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 communication between trustless 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.**

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

I. Introduction

Blockchain technology offers an immutable, decentralized, and transparent mechanism for transaction processing. 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 same network. A blockchain application cannot uniformly use multiple networks and obtain a composition of their guarantees [1]. A core reason of this issue is that each blockchain network has its own state assumptions of the proof-problem [3], and without restrictions, one network cannot verify the information in a different network. Therefore, currently, seamless composition between two blockchain networks is difficult [12].

The cross-blockchain integration would enable the interoperability among distinct, potentially heterogeneous, blockchains [8]. However, with the current protocols of blockchain systems, it is difficult to have direct interoperability between systems [4]. There is no method in the system to provide a cross-chain value transfer; therefore, external third-party services are the preferred solution [1, 12]. However, the blockchain system cannot recognise and verify any such process carried out by a third-party provider. Today most enterprise blockchain applications reside on private blockchains such as Hyperledger and Corda and unable to leverage the full potential of the distributed ledger technology.

To address the above issues, we propose a built-in process that is integrated with the protocol to carry out transactions and facilitate interoperability between systems. We first formalise the concept of interoperability from our previous work [13] and then 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 and an *entryTransaction* to verify the validity of the proof in order to re-create the asset. The contribution of this paper are threefold:

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

A. Assumptions

The following assumptions are made in the proposed protocol:

- The underlying blockchain networks are secure with a concept of transaction finality within finite time, after which the transactions cannot be rolled back [2].
- 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 agreed by both parties.
- The network nodes are motivated to take part in the respective proof-mechanism (mining process) on both the chains.
- 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 transacting.

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 constantly try to produce new blocks and broadcast their achievements to the entire network. In the case of cryptocurrencies, for instance, this behaviour is motivated by mining rewards. Here also we assume there is an appropriate incentive mechanism to support participation by nodes.

II. The Burn-to-Claim protocol

In this section we formally define the proposed Burn-to-Claim protocol and its primitives. A summary of notations used in this paper is available under this Git repository[1].

A. Communication between networks

The Burn-to-Claim protocol consists of two main functions for the communication between networks: **exitTransaction** which locks the asset and serves as a transfer-proof in the source network, **entryTransaction** which verifies the validity of the transfer-proof in order to re-create the asset in the destination network.

In our protocol, the sender who wants to transfer the asset must present a proof that the asset is locked. To achieve this, we adopt the proof-of-burn protocol [10, 14], 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.

Definition 1 (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 there is no private key corresponding to that address.

The process of burning consists of sending crypto asset 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 [10] presents proofs such that if the underlying cryptographic scheme 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 ensure 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 elapsed[2].

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

Algorithm 1 exit-transaction-protocol

```
function exitTransaction(Tx(K_p^S,K_adr^S,β,v,Tx†),H(γ),σ)
    if exportVerifier(K_adr^S, v, σ) is true then
        K_adr^S → β : v
        v  is timelocked(v, t) at β in N_1
        Tx_t ← (Tx, H(γ), σ)
    else
        invalid transaction
    end if
    return transaction receipt
end function
```

The function exitTransaction defined in Algorithm 1 takes a tuple of ($Tx(K_p^S, K_{adr}^S, \beta, v, Tx^\dagger)$, $H(\gamma)$, σ) as inputs where Tx includes the sender's public key (K_p^S) and address (K_{adr}^S), burn-address (β), asset (v) and the previous transaction ($Tx^\dagger$) in which the asset v was spent. $H(\gamma)$ represents the hash value of secret code γ and σ is the digital signature of the Tx .

Definition 2 (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_{adr}^S from K_p^S the function verify(Tx, K_p^S, σ) returns true and getAddress(K_p^S) returns K_{adr}^S;
  - returns false otherwise.
- assetVerifier:
  - returns true if balance(K_{adr}^S) is true and $Tx^\dagger$ is included in a valid state;
  - returns false otherwise.

If exportVerifier returns true then the transaction executes the transfer of the asset to the given burn-address β with conditions such that the asset is time-locked within the source network for a defined 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 validity of the transfer-proof and recreate the asset.

[1] https://github.com/b-pillai/burn-to-claim

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

Algorithm 2 entry-transaction-protocol

function entryTransaction(($Tx(K_p^R$,K_{adr}^R,β,v,$Tx^\dagger$),Γ,σ)
  if importVerifier(Tx, $Tx^\dagger$,σ) **and**
    ($Tx^\dagger$.time-lock is under the time limit) **and**
    decrypt(Γ,K_r^R) = $Tx^\dagger.H(\gamma)$) is true **then**
    $\beta \rightarrow K_{adr}^R : v$
    $Tx_e \leftarrow (Tx,\ H(\gamma),\ \sigma)$
  else
    invalid transaction
  end if
  return transaction receipt
end function

The function entryTransaction defined in Algorithm 2 takes a tuple (Tx (K_p^R, K_{adr}^R, β v, $Tx^\dagger$), Γ, σ) as input where the Tx includes the recipient public key K_p^R, the recipient address K_{adr}^R, the burn-address β, the asset v and the previous transaction $Tx^\dagger$; Γ represents the encrypted secret code and σ denotes the digital signature.

Definition 3 (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 β from K_p^R the function verify(Tx,K_r^R, σ) returns true and getAddress(K_p^R) returns K_{adr}^R (assuming the node use a source chain version of getAddress function);
  - returns false otherwise.
- proofVerifier:
  - returns true if β is generated from K_{adr}^R, balance(β) = $burn$, $Tx^\dagger$ is included in a valid state;
  - 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 clock. If the time is under the time-lock period and the hash of decrypted secret code matches with the hash value embedded in the transaction, the network awards the asset to the recipient address K_{adr}^R.

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 common agreements. 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 presented in [5, 11]. One assumption is that the nodes in sender and receiver networks are synchronized with a global clock.

In the network of our model, some nodes are elected as gateway nodes. We envision each blockchain as an autonomous system, which communicates with each other via a 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 (based on the topology) connected through a gateway mechanism to create a network-of-networks (NoN) [6].

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 require to exchange their public keys via a key exchange mechanism. S uses the public key of *R* to encrypt the secret code (γ) and share the encrypted secret code (Γ) with *R* to initiate *entryTransaction*. Then, they mutually agree on the time-lock period t. After that, Alice S generates the burn address (β). Here, we describe the steps in detail.

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

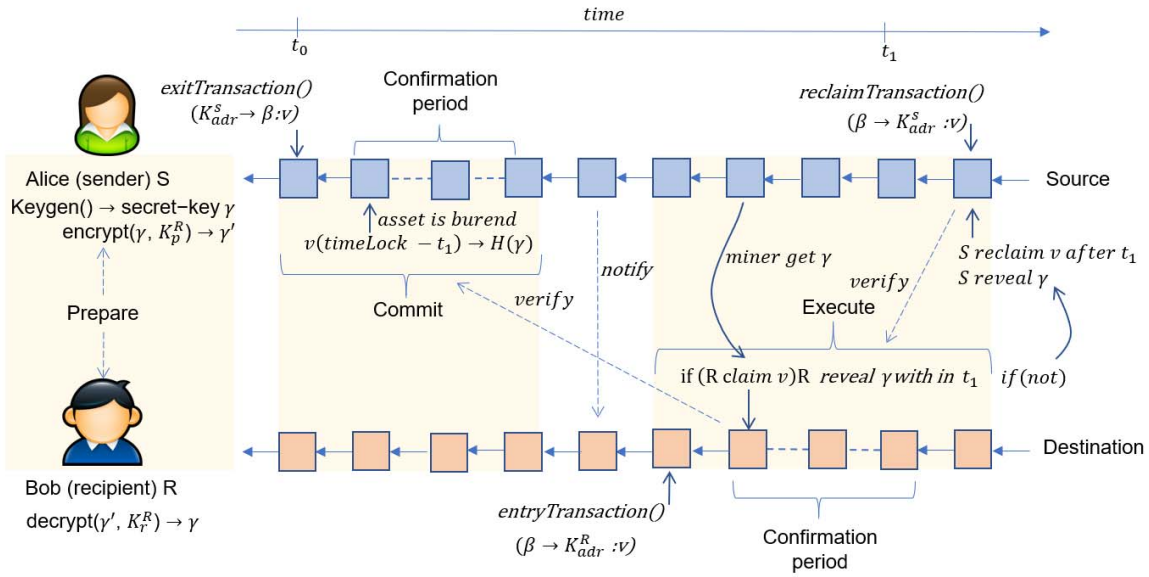

Figure 1. A high level overview of the Burn-to-Claim protocol workflow. The boxes represent chained blocks along the time steps in the source and destination networks. At the **preparation** stage, S generates and shares an encrypted secret code encrypt(γ) with R. At the **commit** stage, S initiates the exitTransaction, burning the asset to β 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 β and reveal γ within the time-lock period. If R failed to claim the v, S reclaims the v after the time-lock period.

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 β. This will guarantee that the address β does not have a private key in the source network and β 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 4. 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 back 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 [9]. In order to prevent double-spending, our protocol uses a secret code and a time-lock value. To claim an asset, the recipient must reveal the secret code whereas the sender is unable to reclaim an asset until the time-lock period is expired.

4) Commit stage: In the commit stage, S creates and broadcasts an exitTransaction to invoke Algorithm 2 in the network N_1.

Algorithm 3 exit-transaction-time-key-lock condition

```
1:  while (time-lock is true) do
2:      if (R claim v) then
3:          R reveal secret γ to N_2
4:          N_1.miner gets γ from N_2
5:          N_1.miner claims his γ-locked fee
6:                              ▷ secret γ 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 γ to N_1
11:     S re-claim v
12:     N_1.miner claim his γ-locked fee
13:                             ▷ secret γ is known to N_1
14: end if
```

Algorithm 3 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 gateway node 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

Tx_t output is time-locked in the source network, if R reveals the secret code γ in her network, the miner who mined this transaction in the source network will reveal it in source network to claim his fees; thereby the γ is known to both networks. After that, S will not be able to claim the asset. Likewise, if R fails to claim the asset within the time frame, then S reveals γ after t_{lock} expired in the source network to reclaim the asset. In our design γ is not known to the network; therefore, once the γ is revealed nobody will be able to claim the asset even after t_{lock} the expiry of 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 expired.

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 β and a previous transaction $Tx^{\dagger}$, which form the proof of commit in the source chain.

Our protocol requires a mining process where some nodes are able to mine in multiple blockchains. With that requirement, among n number of nodes m gateway nodes propose this transaction and eventually one proposal will get 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 [7] 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 γ 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 γ 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 4, 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.

Algorithm 4 reclaim-transaction-protocol

```
function reclaimTransaction((Tx(K_p^S,K_adr^S,β,v,Tx†),Γ,σ)
    if reclaimVerifier() is true and
        (Tx†.t_lock pass the time limit) and
        decrypt(Γ,K_p^S) = Tx†.H(γ)) is true then
        β → K_adr^S : v
    else
        invalid transaction
    end if
    return transaction receipt
end function
```

The function of reclaimVerifier consists of two sub-functions spendVerifier and proofVerifier. The function check if verify(Tx, K_r^R, σ) 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(β) = $burn$ 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.

III. Analysis

In this section, we analyse 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 [2, 10].

It should be noted that the burn-address in our proposed protocol is not generated from a regular keypair, 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.

Lemma 1 (Unspendability)**.** *A burn-address β is unspendable with respect to a blockchain address protocol [11].*

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 being transferred. In exitTransaction the exportVer-

ifier() 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.*

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

The function exitTransaction transfers the asset v to a burn-address β that is derived from the recipient R's address K^R_{adr}. To claim v, the recipient must prove to the network that β is derived from an address he owns. The function spendVerifier checks the signature σ to verify the K_p and checks the whether β 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 2 (Correctness)**.** *The exchange operation only exchange an asset to a 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 σ be a secure signature scheme then the possibility of a distribution entity to decipher the K_r from a K_p or β is unpredictable.

Theorem 3 (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.*

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 party, so as to realise fair cross-chain exchange.

IV. Conclusion

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 using standardised pathways. Furthermore, we briefly showed that Burn-to-Claim protocol is resilient to double-spending by its correctness and fairness properties.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

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

[7] 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*.

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

[9] 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.

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

[11] 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.

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

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

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