Cluster Computing
https://doi.org/10.1007/s10586-022-03737-6

Decentralized cross-blockchain asset transfers with transfer confirmation

Michael Sober[1,4] · Marten Sigwart[2] · Philipp Frauenthaler[2] · Christof Spanring[3] · Max Kobelt[1,4] · Stefan Schulte[1,4]

Received: 1 April 2022 / Revised: 8 August 2022 / Accepted: 25 August 2022

Abstract
Today, several solutions for cross-blockchain asset transfers exist. However, these solutions are either tailored to specific assets or neglect finality guarantees that prevent assets from getting lost in transit. In this paper, we present a cross-blockchain asset transfer protocol that supports arbitrary assets, is adaptable to different means of cross-blockchain communication, and adheres to requirements such as finality. The ability to freely transfer assets between blockchains may increase transaction throughput and provide developers with more flexibility by allowing them to design digital assets that leverage the capacities and capabilities of multiple blockchains. We define the general requirements and specifications for a cross-blockchain asset transfer protocol and provide a proof-of-concept implementation for EVM-based blockchains. Further, we evaluate the protocol concerning costs, transfer duration, and security.

Keywords Blockchain interoperability · Decentralized asset transfers · Cross-blockchain communication · Digital assets

1 Introduction

With its ability to store data and perform computations in a decentralized and immutable manner, blockchain technology shows potential in application areas such as finance [1], supply chain management [2], healthcare [3], business process management [4], smart cities [5], the Internet of Things [6, 7] and others [8]. Multiple independent and unconnected blockchains have been developed [9] to address the diverse requirements of these areas. As it is unlikely that a single blockchain caters to the requirements of all different areas [10], there is a strong need for interoperability between distinct blockchains.

Especially in scenarios where assets, i.e., digital representations of value, are managed on-chain, the lack of interoperability leads to a vendor lock-in as assets cannot leave the blockchain platform on which they were issued. Users would have to continue using the blockchain they initially chose since switching to another blockchain would only be possible with considerable effort or not at all. This vendor lock-in exposes projects to significant risks such as limited scalability [11], the danger that the underlying blockchain sinks into insignificance, and the inability to

✉ Michael Sober
michael.sober@tuhh.de

Marten Sigwart
m.sigwart@dsg.tuwien.ac.at

Philipp Frauenthaler
p.frauenthaler@dsg.tuwien.ac.at

Christof Spanring
christof.spanring@bitpanda.com

Max Kobelt
max.kobelt@tuhh.de

Stefan Schulte
stefan.schulte@tuhh.de

1 TU Hamburg, Hamburg, Germany

2 TU Wien, Vienna, Austria

3 Pantos GmbH, Vienna, Austria

4 Christian Doppler Laboratory for Blockchain Technologies for the Internet of Things, Hamburg, Germany

take advantage of new features offered by novel blockchains [9]. Of course, a centralized entity can be deployed to migrate assets from one blockchain to another. However, this contradicts the blockchain's original idea of decentralization [12].

The ability to transfer assets to arbitrary blockchains in a decentralized way would remove the need to commit to a particular blockchain. Instead, assets could be migrated to new blockchains offering novel functionality or better security at any time [9]. Another potential use case of cross-blockchain asset transfers arises in the context of sidechains [13, 14]. The idea is that an asset can be transferred to and processed on multiple "side" blockchains, thus reducing the workload of the original blockchain.

One way to exchange assets between independent blockchains is via atomic swaps [15]. With atomic swaps, multiple parties can atomically exchange assets, whereby no party ends up worse off if one of them misbehaves. However, atomic swaps do not constitute actual cross-blockchain asset transfers. The different assets are not leaving their respective blockchain but rather ownership of assets changes in an atomic fashion. Cross-blockchain asset transfers, on the contrary, achieve that an asset moves from one blockchain to the other, enabling users to hold different denominations of the same asset type on multiple blockchains.

While schemes for cross-blockchain asset transfers have been proposed before, most of these solutions are designed with specific assets in mind and neglect important requirements for cross-blockchain asset transfers, such as finality to prevent assets from getting lost in transit.

Thus, in this paper, we formally define a set of general requirements that need to be fulfilled by cross-blockchain asset transfers and then propose a protocol that complies with the defined requirements.

This paper extends our previous work [16] by (i) providing more background information about cross-blockchain communication, (ii) adding additional functionality to the protocol, (iii) conducting a more detailed evaluation of the extended protocol while using different cross-blockchain communication mechanisms, and (iv) updating the related work.

The contributions of this paper can be summed up as follows:

- We formally define functional requirements for cross-blockchain asset transfers.
- We define a protocol specification for enabling cross-blockchain asset transfers that are decentralized, secure, and not tailored to specific assets.
- We evaluate the protocol on public Ethereum test networks using a proof-of-concept implementation for blockchains based on the Ethereum Virtual Machine (EVM).

To this end, Sect. 2 provides important background information. In Sect. 3, we formally define the requirements and specify the protocol for cross-blockchain asset transfers. Section 4 evaluates the proposed protocol concerning costs, duration, security, and features using a proof-of-concept implementation. Section 5 provides an overview of the related work. Finally, Sect. 6 concludes the paper.

2 Background

This section introduces some notations and definitions necessary for describing the requirements and specifications of the proposed cross-blockchain asset transfer protocol. Further, we look into different means of cross-blockchain communication, which is an essential building block of the protocol proposed in Sect. 3.

2.1 Notations and definitions

As has already been mentioned in Sect. 1, cross-blockchain asset transfers ideally enable users to hold different denominations of the same asset on multiple blockchains simultaneously. By that, users are free to choose on which blockchain they want to keep their assets. An asset can be seen as anything holding some value with a corresponding representation on a blockchain.

Assets can generally be divided into fungible and non-fungible assets [17]. Fungibility implies that two entities of the same asset are interchangeable. Cryptocurrencies like Bitcoin or Ether are fungible assets. Another example of fungible assets is Ethereum tokens following the ERC20 standard [18]. In contrast, non-fungible assets are uniquely identifiable, i.e., one entity cannot be substituted by another entity. For instance, ERC721 tokens (e.g., Cryptokitties) are non-fungible assets [19].

One can further distinguish between native and user-defined assets [17]. Native assets are inherently part of a particular blockchain. One cannot exist without the other, e.g., the Bitcoin and Ether cryptocurrencies and the Bitcoin and Ethereum blockchains, respectively. On the other hand, certain blockchains allow the implementation of use case-specific assets with their own set of rules, e.g., the already mentioned ERC20 or ERC721 tokens. Contrary to native assets, these user-defined assets are not bound to specific blockchains. Instead, they are implemented using smart contracts and can thus be potentially deployed on any blockchain with the necessary scripting capabilities to express the asset's rules.

This work concentrates on user-defined assets since the goal is to provide an asset that allows users to hold different amounts of the same asset on multiple blockchains at the same time. We formally define an asset A as a set where the set's members represent the asset's smallest indivisible entities (asset entities). For instance, the smallest indivisible entity of a fungible asset like Bitcoin is a Satoshi (i.e., 0.00000001 BTC). For a non-fungible asset like Cryptokitties, the smallest indivisible entity is a single "cryptokitty".

We define the set of blockchains between which asset transfers can take place by the finite set B (also referred to as the *cross-blockchain ecosystem*). Each blockchain $b \in B$ can host multiple smart contracts. Out of these smart contracts, one contract is responsible for managing asset A on b. We denote this particular smart contract as c_b for each $b \in B$.

We assume blockchains to roughly follow the model devised by Satoshi Nakatomo [1]: The state of the blockchain is updated through transactions that can be used to transfer the native asset of the blockchain, store arbitrary data, or trigger the execution of smart contracts. In the latter case, the transaction's payload contains parameters based on which the smart contracts may change their associated state. For instance, a transaction payload containing a sender, a recipient, and an amount could trigger a smart contract causing the transfer of some user-defined asset from the sender to the recipient.

We specify transactions as a tuple containing the elements of the payload, which serve as parameters for the invoked contract. In particular, we use $tx := \langle param_1, \ldots, param_n \rangle$ to denote a transaction tx with a payload containing n parameters. Further, we define the function $calledContract(tx)$ to return the address of the smart contract that was triggered by transaction tx. The execution of transactions may fail, for instance, if a user does not have enough funds for a transfer. For this, we define the function $isSuccessful(tx)$ to return *true* or *false* depending on whether the transaction has been executed successfully or not.

Every transaction is signed by some off-chain user $u \in U$ before being submitted to the blockchain. The function $submitter(tx)$ denotes the user that signed tx. Users can be the owners of a subset of asset A on each participating blockchain $b \in B$. Subsequently, the set $A_u^b \subseteq A$ defines the entities of asset A that are owned by a particular user $u \in U$ on blockchain b.

Finally, for two blockchains $src, dest \in B$, and two users $sender$, $recipient \in U$, we define a cross-blockchain asset transfer as the transfer of some $X \subseteq A$ from user $sender$ on source blockchain src to user $recipient$ on destination blockchain $dest$.

2.2 Cross-blockchain communication

Fundamental to the protocol proposed in Sect. 3 is the ability to communicate state information between blockchains. In particular, the protocol relies on the ability to verify the inclusion of transactions across blockchains, i.e., the destination blockchain $dest$ must be able to verify that certain transactions are included in the source blockchain src. Ideally, this cross-blockchain communication is decentralized such that no trust in a centralized party is required to ensure the validity of the information.

2.2.1 Oracles

Oracles, or more generally, data on-chaining solutions [20], offer one way to realize cross-blockchain communication. An oracle acts as a bridge between a blockchain and external data sources. The task of the oracle is to retrieve the data from the external data source (e.g., another blockchain) and submit it to a smart contract. While different oracle solutions exist, one approach to verify the inclusion of transactions is voting-based oracles. Several concepts and solutions for voting-based oracles have already been proposed [21–25].

A voting-based oracle requires a special oracle contract on the destination blockchain. Clients can call the oracle contract to get state information about the source blockchain src, e.g., to check for the inclusion of a specific transaction. The oracle contract starts a voting period during which other users can post their votes ("yes" or "no"). The answer which reaches a previously defined threshold (e.g., the majority) wins. Users are encouraged to participate in the voting process through crypto-economic incentives. Unfortunately, the voting mechanism incurs high costs if the aggregation of the votes is done on-chain. However, there are already oracle solutions, e.g. [25], that use an off-chain aggregation mechanism and only submit the aggregated result to the oracle contract. The oracle contract then only has to verify the validity of the aggregated result, which is cheaper since it requires only a single transaction.

2.2.2 Blockchain relays

Another technique for realizing transaction inclusion verifications is using blockchain relays [26]. Blockchain relays operate by having a set of off-chain clients relay block headers (see Fig. 1) from a source blockchain to a destination blockchain, replicating the source blockchain within the destination blockchain. Each relayed block header is validated on the destination blockchain according to the validation rules of the source blockchain. As block headers do not contain any transactions, only a small fraction of the space needed to store full blocks is consumed on the destination blockchain.

With the block headers of the source blockchain replicated, clients can query state information about the source blockchain on the destination blockchain, e.g., the current longest branch of the source blockchain, or if a certain block header is confirmed by at least x succeeding block headers.

Further, transactions in a blockchain (i.e., in a block) are stored in a special data structure called Merkle tree [27], or variants thereof. Since the root hash of the Merkle tree is

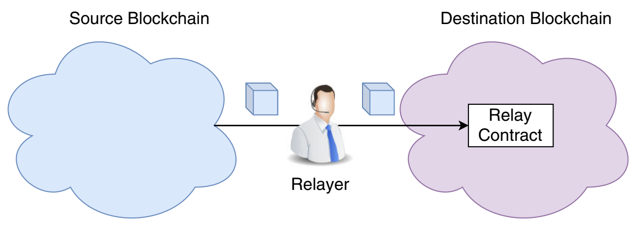

Fig. 1 Blockchain relay

also stored in the block header, it becomes possible on the destination blockchain to verify whether some transaction is included within a particular block of the source blockchain. For that, users construct a so-called Merkle proof of membership [27]. The proof is submitted to the destination blockchain, on which the proof is used to recalculate the root hash of the corresponding Merkle tree. If the calculated root hash matches the root hash of the stored block header, the destination blockchain can be certain that the transaction is included within the corresponding block of the source blockchain. This technique is also known as Simplified Payment Verification (SPV) [1].

Blockchain relays provide an on-chain answer to whether a certain transaction is included in the source blockchain. Contrary to voting-based oracles where multiple clients have to act honestly, blockchain relays only require one honest participating client [28]. However, blockchain relays cause higher costs since relayers have to continuously submit block headers to keep the relay up to date. In the following section, we use blockchain relays and oracles to realize the proposed asset transfer protocol.

3 Cross-blockchain asset transfers

In this section, we define the requirements for cross-blockchain asset transfers. Then, we use these requirements as the foundation to define a decentralized cross-blockchain asset transfer protocol.

3.1 Requirements

As defined in Sect. 2, a cross-blockchain asset transfer for an asset A constitutes the transfer of ownership of some subset $X \subseteq A$ from some user *sender* on a source blockchain *src* to another user *recipient* on a destination blockchain *dest*.

Before the transfer, X must only exist on blockchain *src*, and after the transfer, the asset must only exist on blockchain *dest*. At no point should X exist on both blockchains in parallel since the accidental duplication of asset entities can potentially lead to a deflation of the asset's value. Hence, a cross-blockchain asset transfer should only be successful, i.e., X is created on *dest*, if X has been burned (i.e., destroyed) before by its owner on *src*.

Therefore, before X can be recreated on *dest*, *dest* needs some kind of evidence that X has already been burned on *src*. If we assume that it is possible to provide such evidence guaranteeing that X has been burned on *src* and that this evidence can be used to recreate X on *dest*, two further requirements emerge. First, faking the evidence needs to be prevented at all costs. Users should not be able to counterfeit evidence certifying that X has been burned on *src* without it having occurred. Second, if the evidence is correct, it should only be used once to recreate X on a different blockchain, i.e., on blockchain *dest*. Hence, evidence of X having been burned on *src* cannot be used multiple times to recreate X on other blockchains. Essentially, disregard of any of these requirements would enable users to illegally create new entities of asset A out of nothing—again potentially deflating the value of the asset and decreasing trust in this particular asset.

A further requirement comes up when trying to prevent the opposite, accidental inflation of the asset's value. Accidental inflation could take place if X is burned on *src* without ever being recreated on *dest*. This reduces the total supply of A. Hence, cross-blockchain asset transfers need to be eventually finalized to not decrease the total supply of A over time. That is, either the transfer is executed completely or it fails with no intermediate state persisting.

Finally, the source blockchain *src* possibly needs to perform some action if a certain cross-blockchain asset transfer has been executed successfully (i.e., X has been successfully recreated on destination blockchain *dest*). For instance, imagine a business deal where some user *buyer* wants to buy asset entities $Y \subseteq A'$, with $A' \cap A = \emptyset$, from another user *seller*. While asset A' lives exclusively on the source blockchain *src*, user *buyer* aims to buy Y with cross-blockchain asset A, transferring X from themselves on blockchain *src* to user *seller* on blockchain *dest*. Therefore, ownership of Y can change on *src* if it is ensured that the transfer of X was successful, i.e., blockchain *src* must retrieve a confirmation of the successful transfer.

To sum up, we define the general requirements for a cross-blockchain asset transfer as follows:

Requirement 1 When a user *sender* wants to burn X on blockchain *src*, X should only be burned if $X \subseteq A^{src}_{sender}$.

Requirement 2 When transferring some $X \subseteq A$ from the source blockchain *src* to the destination blockchain *dest*, X should only be recreated on *dest* if it can be proven that X has already been burned on *src*. That is, it should not be possible to counterfeit the burning of asset entities.

Requirement 3 Double spending must be prevented at all times. That is, if X is burned on one blockchain, X can only be recreated once on one other blockchain.

Requirement 4 If X is burned on one blockchain, X is always recreated on another blockchain within a certain time limit t. Further, finality should not be dependent on a single actor (i.e., not be centralized).

Requirement 5 After burning X on source blockchain src, src will eventually receive a confirmation that X has been successfully recreated on another blockchain. Analogous to decentralized finality, any user should be able to submit confirmations. This requirement is *optional* since not every use case requires that the source blockchain knows whether a cross-blockchain asset transfer has been completed successfully.

In the next subsection, we define a cross-blockchain asset transfer protocol that fulfills these requirements. To this end, we first define a base protocol that fulfills Requirements 1 to 4 (Sect. 3.2). We then provide an extension of the protocol to also account for Requirement 5 (Sect. 3.3).

3.2 Base protocol

Protocol 1 Protocol for cross-blockchain asset transfers

Goal: For two blockchains $src, dest \in B$ and two users $sender, recipient \in U$, transfer $X \subseteq A$ from src to $dest$ and change ownership of X from $sender$ to $recipient$.

1. **Burn.** User $sender$ creates a new BURN transaction $tx_{\text{BURN}} := \langle recipient, dest, X \rangle$.
   (a) User $sender$ signs and submits tx_{BURN} to source blockchain src invoking contract c_{src}, i.e., the contract managing asset A on src.
   (b) When being invoked, contract c_{src} performs the following operations.
      i. Verify $dest \in B$ to make sure that the specified blockchain $dest$ is part of the cross-blockchain ecosystem.
      ii. Verify $X \subseteq A^{src}_{sender}$ to make sure that user $sender$ owns the asset entities it wants to transfer on blockchain src.
      iii. When all checks are successful, the asset entities to be transferred are burned, i.e., $A^{src}_{sender} = A^{src}_{sender} \setminus X$.
2. **Claim.** Once tx_{BURN} is included in blockchain src, any user $u \in U$ can construct the CLAIM transaction $tx_{\text{CLAIM}} := \langle tx_{\text{BURN}}, proof_{tx_{\text{BURN}}} \rangle$. Variable $proof_{tx_{\text{BURN}}}$ contains the Merkle proof of membership of tx_{BURN} certifying the inclusion of tx_{BURN} in blockchain src.
   (a) User u signs and submits tx_{CLAIM} to blockchain $b \in B$ invoking contract c_b, i.e., the contract managing asset A on b.
   (b) When being invoked, contract c_b utilizes the verifier contract $c_{verifier}$ to verify the inclusion and confirmation of tx_{BURN} in blockchain src, i.e., c_b calls $c_{verifier}.verifyInclusion(tx_{\text{BURN}}, proof_{tx_{\text{BURN}}}, src)$.
   (c) If $c_{verifier}$ confirms the inclusion of tx_{BURN}, contract c_b performs the following steps.
      i. Verify $b = dest$ to ensure that the executing blockchain b is the intended destination blockchain $dest$. Note that $dest$, $recipient$, and X are available within c_b as these variables are contained within the payload of tx_{BURN}.
      ii. Verify $tx_{\text{BURN}} \notin T_{\text{BURN}}$ where T_{BURN} is the set of BURN transactions that have already been used to claim entities of asset A on $dest$. This ensures that BURN transactions cannot be used multiple times for claiming.
      iii. Verify $calledContract(tx_{\text{BURN}}) = c_{src}$ to make sure that the contract that has been invoked by tx_{BURN} is a contract authorized for managing asset A on blockchain src.
      iv. Verify that $isSuccessful(tx_{\text{BURN}})$ returns true to ensure that the execution of c_{src} has been completed without error.
      v. If $c_{verifier}.confirmations(tx_{\text{BURN}}, src) > t$, user $recipient$ has not submitted tx_{CLAIM} within time t. Hence, the user $u = submitter(tx_{\text{CLAIM}})$ that submitted tx_{CLAIM} receives a transfer fee $X_{fee} \subseteq X$ as reward for finalizing the transfer, i.e., $A^{dest}_u = A^{dest}_u \cup X_{fee}$. Otherwise, no fee will be paid to u (i.e., $X_{fee} = \emptyset$), resulting in all asset entities being transferred to $recipient$.
      vi. (Re-)create the asset entities and assign ownership to user $recipient$, i.e., $A^{dest}_{recipient} = A^{dest}_{recipient} \cup (X \setminus X_{fee})$.
      vii. Add tx_{BURN} to the set of already used BURN transactions, i.e., $T_{\text{BURN}} = T_{\text{BURN}} \cup \{tx_{\text{BURN}}\}$.

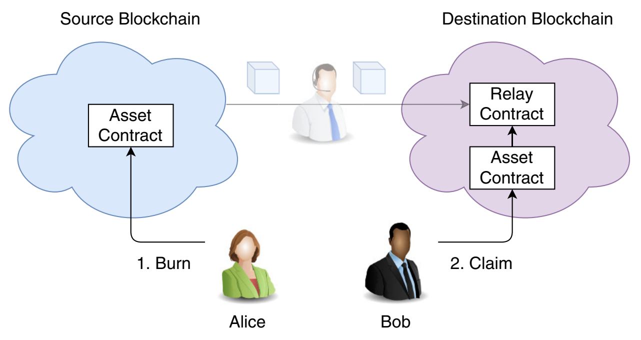

Fig. 2 Base protocol for cross-blockchain asset transfers leveraging a blockchain relay

As mentioned above, a cross-blockchain asset transfer should only be successful if the asset is first burned on the source blockchain and then recreated on the destination blockchain (Requirement 2). Therefore, the transfer requires at least two steps: one "burn" step on the source blockchain and one "claim" step on the destination blockchain. The protocol uses oracles or blockchain relays for decentralized cross-blockchain communication to verify the "burn" step. In particular, these provide an on-chain answer to whether a specific transaction is included in the source blockchain via SPV [1]. With this in mind, we can outline a minimal protocol for cross-blockchain asset transfers (see Fig. 2).

The protocol consists of a BURN transaction tx_{BURN} submitted to source blockchain *src* and a CLAIM transaction tx_{CLAIM} submitted to destination blockchain *dest*. Further, a verifier contract allows the asset contract on blockchain *dest* to verify the inclusion of BURN transactions in blockchain *src*. The exact specification is outlined in Protocol 1.

Initially, some user *sender* creates transaction tx_{BURN}. The payload of tx_{BURN} contains the user intended as the recipient of the transfer (*recipient*), an identifier representing the desired destination blockchain (*dest*), and the asset entities to be transferred (X). It may also be that additional data is required, as is the case with ERC721 tokens, which have a *tokenID* or *tokenURI*. These must also be part of tx_{BURN} or an additional data structure (e.g., the transaction receipt) that can be verified on the destination blockchain.

User *sender* then signs and submits tx_{BURN} to the source blockchain *src* invoking smart contract c_{src}, which manages asset A on blockchain *src* (Step 1a). The smart contract then verifies that the specified destination blockchain *dest* is part of the cross-blockchain ecosystem (Step 1(b)i). Second, the contract ensures that the user *sender* is the current owner of X on blockchain *src* (Step 1(b)ii). If both checks are successful, X is burned on *src* (Step 1(b)iii).

Once tx_{BURN} is included in blockchain *src*, any user $u \in U$ can construct the CLAIM transaction tx_{CLAIM}. The payload of tx_{CLAIM} consists of transaction tx_{BURN} and a Merkle proof of membership of tx_{BURN} ($proof_{tx_{\text{BURN}}}$). The verifier contract $c_{verifier}$ on blockchain *dest* uses the proof to verify the inclusion of tx_{BURN} in blockchain *src*. Note that if only the sender or only the recipient of the transfer were allowed to submit tx_{CLAIM}, the finality of the transfer (Requirement 4) would be entirely dependent on that particular user, e.g., the user could decide not to submit tx_{CLAIM}.

User u then signs and submits tx_{CLAIM} to some blockchain $b \in B$ invoking the contract c_b managing A on blockchain b (Step 2a).

By invoking the verifier contract $c_{verifier}$, the contract c_b checks whether tx_{BURN} is included and confirmed in blockchain *src* (Step 2b). If the verifier contract does not confirm the inclusion of tx_{BURN}, the claim request is rejected. Otherwise, contract c_b performs the following steps: First, it verifies that b is the intended destination blockchain *dest* (Step 2(c)i). Second, it is verified that tx_{BURN} has not been used to claim X on b before (Step 2(c)ii). Third, if both checks are successful, contract c_b verifies that the contract that burned X on *src* is a valid contract authorized for managing A on *src* (Step 2(c)iii). If this is the case, contract c_b further checks that tx_{BURN} was successful, i.e., the execution of contract c_{src} has been completed without any error, e.g., constraint violations (Step 2(c)iv). This check covers the case in which some blockchains transactions may be included even if the triggered smart contract execution was not successful. While Ethereum also includes failed transactions, other blockchains may not include such transactions at all.

The above checks ensure that tx_{CLAIM} is only successful if the corresponding tx_{BURN} was also executed successfully. To further account for Requirement 4 (transfer finality), the protocol must ensure that when transaction tx_{BURN} takes place on blockchain *src*, the corresponding tx_{CLAIM} is eventually submitted to destination blockchain *dest*.

Usually, the incentive for transfer finalization lies with the recipient of the transfer since the recipient wants to receive the transferred asset entities. However, in case the recipient is indisposed to submit tx_{CLAIM} for some reason, the protocol offers an incentive in the form of a transfer fee to other users. That is, any user u that successfully submits tx_{CLAIM} gets assigned a subset $X_{fee} \subseteq X$ as a reward (-Step 2(c)v). However, to provide the user *recipient* with the chance to receive all entities of X, other users are only eligible to receive the fee if they submit tx_{CLAIM} after a certain time t has elapsed.

Time t is defined by the number of blocks that succeed the block containing tx_{BURN} on source blockchain *src*. Hence, when being invoked by tx_{claim}, c_b additionally queries the verifier contract $c_{verifier}$ whether the block containing tx_{BURN} is confirmed by more than t succeeding blocks. If this is the case, the time t is considered elapsed, and the user that submitted tx_{CLAIM} receives the transfer fee

X_{fee}, while user *recipient* receives the rest $X \setminus X_{fee}$. If not, the user *recipient* always receives the entire set X (i.e., $X_{fee} = \emptyset$), even if another user submitted tx_{CLAIM} (Step 2(c)vi). Asset entities are (re-)created on blockchain b ($= dest$) by incrementing the balance of the recipient (in case of fungible assets) or by copying the transferred asset's data structure from tx_{BURN} into the storage of contract c_b (in case of non-fungible assets).

Finally, to ensure that tx_{BURN} cannot be used to claim X on b again, tx_{BURN} is added to the set of already used BURN transactions T_{BURN} (Step 2(c)vii).

3.3 Protocol extension for transfer confirmations

Protocol 2 Extension for transfer confirmations on the source blockchain

Goal: For two blockchains $src, dest \in B$ and two users $sender, recipient \in U$, transfer $X \subseteq A$ from src to $dest$ and change ownership of X from $sender$ to $recipient$. Confirm finalization of transfer on src.

1. **Burn.** Contrary to Protocol 1, tx_{BURN} contains an additional variable $Y \subseteq A$ representing the stake that will act as reward for users submitting a confirmation transaction to src once the transfer has been completed on $dest$.
   (a) User $sender$ signs and submits tx_{BURN} to source blockchain src invoking contract c_{src}.
   (b) When being invoked, contract c_{src} performs the following steps.
      i. Perform all verification steps from Step 1b of Protocol 1.
      ii. Verify $Y \subseteq A^{src}_{sender}$ to make sure that user $sender$ owns the asset entities she wants to use as stake on blockchain src.
      iii. Verify $Y \cap X = \emptyset$, i.e., X and Y are disjoint, to ensure that the asset entities intended to be transferred are not used as stake and vice versa.
      iv. When all checks are successful, the asset entities to be transferred are burned and the stake is locked, i.e., $A^{src}_{sender} = A^{src}_{sender} \setminus (X \cup Y)$.
2. **Claim.** To claim the entities of asset A on destination blockchain $dest$, the same steps as for tx_{CLAIM} in Protocol 1 are performed.
3. **Confirm.** Once tx_{CLAIM} is included in blockchain $dest$, any user $u \in U$ can construct the CONFIRM transaction $tx_{\text{CONFIRM}} := \langle tx_{\text{CLAIM}}, proof_{tx_{\text{CLAIM}}} \rangle$. Variable $proof_{tx_{\text{CLAIM}}}$ contains the Merkle proof of membership of tx_{CLAIM} certifying the inclusion of tx_{CLAIM} in blockchain $dest$.
   (a) User u signs and submits tx_{CONFIRM} to the source blockchain src invoking c_{src}.
   (b) When being invoked, contract c_{src} utilizes the verifier contract $c_{verifier}$ to verify the inclusion and confirmation of tx_{CLAIM} in blockchain $dest$, i.e., c_{src} calls $c_{verifier}.verifyInclusion(tx_{\text{CLAIM}}, proof_{tx_{\text{CLAIM}}}, dest)$.
   (c) If $c_{verifier}$ confirms the inclusion of tx_{CLAIM}, contract c_{src} performs the following steps.
      i. Verify $tx_{\text{CLAIM}} \notin T_{\text{CLAIM}}$ where T_{CLAIM} is the set of CLAIM transactions that have already been used for confirming transfer finalization on src. This ensures that CLAIM transactions cannot be used multiple times.
      ii. Verify $calledContract(tx_{\text{BURN}}) = c_{src}$ to ensure that tx_{CONFIRM} invokes the same contract that has been invoked by tx_{BURN}.
      iii. Verify $calledContract(tx_{\text{CLAIM}}) = c_{dest}$ to make sure that the contract that has been invoked by tx_{CLAIM} is the contract managing asset A on $dest$ as intended by tx_{BURN}.
      iv. Verify that $isSuccessful(tx_{\text{CLAIM}})$ returns true to ensure that the execution of c_{dest} has been completed without any error.
      v. Check if timeout for submitting tx_{confirm} is reached. If so, user $sender$ has not submitted tx_{CONFIRM} in time. Hence, the user $u = submitter(tx_{\text{CONFIRM}})$ that submitted tx_{CONFIRM} receives the locked stake from tx_{BURN} as reward for providing the confirmation of the corresponding claim, i.e., $A^{src}_{u} = A^{src}_{u} \cup Y$. If not, user $sender$ gets back control of the locked stake, i.e., $A^{src}_{sender} = A^{src}_{sender} \cup Y$, regardless of which user submitted tx_{confirm}.
      vi. Add tx_{CLAIM} to the set of already used CLAIM transactions, i.e., $T_{\text{CLAIM}} = T_{\text{CLAIM}} \cup \{tx_{\text{CLAIM}}\}$.

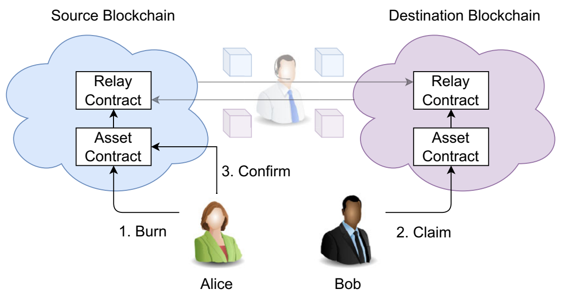

Fig. 3 Protocol extension to provide transfer confirmations leveraging blockchain relays

Protocol 1 provides finality by incentivizing users to submit tx_{CLAIM} for each tx_{BURN}. However, only the destination blockchain *dest* knows whether a cross-blockchain asset transfer has been completed successfully. Source blockchain *src* never actually learns about the finalization of the transfer. To circumvent this, we can augment Protocol 1 by adding a third kind of transaction (CONFIRM) to confirm the successful cross-blockchain asset transfer on *src*, as displayed in Fig. 3.

The extension of the protocol is specified in Protocol 2. At first, transaction tx_{BURN} is submitted to the source blockchain *src*. The payload of tx_{BURN} is the same as presented in Protocol 1—except for one additional field defined as Y. Y is a subset of asset A and represents a certain amount of stake that will act as a reward for users submitting a CONFIRM transaction to *src*.

When user *sender* submits tx_{BURN} to *src* (Step 1a), c_{src} first performs the same verifications as in Protocol 1 (Step 1(b)i). Additionally, a couple of checks concerning the stake Y are performed. First, it is verified that user *sender* is the owner of Y on blockchain *src* (Step 1(b)ii). Second, it is verified that the sets X and Y are disjoint (Step 1(b)iii). That is, the asset entities intended to be transferred should not be used as a stake and vice versa. If all verifications are successful, X is burned on *src*, and Y is locked (Step 1(b)iv).

Once tx_{BURN} is included in *src*, any user can create and submit transaction tx_{CLAIM} containing tx_{BURN} and a corresponding proof data $proof_{tx_{\text{BURN}}}$ to the destination blockchain *dest*. The steps performed when the corresponding smart contract c_{dest} is invoked are identical to those in Protocol 1 (Step 2).

With tx_{CLAIM} being successfully executed and included in blockchain *dest*, the successful transfer is reported back to source blockchain *src*. For that, any user can submit a CONFIRM transaction tx_{CONFIRM} to *src* (Step 3a).

When being invoked by tx_{CONFIRM}, contract c_{src} verifies that tx_{CLAIM} is included in *dest* using transaction inclusion verification (Step 3b). If the verifier contract does not confirm the inclusion of tx_{CLAIM}, the confirm request is rejected. Otherwise, contract c_{src} performs the following steps. First, it verifies that tx_{CLAIM} has not already been used for confirming transfer finalization (Step 3(c)i). Second, it checks that contract c_{src} is the contract that has burned X, i.e., whether c_{src} has been invoked by tx_{BURN} (Step 3(c)ii). This check ensures that contract c_{src} is the intended receiver of the confirmation. Third, c_{src} verifies that the contract invoked by tx_{CLAIM} is the correct contract managing asset A on the intended destination blockchain *dest* (Step 3(c)iii). *dest* can be extracted from tx_{BURN} included in the payload of the provided tx_{CLAIM}. Fourth, contract c_{src} checks that tx_{CLAIM} was successful, i.e., the execution of contract c_{dest} has been completed without any error (Step 3(c)vi).

If all checks are successful, contract c_{src} not only has the information about the successful execution of tx_{BURN} but also that the corresponding tx_{CLAIM} finalized the transfer on blockchain *dest*. To reward user u for submitting tx_{CONFIRM} to *src*, the locked stake Y from tx_{BURN} is assigned to u (Step 3(c)v). This provides an incentive for users to submit tx_{CONFIRM} to *src* for finalized cross-blockchain asset transfers.

The protocol uses a similar approach for the locked stake as for the transfer fee. The initiator of the transfer can withdraw its locked stake with certain conditions. A timeout determines who is eligible to retrieve the locked stake Y when tx_{CONFIRM} is submitted. Before the timeout, only *sender* is eligible for the stake. After the timeout, anyone submitting tx_{CONFIRM} earns Y as a reward. As contract c_{src} executes both the burning of entities and confirmations, it can track the time difference between the two events.

Finally, to ensure that tx_{CLAIM} cannot be used several times, tx_{CLAIM} is added to the set of already used CLAIM transactions T_{CLAIM} (Step 3(c)vi).

4 Evaluation

In this section, we evaluate the proposed cross-blockchain asset transfer protocol and its extension with regard to the defined requirements and conduct a quantitative analysis evaluating transfer costs and duration. For the evaluation, we provide a proof-of-concept implementation for EVM-based blockchains such as Ethereum and Ethereum Classic. The prototype, as well as the evaluation scripts used for obtaining the results presented in Sect. 4.3, are available as an open-source project on GitHub[1].

[1] https://github.com/soberm/x-chain-protocols/

4.1 Prototype

As mentioned above, the prototype is implemented for EVM-based blockchains. The advantages of targeting EVM-based blockchains in a first proof-of-concept are twofold. First, EVM-based blockchains such as Ethereum are among the most popular blockchains concerning decentralized applications (DApps) and digital assets [11, 29]. Cross-blockchain transfer capabilities for EVM-based blockchains can thus enhance the utility of a majority of available assets. The second reason is rather practical. As quite a few EVM-based blockchains exist, multiple blockchains can be targeted with a single implementation. However, as long as a blockchain provides sufficient scripting capabilities to implement the concepts of the protocol, as well as some means of transaction inclusion verification (e.g., via oracles or relays), the solution can be adopted beyond EVM-based blockchains.

For our analysis, we use the ERC20 token standard as asset representation. For transaction inclusion verification, we use two different means of cross-blockchain communication. Initially, we implemented the prototype to work with ETH Relay, a blockchain relay specifically targeting EVM-based blockchains [28]. ETH Relay follows a validation-on-demand pattern where relayers can issue a dispute during a certain lock time to trigger the validation of a block header. This approach saves costs since not every block has to be validated. For simplicity, we set the lock time to 0 as it would only lengthen the experiments and add a constant value to the transfer duration.

Additionally, we apply the oracle solution proposed in [25], which uses an off-chain aggregation mechanism based on threshold signatures. We extended the oracle to return block headers instead of single transactions such that it implements the same interface as ETH Relay. We run three oracle nodes, the majority of which have to agree on the same outcome to construct the threshold signature. With the application of both mechanisms, we can also observe how the asset transfer protocol works with different approaches to cross-blockchain communication.

A transaction in Ethereum consists of the fields *nonce*, *gasPrice*, *gasLimit*, *to*, *value*, *data*, and a signature (v, r, s) [30]. The field *data* contains the payload (e.g., the parameters for a smart contract invocation) of the transaction. The field *to* contains the address of the smart contract that has been invoked by the transaction (i.e., function $calledContract(tx)$ in our protocol). The submitter $submitter(tx)$ of the transaction can be calculated out of the signature fields v, r, and s.

It should be noted that the transaction data does not contain information about the transaction status, i.e., whether the execution succeeded or failed. In Ethereum, this information is stored in another data structure, the so-called transaction receipt. For each transaction, there exists a corresponding receipt. The receipt contains all events that were emitted during the execution of the transaction. Further, a status flag indicates the successful execution of the transaction. Thus, when evaluating the function $isSuccessful(tx)$ in our protocol, the asset contract must have access to the receipt of tx as well.

In Ethereum, a block's transactions and receipts are kept in separate Merkle trees, which do not contain references to each other. However, transaction and receipt are logically linked together as their position in their respective trees is identical. Both the inclusion of the transaction and the receipt can be verified via Merkle proofs of membership using the respective Merkle trees.

Both Merkle proofs must be evaluated along the same path to ensure that a transaction and receipt belong together. Hence, in our prototype, the proof data certifying the inclusion of a transaction (e.g., $proof_{tx_{\text{BURN}}}$) contains a Merkle proof for the transaction and the transaction receipt. Further, it includes the path for the evaluation of the Merkle proofs.

4.2 Requirements analysis

This section evaluates the protocol with regard to the requirements defined in Sect. 3.1.

4.2.1 Requirement 1 (ownership)

When user *sender* submits a BURN transaction tx_{BURN} invoking contract c_{src}, the contract verifies that $X \subseteq A^{src}_{sender}$ (see Step 1(b)ii of Protocol 1), thus making sure that user *sender* is the owner of X on *src*. Hence, we consider Requirement 1 as fulfilled.

4.2.2 Requirement 2 (no claim without burn)

To claim X on *dest*, a user u submits a CLAIM transaction tx_{CLAIM}. As defined by the protocol, the user provides tx_{BURN} as well as some proof data in the payload of tx_{CLAIM}. Before recreating X, the asset contract c_{dest} on blockchain *dest* performs several checks.

First, it is verified that tx_{BURN} is included in the source blockchain *src* and confirmed by enough blocks (Step 2b). Second, the protocol checks that tx_{BURN} indeed invoked asset contract c_{src} on the source blockchain *src* (Step 2(c)iii). Third, contract c_{dest} verifies that the execution of tx_{BURN} was successful (Step 2(c)iv).

These three checks ensure that assets are created on the destination blockchain *dest* if they have been successfully burned by the contract c_{src} on the source blockchain *src*.

Notably, the fulfillment of this requirement strongly depends on the security of the used transaction inclusion verification mechanism. A security analysis of the blockchain relay (ETH Relay), as well as the oracle used in our proof-of-concept implementation, can be found in [25, 28], respectively.

4.2.3 Requirement 3 (double spend prevention)

To ensure that burned assets can not be claimed multiple times, all BURN transactions that have already been used within CLAIM transactions are stored in a set T_{BURN} within asset contract c_{dest}. When c_{dest} is invoked by a new CLAIM transaction tx_{CLAIM}, it only executes the claim if the provided BURN transaction tx_{BURN} is not yet included in T_{BURN} (Step 2(c)ii).

Further, by encoding an identifier of the desired destination blockchain *dest* within BURN transactions, a burned asset can not be claimed on multiple different blockchains. When an asset contract c_b on some blockchain b is invoked by a CLAIM transaction containing tx_{BURN}, c_b can verify whether it is the intended destination contract by comparing $b = dest$ (Step 2(c)i). If not, the claim is rejected. Therefore, Requirement 3 can be considered fulfilled as well.

4.2.4 Requirement 4 (decentralized finality)

For the analysis of finality, we make use of the BAR (Byzantine, Altruistic, Rational) model [31], which has found application in security analysis for blockchain protocols and extensions before, e.g., [28, 32, 33]. Under this model, *Byzantine* users may depart arbitrarily from the protocol for any reason; *altruistic* users always adhere to the protocol rules, and *rational* users will deviate from the protocol to maximize their profit.

The protocol in this paper offers a reward to users submitting the CLAIM transaction. As the reward is at least as high as the submission costs of CLAIM transactions (see Sect. 4.3), rational users have an economic incentive to finalize transfers. However, in any protocol, rational users according to the BAR model, cannot be fully trusted to act rationally in the sense of the protocol's incentive structure since seemingly irrational behavior might be perfectly rational in the context of a larger ecosystem with the protocol being part of it [34]. For instance, rational users might aim at yielding profit in the larger ecosystem by finding ways to bet against the protocol or the value of the asset.

Therefore, rational users are not guaranteed to comply with the protocol rules even with perfectly aligned incentives. Building an open and permissionless system that withstands all participants potentially deviating from the protocol rules appears fundamentally impossible [34]. Thus, in our protocol, not only the users directly involved in a transfer are allowed to post the CLAIM transaction, but rather any user of the system can do it. This provides stronger finalization guarantees as finalization does not depend on a single user acting honestly. It is sufficient if one user out of all users is altruistic to ensure finalization. Notably, the protocol only relies on an altruistic user in case rational users see an incentive in deviating from the proposed protocol.

4.2.5 Requirement 5 (transfer confirmation)

With Requirements 1to 4 fulfilled, cross-blockchain asset transfers with decentralized finality can be realized. However, in Protocol 1, only the destination blockchain *dest* knows when a transfer is finalized. The source blockchain *src* has no way of knowing about the finalization. We have thus proposed Protocol 2 as an extension of Protocol 1 to fulfill the additional requirement of providing *src* with a finalization confirmation. The confirmation takes place in the form of an additional transaction tx_{CONFIRM} being submitted to *src*.

The CONFIRM transaction also has to comply with Requirements 2 to 4. The execution of tx_{CONFIRM} should only be successful if there exists a corresponding CLAIM transaction tx_{CLAIM} successfully executed on *dest* (Requirement 2). Further, it should not be possible to confirm tx_{CLAIM} multiple times (Requirement 3). Additionally, after the successful execution of a CLAIM transaction tx_{CLAIM} on dest, the corresponding CONFIRM transaction tx_{CONFIRM} is eventually submitted to *src* (Requirement 4).

The requirements are fulfilled by Protocol 2 using the same approach that is used for CLAIM transactions. That is, similar to how a CLAIM transaction's payload contains a BURN transaction and corresponding proof data, tx_{CONFIRM} consists of a CLAIM transaction tx_{CLAIM} and corresponding proof data $proof_{tx_{\text{CLAIM}}}$.

The asset contract c_{src} performs the following checks when being invoked by a CONFIRM transaction tx_{CONFIRM} to fulfill Requirement 2: First, c_{src} verifies that tx_{CLAIM} is included in blockchain *dest* and confirmed by enough blocks (see Step 3b of Protocol 2). Second, the contract verifies that the provided CLAIM transaction tx_{CLAIM} has invoked asset contract c_{dest} on blockchain *dest* (see Step 3(c)iii of Protocol 2). Third, c_{src} checks that tx_{CLAIM} has been successfully executed on blockchain *dest* (see Step 3(c)iv of Protocol 2). With these checks in place, Requirement 2 also holds for CONFIRM transactions.

It should not be possible to reuse the same CLAIM transaction tx_{CLAIM} on blockchain *src* to fulfill Requirement 3. Further, it should not be possible to use tx_{CLAIM} on any blockchain other than *src* for confirming transfer finalization. The first condition is ensured by adding each confirmed CLAIM transaction tx_{CLAIM} to a set T_{CLAIM} (see Step 3(c)vi of Protocol 2). Whenever c_{src} is invoked, it checks whether tx_{CLAIM} provided within tx_{CONFIRM} is included in T_{CLAIM} (see Step 3(c)i of Protocol 2). If so, the request is rejected. To ensure the second condition, the asset contract that has been invoked by the BURN transaction contained within tx_{CLAIM} is compared to c_{src} (see Step 3(c)ii of Protocol 2). If they match, c_{src} is the intended receiver of the confirmation.

The applied incentive structure ensures the confirmation of each transfer finalization on *src* (Requirement 4). Essentially, whenever some user *sender* submits a BURN transaction tx_{BURN} to blockchain *src*, it has to provide some stake Y. If user *sender* does not submit tx_{CONFIRM} before a timeout, any user can redeem Y by submitting tx_{CONFIRM} to *src*. Hence, there is an economic incentive to confirm transfer finalizations on *src*. Applying the same chain of reasoning for analyzing eventual finality reveals that as long as at least one altruistic user is participating, each transfer finalization on blockchain *dest* is eventually confirmed on blockchain *src*.

As the CONFIRM transaction tx_{CONFIRM} complies with Requirements 2 to 4, we consider the requirement for transfer confirmations fulfilled by Protocol 2.

4.3 Quantitative analysis

This section analyzes transfer costs and duration using the proof-of-concept implementation. We conduct cross-blockchain asset transfers between the public Ethereum test networks Rinkeby and Ropsten. Rinkeby and Ropsten are identical concerning the underlying EVM and the applied inter-block time (about 15 seconds). The main difference is the used consensus algorithm. Rinkeby uses Proof of Authority (PoA), while Ropsten uses Proof of Work (PoW). However, the execution of smart contracts is independent of the consensus algorithm. Therefore, evaluating transfers in one direction provides a sufficient estimation of transfer costs and duration.

Our evaluation consists of conducting 100 transfers of 1 ERC20 token from Rinkeby to Ropsten, i.e., BURN (CONFIRM) transactions are submitted to Rinkeby, whereas CLAIM transactions are executed on Ropsten. We repeat the experiment for both means of transaction inclusion verification, i.e., the oracle and the blockchain relay. Additionally, we also repeat the experiment without transaction inclusion verification to measure the overhead of the respective mechanisms. These mechanisms usually charge a fee to reward the relayers or the oracle nodes. For simplicity, we set the fee to zero. Further, as the protocol extension also includes the steps of Protocol 1, we only use the implementation of Protocol 2 for our experiment.

4.3.1 Transfer costs

For every performed transfer, we measure the gas consumption of all three transaction types, i.e., BURN, CONFIRM, and CLAIM. The obtained results (see Table 1) are outlined in Figs. 4 and 5. Note that the respective figures contain the gas consumption for the protocol and transaction inclusion verification.

First, we examine the gas consumption of the protocol while using the blockchain relay. Here, we get a total gas consumption of 450.02 kGas (standard deviation of 17.16 kGas) for Protocol 1, calculated as the sum of tx_{BURN} and tx_{CLAIM}. The total gas consumption of Protocol 2 is about 1018.09 kGas (standard deviation of 34.8 kGas) and includes the gas consumption of tx_{CONFIRM}, which accounts for the higher overall costs. With an exemplary exchange rate of about 3030.68 EUR per ETH (as of March 2022) and a gas price of 10 GWei, the transfer costs amount to 13.64 EUR for Protocol 1 and 30.86 EUR for Protocol 2.

After examining the gas consumption with the blockchain relay, we look at the gas consumption with the oracle. Here, we get a total gas consumption of 427.31 kGas (standard deviation of 8.75 kGas) for Protocol 1 and 976.35 kGas (standard deviation of 21.21 kGas) for Protocol 2. Using the same exchange rate and gas price leads to transfer costs of 12.95 EUR for Protocol 1 and 29.6 EUR for Protocol 2.

The execution of tx_{BURN} is the cheapest, followed by tx_{CLAIM} and by tx_{CONFIRM}. The differences can be explained by the different payloads of each transaction type. As the payload of tx_{BURN} does not consist of any other transaction and proof data, less data needs to be passed to and processed by the asset contract leading to lower gas consumption. On the contrary, tx_{CLAIM} contains tx_{BURN}, as well as proof data for tx_{BURN}, and tx_{CONFIRM} contains tx_{CLAIM} together with the corresponding proof data, which leads to higher gas consumption.

Table 1 Average gas consumption of the different operations in kGas

	Burn	Claim	Confirm	Protocol 1	Protocol 2
Relay	39.39 ± 0.00	410.62 ± 17.16	568.07 ± 28.52	450.02 ± 17.16	$1{,}018.09 \pm 34.80$
Oracle	39.39 ± 0.00	364.19 ± 8.75	525.30 ± 14.86	427.31 ± 8.75	976.35 ± 21.21

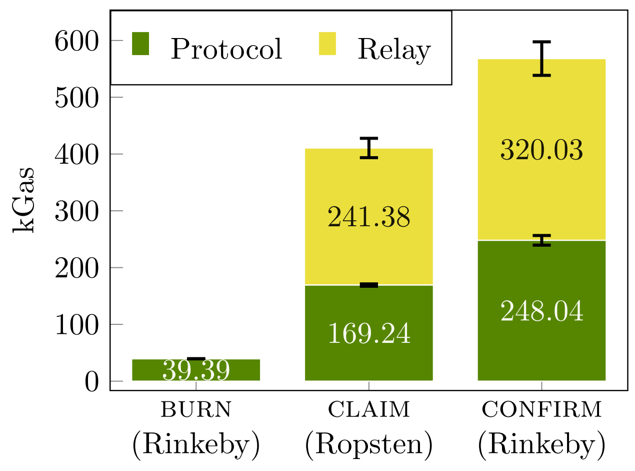

Fig. 4 Average transaction gas consumption with the blockchain relay

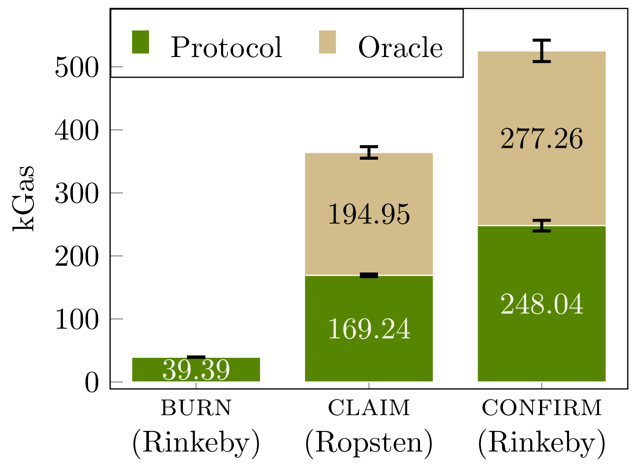

Fig. 5 Average transaction gas consumption with the oracle

The gas consumption of transaction inclusion verification depends on the concrete means of cross-blockchain communication. The oracle checks off-chain if the block header is part of the main chain, thus resulting in lower gas costs than the blockchain relay. Nevertheless, the client has to issue the requests to the oracle separately to provide the hashes of the block headers needed for the proofs. These requests additionally consume 23.4 kGas per request as there are no independent relayers that continuously provide the data. However, an oracle is much cheaper to operate than a blockchain relay since it does not cause any ongoing costs [25]. These operating costs heavily influence the fees charged by the relay or the oracle.

4.3.2 Transfer duration

In this subsection, we analyze the duration of asset transfers, which depends on a few core factors. The current network status and the selected gas price greatly influence the inclusion and confirmation time of a transaction. Therefore, we conduct the experiments at different times over a period of 3 days, while the gas price is estimated automatically to ensure the inclusion of a transaction with a high probability in the next block.

Further, after submitting a BURN transaction, the submitter may wait before posting the corresponding CLAIM transaction. The same applies to a CLAIM transaction, after which the submitter can wait to submit the following CONFIRM transaction. Hence, the overall transfer duration depends on the user executing the transfer. However, despite this uncertainty, we can measure the lowest possible transfer duration by submitting each transaction at the earliest possible time, i.e., as soon as the previous transaction is included in the blockchain and confirmed by enough blocks. In our experiment, we require each transaction on Rinkeby as well as on Ropsten to be confirmed by at least five succeeding blocks.

Finally, the transfer duration also depends on the used transaction inclusion verification mechanism. Therefore, we examine the influence of different transaction inclusion verification mechanisms by measuring the transfer duration using a blockchain relay and an oracle.

As described above, in our experiment, assets are transferred from Rinkeby to Ropsten. Therefore, BURN and CONFIRM transactions are submitted to Rinkeby while CLAIM transactions are submitted to Ropsten. Hence, durations for BURN and CONFIRM transactions were measured on Rinkeby, whereas for CLAIM transactions on Ropsten.

Essentially, CLAIM (CONFIRM) transactions are submitted to Ropsten (Rinkeby) as soon as the corresponding BURN (CLAIM) transactions are included and confirmed on Rinkeby (Ropsten). However, users need to wait until the relay running on Ropsten (Rinkeby) has been brought up to date or the oracle has submitted the data before they can submit the corresponding CLAIM (CONFIRM) transactions. Otherwise, the transactions would not be successful as the transaction inclusion verifier does not have enough information to verify the inclusion of transactions.

To this end, $\Delta_{\text{Inclusion}}$ denotes the duration from the moment a transaction is submitted to Rinkeby (Ropsten) until it is included in some block. $\Delta_{\text{Confirmation}}$ specifies the time it takes for an already included transaction to be confirmed by enough succeeding blocks. Further, Δ_{Relay} denotes the time it takes for the relay to collect enough information to verify the inclusion of transactions.

Alternatively, Δ_{Oracle} specifies the time the oracle takes to query the other blockchain and submit the result.

Figure 6 shows the average transfer duration for each transaction type for both protocols while leveraging a blockchain relay. With an average duration of 55 seconds (standard deviation of 19 seconds), BURN transactions achieve the lowest duration, followed by CONFIRM transactions (average duration of 134 seconds, standard deviation of 49 seconds), and CLAIM transactions (average duration of 210 seconds, standard deviation of 138 seconds). The total duration of Protocol 1 is calculated by summing up the duration of BURN and CLAIM transactions, while the total duration of Protocol 2 also contains the duration of CONFIRM transactions. This calculation leads to an average transfer duration of 265 seconds (standard deviation of 145 seconds) for Protocol 1 and an average duration of 400 seconds (standard deviation of 160 seconds) for Protocol 2. Transfers with Protocol 2 take longer since an additional transaction is required.

The results depicted in Fig. 7 also show the average duration for each transaction type for both protocols. However, an oracle instead of a blockchain relay is used for transaction inclusion verification. Here also, BURN transactions show the lowest transfer duration with 89 seconds (standard deviation of 3 seconds). CLAIM transactions consume on average 248 seconds (standard deviation of 137 seconds), and the execution of CONFIRM transactions averages 117 seconds (standard deviation of 12 seconds). The total duration for Protocol 1 averages 337 seconds (standard deviation of 138 seconds) and 454 seconds (s-tandard deviation of 137 seconds) for Protocol 2.

The results (see Table 2) initially indicate that using a relay leads to a shorter transfer duration. However, looking at the different durations for the inclusion and confirmation of the transactions and the time it takes for the relay and the oracle to include the data, we can discover different results. Further, we also have to consider the lock time, which was set to 0 for the experiments.

Although the inclusion and confirmation times are higher due to the network status, the time it takes for the oracle to provide the necessary data is less than the time it takes for the blockchain relay. This difference is due to the fact that the relay has to be kept up-to-date through the continuous submission of new block headers, which can lead to delays. With the oracle, on the other hand, only two transactions are needed to get the necessary data into the blockchain.

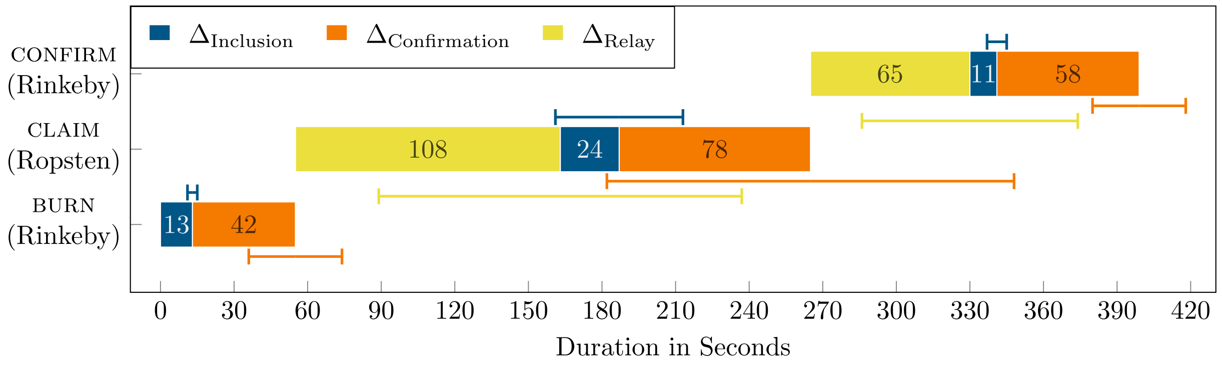

Fig. 6 Average transaction durations with the blockchain relay

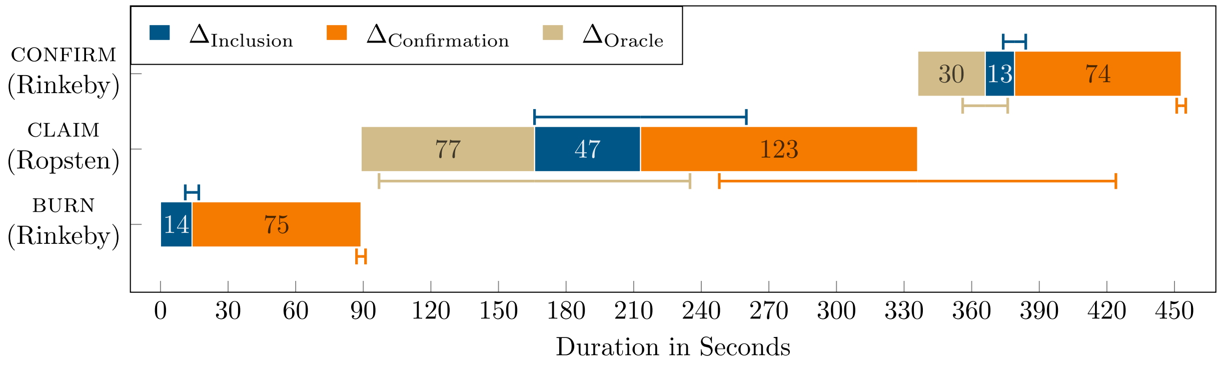

Fig. 7 Average transaction durations with the oracle

Table 2 Average durations for the different operations in seconds

		Relay	Oracle
Inclusion time	Burn	12.84 ± 2.25	14.27 ± 2.83
	Claim	24.26 ± 25.72	47.06 ± 46.87
	Confirm	11.36 ± 4.02	12.57 ± 4.84
Confirmation time	Burn	42.35 ± 19.20	75.16 ± 2.0
	Claim	77.68 ± 83.40	123.18 ± 87.68
	Confirm	57.84 ± 19.37	73.68 ± 2.42
Verification time	Burn	108.04 ± 73.5	77.71 ± 68.98
	Claim	65.14 ± 44.31	30.27 ± 10.30
Protocol 1		265.16 ± 145.05	337.37 ± 137.69
Protocol 2		399.49 ± 160.24	453.88 ± 136.87

The original implementation of ETH Relay foresees a lock time of 5 minutes to allow relayers to dispute invalid block headers. This mechanism causes another delay of at least 5 minutes for Protocol 1 and 10 minutes for Protocol 2. Therefore, using an oracle results in a considerably shorter transfer duration. Overall, the oracle is cheaper and faster, but the blockchain relay provides a higher degree of decentralization. Therefore, it depends on the use case which solution is more viable. In any case, the protocol can support both types of cross-blockchain communication.

5 Related work

Several solutions for cross-blockchain asset transfers have been proposed in the literature [12]. Evaluating the most important existing solutions against the requirements defined in Sect. 3.1 reveals that solutions generally fulfill Requirements 1 to 3. However, they lack with regards to decentralized finality (Requirement 4) or do not provide implementations of the proposed protocols. A summary of the different cross-blockchain asset transfer solutions is provided in Table 3.

In XClaim [35], cross-blockchain asset transfers are realized by first locking assets with a client called “vault” on a “backing” blockchain and reissuing the assets on another “issuing” blockchain. Locking of the assets on the backing blockchain is verified on the issuing blockchain via blockchain relays. However, the locked assets remain with the vault on the backing blockchain. While malicious vaults are penalized, transfer finality still depends on this single actor. In contrast, our protocol enables any client—whether directly involved in the transfer or not—to finalize transfers. Further, transfer confirmations are not foreseen by the XClaim protocol.

Metronome [36] implements cross-blockchain asset transfers for their MET token. Token holders can export MET from one blockchain and then import them on another blockchain via receipts where validators vote on the validity of receipts. While this can prevent illegal transfers, at the time of writing, Metronome cannot be considered decentralized since only authorized nodes can participate as validators. Further, Metronome does not provide concepts for transfer finality and confirmations.

In [37], the authors define a proof-of-burn protocol. The protocol consists of two functions: *GenBurnAddr* and *BurnVerify*. With the former, users can generate new burn addresses with an encoded tag, while the latter allows users to verify the tagged burn. For the verification of the proof-

Table 3 Comparison of different cross-blockchain asset transfer protocols

Reference	Ownership	No claim without burn	Double spend prevention	Decentralized finality	Transfer confirmation	Implementation
XCLAIM [35]	✔	✔	✔			✔
Metronome [36]	✔	✔	✔			✔
Karantias et al. [37]	✔	✔	✔			✔
Pillai et al. [38]	✔	✔	✔			✔
Liu et al. [39]	✔	✔	✔			✔
Kiayias and Zindros [40]	✔	✔				
Gazi et al. [41]	✔	✔				
Zendoo [43]	✔	✔	✔			
van Glabbeek et al. [44]	✔	✔		✔	✔	
DeXTT [46]	✔		✔	✔		✔
Protocol 1	✔	✔	✔	✔		✔
Protocol 2	✔	✔	✔	✔	✔	✔

of-burn, the protocol foresees the usage of Noninteractive Proofs of Proof of Work (NiPoPoWs), oracles, or direct observation of the source blockchain by the miners. In the course of this, the authors show how users can acquire target cryptocurrency (e.g., ERC-20 tokens) by burning the source cryptocurrency. However, the protocol does not provide decentralized finality and transfer confirmations.

Pillai et al. [38] propose a burn-to-claim asset transfer protocol. Also, similar to the protocol presented in Sect. 3, it comprises a burn operation on the source blockchain and a claim operation on the target blockchain. However, to access the transaction on the source blockchain to verify the correct execution of the burn transaction, the protocol requires nodes that can mine in multiple networks and act as gateways. Furthermore, the protocol does not take decentralized finality and transfer confirmations into account.

In [39], the authors present AucSwap, a cross-blockchain token transfer protocol modeled as an auction process. More specifically, the authors leverage the Vickery auction process and atomic swap technology to enable cross-blockchain asset transfers. The protocol consists of two parts: bidding collection and asset exchange. Initially, the seller broadcasts the bidding information and waits for the buyers to return their bids. After receiving all bids, the seller provides the buyers with a list of all bids. Having this information, the buyers can determine who won the auction. After that, the parties use Hash Time-Locked Contracts (HTLCs) to perform the actual exchange of the assets. Compared to our protocol, this solution has more similarities with asset exchanges than asset transfers.

The authors of [40, 41] propose approaches for realizing cross-blockchain transfers between PoW and Proof of Stake (PoS) blockchains, respectively. While [40] verifies transaction inclusions via NiPoPoWs [41, 42] enables transaction inclusion verifications via a novel cryptographic construction called ad-hoc threshold multisignatures (ATMS) ATMS. As such, NiPoPoW and ATMS are used to prove events (e.g., BURN transactions) that occurred on the source blockchain to the destination blockchain. While this satisfies Requirements 1 and 2, Requirements 3 and4 are generally not covered by the protocol. Further, NiPoPoWs currently cannot be implemented in existing PoW blockchains like Bitcoin or Ethereum without introducing a so-called velvet fork, which requires adoption from at least a subset of miners.

Similarly, Zendoo [43] provides a protocol for cross-blockchain asset transfers focussing on zero-knowledge proofs as a method for transaction inclusion verification. However, requirements such as transfer finality are not discussed. Further, the protocol relies on a special sidechain construction and can thus not be easily implemented on existing blockchains.

An approach that takes transfer finality into account is presented by van Glabbeck et al. [44]. The paper proposes a generic protocol for payments across blockchains similar to how multi-hop payment channels operate [45]. The work has a strong focus on finality. Requirements such as double spend prevention are mentioned though not further specified. Also, the protocol has not been implemented and evaluated yet. It does not become clear whether the protocol allows cross-blockchain asset transfers as defined in Sect. 3.1, or only value transfers similar to atomic swaps.

An alternative approach for cross-blockchain asset transfers is introduced in DeXTT [46]. DeXTT describes an asset that can exist on different blockchains at the same time. However, users cannot keep different denominations of the asset on each blockchain. Rather, balances are synchronized across all participating blockchains. While the synchronization process itself is decentralized, the protocol uses a concept called claim-first transactions, where assets are claimed on the other blockchains before being burned on the blockchain on which the transfer was initiated. This clearly violates Requirement 2. Transfer confirmations (Requirement 5) are also not foreseen.

Other works [15, 32, 47] focus on the transfer of value across different blockchains. However, these solutions rather focus on atomic swaps where two different assets are exchanged and do not constitute true cross-blockchain asset transfers as defined by our requirements in Sect. 3.1.

Finally, projects such as Polkadot [48] and Cosmos [49] also aim for generic cross-blockchain interactions. Polkadot implements a Cross-chain Message Passing (XCMP) protocol that enables two separate parachains to communicate with each other. To accomplish this, it makes use of a simple queuing mechanism based on Merkle trees. Cosmos implements the Interblockchain Communication (IBC) protocol proposed in [50]. The IBC protocol is inspired by the TCP/IP protocol and enables the communication between separate ledgers which implement the same interface. Multiple ledgers can establish a connection with each other to create channels over which packages can be transmitted to modules (e.g., smart contracts) on the other ledger. Both protocols have already been implemented by the respective projects. While cross-blockchain asset transfers are mentioned as example use cases, the documentation does not mention specifics on how these transfers are implemented. Further, these projects aim to provide interoperability primarily between specialized blockchains that adhere to certain structures and consensus protocols.

To the best of our knowledge, this work is the first to provide requirements, a specification, and a proof-of-concept implementation of a cross-blockchain asset transfer protocol that also takes transfer finality and transfer confirmations into account.

6 Conclusion

Decentralized cross-blockchain asset transfers are one way to provide interoperability between blockchains. In particular, they prevent vendor lock-ins by allowing blockchain assets to be moved away from the blockchains on which they were originally issued in a completely decentralized way. While a number of solutions for enabling cross-blockchain asset transfers have been proposed, these solutions often focus on specific assets and neglect the fundamental functionality that cross-blockchain asset transfers should offer. In this work, we defined general requirements and specifications for cross-blockchain asset transfer protocols. Providing a proof-of-concept implementation of the proposed protocol, we have shown that requirements such as decentralized finality and transfer confirmations can be fulfilled.

In future work, we will investigate how the concepts of this work can be extended to provide interoperability beyond cross-blockchain asset transfers, e.g., generic message passing between blockchains.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association is gratefully acknowledged.

Author contributions All authors contributed to the conception and design of the protocol. The original paper was written by MS and PF. The first draft of the manuscript was written by MS and is an extended and revised version of the original paper. Data collection and analysis were performed by MS and MK. SS provided supervision, as well as reviewed, and edited this work. All authors read and approved the final manuscript.

Funding Open Access funding enabled and organized by Projekt DEAL. This work was supported by the Austrian Federal Ministry for Digital and Economic Affairs, the National Foundation for Research, Technology and Development as well as the Christian Doppler Research Association.

Data availability The datasets generated during and/or analysed during the current work are available in the Zenodo repository, https://zenodo.org/record/6394523#.YkMj5H9BzJU

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Nakamoto, S.: Bitcoin: A Peer-to-Peer Electronic Cash System. White Paper. Accessed 29 March 2022 (2008) https://bitcoin.org/bitcoin.pdf
2. Tian, F.: An agri-food supply chain traceability system for China based on RFID & blockchain technology. In: 2016 13th International Conference on Service Systems and Service Management (ICSSSM), pp. 1– 6 (2016). IEEE
3. Mettler, M.: Blockchain technology in healthcare: the revolution starts here. In: 2016 IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 1– 3 (2016). IEEE
4. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. Future Gener. Comput. Syst. **107**, 816–831 (2020)
5. Makani, S., Pittala, R., Alsayed, E., Aloqaily, M., Jararweh, Y.: A survey of blockchain applications in sustainable and smart cities. Clust. Comput. **15**, 1–22 (2022)
6. Tseng, L., Yao, X., Otoum, S., Aloqaily, M., Jararweh, Y.: Blockchain-based database in an IoT environment: challenges, opportunities, and analysis. Clust. Comput. **23**(3), 2151–2165 (2020)
7. Al Ridhawi, I., Aloqaily, M., Karray, F.: Intelligent blockchain-enabled communication and services: solutions for moving internet of things devices. IEEE Robot. Automat. Mag. **29**, 10–20 (2022)
8. Berdik, D., Otoum, S., Schmidt, N., Porter, D., Jararweh, Y.: A survey on blockchain for information systems management and security. Inform. Process. Manage. **58**(1), 102329 (2021)
9. Schulte, S., Sigwart, M., Frauenthaler, P., Borkowski, M.: Towards blockchain interoperability. In: Business Process Management: Blockchain and Central and Eastern Europe Forum. LNBIP, vol. 361, pp. 1– 8 (2019)
10. Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E., Moreno-Sánchez, P., Kiayias, A., Knottenbelt, W.J.: SoK: Communication across distributed ledgers. In: International Conference on Financial Cryptography and Data Security. LNCS, vol. 12675, pp. 3–36 (2021)
11. Cai, W., Wang, Z., Ernst, J.B., Hong, Z., Feng, C., Leung, V.C.M.: Decentralized applications: the blockchain-empowered software system. IEEE Access **6**, 53019–53033 (2018)
12. Belchior, R., Vasconcelos, A., Guerreiro, S., Correia, M.: A survey on blockchain interoperability: past, present, and future trends. ACM Comput. Surv. **54**(8), 1–41 (2021)
13. Back, A., Corallo, M., Dashjr, L., Friedenbach, M., Maxwell, G., Miller, A., Poelstra, A., Timón, J., Wuille, P.: Enabling blockchain innovations with pegged sidechains. http://kevinriggen.com/files/sidechains.pdf. White Paper. Accessed 29 March 2022 (2014)
14. Poon, J., Buterin, V.: Plasma: scalable autonomous smart contracts. https://www.plasma.io/plasma.pdf. White Paper. Accessed 29 March 2022 (2017)
15. Herlihy, M.: Atomic cross-chain swaps. In: 2018 ACM Symposium on Principles of Distributed Computing (PODC), pp. 245–254 (2018). ACM
16. Sigwart, M., Frauenthaler, P., Spanring, C., Sober, M., Schulte, S.: Decentralized cross-blockchain asset transfers. In: 2021 Third International Conference on Blockchain Computing and Applications (BCCA), pp. 34–41 (2021). IEEE

17. Pillai, B., Biswas, K., Muthukkumarasamy, V.: Blockchain interoperable digital objects. In: International Conference on Blockchain, pp. 80–94 (2019). Springer
18. Cuffe, P.: The role of the ERC-20 token standard in a financial revolution: the case of Initial Coin Offerings. In: IEC-IEEE-KATS Academic Challenge (2018)
19. Casale-Brunet, S., Ribeca, P., Doyle, P., Mattavelli, M.: Networks of Ethereum Non-Fungible Tokens: A graph-based analysis of the ERC-721 ecosystem. In: 2021 IEEE International Conference on Blockchain, pp. 188–195 (2021). IEEE
20. Heiss, J., Eberhardt, J., Tai, S.: From oracles to trustworthy data on-chaining systems. In: 2019 IEEE International Conference on Blockchain, pp. 496–503 (2019). IEEE
21. Peterson, J., Krug, J., Zoltu, M., Williams, A.K., Alexander, S.: Augur: a decentralized oracle and prediction market platform. arXiv:1501.01042 (2015)
22. Breidenbach, L., Cachin, C., Chan, B., Coventry, A., Ellis, S., Juels, A., Koushanfar, F., Miller, A., Magauran, B., Moroz, D., et al.: Chainlink 2.0: next steps in the evolution of decentralized oracle networks. Accessed 29 March 2022 (2021) https://research.chain.link/whitepaper-v2.pdf?_ga=2.244768454.295607443.1648540372-1480369942.164639185
23. Adler, J., Berryhill, R., Veneris, A., Poulos, Z., Veira, N., Kastania, A.: Astraea: A decentralized blockchain oracle. In: 2018 IEEE International Conference on Blockchain, pp. 1145–1152 (2018). IEEE
24. Kamiya, R.: Shintaku: An end-to-end-decentralized general-purpose blockchain oracle system. https://gitlab.com/shintaku-group/paper/blob/master/shintaku.pdf. Accessed 29 March 2022 (2019)
25. Sober, M., Scaffino, G., Spanring, C., Schulte, S.: A voting-based blockchain interoperability oracle. In: 2021 IEEE International Conference on Blockchain, pp. 160–169 (2021). IEEE
26. Buterin, V.: Chain interoperability. https://www.r3.com/wp-content/uploads/2017/06/chain_interoperability_r3.pdf. Accessed 29 March 2022 (2016)
27. Narayanan, A., Bonneau, J., Felten, E., Miller, A., Goldfeder, S.: Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction. Princeton University Press, Princeton (2016)
28. Frauenthaler, P., Sigwart, M., Spanring, C., Sober, M., Schulte, S.: ETH relay: a cost-efficient relay for ethereum-based blockchains. In: 2020 IEEE International Conference on Blockchain, pp. 204–213 (2020). IEEE
29. Di Angelo, M., Salzer, G.: A survey of tools for analyzing ethereum smart contracts. In: 2019 IEEE International Conference on Decentralized Applications and Infrastructures (DAPPCON), pp. 69–78 (2019). IEEE
30. Wood, G.: Ethereum: a secure decentralised generalised transaction ledger. Accessed 29 March 2022 (2014) https://ethereum.github.io/yellowpaper/paper.pdf
31. Aiyer, A.S., Alvisi, L., Clement, A., Dahlin, M., Martin, J.-P., Porth, C.: BAR fault tolerance for cooperative services. In: 20th ACM Symposium on Operating Systems Principles (SOSP), pp. 45–58 (2005). ACM
32. Herlihy, M., Liskov, B., Shrira, L.: Cross-chain deals and adversarial commerce. VLDB J. **25**, 1–19 (2021)
33. Judmayer, A., Stifter, N., Zamyatin, A., Tsabary, I., Eyal, I., Gazi, P., Meiklejohn, S., Weippl, E.R.: Pay-to-win: incentive attacks on proof-of-work cryptocurrencies. IACR Cryptology ePrint Archive (2019)
34. Ford, B., Böhme, R.: Rationality is self-defeating in permissionless systems. (2019)arXiv:1908.03999
35. Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., Knottenbelt, W.: XCLAIM: trustless, interoperable, cryptocurrency-backed assets. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 193–210 (2019). IEEE
36. Autonomous Software: Metronome: Owner's Manual. Version 0.99 (Last Updated 2019-08-15). Accessed 29 March 2022 (2018). https://github.com/autonomoussoftware/documentation/blob/master/owners_manual/owners_manual.md
37. Karantias, K., Kiayias, A., Zindros, D.: Proof-of-burn. In: International Conference on Financial Cryptography and Data Security, pp. 523–540 (2020). Springer
38. Pillai, B., Biswas, K., Hóu, Z., Muthukkumarasamy, V.: The burn-to-claim cross-blockchain asset transfer protocol. In: 2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS), pp. 119–124 (2020). IEEE
39. Liu, W., Wu, H., Meng, T., Wang, R., Wang, Y., Xu, C.-Z.: AucSwap: a vickrey auction modeled decentralized cross-blockchain asset transfer protocol. J. Syst. Archit. **117**, 102102 (2021)
40. Kiayias, A., Zindros, D.: Proof-of-work sidechains. In: International Conference on Financial Cryptography and Data Security, pp. 21–34 (2019). Springer
41. Gaži, P., Kiayias, A., Zindros, D.: Proof-of-stake sidechains. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 139–156 (2019). IEEE
42. Kiayias, A., Miller, A., Zindros, D.: Non-interactive Proofs of Proof-of-Work. In: 24th International Conference on Financial Cryptography and Data Security—Revised Selected Papers. LNCS, vol. 12059, pp. 505–522 (2020). Springer
43. Garoffolo, A., Kaidalov, D., Oliynykov, R.: Zendoo: a zk-snark verifiable cross-chain transfer protocol enabling decoupled and decentralized sidechains. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), pp. 1257–1262 (2020). IEEE
44. van Glabbeek, R., Gramoli, V., Tholoniat, P.: Cross-chain payment protocols with success guarantees. arXiv preprint arXiv:1912.04513 (2019)
45. Malavolta, G., Moreno-Sanchez, P., Kate, A., Maffei, M., Ravi, S.: Concurrency and privacy with payment-channel networks. In: 2017 ACM SIGSAC Conference on Computer and Communications Security, pp. 455–471 (2017). ACM
46. Borkowski, M., Sigwart, M., Frauenthaler, P., Hukkinen, T., Schulte, S.: DeXTT: deterministic cross-blockchain Token transfers. IEEE Access **7**, 111030–111042 (2019)
47. Thomas, S., Schwartz, E.: A protocol for interledger payments. https://interledger.org/interledger.pdf. Accessed 29 March 2022 (2015)
48. Wood, G.: Polkadot: vision for a heterogeneous multi-chain framework. https://polkadot.network/PolkaDotPaper.pdf. Accessed 29 March 2022 (2016)
49. Kwon, J., Buchman, E.: Cosmos whitepaper: a network of distributed ledgers. https://cosmos.network/resources/whitepaper. Accessed 29 March 2022 (2020)
50. Goes, C.: The interblockchain communication protocol: an overview. arXiv preprint arXiv:2006.15918 (2020)

Michael Sober received his master's degree in Software Engineering & Internet Computing from TU Wien in 2020. He is a research assistant and Ph.D. student at the Institute for Data Engineering at TU Hamburg and working on blockchain interoperability solutions in the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Christof Spanring is currently the Chief Technology Officer of Bonfire Group BV. He was the lead blockchain researcher at Pantos GmbH and worked on blockchain interoperability solutions, contributing to the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Marten Sigwart received his master's degree in Computer Science from TU Berlin, Germany, in 2018. He was a project assistant in the TAST Research Project at TU Wien, working on blockchain interoperability solutions, and is now a full-stack software engineer at Datawrapper, Berlin.

Max Kobelt is currently pursuing his bachelor's degree at TU Hamburg. He is also a student assistant in the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT).

Philipp Frauenthaler received his master's degree in Software Engineering & Internet Computing from TU Wien in 2018, where he was also a project assistant in the Distributed Systems Group. Before joining the Distributed Systems Group in February 2019, he worked for five years as a software engineer, developing enterprise software for insurances. He is now a senior software engineer at the Austrian Federal Computing Center in Vienna, Austria.

Stefan Schulte heads the Institute for Data Engineering at Hamburg University of Technology, Germany, and leads the Christian Doppler Laboratory Blockchain Technologies for the Internet of Things (CDL-BOT). His research interests include the areas of data stream processing, the Internet of Things, and distributed ledger technologies. Findings from his research have been published in more than 100 refereed scholarly publications.