# The Burn-to-Claim cross-blockchain asset transfer protocol

Babu Pillai
*School of ICT*
*Griffith University*
Gold Coast, Australia
babu.pillai@griffithuni.edu.au

Kamanashis Biswas
*Faculty of Law and Business*
*Australian Catholic University*
Brisbane, Australia
kamanashis.biswas@acu.edu.au

Zhé Hóu
*School of ICT*
*Griffith University*
Brisbane, Australia
z.hou@griffith.edu.au

Vallipuram Muthukkumarasamy
*School of ICT*
*Griffith University*
Gold Coast, Australia
v.muthu@griffith.edu.au

***Abstract*—The future of multi-blockchain architecture depends on the emergence of new protocols that achieve trustless communication between cross-chain participants. However, interoperability between blockchains remains an open problem. Existing approaches provide integration through solutions using a middleware system, which makes it harder to gain confidence mainly in terms of security and correctness of the process. A cross-chain protocol needs to provide a self-verifiable state-proof that embeds trust in the transfer process. We propose a Burn-to-Claim cross-chain protocol to seamlessly exchange assets between networks. Our scheme transfers assets from one blockchain system to another in a way that the asset is burned from the source blockchain and claimed on the destination blockchain. Our mechanism employs combinations of crypto mechanisms such as digital signatures and time lock to operate the protocol in a distributed manner. We provide an analysis which proves that our cross-chain protocol transfers assets correctly and securely assuming that the underlying blockchain protocol is secure.**

***Index Terms*—blockchain, interoperability, asset transfer, cross-blockchain protocol**

## I. Introduction

Blockchain technology offers an immutable, decentralized, and transparent mechanism for transaction processing [33]. Interestingly, beyond its role as a protocol for exchanging values within a network, one might reasonably assume that a blockchain system should be able to transfer assets between networks. However, the current architecture of this technology limits the transaction within the network and its users. A blockchain application cannot uniformly use multiple networks and obtain a composition of their guarantees [3]. A core reason of this issue is that each blockchain network has its own state assumptions of the proof-problem [5], and without restrictions, one network cannot trust the informationin a different network. Therefore, currently, seamless composition between two blockchain networks is impossible [23].

A blockchain is *interoperable* with another blockchain if one can provide state composition guarantees of cross-blockchain transactions for the other and vice versa [3]. We formalise the concept of interoperability from our previous work [26] and building on that, we propose a cross-blockchain asset transfer protocol. Our vision of next-generation blockchains is that they should have an *internal* function for composability, the lack of which makes it difficult for applications to commit transactions to multiple networks.

With the current protocols of blockchain systems, it is not possible to have direct interoperability between systems [6]. There is no method in the system to provide a cross-chain value transfer; therefore, external third-party services are the preferred solution [3, 23]. However, such methods come with a trade-off on decentralization, which eventually leads to security issues of the system. Moreover, the blockchain system cannot recognise and verify any such process carried out by a third-party provider. Mainstream enterprises have begun to realise that they cannot use the current public blockchain to streamline their business processes due to this limitation. Today most enterprise blockchain applications reside on private blockchains such as Hyperledger and Corda and loose the full potential of the distributed ledger technology. To address these issues, we propose to use an internal built-in process that is integrated with the protocol to carry out transactions and facilitate interoperability between systems. That way, the system and its users are in control and aware of the assets being transferred.

Decentralised cross-blockchain transfer mechanisms should allow participants to fully utilise the existing variety of blockchains, instead of being locked to a single blockchain type. The cross-blockchain transaction serves as a cornerstone for the next-generation blockchain-based systems. The cross-blockchain integration would enable the interoperability among distinct, potentially heterogeneous, blockchains [14]. This paper introduces the Burn-to-Claim cross-blockchain protocol – a built-in method to address cross-chain interoperability. We define it as a protocol which consists of two components: an exitTransaction to generate a self-verifiable proof that the transaction is committed in the source network and an entryTransaction to verify the validity of the proof in order to re-create the asset in the destination network. A key novelty of a built-in method is that it is not system/ application-specific. Unlike existing approaches that rely on external middleware mechanisms, our method does not require a new crypto-asset or trusted third party. Instead, it implements a 'universal protocol' for all applications and tries to enforce the same integrity requirements across different connected blockchain networks. Our method can serve as a more general framework, in which each application can construct a network based on its trust assumptions and guarantees yet all these heterogeneous networks share the same cross-chain transfer protocol.

A summary of our contributions is as follows:

- **Protocol.** We propose a protocol for cross-blockchain asset transfer where transactions are performed in a decentralised and trustworthy manner.
- **Construction.** We propose a novel and simple construction which is flexible and can be adopted for digital asset transfer among blockchain network without violating the key characteristics of the blockchain technology.
- **Analysis.** We perform a preliminary analysis to show the correctness, security and fairness of our protocol.

### A. *Overview of the proposed protocol*

In order to give an overview of our protocol, let's consider a common example of bank transaction. Typically a bank transaction transfer money from one bank account to another and is initiated by an account holder. Before the transfer takes place, the users should establish knowledge such as account details. The sender proves the her credentials, after validation she initiates the transfer. At the same time, the transfer-amount gets blocked by the bank from further use (preventing double spending). Here the system is generating a proof that the money exists and it is locked; thereby, the counterparties can rely on it and act. In a banking scenario, we deal with money as a commonly accepted asset – this can be different in other applications.

In the blockchain setting, a cross-chain protocol is defined as the interaction between two blockchain networks, namely the *source* and the *destination* network. Our cross-chain 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 conditional time-locked and publicly verifiable transfer-proof. We define this *transfer-proof* to be a committed cross-chain transaction. The conditional *time-lock* means that the transaction output is time-locked in the source network while the transaction is in transit. Finally, at the execute stage, upon presenting the transfer-proof, the destination network nodes verify the validity and correctness of the transfer-proof and execute the exchange. We assume that the recipient's network nodes can validate the transfer-proof through an intermediary middleware mechanism. The recipient can claim the transaction if it is within the conditional time-bound and the transfer-proof has been validated. In case of an unsuccessful transaction, after the expiry of time-bound, the sender is able to reclaim the asset.

### B. *Assumptions*

Our protocol is designed with the following assumptions:

- The underlying blockchain networks are secure with a concept of transaction finality within finite time, after which the transactions cannot be rolled back [4].
- The participants involved in the cross-communication process 'trust' each other to a required minimum level and are willing to process the transaction if valid proof is presented by the other party. The required level of trust varies depending on the application and to be agreeable to both parties.
- The network nodes are motivated to take part in the respective proof-mechanism (mining process) on both the chains (refer to the network assumption in subsection III-B1).
- A transaction output carry a single output which corresponds to a single asset and the networks involved recognize and form a common understanding of the assets they are holding.

Under these assumptions, once a transaction is broadcasted, 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 are constantly trying to produce new blocks and broadcasting their achievements to the entire network. In the case of cryptocurrencies, for instance, this behaviour is motivated by mining rewards. We assume there is an appropriate incentive mechanism to support participation by nodes here as well.

## II. Preliminaries

In the literature, there are slightly different definitions for blockchain operations based on the purpose of the application, but they usually do not cover all the assumptions of blockchain in our context. Therefore, in this section, we define the terminologies and general features of blockchain required to explain the aspects of cross-chain interoperability.

**Definition 1** (Block)**.** A block, written as $B$, represents a list of transactions $B$= $[Tx_1, Tx_2, Tx_3, \cdots]$.

A blockchain is a chain of blocks where each element of the block contains several transactions, a validation proof and some metadata which are omitted for simplicity.

**Definition 2** (Blockchain)**.** A blockchain, written as $C$, is a tuple ($G$,$B$) where $G$ is a genesis state and $B$ is block which consists of a list of transactions representing the global state. The blocks are cryptographically linked such that $H(B_1) \subset B_2$, $H(B_2) \subset B_3$ , $\cdots$, $H(B_{n-1}) \subset B_n$, where $H$ denotes a hash function.

**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 transaction function given as $\Pi$ takes the current state $Q$ and a block $B$ to produce new state $Q'$, is written as $\Pi(Q, B) = Q'$. In the context of a blockchain, a state is valid if it is accepted by the majority of nodes [25]. A block $B$ is valid in a state $Q$ if $B \in Q$ and $Q$ is valid in a blockchain. We assume that the state $Q$ is monotonic once it is settled on a blockchain $C$ with height $k$. We require that our state $Q$ only depends on the stable portion of the blockchain $C$, which are

Table I
SUMMARY OF NOTATIONS USED THROUGHOUT THIS PAPER

| Symbol | Description |
|---|---|
| $C$ | Blockchain |
| $B$ | Block |
| $Q$ | Blockchain state and $Q'$ a new state |
| $N_1$, $N_2$ | Blockchain network 1 and network 2 respectively |
| $\Pi$ | State transaction function |
| $v$ | Asset |
| $K_{adr}$, $\beta$ | Blockchain address and Burn address |
| $K_p$, $K_r$, $K_s$ | Public key, private key and secret key respectively |
| $S$, $R$ | Sender (Alice) and recipient (Bob) |
| $K_p^S$, $K_r^S$, $K_{adr}^S$ | Senders public key, private key and blockchain address respectively |
| $K_p^R$, $K_r^R$, $K_{adr}^R$ | Recipients public key, private key and blockchain address respectively |
| $\sigma$ | Digital signature |
| $Tx$, $Tx_t$, $Tx_e$, $Tx^\dagger$ | Transaction, exit transaction, entry transaction and previous transaction respectively |
| $\gamma$, $H(\gamma)$, $\Gamma$ | Secret code, hash of the secret code and encrypted secret code respectively |
| $t$, $t_{lock}$ | Time and time lock respectively |

the blocks that are buried under $k$ (recommended block height for security [28]) 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** (Address)**.** The blockchain *Address* schema is constructed from a key pair $(K_p,\, K_r)$ , where the public key $K_p$ is concatenated with the network identifier $N_{id}$ and subsequently hashed to generate a network specific address $K_{adr}$.

In a blockchain network, a user is defined by a unique *address* ($K_{adr}$). In this paper, $K_{adr}^S$ and $K_{adr}^R$ is used to represent the sender and the recipient's address respectively. However, this address is constructed from the user's public key ($K_p$) as shown in Algorithm 1, whereas the private key ($K_r$) is used to generate the digital signature [24]. 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 ($\sigma$) as an output. In contrast, the verification function $verify(M,\, K_p,\, \sigma)$ checks whether the signature $\sigma$ is valid for the message $M$ by using the sender's public key $K_p$.

Bitcoin and other similar cryptocurrencies use a transaction-output-based system where each transaction fully spends the outputs of previously unspent transactions. These unspent transaction outputs are called UTXOs. Bitcoin allows a single UTXO to be spent to many distinct outputs, each associated with a different address. To spend a UTXO a user must provide a signature, or more precisely a scriptSig, that enables the transaction SCRIPT to evaluate to be true. Apart from the validity of the scriptSig, miners verify that the transaction spends previously unspent outputs, and that the sum of the inputs is greater than the sum of the outputs.

For a given transaction $Tx$ from a public key $K_p$ referring to an address $K_{adr}$ and a signature $\sigma$, the verification function $verify(Tx,\, K_p,\, \sigma)$ verifies $\sigma$. Once $\sigma$ is verified, the address $K_{adr}$ needs to be verified via Algorithm 1 [8, 19].

**Algorithm 1** blockchain-address-protocol

```
1: function genAddress($K_p$)
2:     fingerPrint ← 0x00 || RIPEMD160(SHA256($K_p$))
3:     checkSum ← SHA256(SHA256(fingerPrint))
4:     $K_{adr}$ ← base58(fingerPrint || checkSum[:4])
5:     return $K_{adr}$
6: end function
```

Here, we present a generic blockchain address generation protocol [8, 19]. We assume an elliptic curve algorithm is used to generate $K_p$. Algorithm 1 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 the sake of simplicity, we combine the digital signature verification process and address generation process using a single function called **spendVerifier**, which 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 $K_{adr}$.

**Definition 5** (Transaction)**.** A transaction[1] $Tx$ is a tuple $(K_p^S,\, K_{adr}^S,\, K_{adr}^R\; v,\, Tx^\dagger)$ where $v$ is an asset and $Tx^\dagger$ is a previous transaction where $v$ is transferred from. Each $Tx$ encode transaction value and relevant parameters associated with it.

The *transaction value* includes the asset and any related transaction fee, which is the fee a spender offers to a miner node who successfully includes that transaction in a block. The relevant parameters include time, date and other metadata information which are ignored for clarity.

[1]As in the literature [7], we loosely use "transaction" to represent both a piece of data and an operation.

A typical transaction protocol in blockchain is given in Algorithm 2, where a valid transfer of $v$ from $K^S_{adr}$ to $K^R_{adr}$ is defined as $K^S_{adr} \rightarrow K^R_{adr} : v$. The symbol : denotes asset $v$ is associated with the address $K^R_{adr}$.

**Algorithm 2** transaction-protocol

```
1: function transactionProtocol($Tx(K^S_p$,$K^S_{adr}$,$K^R_{adr}$,$v$,$Tx^\dagger$),$\sigma$)
2:     if spendVerifier($K^S_p$, $K^S_{adr}$, $\sigma$) and
3:         assetVerifier($Tx^\dagger$, $K^S_{adr}$) is true then
4:         $K^S_{adr} \rightarrow K^R_{adr} : v$
5:     else
6:         invalid transaction
7:     end if
8:     return transaction receipt
9: end function
```

A transfer will only occur if spendVerifier and assetVerifier both return true. The function assetVerifier [2, 25] first checks the validity of the asset $v$ by ensuring that the address $K^S_{adr}$ carries (enough amount of) the asset. We refer to this first step as the balance function *balance*($K^S_{adr}$) which returns $true$ if it is valid, $false$ if it is not valid and $burn$ if it has transferred the asset to a burn-address. Then the function checks whether this $Tx$ referred by the previous transaction $Tx^\dagger$ is included a valid block $Tx^\dagger \in B$ (validate with a Merkle Tree proof[2]) which is in a valid state (check the longest chain). Every transaction returns a transaction receipt which encodes transaction information from its execution that are useful form a zero-knowledge proof, or to search from the index.

## III. The Burn-to-Claim protocol

In this section we formally define the Burn-to-Claim protocol and its primitives.

### A. Communication between networks

The Burn-to-Claim protocol consists of two main functions for the communication between networks: **exitTransaction** defined in Algorithm 3 to lock the asset and serve as a transfer-proof in the source network, **entryTransaction** defined in Algorithm 4 to verify the validity of the transfer-proof in order to re-create the asset in the destination network.

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 [19, 29], which presents a mechanism where the sender transfers the asset to a non-spendable burn-address and is able to present that transaction as a proof for the locked asset. The proof-of-burn protocol is a more energy-saving alternative to the proof-of-work protocol. In proof-of-burn based systems, for security guarantee, cryptocurrencies are intentionally "burned" and the process does not require powerful mining hardware.

**Definition 6** (Burn-address)**.** A burn-address given as $\beta$ is an address to which one can send assets, but from where they can never be recovered because there is no private key corresponding to that address.

The process of burning consists of sending cryptocurrencies to an address where they become inaccessible and useless. Typically, these addresses are randomly generated, where the addresses do not have a corresponding private key therefore the asset at those addresses are not spendable. The burn protocol [19] presents proofs such that if the underlying cryptographic is secure then the probability of finding a private key for a given burn address is nearly negligible.

*1) Exit transaction:* The exit-transaction must be initiated on the source network by the sender. This execution checks the validity of the transaction and generates a transfer-proof. The transaction-validity process checks the authenticity of the asset and the owner's ability to spend. The transfer-proof generator produces a proof that the asset exists and it is locked while the asset is in transit. This transaction aims to create an exit proof for that asset in the source blockchain network. Having an exit proof created by the system as part of the protocol will retain the system's security. Moreover, the network comes to a consensus about the asset transfer, thereby the authenticity of the information is satisfied.

In our protocol, exitTransaction uses a conditional time-lock with a 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 past[3].

Our protocol requires the sender to generate a secret code $\gamma$ using a random key generator function, $keyGen()$. Then the secret code $\gamma$ is encrypted using the public key of the recipient $K^R_p$ before sharing with her. The encrypted secret code is denoted by $\Gamma$. We assume the public key information is shared among users during the preparation stage.

**Algorithm 3** exit-transaction-protocol

```
1: function exitTransaction($Tx(K^S_p$,$K^S_{adr}$,$\beta$,$v$,$Tx^\dagger$),$H(\gamma)$,$\sigma$)
2:     if exportVerifier($K^S_{adr}$, $v$, $\sigma$) is true then
3:         $K^S_{adr} \rightarrow \beta : v$
4:         $v$ is timelocked($v$, t) at $\beta$ in $N_1$
5:         $Tx_t \leftarrow (Tx, H(\gamma), \sigma)$
6:     else
7:         invalid transaction
8:     end if
9:     return transaction receipt
10: end function
```

We design our transaction protocol based on the the blockchain transaction protocol in Algorithm 2. The function exitTransaction defined in Algorithm 3 takes a tuple of $(Tx(K^S_p, K^S_{adr}, \beta, v, Tx^\dagger), H(\gamma), \sigma)$ as inputs where $Tx$ includes the sender's public key ($K^S_p$) and address ($K^S_{adr}$), burn-address ($\beta$), asset ($v$) and the previous transaction ($Tx^\dagger$)

[2]A cryptographic hash tree where every leaf node hash of a data block.

[3]https://bcoin.io/guides/cltv.html

in which the asset $v$ was spent. $H(\gamma)$ represents the hash value of secret code $\gamma$ and $\sigma$ is the digital signature. We denote the data structure of exit-transaction as $Tx_t$, which is a tuple ($Tx$, $h$, $\sigma$) of a transaction, a hash value and a signature.

**Definition 7** (exportVerifier)**.** The exportVerifier function consists of two sub-functions spendVerifier and assetVerifier, it returns true when both sub-functions return true, and it returns false otherwise.

- spendVerifier:
  - returns true if for the given $K^S_{adr}$ from $K^S_p$ the function verify($Tx$, $K^S_p$, $\sigma$) returns true and getAddress($K^S_p$) returns $K^S_{adr}$;
  - returns false otherwise.
- assetVerifier:
  - returns true if balance($K^S_{adr}$) is true and $Tx^\dagger \in B$ and $B \in Q$;
  - returns false otherwise.

Each node performs the mining process. If exportVerifier returns true then the transaction executes the transfer of the asset to the given burn-address $\beta$ with conditions such that the asset is time-locked within the source network for a determined time-lock period $t$ and the hashed secret code $H(\gamma)$ is added to the data structure of the transaction.

*2) Entry transaction:* The purpose of the entry-transaction is to recreate the asset in the destination network. An entry-transaction must be initiated in the destination network by the recipient. Upon initiating the entryTransaction with the transfer-proof from source chain, the network nodes verify the validate of the transfer-proof and re-create the asset.

**Algorithm 4** entry-transaction-protocol

```
1: function entryTransaction(($Tx(K^R_p$,$K^R_{adr}$,$\beta$,$v$,$Tx^\dagger$),$\Gamma$,$\sigma$)
2:     if importVerifier($Tx$, $Tx^\dagger$,$\sigma$) and
3:         ($Tx^\dagger$.time-lock is under the time limit) and
4:         decrypt($\Gamma$,$K^R_r$) = $Tx^\dagger.H(\gamma)$) is true then
5:         $\beta \rightarrow K^R_{adr} : v$
6:         $Tx_e \leftarrow (Tx, H(\gamma), \sigma)$
7:     else
8:         invalid transaction
9:     end if
10:    return transaction receipt
11: end function
```

The function entryTransaction defined in Algorithm 4 takes a tuple ($Tx$ ($K^R_p$, $K^R_{adr}$, $\beta$ $v$, $Tx^\dagger$), $\Gamma$, $\sigma$) as input where the $Tx$ includes the recipient public key $K^R_p$, the recipient address $K^R_{adr}$, the burn-address $\beta$, the asset $v$ and the previous transaction $Tx^\dagger$. The data structure of entry-transaction is written as $Tx_e$, which is also a tuple ($Tx$, $h$, $\sigma$) of a transaction, a hash value and a signature. A $\Gamma$ represent the encrypted version[4] of hash of the secret code and $\sigma$ denotes the digital signature.

[4]Encrypted with recipient private key $\Gamma$ = encrypt($K^R_r$,$\gamma$) and decrypt($\Gamma$,$K^R_p$) = $\gamma$

**Definition 8** (importVerifier)**.** The importVerifier consists of two functions spendVerifier and proofVerifier, it returns true when both sub-functions return true, and false otherwise.

- spendVerifier:
  - returns true if for the given burn-address $\beta$ from $K^R_p$ the function verify($Tx$,$K^R_r$, $\sigma$) returns true and getAddress($K^R_p$) returns $K^R_{adr}$ (assuming the node use a source chain version of getAddress function);
  - returns false otherwise.
- proofVerifier:
  - returns true if $\beta$ is generated from $K^R_{adr}$, balance($\beta$) = $burn$, $Tx^\dagger \in B$, and $B \in Q$;
  - otherwise returns false.

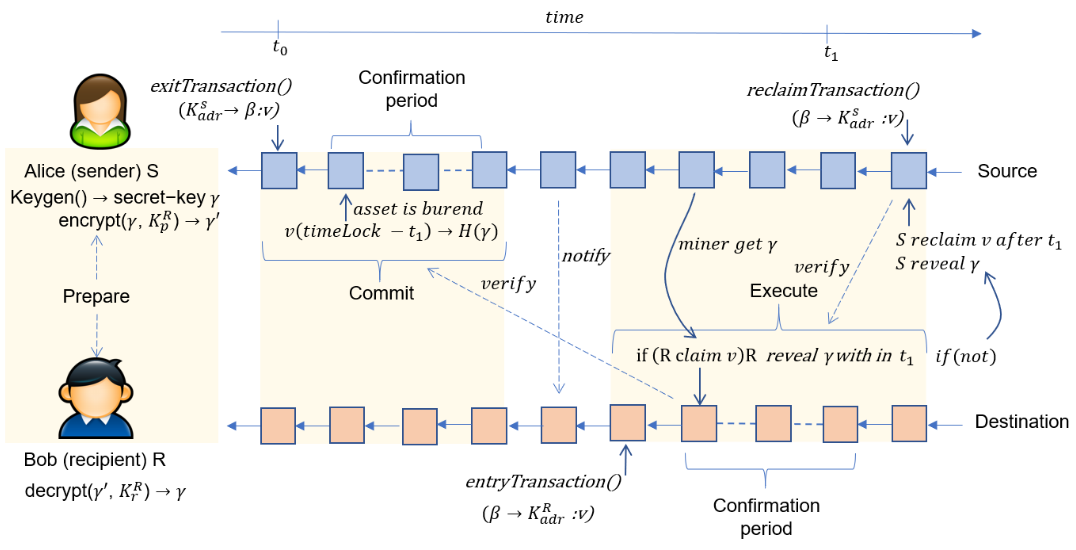
A proofVerifier is an extended version of assetVerifier. Here the nodes on the destination network need to verify the proof from the source network. We assume that through the gateway mechanism, the destination network nodes are able to verify the proof. Through the $Tx^\dagger$ any gateway node can access the specific transaction in the source network. Once the importVerifier returns true, the mining nodes need to check the time-lock and the secret code. We assume that both the source and the destination network run on a global time zone. If the time is under 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 $K^R_{adr}$.

## B. Workflow of the protocol

This section presents a walk-through of the workflow of the burn-to-claim protocol. We begin with a use case of two blockchain systems which are self-sufficient and secure. The two networks run different applications but they want to interoperate. These networks may have distinct consensus participants that employ different agreement protocols. It is assumed that the majority of consensus participants in both networks are honest. Here the assumption is that even though these systems are not connected, they have enough credibility which is governed by a protocol and have some top-level relation. 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 paper is to address the cross-blockchain transaction proof-problem. Therefore, we focus on the construction of consensus on how the transactions are verified, and on what conditions the transactions are valid.

*1) Network assumptions:* To address the state proof-problem, we made some assumptions about the network participants and their ability to mine. We assume that the cryptographic primitives of the networks are secure. For the underlying network, we make the same assumptions as in prior work [9, 20, 22, 32]. For example, a presents of a global clock and the honest nodes are well connected with the able to communicate with each other.

In the network of our model, elected nodes function as gateway nodes. We envision each blockchain as an autonomous system, which communicates with each other via a



time
$t_0$
$t_1$
exitTransaction()
$(K^s_{adr} \rightarrow \beta{:}v)$
Confirmation period
reclaimTransaction()
$(\beta \rightarrow K^s_{adr} :v)$
Alice (sender) S
Keygen() → secret−key $\gamma$
encrypt($\gamma$, $K^R_p$) → $\gamma'$
Source
asset is burend
$v(timeLock - t_1) \rightarrow H(\gamma)$
notify
miner get $\gamma$
verify
S reclaim v after $t_1$
S reveal $\gamma$
Prepare
Commit
verify
Execute
if (R claim v)R reveal $\gamma$ with in $t_1$
if (not)
Destination
Bob (recipient) R
decrypt($\gamma'$, $K^R_r$) → $\gamma$
entryTransaction()
$(\beta \rightarrow K^R_{adr} :v)$
Confirmation period

Figure 1. A high level overview of the Burn-to-Claim protocol workflow. The boxes represent chained blocks along the time steps in the source and destination networks. At the **preparation** stage, $S$ generates and shares an encrypted secret code encrypt($\gamma$) with $R$. At the **commit** stage, $S$ initiates the exitTransaction, burning the asset to $\beta$ generated from $R$'s address, then time-locks the transaction output with the hash value of $H(\gamma)$. After confirmation period $R$ gets notified with the previous transaction $Tx^{\dagger}$. At the **execution** stage, $R$ initiates exitTransaction, claiming asset $v$. To do that $R$ must prove the relation with $\beta$ and reveal $\gamma$ within the time-lock period. If $R$ failed to claim the $v$, $S$ reclaims the $v$ after the time-lock period.

cross-blockchain protocol. To facilitate communication among blockchains, the nodes can rely on decentralised integration mechanisms that can identify and address cross-chain 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 (depends upon the topology) connected through a gateway mechanism to create a network-of-networks (NON) [11].

There will be multiple nodes which function as gateways on the same network; therefore, they will be competing against each other for the mining reward. We assume that not all the nodes will verify the transaction, but a sufficient number of them must do to satisfy the security of the system. The other nodes will accept a gateway node's proposal based on the credibility of the gateway node in the network. We leave further discussions on a decentralised integration mechanisms for future work.

*2) Example:* We set an example where Alice (the sender $S$ in the blockchain network $N_1$) wants to transfer an asset to Bob (the recipient $R$ in the blockchain network $N_2$). Alice first submits an exitTransaction to network $N_1$. Network $N_1$ executes the transaction, effectively burns Alice's asset in network $N_1$. Bob notices that the asset has been burned successfully and submits the proof of burn to network $N_2$. Network $N_2$ verifies the proof and if successful, recreates the asset and assigns it to Bob's account. Figure 1 shows a brief overview of the protocol construction.

*3) Preparation stage:* In the preparation stage, First, Alice $S$ and Bob $R$ shall agree on a shared secret code $\gamma$ via asymmetric cryptography. The $S$ generates the $\gamma$ and encrypts the $\gamma$ with the recipient's $K^R_p$ and then shares the encrypted secret code $\Gamma$ with the recipient; Secondly, they must agree on the time-lock period $t$; Thirdly, Alice $S$ need to generate the burn address.

We now discuss the use and importance of these parameters.

In order to spend value in an address $K_{adr}$, a user needs to prove the ownership of the public key that is 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($K^R_{adr}$) and returns $\beta$. This will guarantee that the address $\beta$ does not have a private key in the source network and the $\beta$ is more specific to the recipient $R$ because it is generated from $R$'s address. Therefore, we can use such a transfer to burn address as a valid proof.

Now that we have the proof we need to add a provision for retrieving the asset in case of an unsuccessful transfer. For instance, the asset is burned on the source chain and not recreated in the recipient network, or the asset is sent to a wrong address. We name this property as reclaim-from-burn and explain the process in Algorithm 6. To ensure this correctness of this property, we impose a time-lock on transaction output in the source network. This will stop the sender from claiming the asset before the recipient claims the asset. The time-lock period is agreed based on factors such as the network latency and consensus timing, which we refer to as transit-time.

We also need to consider the double-spending problem where a dishonest sender commits exit-transaction on the same asset multiple times [18]. In order to prevent double-spending, our protocol requires that the sender and the recipient agree on a shared secret code $\gamma$ which is hashed and encrypted using the recipient's public key. This $\gamma$ is not known to the network. As a result, only the user who has the recipient's private key can decrypt and compare the hash value, which is required to claim the asset. Once secret $\gamma$ reveled no one can claim the asset.

*4) Commit stage:* In the commit stage, $S$ creates and broadcasts an exitTransaction to invoke Algorithm 4 in the network $N_1$.

Algorithm 5 shows the transfer condition logic. The condition here is that anyone claiming the transaction output (includes the asset and the miner's fee) must reveal the secret code. Therefore, the transaction will be mined by the miners who has access to the destination network's data because they need to get the secret code to claim the fee. That means if there is no valid[5] gateway node then this transaction will not go through in the first place.

**Algorithm 5** exit-transaction-time-key-lock condition

```
 1: while (time-lock is true) do
 2:     if ($R$ claim $v$) then
 3:         $R$ reveal secret $\gamma$ to $N_2$
 4:         $N_1$.miner gets $\gamma$ from $N_2$
 5:         $N_1$.miner claims his $\gamma$-locked fee
 6:                                  ▷ secret $\gamma$ is known to $N_1$ & $N_2$
 7:     end if
 8: end while (after the $t_{lock}$)
 9: if ($R$ fail to claim $v$) then
10:     $S$ reveals $\gamma$ to $N_1$
11:     $S$ re-claim $v$
12:     $N_1$.miner claim his $\gamma$-locked fee
13:                                  ▷ secret $\gamma$ is known to $N_1$
14: end if
```

The time-key-lock condition mechanism allows only one party to claim the asset at a time. For example, while the $Tx_t$ output is time-locked in the source network, if $R$ reveals the secret code $\gamma$ in his network, the miner who mined this transaction in the source network will reveal it in source network to claim his fees; thereby the $\gamma$ 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 $\gamma$ after $t_{lock}$ in the source network to reclaim the asset. In our design $\gamma$ is not known to the network; therefore, once the $\gamma$ is revealed nobody will be able to claim the asset even after $t_{lock}$.

*5) Execution stage:* The execution 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 to $N_2$. If $R$ can prove the ownership of $K_r^R$ 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 $\beta$ and a previous transaction $Tx^\dagger$, which form the proof of commit in the source chain.

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

As specified in subsection III-B1, our protocol requires a mining process where some nodes are able to mine in multiple blockchains. With that requirement, among $n$ number of nodes and $m$ gateway nodes propose this transaction and eventually one proposal will be accepted by the network. We assume that these gateway nodes are able to validate the transfer-proof with $Tx^\dagger$. The nodes verify that transaction occurred on the source chain by ensuring that the transaction is contained in a block. This can be done by checking the chain validity [12] of the block and the transaction.

If importVerifier returns true the network awards the asset $v$ to the recipient address $K_{adr}^R$. That means by now the secret code $\gamma$ is known to the network, and after that, the source network miner will take it to the source network to claim his fee, and then $\gamma$ is known to both the networks.

*b) $S$ reclaims the asset:* In case the recipient has not claimed the asset within the time-lock period, the sender gets notified by the network or the mining node who mined the $Tx_t$, and the sender then invoke reclaimTransaction, which is is a variant of entryTransaction and is defined in Algorithm 6, to claim the asset back. The transaction first checks the signature via verify($Tx$, $K_{adr}^S$, $\sigma$), 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.

**Algorithm 6** reclaim-transaction-protocol

```
 1: function reclaimTransaction(($Tx(K_p^S,K_{adr}^S,\beta,v,Tx^\dagger)$),$\Gamma$,$\sigma$)
 2:     if reclaimVerifier() is true and
 3:         ($Tx^\dagger.t_{lock}$ pass the time limit) and
 4:         decrypt($\Gamma$,$K_p^S$) = $Tx^\dagger.H(\gamma)$) is true then
 5:         $\beta \rightarrow K_{adr}^S : v$
 6:     else
 7:         invalid transaction
 8:     end if
 9:     return transaction receipt
10: end function
```

The function of reclaimVerifier consists of two subfunctions spendVerifier and proofVerifier. The function check if verify($Tx$, $K_r^R$, $\sigma$) returns true and getAddress($K_r^R$) returns $K_{adr}^R$. Then the proofVerifier checks the transfer proof of $Tx^\dagger$ referring to if the balance($\beta$) = $burn$ and $Tx^\dagger \in B$ and $B \in Q$ return true else return false. The reclaimTransaction can be included with the entryTransaction but for clarity, we present it as a separate transaction.

## IV. Analysis

In this section, we analysis the correctness and security of the Burn-to-Claim protocol. Cross-chain data guaranty/trust 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. But to build trust about the shared information we must resolve specific properties of the individual transactions involved in the exchange process; that is, security: a cryptographic assurance of transfer commitment of transactions; correctness: each successful transaction commits only one valid outcome and fairness: either the transfer executes the transfer of an asset or return the asset [4, 19, 27, 31].

In our protocol the burn-address is generated uniquely for each input (recipient address) therefore it is clear that it is not generated from a regular keypair. Given that it is manually crafted from the recipient address which is not part of the source network it is not spendable in the current network. Therefore, with the signature scheme of the underlying cryptocurrency, the asset burned through our scheme would remain unspendable.

We now present some lemmas to establish the security of the protocol.

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

The main functionality of entryTransaction is to recreate 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. With the burn protocol and its property of unspendability the asset is permanently burned before claiming on the source chain.

**Lemma 2** (Burn before claim)**.** *An asset $v$ which is transferred from $N_1$ to $N_2$ must be burned in $N_1$ before a user can claim it in $N_2$.*

The sender who initiates the exit-transaction must own the asset he is transferring. In exitTransaction the exportVerifier() checks the transaction validity and owner's ability to spend. This process must be carried by each mining node and must reach the consensus of the network. Therefore, as long as the network is secure, the participants can only exchange the asset of their own.

**Lemma 3** (Ownership)**.** *The function exitTransaction can only be successfully executed if the sender owns the asset.*

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

**Theorem 4** (Security)**.** *The recipient network can rely on the Burn-to-Claim exit-proof guarantee provided by the source network.*

*Proof.* (Outline.) Based on Lemma 2, exitTransaction burns the asset to a burn-address $\beta$ which is unspendable as per Lemma 1. The function 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 an $Tx_t$ successfully executes a transfer $K^S_{adr} \rightarrow \beta : v$ and includes in a block $B$ then is executed by a state transaction function $\Pi$ ($Q$, $B$) = $Q'$ then the out put of the new state $Q'$ is deemed as a valid proof. □

Now we analyse the correctness of the our protocol.

The function exitTransaction transfers the asset $v$ to a burn-address $\beta$ that is derived from the recipient $R$'s address $K^R_{adr}$. To claim $v$, the recipient must prove to the network that $\beta$ is derived from an address he owns. The function spendVerifier checks the signature $\sigma$ to verify the $K_p$ and checks the whether $\beta$ is derived from the given $K_p$ using the function getAddress. Therefore, only the user who owns a private key associated with $K^R_{adr}$ can make the claim of the asset in the destination network.

**Theorem 5** (Correctness)**.** *The exchange operation only exchange an asset to a correct recipient.*

*Proof.* (Outline.) Based on the function exitTransaction of Algorithm 4 the recipient's signature must be correct, then the relation with the $\beta$ must be correct. Finally the recipient must be able to decrypt the encrypted secret code $\Gamma$ using his private key via decrypt($\Gamma$,$K^R_p$). All these measures point out that the $Tx_t$ will only transfer asset to the correct recipient. □

We assume that no adversary can obtain the private key of a secure signature scheme, except with negligible probability. As a result, the correctness of our protocol is dependent on a probabilistic polynomial-time adversary can decipher the key. Let $\sigma$ be a secure signature scheme then the possibility of a distribution entity to decipher the $K_r$ from a $K_p$ or $\beta$ is unpredictable.

Now we analyse the fairness property of the our protocol.

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

Fairness in our context means that both parties receive the item they expect or neither do. The fairness theorem only relies on the fact that a standard hash function is collision resistant. By using hash preimage as the secret code of the conditional payment, the atomicity of the transactions can be guaranteed without the participation of a trusted third parties, so as to realise fair cross-chain exchange.

**Theorem 6** (Fairness)**.** *The exchange operation should only execute one outcome, either the exchange succeeds and the asset is transferred to the recipient, or it is failed and the asset returns to the sender.*

*Proof.* (Outline.) If the recipient $R$ claimed the asset $v$, it must be that $R$ has revealed the secret code $\gamma$. Therefore, $\gamma$ is known to the destination network $N_2$. Now the miner who processes the exit-transaction in source network $N_1$ needs $\gamma$ to get the fee. The miner gets $\gamma$ from $N_2$ to $N_1$ to claiming his fees. Thereafter $\gamma$ is known to both $N_2$ and $N_1$ and no one will be able to claim the asset.

If $R$ fails to claim the asset within the time-lock, $S$ would invoke reclaimTransaction defined in Algorithm 6 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 $\gamma$ 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 is also able to reclaim the asset. With the help of time-lock and the secret code, the protocol can only reach one of the outcomes stated in this theorem. □

## V. Related work

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

Blockchain interoperability is often considered as one of the prerequisites for the massive adoption of blockchains. The existing approaches focus on asset transfer between blockchains using trustless exchange protocols, including side-chains [13, 21], atomic cross-chain swaps [15], and cryptocurrency-backed assets [32]. 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. The cryptocurrency-backed asset transfer protocol relies on a set of reliable smart contracts to control the underlying process.

There is a number of efforts addressing interoperability [16, 27] with different integration architecture. Although they differ in architectures, most of them follow the approach of using the source chain to generate a transfer-proof and the destination chain to recreate the asset based on the transfer-proof. The cross-chain transfer-proofs are either a smart contract [27] or a notary schema [16]. Generally, these approaches makes significant assumptions. For example, in a smart contract holds the transfer proof the verifying smart contract able to trust the communicate with the verifier smart contract. In a notary schema, a pre-defined set of validators is trusted with controlling over assets on the transferring chain. Some other projects [1, 17, 30] are focused on the building of a completely new ecosystem for interoperable blockchains rather than enabling interoperability between existing blockchains.

The proof-of-burn protocol is proposed by [19, 27]. In [19] 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 [19] is the absence of a mechanism to return the asset back to the source chain in case of an unsuccessful transfer. Whereas in [27] the sender on the source chain invokes a burn-transaction to destroy his coin which will be recorded in the smart contract on the source chain. After that, anyone 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, a user invokes the asset management smart contract on the destination network and provides the proof-of-burn transaction reference on the smart contract. Compared to those methods, our work uses a direct reference of the recipient address in the exit transaction which guaranty the correctness property we defined. The time-lock along with secret code and re-claim mechanism satisfies the fairness property. Table II gives a summary of the protocols in related work and an analysis on the properties of interest.

Table II
ANALYSIS SUMMARY

| Reference | Security | Correctness | Fairness |
|---|---|---|---|
| Burn protocol [19] | yes | yes | no |
| Sigwart et al.[27] | yes | low | yes |
| Burn-to-Claim | yes | yes | yes |

## VI. Conclusion and future works

In this paper, we analyse the blockchain transaction protocol and propose to add new features that help solve the problem of interoperability. One of the critical features of our method is that it presents an internal functionality for value/asset exchange. An internal protocol provides a universal language, and via such a protocol, blockchain users can communicate directly and transfer various forms of data via standardised pathways. Furthermore, we briefly showed that Burn-to-Claim protocol is resilient to double-spending by its correctness and fairness properties.

We formalise the cross-blockchain proof-problem and show that it can be solved without a trusted third party, as compare to the assumptions often made in the blockchain community. Our approach is to introduce an built-in function to address the proof-problem. However, it often requires changes in the target blockchains, as most blockchains do not support locking or burning assets.

Future work involves the implementation of the Burn-to-Claim protocol which will show how the protocol will hold up in practice. Further investigations on selfish mining [10] and gateway incentive mechanism will also be conducted.

## References

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, pages 1–15, 2018.

[2] Nicola Atzei, Massimo Bartoletti, Stefano Lande, and Roberto Zunino. A formal model of bitcoin transactions. In *International Conference on Financial Cryptography and Data Security*, pages 541–560. Springer, 2018.

[3] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *arXiv preprint arXiv:2005.14282*, 2020.

[4] Marianna Belotti, Stefano Moretti, Maria Potop-Butucaru, and Stefano Secci. *Game theoretical analysis of Atomic Cross-Chain Swaps*. PhD thesis, Caisse des

dépôts-Institut pour la recherche et Banque des territoires, 2019.

[5] Michael Borkowski, Daniel McDonald, Christoph Ritzer, and Stefan Schulte. 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*, 2018.

[6] Michael Borkowski, Christoph Ritzer, Daniel McDonald, and Stefan Schulte. Caught in chains: Claim-first transactions for cross-blockchain asset transfers. *Technische Universität Wien, Whitepaper*, 2018.

[7] Vitalik Buterin, Jeff Coleman, and Matthew Wampler-Doty. Notes on scalable blockchain protocols (verson 0.3), 2015.

[8] Dmitry Ermilov, Maxim Panov, and Yury Yanovich. Automatic bitcoin address clustering. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 461–466. IEEE, 2017.

[9] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert Van Renesse. Bitcoin-ng: A scalable blockchain protocol. In *13th symposium on networked systems design and implementation*, pages 45–59, 2016.

[10] Ittay Eyal and Emin Gün Sirer. Majority is not enough: Bitcoin mining is vulnerable. In *International conference on financial cryptography and data security*, pages 436–454. Springer, 2014.

[11] Jianxi Gao, Daqing Li, and Shlomo Havlin. From a single network to a network of networks. *National Science Review*, 1(3):346–356, 2014.

[12] Juan Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 281–310. Springer, 2015.

[13] Peter Gaži, Aggelos Kiayias, and Dionysis Zindros. Proof-of-stake sidechains. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 139–156. IEEE, 2019.

[14] Thomas Hardjono, Alexander Lipton, and Alex Pentland. Towards a design philosophy for interoperable blockchain systems. *arXiv preprint arXiv:1805.05934*, 2018.

[15] Maurice Herlihy. Atomic cross-chain swaps. In *Proceedings of the 2018 ACM symposium on principles of distributed computing*, pages 245–254, 2018.

[16] Yiming Jiang, Chenxu Wang, Yawei Wang, and Lang Gao. A cross-chain solution to integrating multiple blockchains for iot data management. *Sensors*, 19(9):2042, 2019.

[17] Luo Kan, Yu Wei, Amjad Hafiz Muhammad, Wang Siyuan, Gao Linchao, and Hu Kai. 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, 2018.

[18] Ghassan O Karame, Elli Androulaki, and Srdjan Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917, 2012.

[19] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof-of-burn. In *International Conference on Financial Cryptography and Data Security*, 2019.

[20] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Annual International Cryptology Conference*, pages 357–388. Springer, 2017.

[21] Aggelos Kiayias and Dionysis Zindros. Proof-of-work sidechains. In *International Conference on Financial Cryptography and Data Security*, pages 21–34. Springer, 2019.

[22] Eleftherios Kokoris-Kogias, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ewa Syta, and Bryan Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 583–598. IEEE, 2018.

[23] Pascal Lafourcade and Marius Lombard-Platet. About blockchain interoperability. *Information Processing Letters*, page 105976, 2020.

[24] Victor S Miller. Use of elliptic curves in cryptography. In *Conference on the theory and application of cryptographic techniques*, pages 417–426. Springer, 1985.

[25] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. Technical report, Manubot, 2019.

[26] Babu Pillai, Kamanashis Biswas, and Vallipuram Muthukkumarasamy. Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review*, 35, 2020.

[27] Marten Sigwart, Philipp Frauenthaler, Christof Spanring, and Stefan Schulte. Decentralized cross-blockchain asset transfers. *arXiv preprint arXiv:2004.10488*, 2020.

[28] Yonatan Sompolinsky and Aviv Zohar. Bitcoin's security model revisited. *arXiv preprint arXiv:1605.09193*, 2016.

[29] Iain Stewart. Proof of burn - bitcoin wiki. Available at: https://en.bitcoin.it/wiki/Proof_of_burn, Dec 2012.

[30] Gavin Wood. Polkadot: Vision for a heterogeneous multi-chain framework. *White Paper*, 2016.

[31] Alexei Zamyatin, Mustafa Al-Bassam, Dionysis Zindros, Eleftherios Kokoris-Kogias, Pedro Moreno-Sanchez, Aggelos Kiayias, and William J Knottenbelt. Sok: Communication across distributed ledgers. Technical report, IACR Cryptology ePrint Archive, 2019: 1128, 2019.

[32] Alexei Zamyatin, Dominik Harz, Joshua Lind, Panayiotis Panayiotou, Arthur Gervais, and William Knottenbelt. Xclaim: Trustless, interoperable, cryptocurrency-backed assets. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 193–210. IEEE, 2019.

[33] Kaiwen Zhang and Hans-Arno Jacobsen. Towards dependable, scalable, and pervasive distributed ledgers with blockchains. In *ICDCS*, pages 1337–1346, 2018.