check whether the transaction is validated within some next blocks. If the transaction is validated within $c\lambda_B$, then the leader of the Listener Λ_B does not have to do anything as the *Receiver* smart contract of blockchain $\mathcal{B}_B$ will emit an *Inter-chain transaction completed* event which will be read by Listener Λ_A. This will complete the inter-chain transaction.

case 2: not validated within some next blocks. If the transaction is not validated within some next blocks $c\lambda_B$, then the leader of the Listener Λ_B must replace the transaction with an empty one before it is executed. This is essential as there is a risk that the transaction will be validated after Listener Λ_A times out and rolls back to blockchain $\mathcal{B}_A$.

This behaviour is not necessary if interconnected blockchains allow for validity times to be associated with pending transactions. For example, Stellar [78] allows specifying the maximum time a transaction can remain in the waiting pool directly within transactions, after which the transaction expires. Ethereum [79], on the other hand, does not have a mechanism to expire transactions, but a similar logic could be implemented via smart contracts.

The timeline diagram of the case where blockchains don't implement an expire time over the transactions is shown in Fig. 3.

5.3. Behaviour in case of transaction failure

In the case of transaction failure, the behaviour of the protocol is quite simple. Since the inter-chain protocol involves exactly one transaction on blockchain $\mathcal{B}_A$ and one transaction on blockchain $\mathcal{B}_B$, we can distinguish two cases:

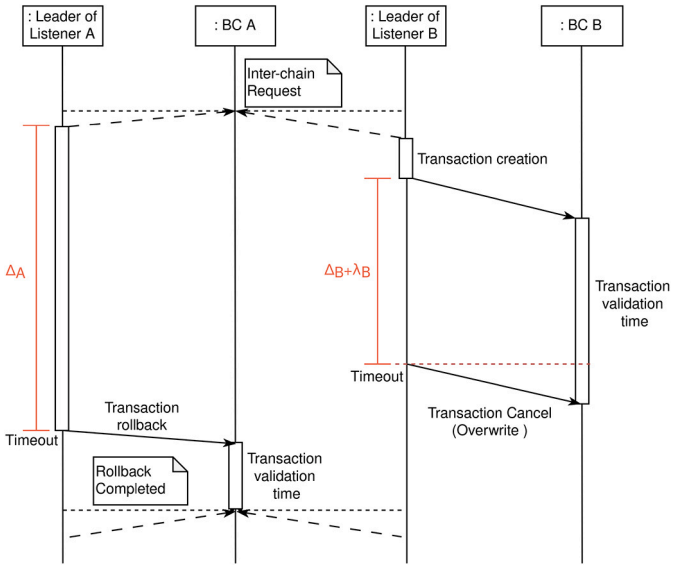

Fig. 3. The figure shows the timeline in the case expressed by the Section 5.2, scenario 3, case 2.

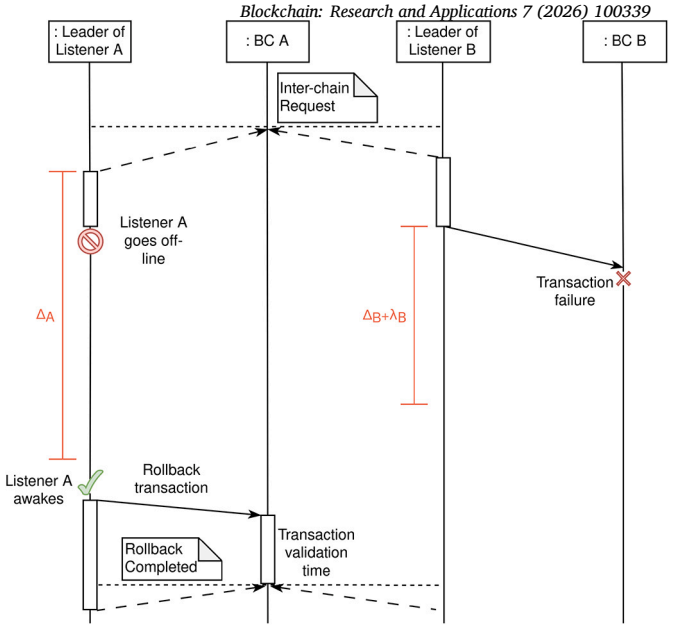

Fig. 4. The figure shows the timeline in the case expressed by the Section 5.4, scenario 3.

Failure on the source blockchain $\mathcal{B}_A$: This case is trivial. If the transaction tx^i_A fails on blockchain $\mathcal{B}_A$, no event is emitted, and the inter-chain transaction will not take place. This means that the transaction $tx^i_A \notin \mathcal{W}_A \cup \mathcal{L}_A$.

Failure on the destination blockchain $\mathcal{B}_B$: If the transaction tx^i_B fails on blockchain $\mathcal{B}_B$, Listener Λ_B will not have to do anything. This means that the transaction $tx^i_B \notin \mathcal{W}_B \cup \mathcal{L}_B$. The Listener Λ_A will not see any *Inter-chain transaction completed* event and at the expiration of Δ_A the Listener Λ_A will roll back on blockchain $\mathcal{B}_A$.

5.4. Behaviour in case of off-line

The behaviour of the protocol must also foresee the different cases in which the Listeners go off-line (i.e., the number of active nodes is less than the threshold t needed to produce the threshold signature). To cover all cases, we must consider the following scenarios:

1. Λ_A **is off-line and no event was emitted**: This case is trivial, no inter-chain transaction can be performed as the Listener Λ_A cannot provide the proof;
2. Λ_A **is off-line and an event *Inter-chain transaction completed* was issued**: In this case the situation is quite trivial. If Λ_A goes off-line and an event *Inter-chain transaction completed* is emitted on the blockchain $\mathcal{B}_B$ it means that the inter-chain transaction was performed correctly, i.e., we meet the condition: $\exists tx^i_A \in \mathcal{L}_A \Rightarrow \exists tx^i_B \in \mathcal{L}_B \mid tx^i_A \in Tx^{\Lambda_B}_A \wedge tx^i_B \in Tx^{\Lambda_A}_B$ expressed by Definition 18. When the Listener Λ_A becomes active, it simply checks for past events and will be aware of the correct completion.
3. Λ_A **is off-line and an event *Inter-chain transaction completed* was not issued**: It means that the inter-chain transaction failed, i.e., we are in the condition where $\exists tx^i_A \in \mathcal{L}_A$, but $\nexists tx^i_B \in \mathcal{L}_B$. Depending on the inter-chain nature, some assets of the participant on the blockchain $\mathcal{B}_A$ are frozen and will remain frozen until the Listener Λ_A awakens and, becoming aware of the failure of the inter-chain transaction, will roll back on the blockchain $\mathcal{B}_A$. This is a bad scenario that could happen, since, depending on the nature of the inter-chain transaction, the asset of some participant could be frozen on the blockchain $\mathcal{B}_A$. (See Fig. 4.)
4. Λ_B **is off-line and an event *Inter-chain transaction request* was issued**: In this case we have a transaction $tx^i_A \in \mathcal{L}_A$. Since the Listener Λ_B is off-line the event *Inter-chain transaction request* cannot be detected. So, after the validity time Δ_A has expired, the Listener Λ_A will perform a rollback on the blockchain $\mathcal{B}_A$;
5. Λ_B **is off-line after reading an event *Inter-chain transaction request***: In this case there are two possible scenarios:
   (a) Λ_B **didn't send** tx^i_B: This case is trivial; Λ_B has nothing to do since the Listener Λ_A will expire and roll back on the source blockchain $\mathcal{B}_A$;
   (b) Λ_B **sent** tx^i_B **before off-line**: This is the worst case in the protocol. Since the number of nodes composing the Listener Λ_B is less than the threshold t, the Listener Λ_B is not able to act on the pending transactions. Therefore, in this case, the protocol always works if the destination blockchain allows associating the validity time with the transactions within the waiting pool. Otherwise, the protocol works only if the selection of transactions made by the consensus algorithm is deterministic or can be prioritised, that is, if we take Definition 13 we have the following cases:
      (*i*) The protocol may not work. In this case, we have no way to influence the priority of the transaction that has to finalise the inter-chain transaction. In practice, we cannot predict how long the transaction will remain in the waiting pool. Therefore, there is a risk that the Listener Λ_A of the source blockchain will rollback before the transaction is validated in the destination blockchain. This is because the Listener Λ_B cannot act as it is unable to generate the threshold signature, so the transaction in the destination blockchain will not be rolled back;
      (*ii*) The protocol works by maximising priority. In fact, even if Listener Λ_B is frozen, we are sure that the transaction will be validated within $c\lambda_B$, i.e., before Listener Λ_A rolls back, and this completes the inter-chain transaction;
      (*iii*) The protocol works by properly setting Δ_A and Δ_B. In this case, the selection of transactions is deterministic. Therefore, it is possible to predict the number of blocks necessary to validate the transaction responsible for finalising the inter-chain transaction. In this case, it must be ensured that the transaction submitted by Listener Λ_B is validated before Listener Λ_A times out.

Fig. 5 shows the worst case expressed by scenario 5.(b), case *i*, for which the protocol may not work.

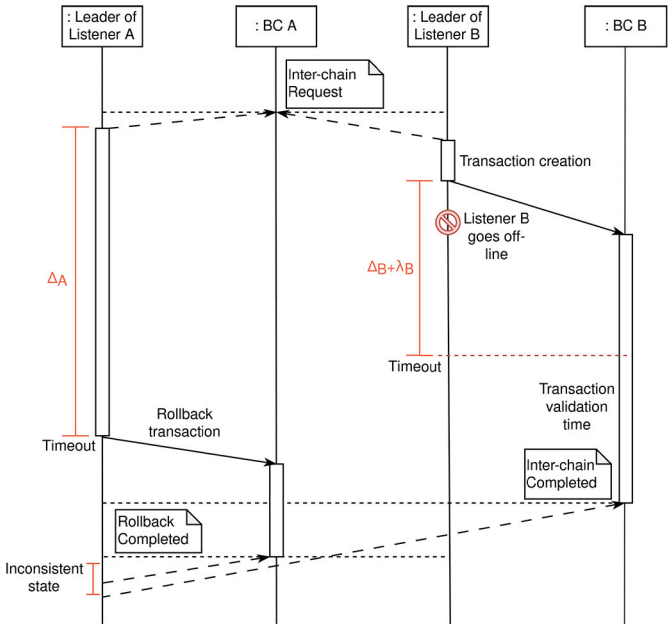

Fig. 5. The figure shows the timeline expressed by the Section 5.4, scenario 5.(b), case (*i*).

5.5. Behaviour in case of Byzantine faults

We need to handle malicious behaviour in case some Listener nodes are compromised. Since the initiation of a transaction occurs only through the presence of events or log systems in the source blockchain, Byzantine behaviours can only occur in the Listener that must finalise the transaction (since we assume that the consensus algorithm of the blockchains involved is not compromised). In general, the protocol should include the following cases:

The leader of the Listener Λ_B attempts to cheat on a transaction: This can happen if the leader of the Listener Λ_B, responsible for finalising an inter-chain transaction, reading an inter-chain request event, tries to alter the content of the data read from the event to generate a transaction that causes an advantage or damage in the destination blockchain. This behaviour is prevented by the presence of the threshold signature. In fact, to send a transaction to the smart contract *Receiver*, the leader of the Listener Λ_B must reach the minimum threshold t required by the signature. Each node composing the Listener Λ_B can check whether the leader's request is correct or fraudulent and decide not to produce the signature. Therefore, correct execution is guaranteed depending on the configuration of the (t, n)-threshold signature. For the system to work, the minimum threshold required for signature generation is at least $n/2 + 1$; otherwise, there is the risk that the Listener Λ_B will be able to correctly sign fraudulent transactions. Other configurations are possible, balancing security and efficiency. For example, the threshold can be unanimous, i.e., $t = n$, which maximises security, but sacrifices fault tolerance. In general, the system works for an arbitrary number of faulty nodes in the interval [0, $n/2$ - 1].

The Listener Λ_B doesn't send the transaction: Means that the leader of the Listener Λ_B doesn't operate correctly. In this case, the next leader has to manage the past events that were not handled. The main problem with this behaviour is that the validity interval Δ_B becomes smaller, and it should be managed adequately. For example, if the window Δ_B becomes too small the leader of the Listener Λ_B may decide not to send the transaction and make the Listener Λ_A expire.

In general, to prevent Byzantine faults, some mechanism of reward/punishment must be implemented and foreseen by the protocol. Here, we want to remain as generic as possible. However, as an example, a countermeasure that could be taken concerns the introduction of a reputation-based mechanism among the nodes composing the Listener [80].

6. Smart listener protocol

This section explains the protocol that Listeners follow. The Listener initialisation procedure and main routine are shown. During the description, we assume that the source blockchain is $\mathcal{B}_A$ and the destination blockchain is $\mathcal{B}_B$. The Listener Λ_A is tied to the source blockchain, and the Listener Λ_B is tied to the destination blockchain.

Procedure 1 shows the initialisation phase of a Listener. Each node has a unique identifier (line 1) and a triple of keys generated by the distributed key generation of the (t, n)-threshold signature (line 2): the private key sk_{id} is owned by the node identified by id, vk and pk are the verification key and the public key, respectively, shared among all nodes of a Listener. The verification key vk will be used by the nodes to verify the partial signature collected to compute the threshold signature. The pair $(h_S,\ h_R)_A$ identifies the smart contracts *Sender* and *Receiver* stored on the source blockchain $\mathcal{B}_A$ (line 3). The pair $(h_S,\ h_R)_B$ identifies the smart contracts *Sender* and *Receiver* stored on the target blockchain $\mathcal{B}_B$ (line 4). Each node composing the Listener holds a list L of all the other nodes in the network (line 5). The initialisation must be done for the Listener Λ_A on the source blockchain and for the Listener Λ_B on the target blockchain. The initialisation phase must be executed until the Listener is composed of at least $n = t + k$ nodes, where t is the minimum number required to compute the threshold signature, while k depends on the configuration of the threshold signature; it can be every integer value in the interval [0, $n/2$-1] (for example possible configurations are unanimity: $k = 0$ and $t = n$; or simple majority: $k = 1/2n$-1 and $t = 1/2n + 1$).

Procedure 1: Initialisation phase.

```
1 id ← Unique identifier of the node in the Listener
2 (sk_id, vk, pk) ← Distributed Key Generation
3 (h_S, h_R)_A ← hashes of Sender and Receiver smart contracts of B_A
4 (h_S, h_R)_B ← hashes of Sender and Receiver smart contracts of B_B
5 L ← list of all nodes composing a Listener
```

Procedure 2 shows the main routine executed by each Listener.

Procedure 2: Main routine.

```
1  l ← Leader Election
2  δ ← Time of current round
3  if l == id then
4      if E_A ≠ ∅ then
5          Recover(E_A)
6      end
7      while δ not expired do
8          Listen()
9      end
10     Broadcast(new Leader Election)
11 else
12     while δ not expired do
13         Wait for ThresholdSignatureRequests
14         if leader goes off-line then
15             Broadcast(new Leader Election)
16         end
17     end
18     Broadcast(new Leader Election)
19 end
```

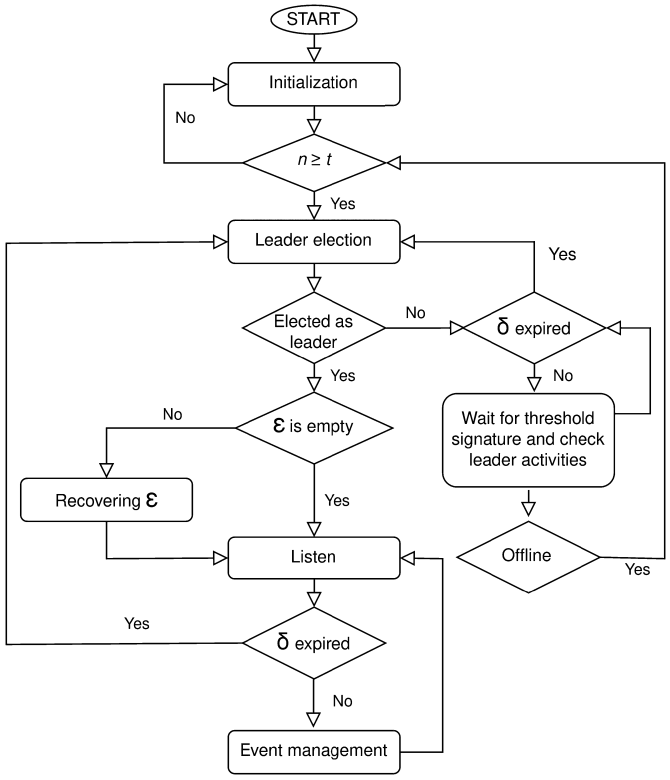

Fig. 6. The figure shows the flow diagram followed by the Listeners during the initialisation phase and during the main routine.

Once the initialisation phase is finished, the Listeners begin the main routine. Listener nodes work in rounds; each round starts with a leader election and has a certain duration. At the beginning, a leader election is performed, which will select one node of the Listener as the leader (line 1). The selected node will have a finite amount of time δ (line 2), after which, a new leader election will take place. The **if** check (line 3) differentiates the tasks of the leader versus non-leader nodes. If a node is a leader, the first operation it must do is check whether there are events that have not yet been terminated (line 4) and if so, it must recover and manage them (line 5). At this point, the Listener remains listening until its turn ends (lines 7-9). This phase involves the management of a series of events, which will be presented in the next subsections. When the time δ has expired, the leader node broadcasts a new leader election (line 10). Nodes that are not leaders wait for requests, the partial threshold signature performed by the leader (line 13). Non-leader nodes must also check the activity of the selected leader. If the leader is offline, non-leader nodes broadcast a new leader election (line 14-16). After the expiration of the δ time the nodes broadcast a new leader election (line 18).

Fig. 6 shows the flowchart of the Procedures 1 and 2 for initialisation and the main routine respectively. The cell *Event management* refers to the management of events that occur within the protocol performed by the individual Listeners and will be presented in detail in the next subsections. The $n \geq t$ check ensures that the number of nodes to initialise is greater than or equal to the threshold required by the threshold signature.

6.1. Procedures on sender Listener Λ_A

This section shows the events that the Listener Λ_A will have to manage on the side of the source blockchain $\mathcal{B}_A$, i.e., the blockchain that initiates inter-chain transactions. Inter-chain transactions can be initiated by independent participants p_k simply interacting with the smart contract $Sender_A$. A participant p_k must prepare a transaction $tx_A = P2C(p_k, Sender_A, f_i, v_i)$ and send it to the blockchain. If the transaction tx_A is valid, it will be processed by the consensus algorithm of the source blockchain, and at the end of its execution, an event will be issued e_i. Procedure 3 shows the steps needed to create the cryptographic proof.

Procedure 3: On Proof Request.

Input: An inter-chain transaction $tx^i_{A\to B} = (tx^i_A, tx^i_B)$
Output: Threshold Signature of Λ_A for tx^i_A
1 **Function** CREATEPROOF(p_k, *nonce*, tx^i_A, $\sigma_{sk_{p_k}}(tx^i_A)$) **is**
2 $p_k \leftarrow$ public key of participant p_k
3 *nonce* $\leftarrow$ current *nonce* registered in the smart contract $Sender$
4 $tx^i_A = P2C(p_k, Sender_A, f_i, v_i) \leftarrow$ is the transaction
5 $\sigma_{sk_{p_k}}(tx^i_A) \leftarrow$ is the signature of the participant p_k
6 **if** *find*(*nonce*, p_k, f_i, $\mathcal{E}_A$) *is not empty* **then**
7 **return** $\varnothing$
8 **end**
9 **if** *find*(p_k, f_i, $\mathcal{E}_A$) *is not empty* $\wedge$ $f_i \in \mathcal{F}_{ind}$ **then**
10 **return** $\varnothing$
11 **end**
12 **if** *find*(f_i, $\mathcal{E}_A$) *is not empty* $\wedge$ $f_i \in \mathcal{F}_{dep}$ **then**
13 **return** $\varnothing$
14 **end**
15 $< \sigma_1, \sigma_2, \ldots, \sigma_t > \leftarrow$ RequestThresholdSignature(p_k, *nonce*, tx^i_A, $\sigma_{sk_{p_k}}(tx^i_A)$)
16 **if** $vk(\sigma_i) == 1$; $\forall \sigma_i$ **then**
17 $\sigma(tx^i_A) \leftarrow$ ComputeThresholdSignature($\{\sigma_1, \sigma_2, \ldots, \sigma_t\}$)
18 **return** $\sigma(tx^i_A)$
19 **end**
20 **return** $\varnothing$
21 **end**

The Listener Λ_A executes the CREATEPROOF function that prepares and outputs the cryptographic proof $\sigma(tx^i_A)$. A participant p_k that wants to initiate an inter-chain transaction $tx_{A\to B} = (tx^i_A, tx^i_B)$ must provide the transaction tx^i_A, the virtual *nonce* of the smart contract $Sender_A$ required by it and the signature to authenticate the participant p_k (lines 2-5). Here, the *nonce* refers to a progressive number registered within the smart contract $Sender_A$, and each participant p_k has a specific *nonce* tied to it. The leader node must check for the validity of the *nonce* provided, and the existence of possible transactions in the event set $\mathcal{E}_A$, in which case the leader returns an empty value (lines 6-14). In particular, the **If** (lines 6-8) verifies the correctness of the nonce, preventing replay attacks. The **If** (lines 9-11) verifies that participant p_k has no pending inter-chain transactions in the case the function f_i is independent, which allows to regulate user activities and better manage the flow of transactions between interconnected blockchains. Finally, the **If** (lines 12-14) prevents dependent functions f_i from executing if there are pending transactions on them. This helps maintain a consistent blockchain state and ensures isolation in interchain transactions. After the checks, the leader node broadcasts a RequestThresholdSignature to the non-leader nodes. Each non-leader node has to verify the correctness of the data for the inter-chain transaction and has to compute the partial threshold signature (line 15). The leader has to collect at least t partial signatures from the non-leader nodes. After collecting the partial signatures, the leader must verify the correctness of the received signatures; if the verification is successful, the leader will prepare the cryptographic proof and output it (line 16-19), otherwise, output nothing (line 20).

Procedure 4 shows the operations that the Listener Λ_A has to follow when an inter-chain transaction request event e_i is issued by the source blockchain $\mathcal{B}_A$.

This means that a participant p_k invoked an inter-chain transaction request from the $Sender_A$ smart contract. The Listener Λ_A, reading the e_i event, will trigger the INTER-CHAINREQUESTEMITTED function where the Listener must associate the waiting time for the event and add it to the set of events to handle. To calculate the waiting time for the validity

Procedure 4: On Inter-Chain Transaction Request.

Input: Inter-Chain Transaction Request Event e_i
Output: Update the event set $\mathcal{E}_A$ adding e_i and $\Delta_A(e_i)$
1 **Function** *InterChainRequestEmitted*(e_i)**is**
2 $block \leftarrow \text{block}(e_i)$
3 $time \leftarrow \text{time}(e_i)$
4 $\Delta_A(e_i) \leftarrow$ start counter for (e_i)
5 **return** $\mathcal{E}_A \leftarrow \mathcal{E}_A \cup (e_i, \Delta_A(e_i))$
6 **end**

of the event, the Listener must use the block that generates the event and the time of the block generation (lines 2-3). At this point, the Listener must set the validity time $\Delta_A(e_i)$ for the event e_i (line 4). The procedure ends adding the new event with its validity time to the event list $\mathcal{E}_A$ (line 5).

Procedure 5 shows the behaviour of the Listeners in case an inter-chain transaction completed event is emitted by the destination blockchain.

Procedure 5: On Inter-Chain Transaction Completed.

Input: Inter-Chain Transaction Completed Event e_j
Output: Stop $\Delta_A(e_i)$ and remove e_i
1 **Function** *InterChainCompletedRead*(e_j)**is**
2 $nonce \leftarrow \text{nonce}(e_j)$
3 $p_k \leftarrow \text{participant}(e_j)$
4 $e_i \leftarrow \text{find}(nonce, p_k, \mathcal{E}_A)$
5 **return** $\mathcal{E}_A \leftarrow \mathcal{E}_A \backslash e_i$
6 **end**

In this case, it means that the inter-chain transaction was successful and the Listener has to execute the InterChainCompletedRead function. The leader node extracts, from the inter-chain transaction completed event e_j emitted by the destination blockchain, the participant p_k that started the inter-chain transaction and the *nonce* related to the inter-chain transaction (lines 2-3). Using this information, the leader node will select the event e_i from the event set $\mathcal{E}_A$ (line 4) and has only to remove the event e_i that was waiting for completion from the set of events $\mathcal{E}_A$ and return the updated set (line 5).

Procedure 6 shows the behaviour of the protocol if the validity time for an inter-chain transaction has expired.

Procedure 6: On Time Out Δ_A.

Input: An event e_i expired
Output: Update the event state with *In rollback*
1 **Function** *TimedOut*Δ_A*()***is**
2 $e_i \leftarrow e_i \in \mathcal{E}_A$ *s.t.* $\Delta_A(e_i)$ expired
3 $tx_A^{-1} \leftarrow$ roll back transaction for e_i
4 $\{\sigma_1, \sigma_2, \ldots, \sigma_t\} \leftarrow \text{RequestThresholdSignature}(tx_A^{-1}, e_i)$
5 **if** $vk(\sigma_i) == 1; \forall \sigma_i$ **then**
6 $\sigma(tx_A^{-1}) \leftarrow \text{ComputeThresholdSignature}(< \sigma_1, \sigma_2, \ldots, \sigma_t >)$
7 $\mathcal{E}_A \leftarrow \mathcal{E}_A \backslash e_i$
8 $\mathcal{W}_A \leftarrow \mathcal{W}_A \cup \sigma(tx_A^{-1})$
9 $(e_i, \not\Delta_A(e_i), H(tx_A^{-1})) \leftarrow$ update event with roll back transaction hash
10 **end**
11 **return** $\mathcal{E}_A \leftarrow \mathcal{E}_A \cup (e_i, \not\Delta_A(e_i), H(tx_{A \to B}^{-1}))$
12 **end**

In this case, the procedure must include the roll back on the source blockchain. The function that takes care of this task is TimedOutΔ_A and has to recover the event e_i that timed out (line 2). We indicate an event that has an expired delta with the symbol $\not\Delta$. The leader node (p_l) of the Listener Λ_A will have to prepare a transaction $tx_A^{-1} = P2C(p_l, Sender_A, f^{-1}, v)$ that reverses the changes that have occurred on the source blockchain (line 3). Once the transaction has been prepared, the leader has to broadcast the RequestThresholdSignature and wait for at least t valid partial signatures (line 4). The leader has to verify the validity of the partial signatures and prepare the threshold signature of the transaction (line 5-6). At this point the leader will remove the event e_i from the event set $\mathcal{E}_A$ (line 7), will submit the inter-chain transaction roll back (line 8) and will update the event e_i with the hash of the roll back transaction (line 9). At the end of the procedure, the leader will output the event set $\mathcal{E}_A$ updated with the event tagged with the hash of the roll back transaction (line 11).

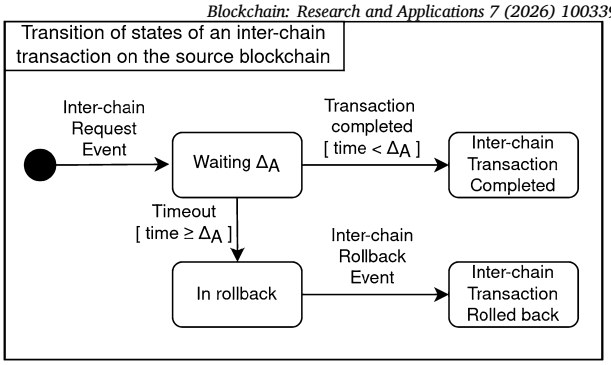

Fig. 7. The figure shows the state transition of an inter-chain transaction on the source blockchain side.

Procedure 7 illustrates the operations that the Listener Λ_A must perform when an inter-chain transaction rolled back event is generated. The leader node p_l has only to remove the event e_i from the event set $\mathcal{E}_A$ as the global state of the source blockchain $\mathcal{B}_A$ has been restored.

Procedure 7: On Transaction Rolled Back Event.

Input: Transaction Rolled Back Event e_k
Output: remove the event from the set $\mathcal{E}_A$
1 **Function** *RollbackCompleted*()**is**
2 $H(tx_A^{-1}) \leftarrow$ transaction hash of e_k
3 $e_i \leftarrow e_i \in \mathcal{E}_A$ *s.t.* $\exists(e_i, \not\Delta_A(e_i), H(tx_A^{-1}))$
4 **return** $\mathcal{E}_A \leftarrow \mathcal{E}_A \backslash e_i$
5 **end**

Fig. 7 shows the state diagram of inter-chain transactions on the source blockchain. The procedures illustrated so far (i.e., Procedures 3–7) have the task of monitoring the state of the inter-chain transaction and leading it to the terminal states, transaction completed (success) or transaction rolled back (abort).

6.2. Procedures on receiver Listener Λ_B

This section shows the events that a Listener Λ_B will need to handle on the side of the target blockchain $\mathcal{B}_B$, i.e., the blockchain that has to complete the inter-chain transaction.

Procedure 8 shows the steps the protocol must follow when an inter-chain transaction is initiated. In this case, the Listener Λ_B reads the inter-chain transaction request event e_i from the source blockchain $\mathcal{B}_A$ and must prepare and send the final transaction on the destination blockchain $\mathcal{B}_B$. The InterChainRequestRead routine shows the necessary steps. Using the data present in the event, the leader maintains the block $block(e_i)$ of blockchain $\mathcal{B}_A$ that generated the inter-chain transaction request event, the generation time $time(e_i)$ of the event which will be used to calculate the Δ_A of the source blockchain (lines 2-4). Starting from the data $data(e_i)$, the leader must prepare the transaction $tx_B = P2C(p_q, Receiver_B, f_i, v)$ which will complete the inter-chain transaction generated by the event e_i (lines 5-6). Once the transaction is prepared, the leader p_q must request the threshold signature (line 7) of at least t non-leader nodes to validate the transaction so that it

can be sent to the target blockchain $\mathcal{B}_B$. If the leader receives at least t valid partial signatures, it can compute the threshold signature and can submit the transaction to the blockchain (lines 9-10). At this point, the leader must start the timer $\Delta_B(e_i)$, related to the event e_i (line 11). At the end of the procedure, the leader updates the set of events with the new event added (line 13).

Procedure 8: On Inter-Chain Transaction Request.

```
   Input: Inter-Chain Request Event e_i
   Output: Updated set of events E_B
1  Function INTER-CHAINREQUESTREAD(e_i) is
2      block_A ← block(e_i)
3      time_A ← time(e_i)
4      Δ_A(e_i) ← calculate counter for e_i
5      data ← data(e_i)
6      tx_B ← PrepareTransaction(data)
7      {σ_1, σ_2, ..., σ_t} ← RequestThresholdSignature(tx_B, e_i)
8      if vk(σ_i) == 1; ∀σ_i then
9          σ(tx_B) ← ComputeThresholdSignature(< σ_1, σ_2, ..., σ_t >)
10         W_B ← W_B ∪ σ(tx_B)
11         Δ_B(e_i) ← Δ_A(e_i) − cλ_B
12     end
13     return E_B ← E_B ∪ (e_i, Δ_B(e_i))
14 end
```

Procedure 9 shows the behaviour of the protocol in the case where the inter-chain transaction was successfully executed on the destination blockchain $\mathcal{B}_B$. In this case, the behaviour of the Λ_B Listener is quite simple. The INTER-CHAINCOMPLETEDEMITTED function shows the steps the leader has to follow. The leader node p_q, using the *nonce* and the participant p_k read from the event e_j, search for the inter-chain transaction request in the event set $\mathcal{E}_B$ (lines 2-4). At this point, the only operation that must be performed is to remove the event from the set of events to be managed (line 5).

Procedure 9: On Inter-Chain Transaction Completed.

```
   Input: Inter-Chain Transaction Completed Event e_j
   Output: Updated set of events E_B
1  Function INTER-CHAINCOMPLETEDEMITTED(e_j) is
2      nonce ← nonce(e_j)
3      p_k ← participant(e_j)
4      e_i ← find(nonce, p_k, E_B)
5      return E_B ← E_B\e_i
6  end
```

Procedure 10 shows the steps performed by the protocol in case of timeout in the Λ_B Listener of the destination blockchain. The symbol $\not{\Delta}_B(e_i)$ means the event e_i has expired. In this case, it means that the transaction was not validated within the Δ_B time and the Listener only needs to monitor the $c\lambda_B$ interval.

Procedure 10: On Time Out Δ_B.

```
   Output: Updated set of events E_B
1  Function TIMEDOUTΔ_B() is
2      e_i ← e_i ∈ E_B s.t. Δ_B(e_i) expired
3      E_B ← E_B\e_i
4      (e_i, ∄Δ_B(e_i), cλ_B^{e_i}) ← monitor block time cλ_B^{e_i}
5      return E_B ← E_B ∪ (e_i, ∄Δ_B(e_i), cλ_B^{e_i})
6  end
```

Procedure 11 shows the behaviour of the protocol while the Listener has to monitor $c\lambda_B$. In this case, the protocol must manage two possible scenarios: 1. the transaction is validated within the λ_B time; 2. the transaction is not validated within the λ_B time. Case 1 is handled by

Procedure 11: Monitoring time $c\lambda_B$.

```
   Output: Updated set of events E_B
1  Function CHECKTIMEDOUTλ_B() is
2      e_i ← e_i ∈ E_B s.t. cλ_B(e_i) expired
3      tx_B ← tx_B ∈ W_B
4      tx_B^∅ ← DeleteTransaction(tx_B)
5      {σ_1, σ_2, ..., σ_t} ← RequestThresholdSignature(tx_B^∅, e_i)
6      if vk(σ_i) == 1; ∀σ_i then
7          σ(tx_B^∅) ← ComputeThresholdSignature(< σ_1, σ_2, ..., σ_t >)
8      end
9      W_B ← W_B ∪ σ(tx_B^∅)
10     return E_B ← E_B\e_i
11 end
```

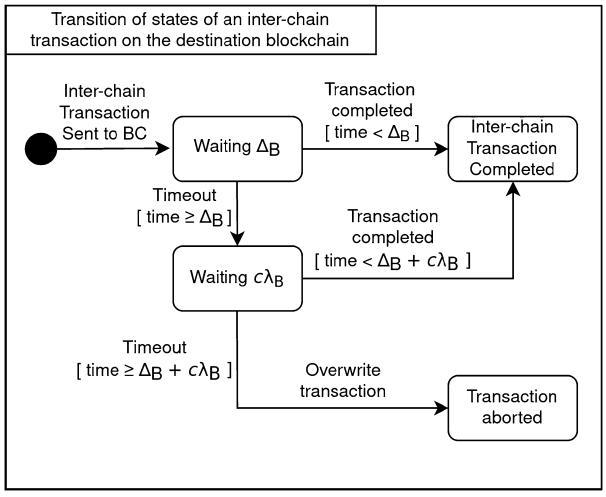

Fig. 8. The figure shows the state transition of an inter-chain transaction on the destination blockchain side.

Procedure 9. Case 2 is different for blockchains that have mechanisms to associate validity times with transactions in the waiting pool and for blockchains that do not allow this. In particular, blockchains that do not allow pending transactions to expire must run the CHECKTIMEDOUT procedure. The Listener Λ_B must check whether the extra time $c\lambda_B$ expected for the transaction has expired (line 2). If $c\lambda_B$ has expired, the Listener must retrieve the transaction from the waiting pool (line 3) and prepare a transaction that replaces the pending transaction with a null transaction (line 4). The new transaction will not change the state of the blockchain and will not emit any events. The Listener proceeds by requesting the threshold signature from the non-leader nodes (line 5). If all partial signatures are valid, the leader node will prepare the transaction that will delete the pending transaction (lines 6-7). At this point, the Listener must send the transaction to the target blockchain (line 9) and update the event set, removing the event e_i (line 10). Otherwise, the procedure will simply remove the transaction from the event set (i.e., only lines 2, 9 and 10 must be executed).

Fig. 8 shows the state diagram of inter-chain transactions on the destination blockchain. The procedures illustrated so far (i.e., Procedures 8–11) have the task of monitoring the state of the inter-chain transaction and leading it to the terminal states, transaction completed (success) and the transaction aborted (abort).

6.3. Correctness of Smart listener protocol

The correctness of the protocol lies in the safety and liveness properties. Given the purpose of the protocol, safety is a property that guarantees that an inter-chain transaction either occurs correctly or does not occur at all. From the Definition 18 of inter-chain transaction we have to guarantee:

success: At the end of the protocol we have:

$$tx_A \in \mathcal{L}_A \wedge tx_B \in \mathcal{L}_B$$

abort: At the end of the protocol we have:

$$tx_A \notin \mathcal{L}_A \wedge tx_B \notin \mathcal{L}_B$$

The protocol must avoid the following situation:

$$(tx_A \in \mathcal{L}_A \wedge tx_B \notin \mathcal{L}_B) \vee (tx_A \notin \mathcal{L}_A \wedge tx_B \in \mathcal{L}_B) \quad (2)$$

As regards liveness, we need to guarantee that the conditions of success or abort are reached within a finite amount of time, which in the protocol is expressed by Δ_A.

In what follows, we discuss all the properties that the protocol should have, that are agreement, validity, termination, atomicity, consistency, isolation and durability. These properties hold under the hypothesis that the number of non-faulty nodes composing the Listeners is at least t, where t is the minimum number of nodes needed by the threshold signature and the transaction priority of the consensus algorithms adopted by the involved blockchains falls under the type $ii.$ or $iii.$ as expressed by Definition 13.

Agreement and validity: To prove the properties of agreement and validity, we have to prove that Λ_A and Λ_B agree on the transactions tx_A and tx_B responsible for the updates of the global states Ω_A and Ω_B, and that are stored on the ledger $\mathcal{L}_A$ and $\mathcal{L}_B$, respectively. We also need to prove that the transactions tx_A and tx_B are valid and that the tx_B transaction finalises the tx_A transaction in the destination blockchain.

We note that for the transaction tx_A, to be sent to the waiting pool $\mathcal{W}_A$ of the source blockchain, the latter must be approved by the Listener Λ_A using a cryptographic proof based on a threshold signature. It means that the nodes of the Listener Λ_A must agree on the validity of the transaction tx_A performed by a certain participant p (Procedure 3). If the transaction tx_A is valid, it can be sent to the blockchain, i.e., $tx_A \in \mathcal{W}_A$. If all the changes foreseen by tx_A are valid, tx_A will be added to the validated transaction ledger $\mathcal{L}_A$ in the source blockchain $\mathcal{B}_A$. After that, tx_A has a certain validity period Δ_A. At this point, we are in the condition $tx_A \in \mathcal{L}_A \wedge tx_B \notin \mathcal{W}_B$. Both Listeners agree on the presence of the transaction tx_A on $\mathcal{L}_A$ (Procedure 4 for Λ_A and Procedure 8 for Λ_B), as it relies on the consensus $\mathcal{A}_A$ of the source blockchain $\mathcal{B}_A$.

Now the Listener Λ_B must produce the tx_B transaction which finalises the inter-chain transaction on the destination blockchain $\mathcal{B}_B$ (Procedure 8). To do so, the nodes of the Listener Λ_B must agree on the transaction data and produce the threshold signature. All nodes can verify the transaction data, moreover, they are certified by the consensus of the source blockchain. If the data is correct and the processes are in agreement, they will produce the threshold signature. The transaction tx_B can be built only using the data contained in the event generated by the consensus algorithm of the source blockchain relating to the transaction tx_A (i.e., the data needed to finalise the inter-chain transaction). If tx_B is valid, it will be signed by all nodes composing the Listener Λ_B and will be sent to the destination blockchain, i.e., $tx_B \in \mathcal{W}_B$. Now we are in the situation $tx_A \in \mathcal{L}_A \wedge tx_B \in \mathcal{W}_B$.

The Listener Λ_B is aware of this condition, while the Listener Λ_A is waiting for the completion of the inter-chain transaction (i.e., it is willing to wait Δ_A). Currently, the control is under the consensus algorithm of the destination blockchain $\mathcal{B}_B$, and depending on the transaction selection provided by the consensus algorithm $\mathcal{A}_B$ can happen: 1. the transaction tx_B is validated within $\Delta_B + c\lambda_B$. In this situation both Listener Λ_A and Λ_B agree on the completion of the transaction (Procedure 5 for Λ_A and Procedures 9 and 10 for Λ_B), as they became aware of the event produced by the destination blockchain that enforces its validity on the consensus algorithm of the destination blockchain. This ends in the correct condition *success*: $tx_A \in \mathcal{L}_A \wedge tx_B \in \mathcal{L}_B$; 2. The transaction tx_B is not validated within $\Delta_B + c\lambda_B$, in this case the Listeners agree on the expiration of the transaction validity (Procedures 6 and 7 for Λ_A and Procedures 10 and 11 for Λ_B). The Listener Λ_A will roll back the global state changes on the source blockchain and the Listener Λ_B will prevent the state update on the destination blockchain by deleting the pending transaction tx_B, or by expiring the validity of the transaction, this ends in the correct state *abort*: $tx_A \notin \mathcal{L}_A \wedge tx_B \notin \mathcal{L}_B$.

In summary, in the *Smart listener* protocol, the conditions of agreement and validity are achieved by combining the threshold signature implemented by the Listeners Λ_A and Λ_B with the consensus algorithm of the source and destination blockchains.

Termination: The termination of the protocol is guaranteed by the Eq. (1). In particular, within the operability interval Δ_A (i.e., the event that triggers an inter-chain transaction), the protocol always arrives in a terminal state, whether *success* or *abort*. It does not happen that the protocol encounters a bad state expressed by the propositional Eq. (2).

Atomicity: This property is reached by combining the functioning of the protocol with the characteristics of the blockchain. In fact, we notice that the protocol foresees only two transactions: one that finalises the inter-chain transaction on the destination blockchain and one that make a roll back on the source blockchain (the transaction that initiates the inter-chain transaction is performed by independent participants). The transactions sent by the Listeners (line 8 of Procedure 6 and line 10 of Procedure 8) are single transactions. In the event that transactions involve cascade calls, as expressed by the Definition 11, the latter are automatically managed by the relevant blockchains that will execute them in an atomic fashion. This ensures that transactions are atomic.

Consistency: To preserve consistency, the inter-chain transaction must preserve the data within the system. In the context of an inter-chain transaction, data integrity concerns the global state of blockchain $\mathcal{B}_A$ and blockchain $\mathcal{B}_B$. This property is preserved because the protocol does not meet the state expressed by the propositional Eq. (2).

Isolation: This property is ensured by the protocol by preventing participants from invoking multiple inter-chain transactions that involve dependent functions. This rule is guaranteed by the constraint $c.2$, as illustrated in Section 5.1 scenario 1.(a) (see also lines 12 - 14 of the Procedure 3).

Durability: To ensure the durability property, special attention must be given to the configurations of Δ_A and Δ_B. If the protocol interconnects blockchains that implement deterministic consensus algorithms, the selection of Δ_A and Δ_B becomes simpler. This is because once a transaction is included in a block, it is irrevocable. In the case where interconnected blockchains are based on probabilistic consensus algorithms, the configurations of Δ_A and Δ_B are more complicated. Once a transaction is added to a block, its persistence is not immediately guaranteed, since probabilistic consensus algorithms allow forks. Therefore, the generated events should be considered valid only after the probability of a fork becomes negligible.

7. Application and experimental result

The protocol presented aims to be blockchain-agnostic. The difference in the cryptographic primitives adopted by particular blockchains is mitigated by the fact that the protocol requires write access only in the blockchain to which it is connected, where compatibility is required for both signature and hash generation. While for the destination blockchain, only read access is required on specific events or system logs, for which there are no particular constraints in the cryptographic primitives adopted. In case the destination blockchain has a much longer block production time than the source blockchain (i.e., $\lambda_B > \lambda_A$), the initiation of many inter-chain transactions could saturate the destination blockchain or create bottlenecks. Differences in block generation times (λ_A and λ_B) are managed by preventing inter-chain transactions from starting if there are inter-chain transactions pending towards a specific user. This is a fundamental point as, on the one hand, it prevents saturation of the slower blockchain, and on the other, it favours the synchronisation of interconnected blockchains.

By construction, the connection between a private blockchain and a public blockchain does not require the presence of two Listeners, but only one managed by the private blockchain is needed. This is due to the fact that public blockchains have no access control mechanisms, so a single Listener is able to read and write to both blockchains. In this case, the Listener, being an off-chain component, can use different interfaces for communication with the different interconnected blockchains.

The case of interconnection between private blockchains is more complex, in which privacy is an indispensable requirement. To date, there are two well-known frameworks mostly adopted in private contexts, which are Hyperledger Fabric [81] and Hyperledger Besu [82]. Both frameworks allow the construction of blockchains with different characteristics, both in the consensus algorithms and in the languages adopted for smart contracts. Fabric allows the development of smart contracts in Java, Go, and Javascript, while Besu is based on Solidity. One of the difficulties concerns the lack of well-defined standards for the representation of data on the different blockchains. For example, Solidity adopts the ERC-20 [83] and ERC-721 [84] standards for the representation of so-called fungible tokens and non-fungible tokens. In the case of inter-chain transactions between Fabric and Besu, special attention must be paid to the correct translation of these objects in the respective blockchains. The limitations of different consensus algorithms will be presented in the next subsection.

7.1. Experimental result

This section shows the results of the experiments conducted to understand the capabilities of the protocol in terms of latency and transactions per second (TPS). During testing, Listeners were simulated using multithreading processes developed in Python, version 3.11. The blockchain adopted is Hyperledger Besu, version 23.1.0-RC1. The blockchain is implemented using Docker, version 27.0.3 and docker-compose, version 1.29.2. The tests were performed on a machine with a Linux pop-os operating system, version 6.9.3, Intel Core i7 CPU, 2.20 GHz, 32 GB RAM, NVIDIA RTX 3070 GPU, 8 GB RAM.

The performance of the Listener in requests for inter-chain transactions from users and event management (see Procedure 3, Procedure 6 and Procedure 8) are shown in Table 1. During testing, the Listeners adopt a (t-of-n) threshold signature scheme (Σ^n) based on ECDSA and the leader has an operability time (δ) of 10 s. During the tests, the number of listener nodes n and the minimum threshold t for generating a valid signature were varied. Table 1 shows the average latency and TPS of the Listener.

The protocol is entirely event-based. This means the Listener performs inter-chain transactions in batches. The number of events contained within a block is dependent on the throughput of the source blockchain. For the purpose of testing, the source blockchain ($\mathcal{B}_A$) is implemented using Hyperledger Besu, and is configured with a block generation time (λ_A) of 4 s, adopts a deterministic consensus algorithm based on IBFT 2.0 [57] and is composed of 5 validator nodes. During the experiments, several transactions were submitted at different rates (from 10 to 1000 per second). The statistics shown in Fig. 9 were calculated over 50 iterations for each transaction rate.

Table 1
Latency and TPS of the Listeners to produce the threshold signature in Procedure 3, Procedure 6 and Procedure 8.

Nodes (n)	Threshold (t)	Latency	TPS
3	2	0.0010 s	>1000
5	3	0.0014 s	>700
10	6	0.0027 s	>375
15	8	0.0038 s	>260
20	11	0.0050 s	>190
30	16	0.0068 s	>140

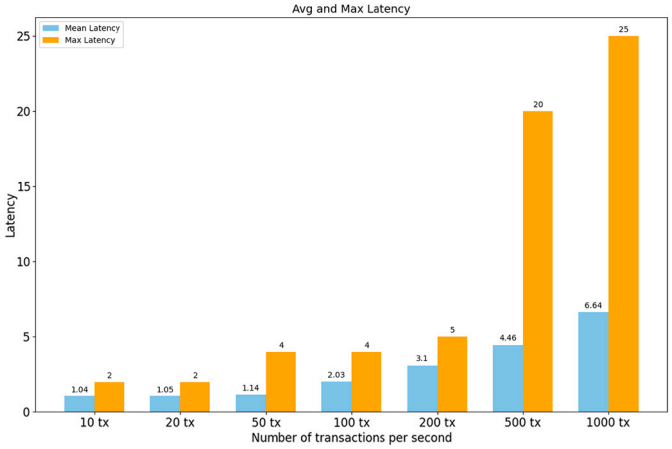

Fig. 9. Average and maximum latency during the interactions with the smart contract *Sender* (h_S).

During interactions with the *Sender* smart contract, the blockchain recorded an average TPS of 119, and a maximum TPS of 159, producing blocks with an average of 476 events. The largest block recorded 639 events. Once the events are generated, the Listeners must read them and set the Δ_A and Δ_B parameters for the validity of inter-chain transactions. These parameters are entirely dependent on the target blockchain, as they are set only after the transactions are validated by the source blockchain.

There are several works in the literature that evaluate the performance of modern blockchains. Shalaby et al. [85] showed the performance of Hyperledger Fabric. Capocasale et al. [86] showed a performance comparison among different blockchain frameworks. Here, we recall the main outcomes of the authors on different blockchain networks. From the results obtained by the authors, the performance of blockchains based on Hyperledger Fabric shows a TPS that varies between 150 and 250 TPS and an average latency that varies between 2 and 4 s. Concerning Hyperledger Besu, the performances are shown in the Figs. 9 and 10, obtaining an average latency of about 6.64 s when the blockchain is stressed. The tests refer to blockchains configured with consensus algorithms with deterministic finality; therefore, once a transaction is included in a block, the latter is crystallised and can no longer be removed. Regarding the configuration of Δ_A and Δ_B, performance metrics can be evaluated. The decision of these parameters is easier on blockchains with deterministic finality since it is sufficient to consider the block generation time and the latency. For example, Fig. 9 shows the maximum latency recorded for validating a transaction is 25 s (i.e., approximately 6.25 blocks under the configuration adopted). We could configure $\Delta_B + c\lambda_B$ as seven times the time required to produce a block on the destination blockchain. The configuration of Δ_A could be set

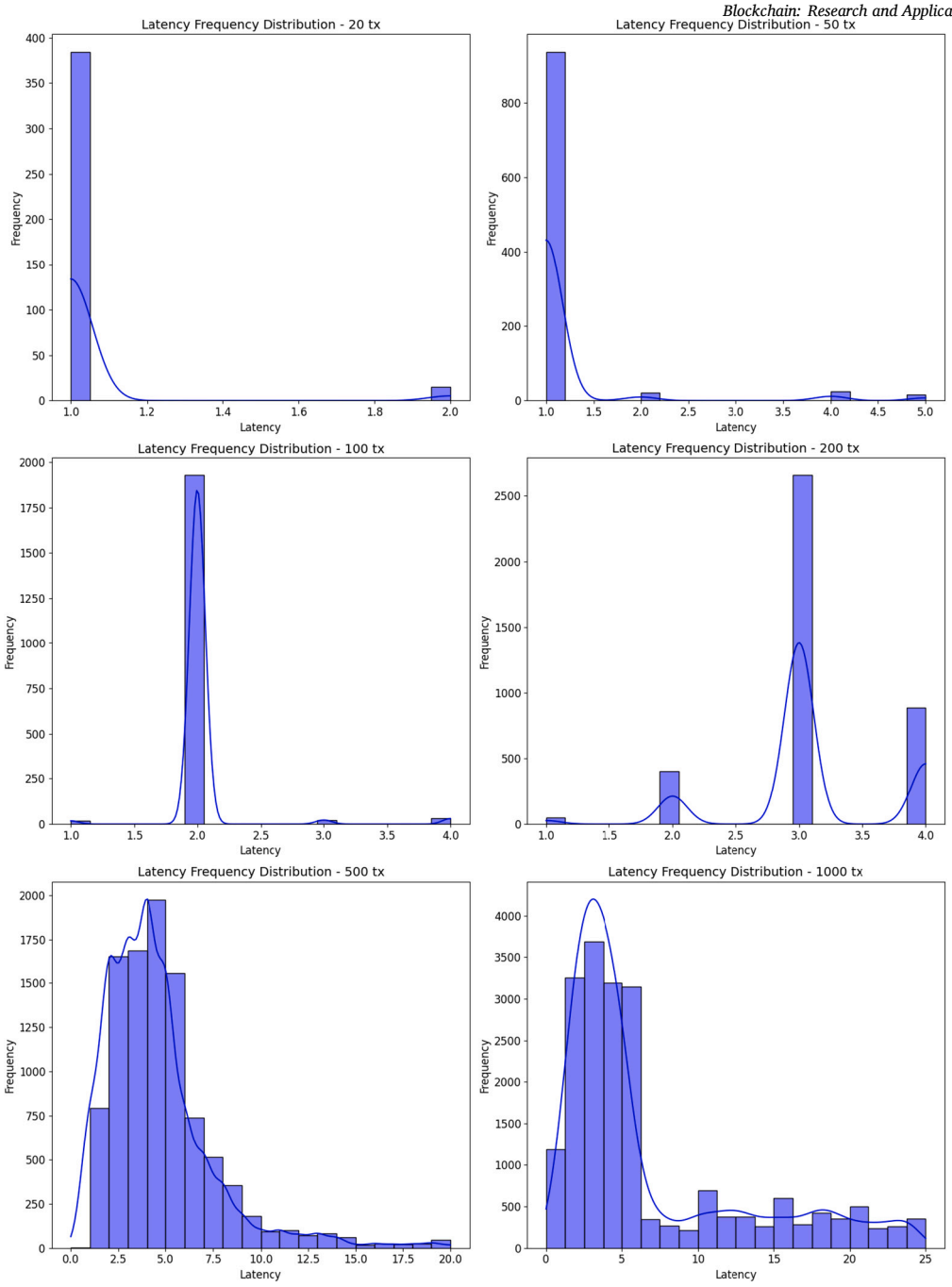

Fig. 10. Latency rate per single transaction during interactions with the *Sender* (h_S) smart contract.

to nine times the time required to produce a block on the destination blockchain. In general, the larger the Δ_A and $\Delta_B + c\lambda_B$ parameters are selected, the less likely it is to encounter a timeout.

The selection of the parameters Δ_A and $\Delta_B + c\lambda_B$ is more difficult to estimate in blockchains that implement consensus algorithms with probabilistic finality, since in such blockchains, there is a risk of possible forks. For example, if considering a Hyperledger Besu blockchain configured with Ethash as the consensus algorithm, the parameters Δ_A and Δ_B should be configured taking into account the time needed to make the transactions, responsible for generating events, final and irreversible. It is known that the Ethash protocol requires "12 confirmations" to make the probability of a fork negligible. This means a transaction can be considered final after 12 blocks from its validation. In this case, a possible configuration of Δ_A and Δ_B could be as follows: Δ_B stores the number of confirmations needed to make the transaction final, $c\lambda_B$ stores some extra blocks to further decrease the probability of a fork, then Δ_A can be configured arbitrarily larger than $\Delta_B + c\lambda_B$.

8. Discussion

In this section, we summarise the basic concepts that characterise the proposed protocol and we answer the questions posed in the introduction. As already underlined in the paper, the main obstacles to the design of interoperability protocols concern the heterogeneity of the techniques adopted for the consensus algorithms and the cryptographic primitives.

Consensus algorithms determine the persistence of the data of the relevant blockchains, influenced by the finality of the underlying protocol (deterministic or probabilistic). However, one of the points that could be problematic also concerns the throughput of the relevant blockchains. Making two blockchains that have profoundly different throughput in-

teroperate is very complicated. Depending on the behaviour of the protocol, there is the risk that the faster blockchain ends up saturating the slower blockchain before completing the inter-chain transaction, therefore attention must be paid both to the number of transactions that the protocol provides but also to the methods of submission of transactions. From this point of view, our protocol is optimal, as it employs the minimum number of necessary transactions, that is, one transaction in the source blockchain and one transaction in the destination blockchain. Furthermore, the protocol does not allow sending inter-chain transactions from the same participant if there are pending inter-chain transactions for that participant. Moreover, the throughput of the off-chain component is exclusively characterised by the throughput of the slowest blockchain.

Regarding the use of cryptographic primitives, our protocol requires that Listeners are compatible only with blockchains to which they have write access. Furthermore, by using events generated by target blockchains, the protocol exploits the properties of the consensus algorithms of the blockchains involved. In fact, events or log systems are generated only in the case in which a transaction is validated, and if this happens, it means that the transaction is accepted by the consensus algorithm of the particular blockchain.

At the same time, the strengths of our protocol are based on fairly strong assumptions about the blockchains involved. In fact, it is desired that blockchains support development languages that are Turing complete, that smart contracts are able to generate events or log systems and, preferably, that the consensus protocols implemented allow the priority of a transaction to be influenced. Furthermore, there are still no standards to follow to define the messages contained in the events, and again, there are no semantics that can define how the data must be interpreted by the various Listeners. The authors in Ref. [74] propose a possible interoperability semantics, but other solutions can be explored.

Finally, we want to answer the questions posed in the introduction, which are:

Q.1: *What are the technical requirements that a blockchain should meet to enable general-purpose interoperability protocols?*

Our response is based on the belief that a blockchain should enable the applicability of a general-purpose inter-chain protocol. So, the protocol should cover all necessary use cases. Therefore, in our opinion, a blockchain should meet the following requirements:

R.1-*Required*: The main requirement that a blockchain should meet to allow the applicability of general-purpose interoperability protocols is the presence of an off-chain component that is able to verify particular global state changes of the destination blockchains, and so be compliant with the Definition 17. This is a fundamental requirement. Otherwise generic interoperability is impossible, as also demonstrated by the authors in Ref. [26]. Each existing blockchain could extend its functionality by implementing its own Listener (Definition 16) or a different off-chain component.

R.2-*Desired*: The blockchain supports smart contracts based on a Turing-complete programming language. This is mandatory within our protocol, as it allows the verification of the threshold signature provided by the off-chain Listeners. More in general, the presence of smart contracts developed with Turing-complete languages provides versatility that allows extending the functionality of the blockchain. This allows the representation of complex objects and data structures needed to cover different use cases, simplifying the interaction between different blockchains. Furthermore, this allows for the implementation of complex operations, such as the verification of cryptographic schemes not integrated into the blockchain itself. This not only concerns signatures for authentication, but also any digests to verify data integrity.

R.3-*Desired*: Blockchains allow for the generation of system events or logs. This is mandatory in our protocol, but not mandatory in general. Since blockchains cannot communicate with the outside, the presence of events or log systems could increase their expressiveness. As shown in our protocol, events could be requested externally and allow for the construction of more versatile inter-chain protocols. Semantics for defining generic interoperability standards could be built on top of such events, as shown by the authors in Ref. [74]. Furthermore, the use of events could allow off-chain components to access only the necessary portion of data while preserving the privacy of the target blockchain.

R.4-*Desired*: The transaction selection of the consensus algorithm implemented in the blockchain allows transactions to be prioritized or to predict the average duration of a transaction in the waiting pool. This point is preferable in our protocol, but it is not in general. Working with consensus protocols that are deterministic or that allow to increase the priority of a transaction may help in the development of inter-chain protocols.

R.5-*Desired*: The blockchain allows waiting pool transactions to be associated with an expiration time or allows to replace pending transactions. This property is mandatory in our protocol, as it allows listeners to manage the timing of inter-chain transactions. This is generally desirable as it allows for the timing of inter-chain transactions to be limited, preventing any state of the source blockchain from being stuck for too long in the event of transaction completion delays in the destination blockchain.

Q.2: *What are the technical requirements an inter-chain protocol should provide to assure safety and liveness properties?*

The answer to this question is more complicated, as design choices can vary depending on needs, use cases and other parameters that are difficult to convey. Remaining as generic as possible, an inter-chain protocol should consider every possible use case, and an inter-chain transaction should be characterised by the Definition 18. Under these assumptions, a safety property must avoid the condition expressed by the propositional Eq. (2) and the liveness property must ensure that the protocol terminates in a correct state (*success* or *abort*) within a finite period of time.

In our protocol, safety is guaranteed by the presence of the (t, n)-threshold signature adopted by the Listeners and the robustness of the consensus algorithms implemented by the involved blockchains. Depending on the configuration adopted for the (t, n)-threshold signature, the protocol guarantees agreement and validity, tolerating up to $f = n/2 - 1$ faulty nodes. In general, any configuration of t such that $t \geq n/2 + 1$ is possible. Moreover, the liveness is guaranteed by the operability interval expressed by inequality (1), within which the protocol ends in a *success* or *abort* state.

However, protocols adopting different assumptions might meet the safety and liveness properties in different ways.

9. Conclusion and future works

In this paper, we propose a formalisation of blockchain that highlights its fundamental components: consensus algorithms and cryptographic primitives. Using the proposed formalisation, we designed an interoperability protocol capable of finalising an inter-chain transaction using only two transactions, one on the source blockchain and one on the destination blockchain. We provided a benchmark on the performance of the proposed protocol, then we discussed its correctness, both considering the safety and liveness properties, but also considering the ACID properties of distributed transactions. From the evidence obtained, we answered questions *Q.1*: What are the technical requirements that a blockchain should meet to enable general-purpose interoperability protocols? and *Q.2*: What are the technical requirements an inter-chain protocol should provide to assure safety and liveness properties?

However, nowadays, the proposal of protocols for interoperability remains an open challenge, since satisfying all the needs of heterogeneity in blockchain properties and implementation is very complicated. Our theoretical contribution aims to provide a starting point for greater design homogeneity in the future and, at the same time,

to understand which blockchains can implement general interoperability protocols today. An interesting step following this work consists in identifying which, among the existing blockchains, satisfy the requirements expressed in the answer to question *Q.1*. This would give an estimate of how ready current blockchains are for the implementation of blockchain-agnostic interoperability protocols.

Another aspect that could help in the development of interoperability protocols concerns the research and design of standards that can simplify communication between blockchains. In this context, one should try to abstract as much as possible from the particularity of blockchains, and facilitate communication with semantics and data formats. A possible future work involves defining event semantics adopted by the protocol that can define the data that should be expected in each event to satisfy any intended interoperability use case.

Declaration of competing interest

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

CRediT authorship contribution statement

Alessandro Bigiotti: Writing – original draft, Conceptualization, Formal analysis. **Leonardo Mostarda:** Writing – review & editing, Supervision, Validation. **Alfredo Navarra:** Validation, Supervision, Writing – review & editing. **Andrea Pinna:** Methodology, Conceptualization, Writing – review & editing. **Roberto Tonelli:** Validation, Writing – review & editing, Supervision. **Matteo Vaccargiu:** Conceptualization, Writing – review & editing, Data curation.

Funding

This work has received funding from the European Union's Horizon 2020 research and innovation program through the NGI TRUSTCHAIN program under a cascade funding agreement No. (101093274) Next Generation Internet.

Declaration of competing interest

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

The author Roberto Tonelli is an Editorial Board Member for "Blockchain: Research and Applications" and was not involved in the editorial review or the decision to publish this article.

Appendix A. Notations and abbreviations description

Notations and abbreviations descriptions are given in Tables A.1 and A.2.

Table A.1
The table shows all abbreviations used throughout the paper.

Abbreviation	Meaning
ACID	Atomicity, consistency, isolation, and durability
BFT	Byzantine fault tolerant
CCIP	Cross-chain interoperability protocol
CCC	Correct cross-chain communication
ECDSA	Elliptic curve digital signature algorithm
IBC	Inter-blockchain communication
IBFT	Istanbul Byzantine fault tolerant
PBFT	Practical Byzantine fault tolerant
PoA	Proof of authority
PoS	Proof of stake
PoW	Proof of work
SHA	Secure hash algorithm
2PC	Two-phase commit
TCC	Try-confirm-cancel
UTXO	Unspent transaction output
XCMP	Cross-chain message passing

References

[1] S. Nakamoto, Bitcoin: a peer-to-peer electronic cash system, http://www.bitcoin.org/bitcoin.pdf, 2009. (Accessed 29 November 2023).
[2] G. Wood, Ethereum: a secure decentralised generalised transaction ledger, https://ethereum.github.io/yellowpaper/paper.pdf, 2014. (Accessed 29 November 2023).
[3] J. Chen, S. Micali, Algorand, arXiv, 2016, preprint, arXiv:1607.01341.
[4] M. Alharby, A. Aldweesh, A.v. Moorsel, Blockchain-based smart contracts: a systematic mapping study of academic research, in: Proceedings of the 2018 International Conference on Cloud Computing, Big Data and Blockchain (ICCBB), 2018, pp. 1–6, https://doi.org/10.1109/ICCBB.2018.8756390.
[5] S. Solat, P. Calvez, F. Naït-Abdesselam, Permissioned vs. permissionless blockchain: how and why there is only one right choice, J. Softw. 16 (2021) 95–106.
[6] M. Herlihy, Atomic cross-chain swaps, in: Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing, PODC '18, ACM, 2018, pp. 245–254, https://doi.org/10.1145/3212734.3212736.
[7] P. Hoenisch, L.S. del Pino, Atomic swaps between bitcoin and monero, arXiv, 2021, preprint, arXiv:2101.12332.
[8] P. Hoenisch, S. Mazumdar, P. Moreno-Sanchez, et al., Lightswap: an atomic swap does not require timeouts at both blockchains, in: J. Garcia-Alfaro, G. Navarro-Arribas, N. Dragoni (Eds.), Data Privacy Management, Cryptocurrencies and Blockchain Technology, Springer International Publishing, Cham, 2023, pp. 219–235, https://doi.org/10.1007/978-3-031-25734-6_14.
[9] M.H. Miraz, D.C. Donald, Atomic cross-chain swaps: development, trajectory and potential of non-monetary digital token swap facilities, Ann. Emerg. Technol. Comput. 3 (1) (2019) 42–50, https://doi.org/10.33166/aetic.2019.01.005.
[10] T. Nadahalli, M. Khabbazian, R. Wattenhofer, Grief-free atomic swaps, in: Proceedings of the 2022 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2022, pp. 1–9, https://doi.org/10.1109/ICBC54727.2022.9805490.
[11] J.-Y. Zie, J.-C. Deneuville, J. Briffaut, et al., Extending atomic cross-chain swaps, in: C. Pérez-Solà, G. Navarro-Arribas, A. Biryukov, et al. (Eds.), Data Privacy Management, Cryptocurrencies and Blockchain Technology, Springer International Publishing, Cham, 2019, pp. 219–229, https://doi.org/10.1007/978-3-030-31500-9_14.
[12] S.A. Back, M. Corallo, L. Dashjr, et al., Enabling blockchain innovations with pegged, https://api.semanticscholar.org/CorpusID:18659636, 2014. (Accessed 29 November 2023).
[13] J. Poon, Plasma: scalable autonomous smart contracts, https://api.semanticscholar.org/CorpusID:13266881, 2017. (Accessed 29 November 2023).
[14] P. Robinson, Requirements for Ethereum private sidechains, arXiv, 2018, preprint, arXiv:1806.09834.
[15] A. Singh, K. Click, R.M. Parizi, et al., Sidechain technologies in blockchain networks: an examination and state-of-the-art review, J. Netw. Comput. Appl. 149 (2020) 102471, https://doi.org/10.1016/j.jnca.2019.102471.
[16] C. Goes, The interblockchain communication protocol: an overview, arXiv, 2020, preprint, arXiv:2006.15918.
[17] Polkadot, Cross-consensus message format (xcm), https://wiki.polkadot.network/docs/learn-xcm, 2022. (Accessed 9 July 2024).
[18] I. Foundation, Cosmos, https://v1.cosmos.network/resources/whitepaper, 2019. (Accessed 9 July 2024).
[19] G. Wood, Polkadot: vision for a heterogeneous multi-chain framework, https://assets.polkadot.network/Polkadot-whitepaper.pdf, 2019. (Accessed 9 July 2024).
[20] L. Breidenbach, C. Cachin, B. Chan, et al., Chainlink 2.0: next steps in the evolution of decentralized oracle networks, Whitepaper v2.0, https://research.chain.link/whitepaper-v2.pdf, 2021. (Accessed 9 July 2024).
[21] C. Foundation, Ccip-cross chain interoperability protocol, https://blog.chain.link/introducing-the-cross-chain-interoperability-protocol-ccip/, 2021. (Accessed 9 July 2024).
[22] A. Pupyshev, D. Gubanov, E. Dzhafarov, et al., Gravity: a blockchain-agnostic cross-chain communication and data oracles protocol, arXiv, 2020, preprint, arXiv:2007.00966.
[23] M. Javaid, A. Haleem, R. Pratap Singh, et al., Blockchain technology applications for industry 4.0: a literature-based review, Blockchain Res. Appl. 2 (4) (2021) 100027, https://doi.org/10.1016/j.bcra.2021.100027.
[24] T.B.d. Silva, E.S.d. Morais, L.F.F.d. Almeida, et al., Blockchain and industry 4.0: overview, in: R.R. Righi, A.M. Alberti, M. Singh (Eds.), Blockchain Technology for Industry 4.0, Springer, Singapore, 2020, pp. 27–58, https://doi.org/10.1007/978-981-15-1137-0_2.
[25] F. Liu, S. He, Z. Li, et al., An overview of blockchain efficient interaction technologies, Front. Blockchain 6 (2023), https://doi.org/10.3389/fbloc.2023.996070.
[26] A. Zamyatin, M. Al-Bassam, D. Zindros, et al., Sok: Communication across distributed ledgers, in: N. Borisov, C. Diaz (Eds.), Financial Cryptography and Data Security, Springer, Berlin, 2021, pp. 3–36, https://doi.org/10.1007/978-3-662-64331-0_1.

Table A.2
The table shows all the symbols used throughout the paper.

Symbol	Meaning
Σ	Digital signature scheme
$\Sigma^{(t,n)}$	Threshold signature scheme
G	Key generation algorithm
G_n	Threshold signature key generation algorithm
S	Signing algorithm
V	Verification algorithm
H	Hash function
σ	Digital signature
pk	Public key
sk	Private key
vk	Verification key
$\mathcal{P}$	All possible participant addresses and key pairs generated by a signature scheme Σ
Γ_C	Set of primitives for building smart contracts
C	Smart contract
$\mathcal{F}$	Set of functions defined in a smart contract
$\mathcal{E}$	Set of events, or log system, defined in a smart contract
h	Hash value
Ω	Global state of a blockchain
$P2P$	Participant to participant operation
$P2C$	Participant to smart contract operation
$C2C$	Smart contract to smart contract operation
$C2P$	Smart contract to participant operation
tx	Generic transaction
$\sigma(tx)$	Threshold signature of a transaction
$\mathcal{A}$	Consensus algorithm
$\mathcal{W}$	Set of pending transactions
$\mathcal{L}$	Ledger of validated transactions
Tx	Generic set of transactions
$p(\cdot)$	Priority of a transaction
$\mathcal{B}$	Blockchain
$tx_{A\to B}$	Inter-chain transaction from a blockchain A to a blockchain B
Λ	Listener
$Tx_B^{\Lambda_A}$	Set of transactions on a destination blockchain B that can be accessed by the Listener of a source blockchain A (and vice versa $Tx_A^{\Lambda_B}$)
Δ	Validity time for an inter-chain transaction
λ	Block generation time
h_S	Hash of the smart contract $Sender$
h_R	Hash of the smart contract $Receiver$
δ	Time of a round for a Listener's node elected as leader

[27] R. Belchior, L. Riley, T. Hardjono, et al., Do you need a distributed ledger technology interoperability solution?, Distrib. Ledger Technol. 2 (1) (2023) 1–37, https://doi.org/10.1145/3564532.

[28] A.F. Anta, K. Konwar, C. Georgiou, et al., Formalizing and implementing distributed ledger objects, ACM SIGACT News 49 (2) (2018) 58–76, https://doi.org/10.1145/3232679.3232691.

[29] P. Lafourcade, M. Lombard-Platet, About blockchain interoperability, Inf. Process. Lett. 161 (2020) 105976, https://doi.org/10.1016/j.ipl.2020.105976.

[30] S. Dziembowski, L. Eckey, S. Faust, Fairswap: how to fairly exchange digital goods, in: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS '18, ACM, 2018, pp. 967–984, https://doi.org/10.1145/3243734.3243857.

[31] Y. Chen, J. Guo, C. Li, et al., Fade: a blockchain-based fair data exchange scheme for big data sharing, Future Internet 11 (11) (2019), https://doi.org/10.3390/fi11110225.

[32] H. Pagnia, F.C.G. Darmstadt, On the impossibility of fair exchange without a trusted third party, https://api.semanticscholar.org/CorpusID:11671049, 1999. (Accessed 29 November 2023).

[33] H. Pagnia, H. Vogt, F.C. Gärtner, Fair exchange, Comput. J. 46 (1) (2003) 55–75, https://doi.org/10.1093/comjnl/46.1.55.

[34] M. Castro, B. Liskov, Practical Byzantine fault tolerance and proactive recovery, ACM Trans. Comput. Syst. 20 (4) (2002) 398–461, https://doi.org/10.1145/571637.571640.

[35] L. Lamport, R. Shostak, M. Pease, The Byzantine generals problem, ACM Trans. Program. Lang. Syst. 4 (3) (1982) 382–401, https://doi.org/10.1145/357172.357176.

[36] B. Alpern, F.B. Schneider, Defining liveness, Inf. Process. Lett. 21 (4) (1985) 181–185, https://doi.org/10.1016/0020-0190(85)90056-0.

[37] B. Alpern, F.B. Schneider, Recognizing safety and liveness, Distrib. Comput. 2 (3) (1987) 117–126, https://doi.org/10.1007/BF01782772.

[38] L. Lamport, Proving the correctness of multiprocess programs, IEEE Trans. Softw. Eng. SE-3 (2) (1977) 125–143, https://doi.org/10.1109/TSE.1977.229904.

[39] A. Momose, L. Ren, Multi-threshold Byzantine fault tolerance, in: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, CCS '21, ACM, 2021, pp. 1686–1699, https://doi.org/10.1145/3460120.3484554.

[40] G. Falazi, U. Breitenbücher, F. Leymann, et al., Transactional cross-chain smart contract invocations, Distrib. Ledger Technol. 4 (2) (2025) 1–26, https://doi.org/10.1145/3616023.

[41] A.S. Tanenbaum, Distributed Systems Principles and Paradigms, Pearson Education. Inc., 2007, Ch. 1.

[42] Y. Al-Houmaily, G. Samaras, in: L. Liu, M. Tamerözsu (Eds.), Encyclopedia of Database Systems, Springer, Boston, 2009, pp. 3204–3209, https://doi.org/10.1007/978-0-387-39940-9_713.

[43] G. Pardon, C. Pautasso, Atomic distributed transactions: a restful design, in: Proceedings of the 23rd International Conference on World Wide Web, ACM, 2014, pp. 943–948, https://doi.org/10.1145/2567948.2579221.

[44] D. Pritchett, Base: an acid alternative: in partitioned databases, trading some consistency for availability can lead to dramatic improvements in scalability, Queue 6 (3) (2008) 48–55, https://doi.org/10.1145/1394127.1394128.

[45] A. Redkar, C. Walzer, S. Boyd, et al., Transactional Messaging, in: A. Redkar, C. Walzer, S. Boyd, et al. (Eds.), Pro MSMQ: Microsoft Message Queue Programming, Apress, Berkeley, CA, 2004, pp. 211–234, https://doi.org/10.1007/978-1-4302-0732-0_4.

[46] S. Floyd, M. Allman, Comments on the usefulness of simple best-effort traffic, Tech. Rep., 2008.

[47] H. Garcia-Molina, K. Salem, Sagas, in: Proceedings of the 1987 ACM SIGMOD International Conference on Management of Data, SIGMOD '87, ACM, 1987, pp. 249–259, https://doi.org/10.1145/38713.38742.

[48] D.K. Black, The digital signature standard: overview and current status, Comput. Secur. 12 (5) (1993) 437–446, https://doi.org/10.1016/0167-4048(93)90062-A.

[49] S.M. Matyas, Digital signatures — an overview, Comput. Netw. 3 (2) (1976) 87–94, https://doi.org/10.1016/0376-5075(79)90007-2.

[50] V. Srivastava, A. Baksi, S.K. Debnath, An overview of hash based signatures, Cryptology ePrint Archive, Paper 2023/411, 2023, Preprint. https://eprint.iacr.org/2023/411.

[51] V. Daza, J. Herranz, G. Sáez, Some applications of threshold signature schemes to distributed protocols, Cryptology ePrint Archive, Paper 2002/081, 2002, Preprint. https://eprint.iacr.org/2002/081.

[52] B. Preneel, Cryptographic hash functions, Eur. Trans. Telecommun. 5 (4) (1994) 431–448, https://doi.org/10.1002/ett.4460050406.

[53] C. Paar, J. Pelzl, Hash Functions, in: C. Paar, J. Pelzl (Eds.), Understanding Cryptography, Springer, Berlin, 2010, pp. 293–317, https://doi.org/10.1007/978-3-642-04101-3_11.

[54] R. Sobti, G. Geetha, Cryptographic hash functions: a review, Int. J. Comput. Sci. Issues 9 (2) (2012) 461.

[55] D. Johnson, A. Menezes, S. Vanstone, The elliptic curve digital signature algorithm (ecdsa), Int. J. Inf. Secur. 1 (1) (2001) 36–63, https://doi.org/10.1007/s102070100002.

[56] M.J. Dworkin, Sha-3 standard: Permutation-based hash and extendable-output functions, 2015.

[57] R. Saltini, D. Hyland-Wood, Ibft 2.0: a safe and live variation of the ibft blockchain consensus protocol for eventually synchronous networks, arXiv, 2019, preprint, arXiv:1909.10194.

[58] S. Joshi, Feasibility of proof of authority as a consensus protocol model, arXiv, 2021, preprint, arXiv:2109.02480.

[59] C. Gupta, A. Mahajan, Evaluation of proof-of-work consensus algorithm for blockchain networks, in: Proceedings of the 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT), IEEE, 2020, pp. 1–7, https://doi.org/10.1109/icccnt49239.2020.9225676.

[60] C.T. Nguyen, D.T. Hoang, D.N. Nguyen, et al., Proof-of-stake consensus mechanisms for future blockchain networks: fundamentals, applications and opportunities, IEEE Access 7 (2019) 85727–85745, https://doi.org/10.1109/ACCESS.2019.2925010.

[61] C. Hoskinson, Why we are building Cardano, https://whitepaper.io/document/581/cardano-whitepaper, 2020. (Accessed 9 July 2024).

[62] F. Liu, Z. Li, K. Jia, et al., Bitcoin address clustering based on change address improvement, IEEE Trans. Comput. Soc. Syst. 11 (6) (2024) 8094–8105, https://doi.org/10.1109/TCSS.2023.3239031.

[63] M. Khabbazian, T. Nadahalli, R. Wattenhofer, Outpost: a responsive lightweight watchtower, in: Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT '19, ACM, 2019, pp. 31–40, https://doi.org/10.1145/3318041.3355464.

[64] B. Liu, P. Szalachowski, S. Sun, Fail-safe watchtowers and short-lived assertions for payment channels, in: Proceedings of the 15th ACM Asia Conference on Computer and Communications Security, ASIA CCS '20, ACM, 2020, pp. 506–518, https://doi.org/10.1145/3320269.3384716.

[65] J. Zhang, Y. Ye, W. Wu, et al., Boros: secure and efficient off-blockchain transactions via payment channel hub, IEEE Trans. Dependable Secure Comput. 20 (1) (2023) 407–421, https://doi.org/10.1109/TDSC.2021.3135076.

[66] J. Poon, T. Dryja, The bitcoin lightning network: scalable off-chain instant payments, http://lightning.network/lightning-network-paper.pdf, 2016. (Accessed 9 July 2024).

[67] B. Kachouh, L. Sliman, A.E. Samhat, et al., Demystifying threshold elliptic curve digital signature algorithm for multiparty applications, in: Proceedings of the 2023 Australasian Computer Science Week, ACSW '23, ACM, 2023, pp. 112–121, https://doi.org/10.1007/978-3-319-99058-3_12.

[68] H. Yu, H. Wang, Elliptic curve threshold signature scheme for blockchain, J. Inf. Secur. Appl. 70 (2022) 103345, https://doi.org/10.1016/j.jisa.2022.103345.

[69] S. Garg, A. Jain, P. Mukherjee, et al., Hints: threshold signatures with silent setup, Cryptology ePrint Archive, Paper 2023/567, 2023, Preprint. https://eprint.iacr.org/2023/567.

[70] F. Liu, Z. Feng, J. Qi, A blockchain-based digital asset platform with multi-party certification, Appl. Sci. 12 (11) (2022), https://doi.org/10.3390/app12115342.

[71] A. Bigiotti, L. Mostarda, A. Navarra, et al., Threshold signature in off-chain components to manage inter-chain transactions, in: Proceedings of the 2024 6th Conference on Blockchain Research & Applications for Innovative Networks and Services (BRAINS), IEEE, 2024, pp. 1–4, https://doi.org/10.1109/BRAINS63024.2024.10732513.

[72] L. Richardson, S. Ruby, RESTful Web Services, O'Reilly Media, Inc., 2008.

[73] A. Rodriguez, Restful web services: the basics, IBM developerWorks 33 (2008) 18.

[74] A. Bigiotti, L. Mostarda, A. Navarra, et al., Interoperabily between evm-based blockchains, in: L. Barolli (Ed.), Advanced Information Networking and Applications, Springer, Cham, 2024, pp. 98–109, https://doi.org/10.1007/978-3-031-57853-3_9.

[75] A. Hope-Bailie, S. Thomas, Interledger: creating a standard for payments, in: Proceedings of the 25th International Conference Companion on World Wide Web, International World Wide Web Conferences Steering Committee, ACM, 2016, pp. 281–282, https://doi.org/10.1145/2872518.2889307.

[76] S. Schulte, M. Sigwart, P. Frauenthaler, et al., Towards blockchain interoperability, in: C. Di Ciccio, R. Gabryelczyk, L. García-Bañuelos, et al. (Eds.), Business Process Management: Blockchain and Central and Eastern Europe Forum, Springer, Cham, 2019, pp. 3–10, https://doi.org/10.1007/978-3-030-30429-4_1.

[77] M. Sigwart, P. Frauenthaler, C. Spanring, et al., Decentralized cross-blockchain asset transfers, in: Proceedings of the 2021 Third International Conference on Blockchain Computing and Applications (BCCA), 2021, pp. 34–41, https://doi.org/10.1109/bcca53669.2021.9657007.

[78] E. Voris, Stellar transaction mechanism, https://developers.stellar.org/docs/learn/fundamentals/transactions/operations-and-transactions, 2021. (Accessed 29 November 2023).

[79] G. Wood, et al., Ethereum transaction mechanism, https://ethereum.org/en/developers/docs/transactions/, 2016. (Accessed 29 November 2023).

[80] K. Lei, Q. Zhang, L. Xu, et al., Reputation-based Byzantine fault-tolerance for consortium blockchain, in: Proceedings of the 2018 IEEE 24th International Conference on Parallel and Distributed Systems (ICPADS), 2018, pp. 604–611, https://doi.org/10.1109/PADSW.2018.8644933.

[81] Hyperledger, Hyperledger fabric, https://www.hyperledger.org/projects/fabric, 2016. (Accessed 29 November 2023).

[82] Hyperledger, Hyperledger besu, https://www.hyperledger.org/use/besu, 2022. (Accessed 29 November 2023).

[83] R. Norvill, B. Fiz, R. State, et al., Standardising smart contracts: automatically inferring erc standards, in: Proceedings of the 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019, pp. 192–195, https://doi.org/10.1109/bloc.2019.8751350.

[84] S. Bradić, D. Delija, G. Sirovatka, et al., Creating own nft token using erc721 standard and solidity programming language, in: Proceedings of the 2022 45th Jubilee International Convention on Information, Communication and Electronic Technology (MIPRO), 2022, pp. 1053–1056, https://doi.org/10.23919/MIPRO55190.2022.9803593.

[85] S. Shalaby, A.A. Abdellatif, A. Al-Ali, et al., Performance evaluation of hyperledger fabric, in: Proceedings of the 2020 IEEE International Conference on Informatics, IoT, and Enabling Technologies (ICIoT), 2020, pp. 608–613, https://doi.org/10.1109/iciot48696.2020.9089614.

[86] V. Capocasale, D. Gotta, G. Perboli, Comparative analysis of permissioned blockchain frameworks for industrial applications, Blockchain Res. Appl. 4 (1) (2023) 100113, https://doi.org/10.1016/j.bcra.2022.100113.