Formal Verification of the Burn-to-Claim Blockchain Interoperable Protocol

Babu Pillai[1,2], Zhé Hóu[1], Kamanashis Biswas[1,3], and Vallipuram Muthukkumarasamy[1]

[1] Griffith University, Gold Coast, Australia
[2] Southern Cross University
[3] Australian Catholic University, Brisbane, Australia
babu.pillai@outlook.com

Abstract. This paper introduces an abstract blockchain model that employs the Burn-to-Claim cross-blockchain protocol [1]. This multi-level simulator models a virtual environment of nodes running on the Ethereum Virtual Machine (EVM). Developed using the *CSP#* language [2], it has undergone formal verification with the model checker *PAT*. Focusing on inter-network operations, our model[4] examines the properties of correctness, security, and atomicity using *PAT*. Surprisingly, atomicity, assumed to be inherent in the time-lock mechanism of the Burn-to-Claim protocol, does not always hold. We establish its validity under specific assumptions while confirming the protocol's correctness and security under the added assumptions.

Keywords: Burn-to-Claim · blockchain · interoperability · formal verification

1 Introduction

Despite the recent surge in the number of proposed interoperable protocols, there is a lack of formal guarantees for the properties of those protocols. This is also a common issue in the field of (cyber)security, where much of the protocol design is proved by hand, which is error-prone. Commercial developments of such cyber and network systems are often validated by extensive software testing, but testing can only show the presence of bugs, not their absence. In recent years, formal verification has been used for verifying highly sensitive systems and protocols.

In this paper, we employ a tool named Process Analysis Toolkit (PAT) [3], which supports modelling, simulation and verification of many forms of systems. The supported modelling languages include Hoare's Communicating Sequential Processes [4] extended with C# (CSP#), timed automata, real-time systems, probabilistic systems and hierarchical systems.

The main contributions of this paper include:

[4] https://github.com/b-pillai/Burn-to-Claim-formal-verification

- Modelling: We build a formal model for the Burn-to-Claim blockchain interoperable protocol [1] using CSP# that focused on inter-network operations within a network.
- Verification: Using the developed models, we specify several correctness and security properties using LTL and reachability and verify them using the PAT system.
- Findings: We discuss the verification results. In particular, the atomicity property, which contrary to the common understanding of the adopted timelock technique, does not hold in general. However, it holds under certain assumptions.

2 An overview of the Burn-to-Claim protocol

At the highest level, our objective is to construct a model encompassing a scenario wherein a *user* intends to transfer an asset from one blockchain network to another. The *Burn-to-Claim* protocol [1, 5] facilitates this transfer, ensuring that the asset is destroyed (or removed) from the originating blockchain network and subsequently re-created on the destination network. Within this process, the networks from which the asset is removed are termed *source* networks, while those to which the asset is transferred are labelled *destination* networks. This transfer procedure is twofold: initially, the source network produces a self-verifiable transfer-proof, and subsequently, upon verifying this proof, the destination network reconstitutes the asset.

Workflow Upon initiating the *exitTransaction* in the source network, nodes within this network validate the transaction request. The primary objective is to generate a *proof*, which allows verification of the specific transaction without needing to reference the entire history of the associated asset. After this transaction is committed in the source chain, the *transfer-proof* triggers an *entryTransaction* in the destination network, paving the way for the asset's recreation in the recipient network. Nodes within the destination network then validate the *transfer-proof* and proceed to recreate the asset. We model the above system for formal verification in different modules described below.

Module 1 — exitTransaction This transaction triggers a transfer request on the source network, creating an exit point for an asset on the sender's blockchain through network consensus. The system's generation of a *transfer-proof* ensures security. Once the network agrees on the transaction's authenticity, a *transfer-proof* log is added to the next block, and the asset is locked to prevent further extensions.

Module 2 — entryTransaction This transaction aims to replicate the asset in the destination network. Executing the *entryTransaction* and *transfer-proof* functions from the source chain, the network validates the *transfer-proof* and reproduces the asset.

Let us assume the *exitTransaction* log and timestamp on the sender network created of t_1 time will be delivered to the recipient network through a gateway node with a time latency of t_2.

Module 3 — reclaimTransaction If the recipient does not claim the asset within the time-lock period t_2, the sender can use *reclaimTransaction* to retrieve it. The function verifies the signature and time-lock before returning the asset to the sender.

3 Specifications of the protocol

In order to perform verification and show that the system model satisfies a set of desired requirements, we define a set of four scenarios to check the functional security of the system. 1) A sender sends a transaction and the correct recipient makes the claim. 2) A sender sends a transaction and after the time lock period reclaims the asset. 3) A sender sends a transaction and a malicious recipient tries to make the claim. 4) A malicious sender sends a transaction and tries to reclaim it within the time-lock period. Based on the above four cases, we have defined a set of properties that should hold in the Burn-to-Claim protocol [5, 1].

Property 1 (Burn-Before-Claim). An asset that is transferred from the source network must be burned on the source network before the recipient can claim it on the recipient network.

Property 2 (No-Double-Spend). Double spending is not permitted in the Burn-to-Claim protocol.

Property 3 (Correctness). The Burn-to-Claim protocol only transfers an asset to the correct recipient.

Property 4 (Strong-Atomicity). The transfer operation should only obtain one of the following outcomes: either the transfer succeeds, and the asset is transferred to the recipient, or it fails, and the asset returns to the sender.

Property 5 (Weak-Atomicity). Under the assumption that either the recipient or the sender is guaranteed to make the (re)claim, a transfer operation should only have one of the following outcomes: either the transfer succeeds, and the asset is transferred to the recipient, or it fails, and the asset returns to the sender.

4 The model for cross-blockchain interactions

In this model, we focus on the inter-network events, and we only consider high level operations of transaction and mining, and leave the detailed intra-network operations to a different model that will be discussed in the next section.

The two networks are defined as constants N_1 and N_2 representing the source network as N_1 and destination network as N_2. A set of variables defined are

TxItems to hold the number of items in a transaction, *MaxTx* set the maximum number of transactions, *MaxMiners* set the number of miners, *TxAmount* to hold defaults value to transfer, *InitAmount* to hold initial wallet balance, *MaxUsers* number of users and *ChannelBufferSize* for channel buffer size.

The structure of blockchain is not critical in this model; therefore, for simplicity, we view blockchain data structure as a list of transactions as in *tx[0]*, as transactions in N_1, *tx[1]* as transactions in N_2.

Users. There are four types of users in the model: *User1*, *User2*, *Sender* and *Recipient*. *User1* and *User2* are network specific participants. That is, *User1* exists only in N_1 and is able to send transactions within the same network. Similarly, *User2* exists and operates only within N_2. We model the transaction as a tuple of six items in the order of *sender's network*, *senders address*, *recipient network*, *recipient address*, *beta*, *value*, *gamma*, *miner address*. In reference to the transaction Tx defined in the Burn-to-Claim paper [1], we omitted previous transaction $Tx^{\dagger}$ as it is not important in this model. Based on the role and requirement, the network participants send separate transactions. The *Sender* and *Recipient* are participants that can send cross network transactions to another network. The sender and recipient use separate channels to broadcast the transaction.

Miners. We model two types of miners *Miner1* and *Miner2*. Miner1 is a miner on the network N_1, listening to the channels of [*trans1*, *exit* and *rec*]. This miner execute a relevant function based on the channel the message is coming from. The Miner2 is mining on N_2, listening to the channels of [*trans2* and *entry*] and executes a relevant function based on the channel the message is coming from.

The mining process. There are five processes defined which will be executed by the miner based on the input request. The *minerVerify1* process facilitates value transfers within N_1. It first verifies conditions like transfer networks and user limits before executing the transfer. *minerVerify2* serves a similar purpose for N_2. When *exitTransaction* is invoked, miners ensure the user has sufficient funds. The *exportVerifier* process, in this model, is simplified, excluding signature verification. If conditions are met, tokens are transferred to a burn-address. For the *entryTransaction*, miners validate time-lock, burn-address status, and the provided security code. If verified, the recipient's wallet increases. For the *reclaimTransaction*, similar checks are made, and upon validation, the sender's wallet is refunded.

Process execution. We model the execution of cross-blockchain transfer in a blockchain as a *CrossBlockchains()* process. The process is structured such that the *mining*, *user*, and *BurnToClaim()* processes run concurrently. The *mining* process begins with miners mining on both N_1 and N_2. Meanwhile, users process their transactions on their respective chains. The *BurnToClaim()* process then executes the defined scenarios based on the user's selection.

5 System verification

In this section, we cover the model's targeted assertions and properties, outline the verification procedures and present results. An assertion examines system behaviours. For a given process, it checks if a state meets a specific condition. When process p runs, if e is true, then f remains true.

To verify Property 1 (Burn-Before-Claim), we define an assertion in PAT as below. This assertion verifies that an asset is burned before the recipient's claim using $[(pClaimed -> [\,](!pBurned))]$. Results confirm the asset must be burned prior to a claim. In our model, the sender commits the transfer to a burn address, and once the network accepts it, the decision is deemed final.

```
#assert CrossChains |= [](pClaimed -> [](!pBurned));
```

We define the three assertions given below to check Property 2, *double spending*. The initial assertion examines if an asset, once claimed by the recipient, cannot be reclaimed by the sender using $[(pClaimed- > \ )]$. The next assertion observes whether the recipient's wallet increases, while the sender remains unchanged using $[(recipientClaimed \ \&\ senderReclaimed)]$. Lastly, an assertion confirms against double spending scenarios, ensuring that both recipient claims and sender reclaims cannot coexist, as evidenced by $[(pReclaimed- > \ )]$. Verification results from these three assertions confirm the impossibility of double spending in this model's configuration.

```
#assert CrossChains |= [](pClaimed -> [](!pReclaimed));
#assert CrossChains |= [](!(rClaimed && sReclaimed));
#assert CrossChains |= [](pReclaimed -> [](!pClaimed));
```

To check the Property 3 (correctness), we define the assertion given below. This assertion examines if a burned asset remains in the burn address, is claimed by the recipient, or is reclaimed by the sender, represented by $[(pBurned -> [\,](burnValueExists \ || \ pClaimed \ || \ pReclaimed))]$. Verification confirms that a burned asset consistently stays in its burn address throughout the process.

```
#assert CrossChains |=
[](pBurned -> [](burnValueExists || pClaimed || pReclaimed));
```

We define below two assertions to check strong atomicity (Property 4) and week atomicity (Property 5) atomicity:

```
#assert CrossChains|= [](rClaimed || senderReclaimed));
#assert CrossChains|= [](protocolCompleted ->
    (rClaimed || senderReclaimed));
```

The first assertion checks strong atomicity using $(recipientClaimed \ || \ senderReclaimed)$. The verification of Property 4 fails as PAT found a counterexample event sequence, which we analyse the explain below.

5.1 Discussion

Why general atomicity not hold? Strong atomicity requires all related operations to succeed. In cross-blockchain contexts, this means assets are burned by the sender and claimed by the recipient. However, distributed environments face potential network partitions or system crashes, preventing participants from engaging. Apart from technical issues, intentional recipient inaction or incorrect asset burns by the sender can also compromise strong atomicity.

Cross-blockchain operators facilitate transactions for multiple self-interested parties. A single party's actions can have intricate impacts on others. Given the potential for malicious or irrational behaviour, the assumption is that both sender and recipient act in mutual interest. The sender burns the asset, and the recipient mints an equivalent asset. To validate weak atomicity, we use the second assertion (Property 5).

Property 5 (weak atomicity) assumes that if the sender burns the asset, the recipient must mint it. This is expressed using the second assertion *(protocolCompleted − > (RecipientClaimed || senderReclaimed))*. We introduce *protocolCompleted* to address network and node failures. This property holds.

6 Conclusion and future works

The current model focuses on verifying burn-before-claim (Property 1), double-spending (Property 2), correctness (Property 3), and both strong (Property 4) and weak atomicity (Property 5). While detailed transaction and block data structures are not considered, we model transaction verification's essential operations. Detailed transaction verification will be addressed in future work that incorporates a merge mining process.

References

1. B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "Burn-to-claim: An asset transfer protocol for blockchain interoperability," *Computer Networks*, vol. 200, p. 108495, 2021.
2. J. Sun, Y. Liu, J. S. Dong, and C. Chen, "Integrating specification and programs for system modeling and verification," in *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*. IEEE, 2009, pp. 127–135.
3. J. Sun, Y. Liu, J. Dong, and J. Pang, "Towards flexible verification under fairness," in *CAV '09: 21th International Conference on Computer Aided Verification*, 2009.
4. C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666–677, 1978.
5. B. Pillai, K. Biswas, Z. Hóu, and V. Muthukkumarasamy, "The burn-to-claim cross-blockchain asset transfer protocol," in *2020 25th International Conference on Engineering of Complex Computer Systems (ICECCS)*. IEEE, 2020, pp. 119–124.