

Burn-to-Claim: an asset transfer protocol for blockchain interoperability

Babu Pillai^a, Kamanashis Biswas^b, Zhé Hóu^a, Vallipuram Muthukkumarasamy^a

^a*School of ICT, Griffith University, Gold Coast, Australia*

^b*Australian Catholic University, Brisbane, Australia*

Abstract

The future of multi-blockchain architecture depends on the emergence of new protocols that enable consensus between trustless cross-blockchain participants. However, interoperability between blockchains remains a research challenge. The existing interoperability approaches provide integration through solutions using a middleware system, making it difficult to gain confidence in the security and correctness of the process. A cross-blockchain protocol must provide a self-verifiable state proof that encodes trust in the transfer process to guarantee consensus. Inspired by the burn-address concept, we propose a *Burn-to-Claim* cross-blockchain protocol to exchange assets between two networks. The proposed protocol transfers assets from one blockchain system to another so that the asset is burned from the source blockchain and recreated on the destination blockchain. Our protocol makes use of digital signatures, hash-time-locks and integration mechanisms to perform cross-blockchain transactions in a distributed manner. We theoretically prove that the proposed cross-blockchain protocol transfers assets securely and correctly. In addition, the experimental results demonstrate the feasibility of the *Burn-to-Claim* protocol when used in an application environment.

Keywords: blockchain, interoperability, cross-chain protocol, multi-chain asset transfer, multi-blockchain architecture

1. Introduction

Blockchain technology offers an immutable, decentralized, and transparent mechanism for transaction processing. It has the potential to improve business processes and transaction integrity for enterprises. Interestingly, beyond its role as a mechanism for exchanging values within a particular network, one might reasonably assume that a blockchain system should be able to transfer assets between networks as well. However, the current architecture of this technology effectively limits transactions to a single network. As a result, a blockchain application cannot uniformly use multiple networks and obtain a composition of their guarantees (Belchior et al., 2020). An underlying reason for this issue is that each blockchain network has its own state assumptions (Borkowski et al., 2018a), and without restrictions, one network cannot verify the information on a different network. Therefore, currently, seamless transactions between two blockchain networks are difficult (Lafourcade and Lombard-Platet, 2020).

The concept of interoperability allows two blockchains to function together. A blockchain is interoperable with another blockchain if it can provide a state-guarantee for cross-blockchain transactions to the other network, and vice versa (Belchior et al., 2020). We formalise a seamless transaction of interoperability by extending the previous works of (Lafourcade and Lombard-Platet, 2020; Pillai et al., 2020b).

Interoperability brings the possibility of cross-blockchain communication and the ability to share, reveal, and access information within the trust model of a blockchain. Our

vision of next generation blockchains is that they should have an *internal* or *built-in mechanism* for composability, the lack of which makes it difficult for applications to commit transactions across multiple networks. **Currently, the blockchain landscape has different types of networks, such as permissionless (Bitcoin and Ethereum) and permissioned (Hyperledger fabric, Besu and Sawtooth) networks. Such a distinction exists because some applications require different properties. For instance, Bitcoin offers security through a higher number of participating nodes which results in low performance. In contrast, Hyperledger fabric offers better performance with a fixed set of participating nodes, implying lower security.**

Decentralised cross-blockchain transfer protocols should allow participants to fully utilise the existing variety of blockchains, instead of being locked to a single blockchain type. These cross-blockchain transaction will serves as a cornerstone for the next-generation of blockchain-based systems. Cross-blockchain integration would enable interoperability among distinct and potentially heterogeneous blockchains (Hardjono et al., 2018).

With the current protocols of blockchain systems, it is difficult to have direct interoperability between systems (Borkowski et al., 2018b). There is no method in the system to provide a cross-blockchain value transfer; therefore, external third-party services are the preferred solutions (Belchior et al., 2020). However, such methods come with a trade-off reduction in decentralisation, which eventually leads to security issues on the system (Wang et al., 2019b). Moreover, current blockchain systems can-

not recognise or verify any such process carried out by a third-party provider. To address these issues, we propose to use a carefully designed internal process that is integrated with the core blockchain protocol to carry out asset transfer transactions and facilitate interoperability between systems. That way, the system and its users are in control and aware of the assets being transferred.

This paper introduces the *Burn-to-Claim* protocol — a built-in method to address cross-blockchain interoperability. This protocol is built upon our preliminary work on cross-blockchain transactions and interoperability (Pillai et al., 2020a,b). It consists of two components: an *exitTransaction* function to generate a self-verifiable proof that the transaction is committed in the source network and an *entryTransaction* function to verify the validity of the proof in order to recreate the asset in the destination network. A key advantage of this built-in method is that it is not system/application-specific. Unlike existing approaches which rely on external middleware mechanisms, our protocol does not require a new crypto-asset or trusted third party. Instead, it implements a ‘universal mechanism’ for all applications and tries to enforce the same integrity requirements across different connected blockchain networks. The proposed protocol can serve as a framework for cross-chain asset transfer, where applications can construct a network based on its trust assumptions and guarantees, yet all these heterogeneous networks share the same cross-blockchain transfer protocol.

A summary of our contributions is as follows:

- **Protocol:** We propose a new protocol, i.e., an agreed set of processes, for cross-blockchain asset transfer where transactions are performed in a decentralised and trustworthy manner.
- **Construction:** We propose a novel and simple construction that is flexible and can be adopted for digital asset transfer among blockchain networks without violating the blockchain technology’s key characteristics.
- **Analysis:** We perform a theoretical analysis to demonstrate that the properties of the proposed protocol are correct, secure and fair.
- **Experiment:** We provide a reference implementation of the proposed protocol in test networks to prove the concept.

1.1. Overview of the proposed protocol

Let us consider a common example of a bank transaction. Typically it transfers money from one bank account to another and is initiated by an account holder. Before the transfer takes place, the users should establish certain knowledge such as account details. The sender proves her credentials, and, after validation, the sender can initiate the transfer. At the same time, the transfer-amount gets

blocked by the sender’s bank from further use (preventing double-spending). Here the system is generating a proof that the money exists and has been locked; thereby, the intermediaries/ counterparties can rely on it and act. In a banking scenario, we deal with money as a commonly accepted asset, but the form of the asset can be different in other applications.

In the blockchain setting, a cross-blockchain protocol interacts with two blockchain networks, namely the source and the destination networks which have different ledger data. Therefore, we need to create a universal standalone proof of transfer receipt that can be stored in the ledger, not as a state typically verified by state-validation system such as Merkle tree. Our cross-blockchain protocol has three stages: **prepare**, **commit** and **execute**. At the prepare stage, the users agree and establish the transfer parameters. We assume this process occurs out-of-band through a secure channel. At the commit stage, the source network generates a time-locked and publicly verifiable receipt as a proof-of-transfer using *exitTransaction* function. We define this proof-of-transfer as a committed cross-blockchain transaction.

Using the *time-lock* mechanism, the transaction locks the asset in the source network for a pre-defined/ certain period of time. Finally, at the execute stage, upon presenting the proof-of-transfer, the destination network nodes verify the validity and correctness of the proof-of-transfer and execute the exchange through an *entryTransaction* function. We assume that the recipient’s network nodes can validate the proof-of-transfer through gateway nodes. The recipient can claim the transaction if it is within the conditional time-bound, and then the proof-of-transfer has been validated. In case of an unsuccessful transaction, which means that the recipient did not claim the asset within the time-lock period, the sender is able to reclaim the asset.

In summary, we present a protocol for any blockchain based systems to ensure the integrity of cross-chain transfers. To illustrate the concept, we use an Ethereum-based blockchain setting with smart contracts in this paper. However, theoretically our protocol can be incorporated in other DLT-based systems through the modification of the gateway nodes’ functionalities and mining process. Technically, interoperability in any DLT based systems aims to ensure the integrity of cross-chain data. Therefore, our proposed protocol can serve as a general interoperability solution for asset transfer under certain assumptions listed below. *Burn-to-Claim* is flexible in terms of the choice of public and private networks; however, the performance and security guarantees depend on the properties of the underlying blockchain technologies and consensus protocols.

1.2. General assumptions

This section lists all general assumptions considered in the design of the proposed cross-blockchain protocol. How-

ever, the network specific assumptions and requirements are discussed in Section 4.3.1.

- The underlying blockchain networks are permissionless and secure with a concept of transaction finality within finite time, after which the transactions cannot be rolled back (Belotti et al., 2019).
- The blockchain networks involved have agreed to participate in the cross-blockchain transaction processes and are able to process digital asset transaction within their networks if the other party presents a valid proof. The exact asset and transfer policy depends on the application and must be agreed on by both parties.
- Regarding mining nodes, we envision the concept of merge mining where network nodes are motivated to take part in the respective proof-mechanism (mining process) on both chains (refer to network assumptions (Section 4.3.1) for details).
- We do not consider consensus aspects of the participating networks, we assume network nodes comply with the consensus requirement of its own network.
- There exist nodes that are functionally capable of acting as gateway nodes. There is at least one gateway node in each network capable of replicating cross-blockchain data between networks.
- A transaction output carries a single output which corresponds to a single asset, and the networks involved recognise and form a common understanding of the assets they are exchanging (Pillai et al., 2019).

Under these assumptions, once a transaction is broadcast, the network nodes verify the transaction and include it in a block. A blockchain system tends to synchronise using a protocol in which the network nodes constantly try to produce new blocks and broadcast their achievements to the entire network. For example, this behaviour is motivated by mining rewards in cryptocurrency applications. We also assume that there is an appropriate incentive mechanism to support participation by nodes.

2. Related Work

In this section, we briefly discuss the related work on developing and formalising protocols for cross-blockchain asset transfers.

Blockchain interoperability is identified as one of the primary business requirements to address. The existing approaches focus on asset transfer between blockchains using trustless exchange protocols, including side-chains (Gazi et al., 2019; Kiayias and Zindros, 2019), atomic cross-blockchain swaps (Herlihy, 2018), and cryptocurrency backed assets (Zamyatin et al., 2019b). However, those

solutions have specific constraints. For example, the side-chain protocols can only support transactions between parent and child chains, the atomic swaps protocol requires interactive monitoring during execution, and finally the cryptocurrency-backed asset transfer protocol assumes a set of reliable smart contracts to control the underlying process.

There are a number of efforts being made to address interoperability using/through different integration architecture (Borkowski et al., 2019; Jiang et al., 2019; Sigwart et al., 2020). Although they differ in architectures, most of them follow the approach of using the source chain to generate a proof-of-transfer and the destination chain to recreate the asset based on it. The cross-blockchain proof-of-transfer are either a smart contract (Sigwart et al., 2020) or a notary scheme (Jiang et al., 2019). Generally, these approaches must make significant assumptions. For example, a smart contract needs to hold the transfer proof and the verifying system must trust the communication with the smart contract. In a notary scheme, a pre-defined set of validators has temporary control over assets on the transferring chain. Some other projects (Kan et al., 2018; Wood, 2016) focused on building a completely new ecosystem for interoperable blockchains rather than enabling interoperability between existing blockchain systems.

In Sigwart et al. (2020), assets are transferred through a burning process on one chain and uses the destruction of the data on the sender system as proof-of-transfer to trigger the recreation of the data on the receiver system. Their scheme proposes a relay mechanism to replicate and cross-verify the data across networks. Once BURN is confirmed, any participating node can initiate the CLAIM on the recipient network to transfer the asset. The motivation of participants who mine the blockchain is the transaction fee. On an updated version of their protocol, they include a CONFIRM process where, once the claim is confirmed by the destination chain, a confirmation transaction is initiated in the source chain. For that, the burn transaction locks some incentive which will be awarded to the nodes who successfully submit a claim transaction.

In Borkowski et al. (2019), tokens are transferable between networks through a coordination process of account balancing within a shared wallet. In their scheme, a wallet associates with multiple networks and records the balance of token across the networks. When transferring a token, the sender's wallet initiates the transfer, and the recipient wallet must acknowledge and signing it. Once both the sender and the recipient sign the transaction, the recipient makes a CLAIM. Then the network participants witness the request and make a CONTEST transaction within a given time period. At the end of the time period, through a selection process, the network assigns witness awards to the miner who proposed the transaction.

In Kan et al. (2018), a router network helps to create connections between different networks. The router network consists of nodes within the connected networks. A minimum requirement to join this ecosystem of con-

Table 1: Summary of notations used in this paper

Notations	Description
C	Blockchain
$\mathbb{B}, b, \mathbb{T}^{b\dagger}$	List of blocks, single block and list of transactions in a previous block respectively
Q, Q'	Current blockchain state and new state respectively
N_1, N_2	Blockchain network 1 and network 2 respectively
Π	State transition function
v	Asset
K_p, K_r	Public key and private key respectively.
K_{adr}, β	Blockchain address and Burn address.
σ	Digital signature
$Tx, Tx_t, Tx_e, Tx_r, Tx^\dagger$	Transaction, exit, entry, reclaim and previous transaction respectively
$\gamma, H(\gamma), \bar{\gamma}, \hat{\gamma}$	Secret code, hash of secret code, encrypted secret code with private and public keys respectively
t, t_{lock}	Time and time lock respectively

nected networks is to make one node into a router node. The router node then routes information to the connected chains. For example, when the chain A wants to make a transaction with the chain B , chain A 's router establishes a connection with chain B 's router and sends the transaction. The process of cross-chain transactions happens in three stages of prepare, pre-commit and commit. In the prepare stage, the transaction is packaged and forwarded to the destination chain through the router node. The destination chain issues a pre-commit, which then makes the final commit on both chains.

In our protocol, the sender who wishes to transfer the asset must present a proof that the asset is locked. To achieve this, we adopt the proof-of-burn protocol (Karantias et al., 2019; Stewart, 2012), which presents a mechanism where the sender transfers the asset to a non-spendable burn-address which is inaccessible to anybody and the sender can then present that transaction as a proof for the locked asset. Importantly proof-of-burn protocol is a more energy-saving alternative to the proof-of-work protocol. In proof-of-burn based systems, for security guarantees, digital assets are intentionally ‘‘burned’’ and the process does not require powerful mining resources.

Based on these existing works, a number of slightly different approaches have been proposed for cross-blockchain interactions. Generally, due to the distributed nature of blockchain, it is impossible to validate data across different networks without a middleware mechanism. However, providing state composition guarantees for a cross-blockchain transaction is challenging. To avoid these complications, our protocol utilises a ‘burn’ concept that can provide a cryptographic guaranty for transactions (Karantias et al., 2019). In Lafourcade and Lombard-Platet (2020) the ‘burn’ concept is described as ‘bin’ address, whereas the ‘burn’ concept presented in Sigwart et al. (2020) is complementary to techniques we used in our approach. Our protocol utilises the burn concept in combination with a hash-time-lock to build a cross-blockchain system.

3. Preliminaries

There are various definitions for blockchain operations in the literature based on the type of applications, but they usually do not cover all the assumptions required to define our protocol. Therefore, in this section, we define the terminology and general features of blockchains required to explain the aspects of cross-blockchain interoperability. We list the notations used in this paper in Table 1.

A blockchain implementation can be seen as a series of concurrent mutations of its global state through a state transition function that takes blocks as input.

Definition 1 (Block). A block, written as b , is a tuple of three elements where the first element is the Merkle root of a list of transactions, and we denote the list of transactions in the block by \mathbb{T}^b ; the second element is the hash reference to a previous block; and the third element is the metadata including some auxiliary verification data.

A blockchain is a chain of blocks where each block contains several transactions, a validation proof and the hash of the previous block.

Definition 2 (Blockchain). A blockchain, written as C , is a tuple (G, \mathbb{B}) where G is a genesis state and $\mathbb{B} = [b_1, b_2, \dots, b_n]$ is a list of blocks.

The blocks are cryptographically linked such that $H(b_1)$ is included in b_2 and $H(b_2)$ is included in $b_3, \dots, H(b_{n-1})$ is included in b_n .

Definition 3 (State). A state, written as Q , is the set of transactions and the data associated with on-chain objects in a blockchain at a certain time t .

A state transition function denoted by Π takes the current state Q and a block b to produce new state Q' , we define this operation as $\Pi(Q, b) = Q'$. In the context of a blockchain, a state is valid if it is accepted by the majority of nodes (Nakamoto, 2008). A block b is valid in a state Q if $\mathbb{T}^b \subseteq Q$ and Q is valid in a blockchain. Here we loosely use set operations on lists when the context is understood. We assume that the state Q is monotonic once it is settled on a blockchain C with height k . It is required

that the state Q only depends on the stable portion of the blockchain C , i.e., the blocks that are buried under k (recommended block height for security (Sompolinsky and Zohar, 2016)) subsequent blocks. This means that once a transaction Tx is settled under a state Q of a blockchain C , it will not change due to a blockchain reorganization.

Definition 4 (Blockchain address). A blockchain *address* is a hashed value generated by Algorithm 1 from a key pair (K_p, K_r) .

In a blockchain network, a user is defined by a unique *address* (K_{adr}) . Let K_{adr}^S and K_{adr}^R be the sender and the recipient’s address respectively where the superscripts S and R represent the sender and the recipient respectively. The address is constructed from the user’s public key (K_p) as shown in Algorithm 1. The private key (K_r) is used to generate the digital signature (Wang et al., 2019a). The digital signature is generated through a function, $sign(M, K_r)$, that takes a message (M) and the private key of the sender (K_r) as inputs to create the signature (σ) as an output. On the other hand, the verification function $verify(M, K_p, \sigma)$ checks whether the signature σ is valid for the message M by using the sender’s public key K_p .

Algorithm 1 A blockchain address generation method.

```

1: function genAddress( $K_p$ )
2:   fingerprint  $\leftarrow$  0x00 || RIPEMD160(SHA256( $K_p$ ))
3:   checksum  $\leftarrow$  SHA256(SHA256(fingerprint))
4:    $K_{adr} \leftarrow$  base58(fingerprint || checksum[:4])
5:   return  $K_{adr}$ 
6: end function

```

In Algorithm 1, we assume that an elliptic curve algorithm is used to generate the public key. The algorithm takes the public key as an input to generate the corresponding address from the key. First, the public key, K_p is hashed by the SHA256 algorithm, which is hashed again using the RIPEMD160 hash function. After that, the output is concatenated (denoted by ||) with a network specific ID to create a fingerprint. The fingerprint is then hashed twice to generate a *checksum*. Finally the fingerprint and first 4 bytes (denoted by [:4]) of the *checksum* are concatenated and formatted through a base58 encoder to generate the blockchain address (K_{adr}) for a user.

For a given transaction Tx from a public key K_p referring to an address K_{adr} and a signature σ , the verification function $verify(Tx, K_p, \sigma)$ verifies σ . Once σ is verified, the address K_{adr} needs to be verified via Algorithm 1 (Karantias et al., 2019).

For the sake of simplicity, we combine the digital signature verification process and address generation process using a single function as follows.

Definition 5. (*spendVerifier*) The function *spendVerifier* is defined as below.

- *spendVerifier* returns true if $verify(Tx, K_{adr}, \sigma)$ returns true and $genAddress(K_p)$ returns K_{adr} for a given transaction Tx from a public key K_p referring to address K_{adr} .
- Otherwise *spendVerifier* returns false.

Definition 6 (Burn-address). A burn-address given as β is an address to which one can send assets, but from where they can never be recovered because the private key of the corresponding address is not known/ accessible.

Typically, burn addresses are verifiable but unspendable because those addresses do not have a corresponding private key. The proof-of-burn protocol (Karantias et al., 2019) presents proofs that if the underlying cryptography is secure, then the probability of finding a private key for a given burn address is negligible.

We design the burn address to have a reference of the recipient’s identity, therefore, we take the recipient’s blockchain address and run it through the blockchain address generation function *getAddress* which is similar to *genAddress* function presented in Algorithm 1. The only difference is that the *getAddress* function takes the recipient’s blockchain address K_{adr}^R as input and returns β as output.

Definition 7 (Assets). We define an asset v as a digital representation of objects available on a blockchain in the form of tokens which are tradable units used in addition to native currencies of a given blockchain.

Definition 8 (Transaction). A transaction¹ Tx is a tuple $\langle K_p^S, K_{adr}^S, K_{adr}^R, v, Tx^\dagger \rangle$ where v is an asset and Tx^\dagger is the previous transaction where v is transferred from. Each Tx encodes a transaction value and any relevant parameters associated with it.

A *transaction value* includes the transfer value and any related transaction fee, which is the fee a sender offers to a miner node who successfully includes that transaction in a block. The parameters such as time, date and other metadata information are omitted for simplicity.

A typical transaction protocol (Nakamoto, 2008) in blockchain is presented in Algorithm 2, where a valid transfer of v from K_{adr}^S to K_{adr}^R is defined as $K_{adr}^S \rightarrow K_{adr}^R : v$.

Algorithm 2 A general blockchain transaction protocol.

```

1: function trans( $Tx \langle K_p^S, K_{adr}^S, K_{adr}^R, v, Tx^\dagger \rangle, \sigma$ )
2:   if spendVerifier( $K_p^S, K_{adr}^S, \sigma$ ) and
3:     assetVerifier( $Tx^\dagger, K_{adr}^S$ ) is true then
4:      $K_{adr}^S \rightarrow K_{adr}^R : v$ 
5:   else
6:     invalid transaction
7:   end if
8:   return transaction receipt
9: end function

```

¹As in the literature (Buterin et al., 2015), we loosely use “transaction” to refer to both a data structure and an operation.

A transfer will only occur if both *spendVerifier* and *assetVerifier* functions return true.

Definition 9. (*assetVerifier*) The function *assetVerifier* is defined as follows.

- It first checks the validity of the asset v by ensuring that the address K_{adr}^S indeed carries (enough amount of) the asset (Nakamoto, 2008). We refer to this first step as the balance function $balance(K_{adr}^S)$, which outputs the following:
 1. *true* if enough unit of the asset is in K_{adr}^S ,
 2. *false* if not enough unit of the asset is in K_{adr}^S .
- Then it checks whether the transaction Tx referred by the previous transaction Tx^\dagger is included in a valid previous block $Tx^\dagger \in \mathbb{T}^{b\dagger}$ (validate with a Merkle Tree proof²) which is in a valid state (check the longest chain).
- The function *assetVerifier* returns *true* if both of the above steps return *true*; otherwise it returns *false*.

Finally, every transaction returns a transaction receipt which includes transaction information from its execution.

Definition 10 (Transaction finality). Transaction finality refers to the guarantee that a transaction is permanently accepted by the network. In effect, it is computationally infeasible to revert or alter that transaction afterwards.

Transaction finality can be either deterministic or probabilistic (Saito and Yamada, 2016). Deterministic finality is when a transaction is immediately considered finalised once it is included in a block and appended to the blockchain. For this, an elected leader with sufficient authority proposes the block.

Probabilistic finality is a protocol-driven process where the probability of a transaction’s finality increases as more blocks are added to the blockchain after the transaction. Therefore, for most protocols providing probabilistic finality, a recommended number of blocks must be added following the transaction until it can be considered securely complete. For example, in Bitcoin, it is recommended to wait until six additional blocks are mined before considering a transaction to be final.

Definition 11 (Node). A node is a computing device within the distributed network. Some are full nodes that keep a local copy of the full blockchain data, and others are light nodes that depend on a full node when performing transactions.

Typically the nodes do not trust the state change information that they receives, so they performs their own validation. A state is valid if it is accepted by the majority³ of participants in the network.

²A cryptographic hash tree where every leaf node is the hash of a data block.

³The definition of majority is protocol-dependent.

Definition 12 (Blockchain Interoperability). Assuming that independent networks of blockchain are able to provide self-verifiable state proofs, then two blockchain systems N_1 and N_2 are interoperable if a transaction Tx which belongs to N_1 can be accepted by a set of verifiers/validators in N_2 , and vice versa.

It has been proved that general interoperability between blockchain systems is not possible. In this paper, we adopt a relaxed definition of blockchain interoperability (Lafourcade and Lombard-Platet, 2020). In this relaxed definition, under the assumption that participants from one network are able to verify and validate data from the other network, interoperability can be achieved.

4. Proposed cross-blockchain protocol

The *Burn-to-Claim* protocol consists of two main functions for communication between networks: *exitTransaction* function for locking the asset and generating a proof-of-transfer in the source network, and *entryTransaction* function for verifying and validating the proof-of-transfer in order to recreate the asset in the destination network.

Our protocol requires the sender to generate a secret code γ using a random key generator function, *keyGen()*. The sender then encrypts the secret code using the recipient’s public key via $encrypt(K_p^R, \gamma) = \hat{\gamma}$ and shares the encrypted code $\hat{\gamma}$ with the recipient. It will then be decrypted via $decrypt(K_r^R, \hat{\gamma})$ by recipient to get the γ . We assume the public key information is shared among users during the preparation stage. After the completion of the *exitTransaction* the recipient uses the knowledge of this γ as part of the validation process within the *entryTransaction*. For a successful transfer the recipient must reveal the γ within a time limit through the *entryTransaction*. For this part, the recipient encrypts the secret code using his/her private key as follows: $encrypt(K_r^R, \gamma) = \bar{\gamma}$ and includes $\bar{\gamma}$ in the *entryTransaction*. The mining nodes decrypts $\bar{\gamma}$ using $decrypt(K_p^R, \bar{\gamma})$ to get the γ and validate with the received hash of γ included in the *exitTransaction*.

4.1. The *exitTransaction* function

The *exitTransaction* function must be initiated on the source network by the sender. This execution checks the validity of the transaction and generates a proof-of-transfer. The transaction-validity process checks the authenticity of the sender’s signature and the owner’s ability to spend. The proof-of-transfer generator produces a proof that the asset exists and it is locked while the asset is in transit. This transaction aims to create a proof-of-transfer for that asset in the source blockchain network. The proof-of-transfer created by the system as part of the protocol is essential for the system to maintain security over the transaction information. Moreover, the network should eventually come to a consensus about the asset transfer

and thereby the consistency of the information is guaranteed.

In our protocol, the *exitTransaction* uses a conditional time-lock with a hashed secret code. The former determines a time frame for the transaction, and the latter is used to claim the asset in the destination network. A time-lock is defined as a function that locks the output of a transaction for a period of time such that the asset cannot be spent until the time has elapsed⁴.

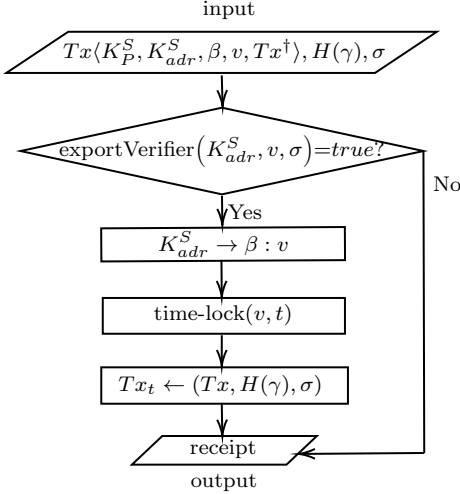


Figure 1: The *exitTransaction*

We design our transaction protocol based on blockchain transaction protocol presented in Algorithm 2 (Section 3). The *exitTransaction* defined in Figure 1 takes a tuple of $(Tx, H(\gamma), \sigma)$ as input arguments where $Tx \langle K_p^S, K_{adr}^S, \beta, v, Tx^\dagger \rangle$ includes the sender's public key (K_p^S) , address (K_{adr}^S) , burn-address (β) , asset (v) and the previous transaction (Tx^\dagger) in which the asset v was last spent. $H(\gamma)$ represents the hash value of secret code γ and σ is the digital signature. We denote the data structure of exit transaction as Tx_t , which is a tuple $(Tx, H(\gamma), \sigma)$ consisting of a transaction, a hash of secret code and a signature.

The *exitTransaction* initiates the *exportVerifier*. The *exportVerifier* function consists of two sub-functions *spendVerifier* and *assetVerifier*, and returns true when both sub-functions return true, otherwise it returns false.

The *exitTransaction* is executed by the mining nodes. If *exportVerifier* returns true then the transaction executes the transfer of the asset to the given burn-address $K_{adr}^S \rightarrow \beta : v$ with conditions such that the asset is time-locked within the source network for a predetermined time-lock period t and the hashed secret code $H(\gamma)$ is added to the data structure of the transaction. In either case of valid or invalid transaction it produces a transaction receipt.

4.2. The *entryTransaction* function

An *entryTransaction* function must be initiated in the destination network by the recipient. Upon initiating the

entryTransaction with the proof-of-transfer from the source network, the network nodes verify the proof-of-transfer and recreate the asset's validity.

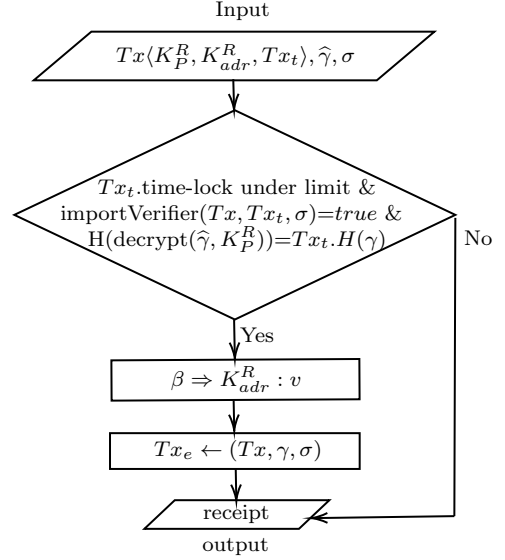


Figure 2: The *entryTransaction*

The *entryTransaction* defined in Figure 2 takes a tuple $(Tx, \tilde{\gamma}, \sigma)$ as input where the transaction $Tx \langle K_p^R, K_{adr}^R, Tx_t \rangle$ includes the recipient public key K_p^R , the recipient address K_{adr}^R , and the previous transaction Tx_t . The burn-address β and the asset v are included in the Tx_t . The data structure of the entry transaction is written as Tx_e , which is also a tuple (Tx, γ, σ) of a transaction, secret code, and a digital signature of R .

The *importVerifier* consists of two functions: *spendVerifier* and *proofVerifier*; it returns true when both sub-functions return true, and returns false otherwise. The function *spendVerifier* returns true if both the given burn-address β and K_{adr}^R are generated from K_p^R , otherwise, it returns false. A *proofVerifier* is an extended version of *assetVerifier*. The *proofVerifier* returns true if the balance $(\beta) = \text{burn}$, in the Tx_t and that $Tx_t \in \mathbb{T}^b$, and $\mathbb{T}^b \subseteq Q$, otherwise returning false. Here the nodes on the destination network need to verify the proof from the source network. We assume that through the gateway mechanism, the participating network will be able to replicate relevant cross-blockchain proof data. In addition, through the previous transaction, any node can access the specific transaction in the source network for verification. Once the *importVerifier* function returns true, the mining nodes need to check the time-lock and the secret code. If the previous transaction's time lock $Tx_t.\text{time-lock}$ is under the time limit and $H(\text{decrypt}(\tilde{\gamma}, K_p^R)) = Tx_t.H(\gamma)$ then the network issues an equivalent value of the asset in the burn address β to the recipient. Note that technically the asset cannot be transferred from β to the recipient — the asset is recreated on the recipient's network. To distinguish it from a normal transfer denoted by \rightarrow , this operation is denoted by $\beta \Rightarrow$

⁴<https://bcoin.io/guides/cltv.html>

$K_{adr}^R : v$. We assume that both the source and the destination networks (synchronised with a global clock) run on a global time zone Zamyatin et al. (2019a). Supposing that the time is within the time-lock period and the encrypted secret code matches with the hash value embedded in the transaction, the network awards the asset to the recipient address K_{adr}^R .

4.3. Workflow of the proposed protocol

This section presents a walk-through of how the *Burn-to-Claim* protocol works. We begin with a use case of two blockchain systems which are self-sufficient and secure (Lafourcade and Lombard-Platet, 2020). The two networks run different applications, but they want to interoperate. These networks may have distinct consensus mechanisms. It is assumed that the majority of consensus participants on both networks are honest. Additionally, even though these systems are not functionally connected, they have enough credibility and both are governed by a higher-level operation scheme/ protocol. For example, they can be two different businesses with a collaborative business interest, different branches of a company or different departments in an organisation. The main objective of this research is to address the cross-blockchain transaction proof-problem. Therefore, we focus on the construction of consensus and how the transactions are verified.

4.3.1. Network assumptions

To address the interoperability we made some assumptions about the network participants and their ability to mine the transaction. We assume that the cryptographic primitives (Wang et al., 2019a) of the networks are secure. We make the same assumptions for the underlying network as in Eyal et al. (2016); Kiayias et al. (2017); Kokoris-Kogias et al. (2018); Zamyatin et al. (2019b). For example, there is a global clock, and the honest nodes are well connected and are able to communicate with each other.

In our protocol, elected nodes function as gateway nodes on both the networks. These gateway nodes listen to events triggered by the cross-blockchain transaction and relay relevant data to their respective networks. [A relaxed assumption is considered where a gateway node may not be a trusted node in the network therefore its activities are regulated using a penalty for malicious activities. Hence, during data sharing, both the source and target gateway nodes must follow their corresponding networks policy and rules. In a private network, gateways are accepted as a trusted node whereas in other networks gateways must prove its trustworthiness. Therefore, we assume to support reliable data sharing, and that networks can use some mechanism to produce the malicious misbehaviour report and corresponding penalty process \(Hardjono, 2021\).](#)

Regarding mining nodes, we envision the concept of merge mining, where nodes are participating in the mining process of multiple networks for financial benefit (Judmayer et al., 2017; Zamyatin, 2016). The nodes can rely on

decentralised integration mechanisms that can identify and address cross-blockchain integration. This decentralised integration mechanism acts as a verification method for the cross-blockchain protocol. The architecture of our model can be seen as a number of networks (depending on the topology) connected through an integration mechanism to create a network-of-networks.

There will be multiple nodes performing merge mining on the same network; therefore, they will be competing against each other for the mining reward. We assume that not all nodes will verify the transaction, but a sufficient number of them must do so to satisfy the security of the system. The other mining nodes will accept the proposal made by a merge mining node based on the credibility (Ferdous et al., 2011) of that node in the network. We consider detailed discussions of the integration mechanism as outside the scope of this paper.

4.3.2. Running Example

We consider an example where a sender S in the blockchain network N_1 wants to transfer an asset to a recipient R in the blockchain network N_2 . S first submits an *exitTransaction* to network N_1 . The network N_1 executes the transaction and effectively burns S 's asset on N_1 . R notices that the asset has been burned successfully and submits the proof of burn to the network N_2 . Network N_2 verifies the proof and, if successful, recreates the asset and assigns it to R 's account. Figure 3 shows a brief overview of the protocol construction. [This consists of three stages: Prepare, Commit, and Execute.](#)

Prepare stage. In the prepare stage, first, Sender S and Recipient R shall agree on a shared secret code γ via an asymmetric key exchange mechanism. S generates the secret code γ and encrypts it with the recipient's public key, K_p^R and then shares the encrypted secret code $\hat{\gamma}$ with the recipient; Secondly, they must agree on the time-lock period t ; Thirdly, S needs to generate the burn address.

We now discuss the use and importance of these parameters. In order to use an asset from an address K_{adr} , a user needs to prove the ownership of the public key that was used to generate the address. That means any value sent to an address with no private key can be considered as burned. However, how do we know that an address does not have a private key? To address this issue and to make a more specific proof in our protocol, S uses R 's address to generate the burn address which is given as *getAddress()* and this function takes a K_{adr}^R and returns β . This will guarantee that the address β does not have a private key in the source network and the β is specific to the recipient R because it is generated from R 's address. Therefore, we can use this transfer to specific burn address as a valid proof.

Now we have the proof, we need to add a provision for retrieving the asset in case of an unsuccessful transfer, which could include the following cases: the asset is burned on the source network and not recreated in the

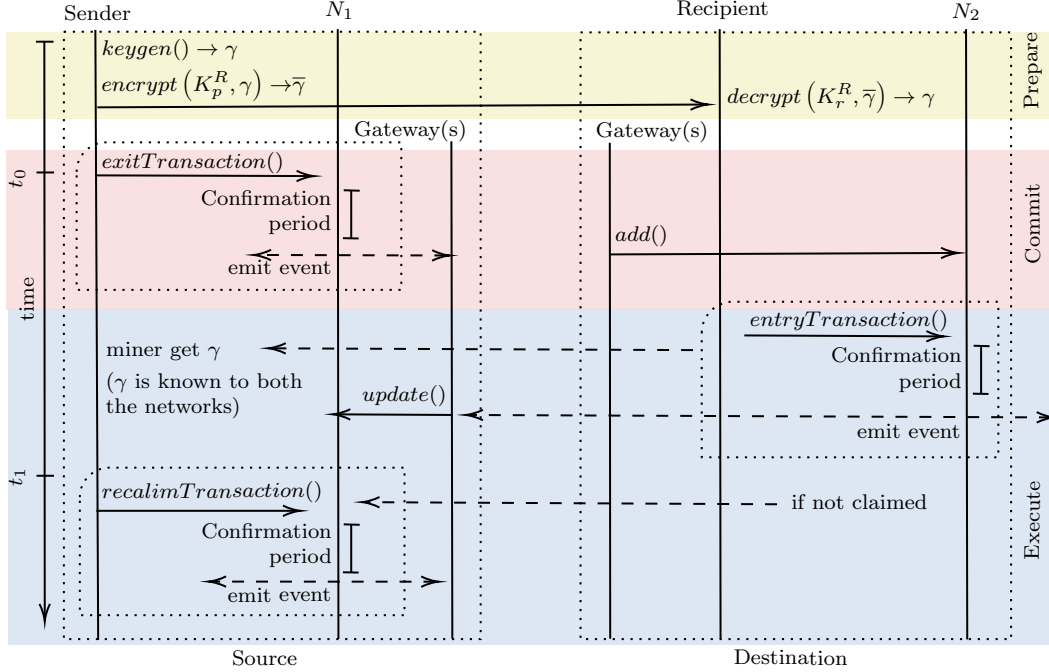


Figure 3: A high level overview of the *Burn-to-Claim* protocol workflow. The vertical lines represent actors and horizontal arrows represent message activity along the time line. At the **prepare** stage (Section 4.3.2), sender generates and shares a secret key with recipient. At the **commit** stage (Section 4.3.2), sender initiates the *exitTransaction* function defined in figure 1 that transfers an asset v to recipient. The transaction gets confirmed after reaching consensus of the network and emits the transaction event. The gateway nodes then add the relevant cross-blockchain transaction data on to the destination network. At the **execute** stage (Section 4.3.2), recipient initiates *exitTransaction* defined in figure 2 to claim v . The destination network nodes cross-verify and propagate the transaction to be included in the next block. If recipient fails to claim v , sender reclaims v via *reclaimTransaction* defined in figure 5 after the time-lock period.

recipient network, or the asset is sent to an incorrect address. We name this procedure as *reclaimTransaction* and explain the process in Figure 5. To ensure the correctness of the reclaim process, we impose a time-lock on transaction output in the source network. The time-lock mechanism will prevent the sender from reclaiming the asset before the recipient claims the asset. The time-lock period is agreed based on factors such as network latency and block creation timing, which we refer to as transit-time.

Commit stage. In the commit stage, S creates and broadcasts an *exitTransaction* to the network N_1 . Figure 4 shows the time-key-lock condition logic. The condition here is that anyone claiming the transaction output (including the asset and the miner’s fee) must reveal the secret code. Therefore, the transaction can only be mined by those miners who have access to the destination network’s data because they need to get the secret code γ to claim the fee. That means if no miner knows the destination network, then this transaction will not go through in the first place.

The time-key-lock condition mechanism allows only one party to claim the asset at a time. For example, while the asset in the Tx_t is time-locked in the source network, if recipient R network reveals the secret code γ , the miner who mines this transaction in the source network will be able to reveal the code γ on the source network to claim

his/her fees; thereby γ is known to both the networks. After that, S will not be able to claim the asset. Likewise, if the R fails to claim the asset within the time frame, then S reveals γ after t_{lock} in the source network to reclaim the asset. In our design γ is not known to the network in the beginning; therefore, once γ is revealed, then nobody will be able to claim the asset even after t_{lock} .

Execute stage. The execute stage has two possibilities: either R claims the asset within the transit time-lock period or S reclaims the asset after the time-lock period.

a) *R claims the asset:* R constructs an entry transaction and broadcasts it to N_2 . If R can prove the ownership of K_p^R and K_{adr}^R , the network will be able to process the transaction. However, our protocol requires a solid evidence that S has burned the asset v on the source network. Here R provides a burn-address β and Tx_t , which works as proof of commit on the source network.

As specified in subsection 4.3.1, our protocol requires a mining process where some nodes are able to mine in multiple blockchains. With that requirement, out of the total number of nodes, a valid gateway⁵ nodes propose this transaction, and eventually, one proposal will be accepted

⁵A valid gateway node refers to a gateway accepted by the network, this could be using a trust-based approach to elect the gateway node.

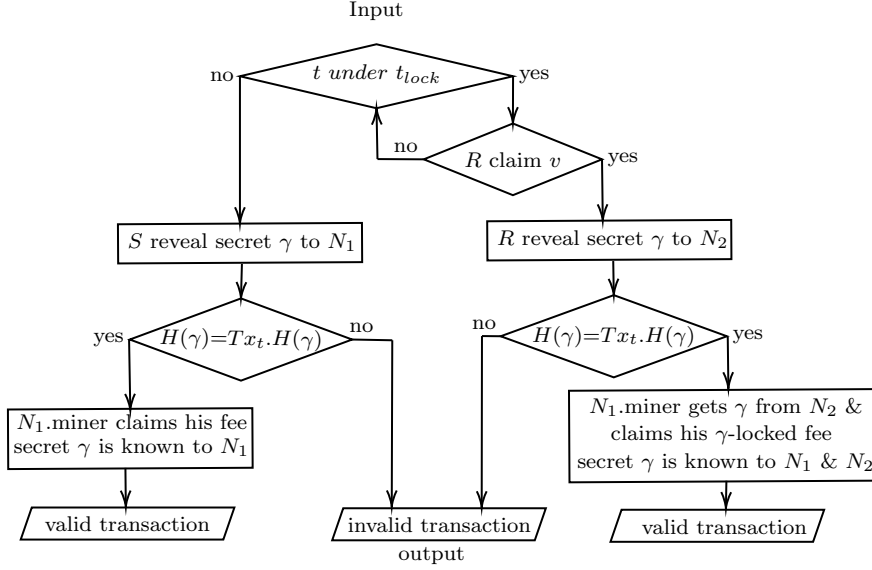


Figure 4: The *exitTransaction* time-key-lock condition

by the network. We assume that these gateway nodes are able to validate the proof-of-transfer with Tx_t . The nodes verify that the transaction occurred on the source network by ensuring that the transaction is contained in a block. This can be done by checking the chain validity (Garay et al., 2015) of the block and the transaction.

If *importVerifier* returns true, the network awards the asset v to the recipient address K_{adr}^R . At this point the secret code γ is known to the R network, and after that, the source network miner can take it to the source network to claim his/her fee, after which γ is known to both the networks.

b) *S reclaims the asset*: If the recipient has not claimed the asset within the time-lock period, the sender is notified through an event listener or the mining node who mined the Tx_t , and the sender then invokes the *reclaimTransaction* function, which is a variant of the *entryTransaction* function and is defined in Figure 5, to claim the asset back. The transaction first checks the signature via *verify*(Tx, K_{adr}^S, σ), then checks the time-lock period and secret code hash. As we stated earlier, our protocol requires some miners to mine in both the chains. Therefore, we assume that miners are able to check with the network N_2 before approving this transaction.

The *reclaimVerifier* consists of two parts: *spendVerifier* and *proofVerifier*. The function *spendVerifier* checks if *verify*(Tx, K_{adr}^S, σ) returns true and *getAddress*(K_{adr}^R) returns K_{adr}^R . Then the *proofVerifier* returns true if the transfer proof of Tx_t referring to the balance(β) = *burn* and $Tx_t \in \mathbb{T}^b$ and $\mathbb{T}^b \subseteq Q$, otherwise it returns false. The *reclaimTransaction* is a variation of *entryTransaction*, and can be included in the *entryTransaction* but, for clarity, we present it as a separate transaction. If the *spendVerifier* and *proofVerifier* return true, then the network issues an equivalent value of the asset in the β to the sender β

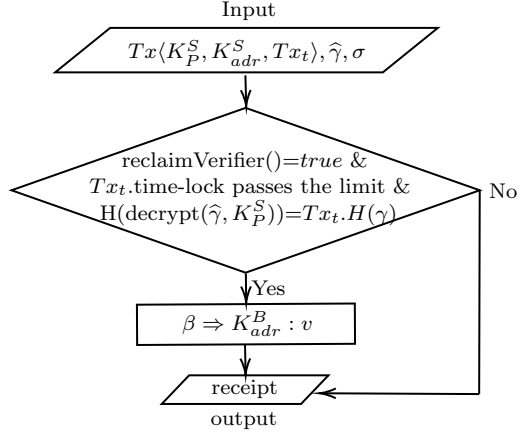


Figure 5: The *reclaimVerifier* function

$\Rightarrow K_{adr}^S : v$.

5. Properties of the *Burn-to-Claim* protocol

This section analyses the properties such as security, correctness, atomicity, and interoperability of the proposed *Burn-to-Claim* protocol. Introducing trust in the cross-blockchain data is one of the most important means to enable interoperability among blockchain networks. The integration process for exchanging information may be based on other existing techniques (Belchior et al., 2020). However, to build trust about the shared information, we must resolve four specific properties of the individual transactions involved in the exchange process. The first is Security: a cryptographic assurance of transfer commitment of transactions; the second is Correctness: each successful transaction commits only one valid outcome; and the third Atomicity: either the transfer executes the transfer of an

asset or returns the asset (Belotti et al., 2019; Karantias et al., 2019; Sigwart et al., 2020; Zamyatin et al., 2019a). Finally and above all comes Interoperability: where one network validates cross-blockchain transaction from another network.

It should be noted that the burn-address in our proposed protocol is not generated from a regular key-pair. Rather a unique address is generated each time by combining the recipient address and other parameters. Therefore, it is not spendable in the current network. This means, with the signature scheme of the underlying cryptocurrency, the asset burned in the proposed scheme would remain unspendable. We state this result below.

Lemma 1 (Unspendability). *A burn-address β is unspendable with respect to a blockchain address protocol in Algorithm 1.*

The main functionality of *entryTransaction* is to recreate an asset, but only after the asset is burnt on the source chain. In our protocol, the asset must be permanently burned at the commit stage. As a result, we have the lemma below.

Lemma 2 (Burn before claim). *An asset v which is transferred from N_1 to N_2 must be burned on N_1 before the recipient can claim it on N_2 .*

Next, we show that the sender who initiates the *exitTransaction* must own the asset he/she is transferring. In *exitTransaction* the *exportVerifier* checks the transaction validity and the owner’s ability to spend. This process must be carried out by each mining node and must reach the consensus on the network. Therefore, as long as the network is secure, the participants can only exchange their own asset.

Lemma 3 (Ownership). *The function Tx_t can only be successfully executed if the sender S owns the v . Let S be a domain of the user, and v a domain of asset. An asset has exactly one owner at a time: $Owner(S, v)$ is true if and only if S owns v .*

A malicious sender may wish to send a conflicting transaction and spend the same asset again resulting in double-spending. In order to prevent such behaviour, we introduce the time-lock period where the recipient can make a claim, and the sender needs to wait before making a reclaim. After the time-lock period, the recipient can not make a claim.

Lemma 4 (Double spend). *The validity of the given token only exists in one chain at a time.*

An asset v is recreated on the recipient network R only if a corresponding v has been burned on the sender network S . Also once the asset has been recreated, S can not reclaim it back on his/her chain. In other words, the state of an asset’s validity in a multi-blockchain network, only exists on one network at a time.

Lemma 5 (Decentralized finality). *The distributed participants of the network must agree on the process of finalising a state change.*

We will now apply the above lemmas to show the security property of our protocol.

Theorem 1 (Security). *The recipient network can rely on the Burn-to-Claim proof-of-transfer guarantee provided by the source network.*

Proof. Based on Lemma 2, *exitTransaction* burns the asset to a burn-address β which is unspendable as per Lemma 1. The *exitTransaction* also checks the sender’s ability to spend the asset (Lemma 3). Under the assumption of a secure blockchain C , for a given Tx , committed in a valid state Q of a C can serve as a reliable proof. Therefore, if a Tx_t successfully executes a transfer $K_{adr}^S \rightarrow \beta : v$ and includes it in a block b , which executed by a state transition function $\Pi(Q, b) = Q'$ then the output of the new state Q' is deemed as a valid proof. \square

Now we analyse the correctness of the our protocol.

The *exitTransaction* transfers the asset v to a burn-address β that is derived from the recipient R ’s address K_{adr}^R . To claim v , the recipient must prove to the network that β is derived from an address he/she owns. The function *spendVerifier* checks the signature σ to verify the K_{adr}^R and checks the whether β is derived from the given K_{adr}^R using the function *getAddress*. Therefore, only the user who owns a private key associated with K_{adr}^R can make a valid claim for the asset on the destination network.

Theorem 2 (Correctness). *The Burn-to-Claim protocol only transfers an asset to the correct recipient.*

Proof. Based on the *exitTransaction* defined in Figure 2 the recipient’s signature must be correct, then the relation with the β must be correct. Finally the recipient must be able to decrypt the encrypted secret code $\bar{\gamma}$ using his/her private key via $decrypt(\bar{\gamma}, K_r^R)$. All these measures ensure that the Tx_t will only transfer the asset to the correct recipient. \square

Now we analyse the atomicity property of our protocol.

Whenever transfer parties do not trust each other, it is in their interest to ensure that no participant can take advantage of the agreed transfer. The protocol must be constructed in such a way that the transfer is performed in its entirety or no not at all. This property is sometimes referred to as “all-or-nothing”. In the case of failures during the protocol execution, every transfer participant must be able to regain possession of the originally owned assets.

Theorem 3 (Atomicity). *The transfer operation should only execute one outcome, either the transfer succeeds and the asset is transferred to the recipient; or it fails and the asset returns to the sender.*

Proof. If the recipient R claimed the asset v , it must be that R has revealed the secret code γ . Therefore, γ is known to the destination network N_2 . Now the miner who processes the *entryTransaction* in source network N_1 needs γ to get the fee. The miner gets γ from N_2 to N_1 to claim his/her fees. Thereafter, γ is known to both N_2 and N_1 and no one else will be able to claim the asset.

If R fails to claim the asset within the time-lock, S would invoke *reclaimTransaction* defined in Figure 5 to re-claim the asset. Due to the time-lock mechanism, S can only reclaim after the time-lock period. During the time lock period, γ is not known to the network.

In other cases of a transfer failure (e.g., the recipient goes offline or the sender uses a wrong recipient address to generate the burn address), the sender can also reclaim the asset. With the help of time-lock and the secret code, only the two outcomes stated in this theorem are possible with our protocol. \square

Now we analyse the interoperability property of the proposed protocol.

For a blockchain system, interoperability refers to cross-communication between different blockchain networks. In our context, interoperability is the ability to transfer assets from one blockchain to another. Technically, native coins such as BTC or ETH created by blockchains are not transferable. However, the tokens on a blockchain that represent digital assets can be transferred between networks of blockchains (Pillai et al., 2019).

Theorem 4 (Interoperability). *The Burn-to-Claim protocol is interoperable w.r.t. Definition 12.*

Proof. Let N_1 be the sender’s blockchain network and N_2 the recipient’s blockchain network; we show that N_1 is interoperable with N_2 and vice versa, provided that N_1 ’s transaction can be verified by N_2 ’s miner and N_2 ’s transaction can be verified by N_1 ’s miner.

Let a sender S burn an asset in the network N_1 . Then, through an integration mechanism of gateway nodes, the nodes in network N_2 get access to the specific cross-blockchain data emitted from the burn algorithm (cf. Figure 1). Similarly, if recipient R claims the asset on N_2 , the claim algorithm (cf. Figure 2) data get updated on the network N_1 . Thus, under the scheme of a multi-blockchain architecture where some mining participants of N_1 and N_2 have an interest in participating in the required mining process, and both the network and have a gateway mechanism that can update specific cross-blockchain data between networks, then the two blockchain networks are interoperable. \square

Comparison with related work. In this section we give a theoretical comparison of the properties we have defined and summarised in Table 2. The proof-of-burn protocol is proposed by Karantias et al. (2019) and Sigwart et al. (2020). In Karantias et al. (2019) the burn-address is generated from a reference of the recipient’s address and a *tag*.

Therefore, the burn-verifier returns true if and only if the address and the tag match. However, one of the limitations we found in Karantias et al. (2019) is the absence of a mechanism to return the asset back to the source chain in case of an unsuccessful transfer, which is why the atomicity of their approach is a “no”. By contrast, in Sigwart et al. (2020) the sender on the source chain invokes a burn-transaction to destroy his/her coin, which will be recorded in the smart contract on the source chain. After that, one can query the smart contract to check the validity of the burn-proof and whether the burn-proof is used or not. To claim the asset, any user can invoke the asset management smart contract on the destination network and provide the proof-of-burn transaction reference on the smart contract. This makes it possible for multiple participants to make a claim. Thus, the correctness is determined as “partial”. Compared to those methods, our work uses a direct reference to the recipient address in the exit transaction and the recipient to invoke the claim, which guarantees the correctness property we defined. The time-lock, along with a secret code and the re-claim mechanism, satisfies the atomicity property.

Table 2: Theoretical comparison summary

Reference	Properties		
	Security	Correctness	Atomicity
Karantias et al. (2019)	yes	yes	no
Sigwart et al. (2020)	yes	partial	yes
<i>Burn-to-Claim</i>	yes	yes	yes

6. Implementation and experimental results

In this section, we discuss the implementation architecture, experimental testing, and our evaluation of the performance of our protocol.

6.1. Contract implementation

For testing, we use a reference implementation of the *Burn-to-Claim* protocol in Solidity⁶. The protocol logic is defined in a smart contract and implemented in an Ethereum-based test network of Rinkeby⁷ and Ropsten⁸. For evaluation, we investigate the performance and cost impact of cross-blockchain transactions in an ecosystem of blockchain networks where participants perform repeated token transfers. In addition, we also measure the application’s communication overhead (Malavolta et al., 2019) when making a transfer.

The implementation consists of an application which acts as an interface that ties all the components together. A brief overview of the project architecture is presented in Figure 6. The application is written in JavaScript which

⁶<https://solidity.readthedocs.io>

⁷<https://rinkeby.io>

⁸<https://ropsten.etherscan.io>

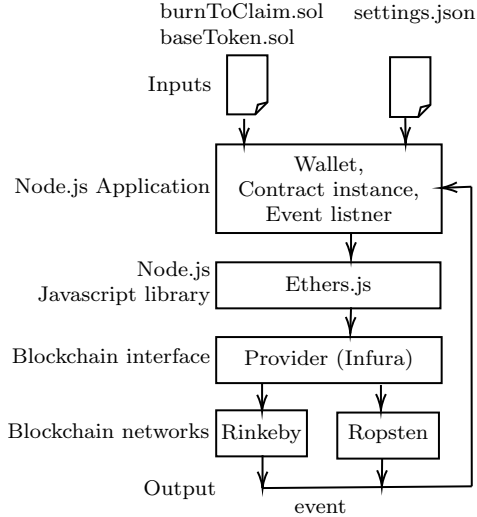


Figure 6: The architecture of experimental implementation

utilises ethers.js⁹ and node.js¹⁰ libraries to interact with the blockchain node. We utilise the Infura¹¹ interface service to connect to a blockchain node. The application input includes a burn-to-claim contract, a base-token contract and a settings file. The burn-to-claim contract is the main contract with all the protocol-specific functions of *exitTransaction*, *entryTransaction* and *reclaimTransaction*. The base-token contract is an ERC20 reference contract inherited from the openzeppelin¹² library. The base-token contract provides token supply and token specific functionalities. Based on the sequence of the transfer, the application creates contracts and wallet instances for different users. For example, to deploy it creates an admin instance of the wallet, to transfer, sender and recipient instance of wallet are created. The settings file stores necessary information to create the individual instances of the wallet. The test implementation acts as a client for deploying the contract and testing the transfer of tokens. The contract code and test script are available under this Git repo¹³.

The main functions of the test application are listed in Table 3. We denote the sender instance of the contract as S and recipient instance of the contract as R . The test script first deploys an instance of burn-to-claim and base-token contracts in the respective networks through an admin account. We assume each network will have its admin to deploy the contract. The admin then transfers the tokens generated by the base-token contract to the burn-to-claim contract address. The burn-to-claim contract will then issue tokens to the participants. In our case, the burn-to-claim contract will issue tokens to Alice, and the test we are demonstrating is Alice transferring her tokens to

Bob on another network. The current evaluation is for demonstrating the overall end-to-end functionality of the *Burn-to-Claim* protocol. This test includes Alice burning her token using Tx_t , and Bob claiming his token using Tx_e .

6.2. Experimental set-up

The experimental cross-blockchain asset transfers test is conducted between two different Ethereum-based test networks of Rinkeby and Ropsten. Both Ropsten and Rinkeby are shared public Ethereum blockchain test networks that are configured to simulate real-world solutions as closely as is practical. Even though these networks use different network parameters, the EVM execution of the smart contract on these networks is identical. Furthermore, compared to the main network where the mining difficulty is dynamically adjusted based on the network hash power, the mining difficulty of test networks are stable with a lower mining time. Therefore, evaluation using test networks provides sufficient estimation of transaction execution time and gas usage on Ethereum based networks.

The test is conducted in intervals of 10 x 5 times, (total 50 times) transfers of 1 ERC20 token from Alice on the Rinkeby network to Bob on the Ropsten network, and 50 times from Bob on the Ropsten network to Alice on the Rinkeby network through a light client machine (Dell Inspiron 15 7000 series laptop). The configurations are 64 GB RAM running on an Intel i7 processor. The standard deviation of results between tests of 10's, 20's, 40's and 50's are very marginal; therefore, 50 times of testing suffice.

6.3. Performance metrics

An experimental performance test is conducted transferring value from one network to another. From an application perspective, it measures how the system performs a given operation. In our case, an integration test is essential for the exchange of value (token) between networks. The application is deployed across different networks which run asynchronously through integration interfaces. Since there are multiple integration interfaces, the performance of end-to-end scenarios (that is, a complete transfer from the sender to the receiver) is more suitable for a comprehensive performance test of the application.

6.4. Transaction time

The time for transferring a token between networks is measured in seconds. This measure is used in performance bench-marking with respect to the duration of the transfer process for the application. However, typically in a distributed network, the latency depends on the network topology. Therefore these results are used as a guideline to estimate the cost of a cross-blockchain transfer. In our experiment, we considered an end-to-end transfer as the

⁹<https://docs.ethers.io/v5/>

¹⁰<https://nodejs.org/en/>

¹¹<https://infura.io>

¹²<https://openzeppelin.com>

¹³<https://github.com/b-pillai/burn-to-claim>

Table 3: Smart contract function

Contract steps	Function
<i>Sender.deploy()</i> and <i>Recipient.deploy()</i>	Deploy the contract on the networks
<i>Sender.exitTransaction()</i>	Initiate by the sender to burn the token
<i>Recipient.add()</i>	Burn token event, gateway node to update
<i>Recipient.entryTransaction()</i>	Initiate by the recipient to claim the token
<i>Sender.update()</i>	Token claim event, gateway node to update
<i>Sender.reclaimTransaction()</i>	Initiate by the sender if not claimed by recipient

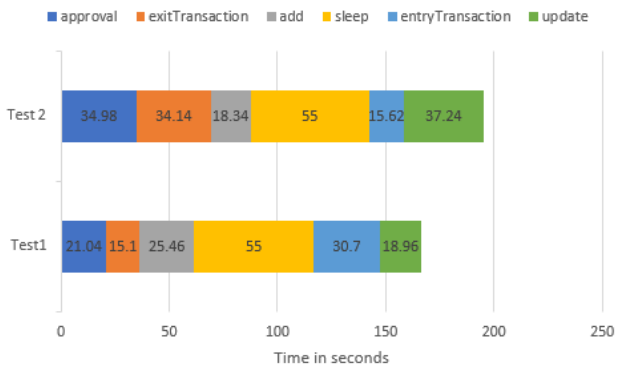


Figure 7: Experimental transaction duration

result of two transactions, the *exitTransaction* on source network and the *entryTransaction* on destination network.

Figure 7 shows two sets of results from the actual transfer experiments. The first result from Test 1 transfers a token from Alice on the Ropsten network to Bob on the Rinkeby network, and the second result Test 2 is from Bob on the Rinkeby network to Alice on the Ropsten network. Each transfer goes through of a sequence of steps: the first step is the approval (the sender approves the contract address to spend the token), then execute *exitTransaction*, after that, we wait (sleep) for 55 seconds in order to reach a probabilistic finality, and finally execute *entryTransaction*. We assume that testing on a single network may not reflect the real-life scenario, whereas taking an average result from two different networks may yield more realistic results.

The implementation is scripted to send each transaction after a previous one was confirmed. During the test, the network status¹⁴ recorded active participation of 43 nodes in the verification process. In order to maintain consistent gas-limit, mining parameters are fixed for every transaction throughout the test. As we explore the end-to-end time of cross-blockchain transfer on different networks, there are significant differences in response and confirmation time, and even multiple tests on the same network recorded varying results. We experiment with multiples sets of tests and reached a point where the standard deviations of 30 and 50 transfers are 27.531 and 27.276 with 95% confidence levels of 9.7 and 9.8 for test 1 and standard deviations of 41.470 and 44.231 with 95% confidence

levels of 15.82 and 11.494 for test 2 respectively. Since the difference between 30 and 50 transfers is very similar, we conclude that the experiment has covered most cases. Therefore, there is no need to extend the experiment with more tests. The results presented are from the average of 100 transactions from two different tests (50 on each network).

For evaluation, we omit the timing of sleep, *add*, *update* and only consider the transaction finality time of *exitTransaction* and *entryTransaction* function. Thus the experimental results present an average time of 86.74 seconds for the *Burn-to-Claim* protocol. The results, shown in Figure 7, were an average end-to-end transfer time of 66.84 seconds for test 1, and of 84.74 for test 2.

6.5. Transaction cost

In Ethereum, smart contracts are deployed as bytecode in blockchain and accessed via a contract address. The contracts are then executed in EVM (Ethereum Virtual Machine) on each client. The transaction execution cost in Ethereum based blockchain networks is measured in the unit of gas. The cost is calculated based on the operation steps of the smart contract and its storage size (Wood et al., 2014).

From our test, the gas consumption for the functions described in Section 6.1 are outlined in Table 4. We have set a Gas limit of 200000 per unit and a gas price of 0.00000002 Ether (20 Gwei) per unit of gas for the test. For the analysis, we take into consideration of the end-to-end transaction cost of *approve*, *exitTransaction* and *entryTransaction*. There are also additional costs involved in keeping the gateway node running. We assume these are included in the form of an incentive mechanism; therefore, they are ignored for now. For the test purpose the *add* and *update* cost are paid by the respective contract. From the experiment, our protocol requires 316507 units of gas (316 kGas) for an end-to-end transfer (the sum of Tx_t and Tx_e).

Table 4: Cost analysis of contract functions

Contract function	Gas used
<i>approve</i>	44114
<i>exitTransaction</i>	184444
<i>add</i>	152313
<i>entryTransaction</i>	87949
<i>update</i>	64947

¹⁴<https://www.rinkeby.io/#stats>

6.6. Communication overheads

In regards to communication overheads, we evaluate our protocol under the number of messages required to complete the exchange process. Further, we divide the communication overheads into two categories: users and integration participants. Users communication overheads include transactions and messages from the users (this does not include private message between sender and recipient) and integration participants communication overheads include messages between integration participant mining or gateway nodes. For evaluation, we only consider the messages initiated by the sender and recipient. All other messages across network participants are as part of the system, and therefore omitted for now.

Based on the design, some protocol require more cross-blockchain tasks than others. With more tasks, more messages need to be handled, which results in time and communication costs. Generally, one successful cross-blockchain task requires at least two messages request, a *commit* and an *execute* whereas a *pre-commit* and *coknowledgenfirm* support more features but bring its own complexity.

The proposed cross-blockchain protocol uses multi-chain architecture based on some integration mechanisms. The transactions are initiated on the relevant network based on some pre-agreed knowledge. Some approaches make a pre-commit and confirmation to ensure efficiency. In other words they do a check before and after making the commit. This process may avoid aborting the transaction if it is not valid. However, this pre-checking process or confirmation process is executed at the cost of the network, which increases complexity. Whereas a two-phase process that satisfies the property bring a multi-phase process can reduce network communicate cost and complexity.

In our protocol, there are three steps: prepare, commit and execute. Prepare happens on a private channel therefore not burdening the blockchain network. The commit and execute steps carry out the burn and claim on the network and satisfy the properties defined in Section 5.

6.7. Performance comparison

In this section, we briefly compare the experimental performance results in accordance with the existing works in Borkowski et al. (2019); Kan et al. (2018); Sigwart et al. (2020) and present a summary in Table 5. Performance refers to an outcome obtained from a process. Here the performance is defined as the outcome of a cross-blockchain transaction’s response *time*, *cost* and *number-of-messages*. This initial analysis can provide an overview to understand the feasibility of different cross-blockchain protocols. [The existing works’ performance value are obtained from the relevant paper; however, not all these works provide details for all three parameter, therefore, we omit unknown data.](#)

In regards to time, in Sigwart et al. (2020) the cross-communication experiments recorded 91 seconds for the burn process, and 117 seconds for the claim process, which

totals 208 seconds for an end-to-end transfer for Protocol 1. For Protocol 2, there is an additional time for the confirmation process of 89 seconds, which total to 297 seconds. In contrast, our protocol records 86.74 seconds to execute an end-to-end transfer process. Even though it is the nature of blockchain-based systems that a transaction times may vary based on network parameters, we have taken an average of 50 transactions and still show significant improvement. Both the tests are conducted in the Ethereum-based test network; therefore, the network parameters are the same for the tests.

In regards to gas cost, in Sigwart et al. (2020), the experiment records a gas consumption of 40.79 kGas for the burn process, 121.47 kGas for the claim process and 104.62 kGas for the confirmation process, which add up to 266.88 kGas of end-to-end costs excluding the integration cost. In Borkowski et al. (2019), the experiment shows a 57.7 kGas for the claim, 81.5 kGas for the contest and 45.5 kGas for finalising the transfer, which add up to 187.7 kGas for an overall end-to-end transfer.

Regarding the communication overheads, in Borkowski et al. (2019), the sender initiates the transfer and waits for the recipient’s acknowledgement before considering it as a valid *Proof of Intent*. The CLAIM transaction is then initiated and CONTEST (witnessed) by the network participants. Therefore, we consider it to be a two-stage process. In (Sigwart et al., 2020), the basic protocol utilises a burn and claim, whereas the updated version utilises the CONFIRM process. Therefore, we consider it to be a three-stage process. In Kan et al. (2018), the process of executing crossing-chain transactions goes through a three-phase process. First, there is a prepare stage, where a transaction is put forward to the destination chain. If valid, a pre-commit is issued by the destination chain, and finally a transaction commit is issued on both the chain. Therefore, we consider it to be a three-stage process. In comparison, our protocol consists of a two-stage process: *exitTransaction* to burn the token and *entryTransaction* to reclaim the asset.

Table 5: Performance comparison summary

Reference	Time Seconds	Cost kGas	Message Numbers
Borkowski et al. (2019)	N/A	187.7	2
Sigwart et al. (2020)	297	266.88	3
<i>Burn-to-Claim</i>	86.74	177	2

7. Conclusion and future work

In this paper, we analysed blockchain transaction protocol for permissionless systems such as Ethereum and we formalised a protocol for cross-blockchain asset transfer. One of the critical features of our proposal is that it presents an internal functionality for interoperability. An internal (built-in) protocol provides a common mechanism. Via such a protocol, blockchain users can com-

municate directly and transfer various forms of data and digital assets using standardised pathways. Furthermore, we briefly showed that the *Burn-to-Claim* protocol is resilient to double-spending because of its correctness and fairness properties.

We formalised the cross-blockchain interoperability proof-problem and showed that it can be solved without needing a trusted third party in contrast to the assumptions often made in the blockchain community. Our approach is to introduce a built-in function to address the proof-problem. However, our protocol will require revisions to the architecture of many target blockchains as most do not support time-locking, asset-burning, or gateway nodes.

Future work will include formal model analysis and verification of our protocol in an environment with network nodes of malicious actors.

Acknowledgement. We acknowledge Paul Collins, who assisted in the development of the test application.

References

- Belchior, R., Vasconcelos, A., Guerreiro, S., and Correia, M. (2020). A survey on blockchain interoperability: Past, present, and future trends. *arXiv preprint arXiv:2005.14282*.
- Belotti, M., Moretti, S., Potop-Butucaru, M., and Secci, S. (2019). *Game theoretical analysis of Atomic Cross-Chain Swaps*. PhD thesis, Caisse des dépôts-Institut pour la recherche et Banque des territoires
- Borkowski, M., McDonald, D., Ritzer, C., and Schulte, S. (2018a). Towards atomic cross-chain token transfers: State of the art and open questions within tast. *Distributed Systems Group TU Wien (Technische Universit at Wien), Report*.
- Borkowski, M., Ritzer, C., McDonald, D., and Schulte, S. (2018b). Caught in chains: Claim-first transactions for cross-blockchain asset transfers. *Technische Universit at Wien, Whitepaper*.
- Borkowski, M., Sigwart, M., Frauenthaler, P., Hukkinen, T., and Schulte, S. (2019). Dextt: Deterministic cross-blockchain token transfers. *IEEE Access*, 7:111030–111042.
- Buterin, V., Coleman, J., and Wampler-Doty, M. (2015). Notes on scalable blockchain protocols (version 0.3).
- Eyal, I., Gencer, A. E., Sirer, E. G., and Van Renesse, R. (2016). Bitcoin-ng: A scalable blockchain protocol. In *13th {USENIX} symposium on networked systems design and implementation ({NSDI} 16)*, pages 45–59.
- Ferdous, R., Muthukkumarasamy, V., and Sithirasanen, E. (2011). Trust-based cluster head selection algorithm for mobile ad hoc networks. In *2011 IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, pages 589–596. IEEE.
- Garay, J., Kiayias, A., and Leonardos, N. (2015). The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer.
- Gazi, P., Kiayias, A., and Zindros, D. (2019). Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 139–156. IEEE.
- Hardjono, T. (2021). Blockchain gateways, bridges and delegated hash-locks. *arXiv preprint arXiv:2102.03933*.
- Hardjono, T., Lipton, A., and Pentland, A. (2018). Towards a design philosophy for interoperable blockchain systems. *arXiv preprint arXiv:1805.05934*.
- Herlihy, M. (2018). Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254.
- Jiang, Y., Wang, C., Wang, Y., and Gao, L. (2019). A cross-chain solution to integrating multiple blockchains for iot data management. *Sensors*, 19(9):2042.
- Judmayer, A., Zamyatin, A., Stifter, N., Voyiatzis, A. G., and Weippl, E. (2017). Merged mining: Curse or cure? In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*, pages 316–333. Springer.
- Kan, L., Wei, Y., Muhammad, A. H., Siyuan, W., Linchao, G., and Kai, H. (2018). A multiple blockchains architecture on inter-blockchain communication. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 139–145. IEEE.
- Karantias, K., Kiayias, A., and Zindros, D. (2019). Proof-of-burn. In *International Conference on Financial Cryptography and Data Security*.
- Kiayias, A., Russell, A., David, B., and Oliynykov, R. (2017). Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer.
- Kiayias, A. and Zindros, D. (2019). Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*, pages 21–34. Springer.
- Kokoris-Kogias, E., Jovanovic, P., Gasser, L., Gailly, N., Syta, E., and Ford, B. (2018). Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE.
- Lafourcade, P. and Lombard-Platet, M. (2020). About blockchain interoperability. *Information Processing Letters*, page 105976.
- Malavolta, G., Moreno-Sanchez, P., Schneidewind, C., Kate, A., and Maffei, M. (2019). Anonymous multi-hop locks for blockchain scalability and interoperability. In *NDSS*.
- Nakamoto, S. (2008). A peer-to-peer electronic cash system. Available at: <https://bitcoin.org/bitcoin.pdf> (Date last accessed 28-February-2020).
- Pillai, B., Biswas, K., Hóu, Z., and Muthukkumarasamy, V. (2020a). The burn-to-claim cross-blockchain asset transfer protocol. In *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*, pages 119–124. IEEE.
- Pillai, B., Biswas, K., and Muthukkumarasamy, V. (2019). Blockchain interoperable digital objects. In *International Conference on Blockchain*, pages 80–94. Springer.
- Pillai, B., Biswas, K., and Muthukkumarasamy, V. (2020b). Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review*, 35.
- Saito, K. and Yamada, H. (2016). What’s so different about blockchain?—blockchain is a probabilistic state machine. In *2016 IEEE 36th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 168–175. IEEE.
- Sigwart, M., Frauenthaler, P., Spanring, C., and Schulte, S. (2020). Decentralized cross-blockchain asset transfers. *arXiv preprint arXiv:2004.10488*.
- Sompolinsky, Y. and Zohar, A. (2016). Bitcoin’s security model revisited. *arXiv preprint arXiv:1605.09193*.
- Stewart, I. (2012). Proof of burn - bitcoin wiki. Available at: https://en.bitcoin.it/wiki/Proof_of_burn.
- Wang, L., Shen, X., Li, J., Shao, J., and Yang, Y. (2019a). Cryptographic primitives in blockchains. *Journal of Network and Computer Applications*, 127:43–58.
- Wang, W., Hoang, D. T., Hu, P., Xiong, Z., Niyato, D., Wang, P., Wen, Y., and Kim, D. I. (2019b). A survey on consensus mechanisms and mining strategy management in blockchain networks. *IEEE Access*, 7:22328–22370.
- Wood, G. (2016). Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*.
- Wood, G. et al. (2014). Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32.
- Zamyatin, A. (2016). *Merged Mining: Analysis of Effects and Implications*. PhD thesis, PhD thesis. Master’s thesis, Vienna University of Technology, 2016 (cit. on 26-11-2020).
- Zamyatin, A., Al-Bassam, M., Zindros, D., Kokoris-Kogias, E.,

- Moreno-Sanchez, P., Kiayias, A., and Knottenbelt, W. J. (2019a). Sok: Communication across distributed ledgers. Technical report, IACR Cryptology ePrint Archive, 2019: 1128.
- Zamyatin, A., Harz, D., Lind, J., Panayiotou, P., Gervais, A., and Knottenbelt, W. (2019b). Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 193–210. IEEE.