

Architecture for Stablecoins with Cross-Chain Interoperability

Éric Bastos Costa Machado, Juliana de Melo Bezerra^a and Celso Massaki Hirata^b

Department of Computing Science, Instituto Tecnológico de Aeronáutica (ITA), São José dos Campos, Brazil

Keywords: Stablecoin, Blockchain, Smart Contract, Cross-Chain Interoperability, Architecture for Stablecoin Service.

Abstract: Blockchain is the enabling technology that implements the operations of cryptocurrencies. Stablecoin is a type of cryptocurrency designed to reduce price volatility. This stability is achieved by tethering the value of the stablecoin to a reserve of assets, often in the form of a fiat currency like the US dollar. Implementing a stablecoin involves various technical challenges related to the design and architecture, which include smart contract complexity and cross-chain interoperability. This work presents an architecture for the backend of stablecoin services that address these two challenges. In the architecture, the bridge component enables seamless cross-chain interoperability, allowing to move of stablecoins from one blockchain to another without the need to be reverted to fiat currency. We developed a proof of concept, using the stablecoins deployed on both Ethereum and Polygon testnets. The proof of concept demonstrated that the architecture offers a design reference to implement other similar stablecoin systems.

1 INTRODUCTION

Blockchain technology has made an impact across various sectors, including supply chain management, healthcare, real estate, government and public services, and financial services. Blockchain's key elements are decentralization, cryptography, distributed ledger, and chained blocks (Yaga et al., 2018). It has been shown that this technology can provide trust, transparency, and shareable information without resorting to intermediaries. Security is provided by the use of cryptography and the consensus mechanisms maintain the integrity of the blockchain and record's immutability.

Bitcoin (Nakamoto, 2008) laid the way for many other cryptocurrencies, such as Litecoin and Ethereum. These currencies offer the advantages of decentralization, security, and transparency; however, their values can fluctuate significantly, which poses challenges for day-to-day transactions and store-of-value purposes. Stablecoins (Phillips, 2020) address the currency volatility by pegging their values to an underlying asset, such as a fiat currency like the US dollar or a commodity like gold.


Stablecoins built on blockchain technology can facilitate fast and low-cost cross-border transactions, and businesses can benefit from faster settlement


times, lower fees, and improved cash flow management. The idea behind stablecoins of creating programmable tokens pegged to an asset (tokenization) has shown huge potential. For instance, the Federal Reserve (the central bank of the United States) is studying the implications of a Central Bank Digital Currency (CBDC) (Infante et al., 2022). Brazil's Central Bank is currently developing the Digital Real (Banco Central do Brasil, 2023).

Fulfilling the stablecoins' potential is not a simple task. Stablecoins need to be properly designed and regulated to offer a compliant and transparent solution for digital transactions. Regulatory frameworks for stablecoins are evolving (Bains et al., 2022), and businesses that operate within established legal frameworks can gain credibility and build trust with regulators, financial institutions, and customers.

Two technical challenges that deserve attention include smart contract complexity and cross-chain interoperability. The majority of stablecoins operate on blockchain platforms and utilize smart contracts. Designing and implementing complex smart contracts to manage collateral, stability mechanisms, and governance is challenging and requires careful consideration. Besides, stablecoins face technical challenges in ensuring seamless integration and communication between different blockchain networks. This is achieved by cross-chain interoperability mechanisms.

In this paper, we propose a general architecture to

^a  <https://orcid.org/0000-0003-4456-8565>

^b  <https://orcid.org/0000-0002-9746-7605>

support the implementation of key functionalities of stablecoins, dealing with smart contract complexity and cross-chain interoperability. The key functionalities include keeping the peg and providing transactions of purchase and sale of tokens. Although the proposal is agnostic to the currency to which it is pegged, we choose the Brazilian currency *Real (BRL)* as the fiat currency backing the stablecoin reserve. The smart contracts are described as considering Ethereum-virtual-machine (EVM) compatible chains, for instance, Polygon and Ethereum.

This paper is organized as follows. The next section describes the related work. Section 3 introduces the proposed architecture. The proof of concept is described in Section 4. Section 5 concludes our work.

2 RELATED WORK

A comprehensive reference for blockchain can be found in (Antonopoulos, 2017). In (Yaga et al., 2018), the general functionality of the blockchain is described, which includes categories of blockchain, its core components, such as the blocks and the encryption, and consensus mechanisms.

Blockchain is a decentralized and distributed ledger technology that enables secure and transparent record-keeping of transactions across a network of computers. Consisting of a chain of blocks, each containing a cryptographic link to the previous block, this technology ensures the immutability and integrity of the data stored within. Utilizing consensus mechanisms among network participants, such as Proof of Work (PoW) or Proof of Stake (PoS), blockchain eliminates the need for a central authority, fostering trust in the system. Transactions, once added to the blockchain, become permanent and tamper-resistant, providing a transparent and verifiable history of all interactions. Beyond its association with cryptocurrencies, blockchain finds applications in various industries, offering solutions for secure data management, smart contracts, and decentralized applications.

On the topic of stablecoins, the current literature emphasizes much more on definitions and presents a more economic view of the subject. In (Baughman et al., 2022), the authors explain the role of stablecoins and the rules for the issuance and redeeming of tokens. Stabilization mechanisms are discussed together with the most common collateralization mechanisms: on-chain collateralized, off-chain collateralized and algorithmic stablecoins.

In (Mell and Yaga, 2022), the authors present many considerations regarding the security and the trust of the reserves and bring attention to funds

movement in the secondary market of centralized and decentralized exchanges, where users trade tokens amongst themselves and are subject to certain attacks, such as malicious smart contracts or other exploits.

Regarding technical implementation, in (Nageswaran et al., 2019), the authors present a minimum viable product for the implementation of a custom stablecoin named Digipound. The authors design the token in the blockchain, a web application to interact with an API that handles the interaction with the blockchain and Stripe (Stripe, 2010), a payment processing system, and also covers an auditing mechanism that crosses the information of the current reserve backing the stablecoin and the current circulating supply in the blockchain. Their work touches on some of the necessary steps to implement and release the custom stablecoin, considering the implementation of a trading service, constructing the smart contract, running a local blockchain node, and potential security concerns. We go a step further by diving deeper into the smart contract details, proposing and designing a complete non-monolithic system architecture, and implementing a cross-chain solution for deploying and integrating the token across multiple blockchains.

Finally, on the subject of cross-chain interoperability, in (Pillai et al., 2020), the authors comment about how different blockchains have different trade-offs and how the notion of “one blockchain to rule them all” is simply unreal. The work further discusses the different strategies for chain interoperation, such as sidechains (systems inside a blockchain that can read the state of other blockchains) or hash-locking (operations set to trigger after the revelation of some kind of secret), and the theory and implementation of these techniques from a computer science point of view.

On more practical terms, a particularly interesting and technical solution is exhibited by (Xie et al., 2022): a blockchain bridge based on zero-knowledge proofs (a way of proving the validity of a statement without revealing the statement itself), focusing on decentralization and the efficiency of proof validation. The solution is validated even in a scenario of a non-EVM-compatible chain (Cosmos) bridging information to an EVM-compatible chain (Ethereum). In our work, we implement a simpler centralized bridge, since the token issuer entity is naturally centralized in the case of fiat-backed stablecoins.

Our work aims to contribute to the stablecoin literature by presenting a technical approach to the subject. In general, the broader part of the current literature addresses a more economic perspective on this topic, focusing on the definitions, implications, and

use cases whereas our proposal includes the cross-chain interoperability mechanism and can serve as the technical reference for developing stablecoin solutions.

3 THE PROPOSED ARCHITECTURE FOR STABLECOIN SERVICE

Figure 1 shows the proposed architecture for the stablecoin service. The architecture counts on microservices to encapsulate different responsibilities and ways to communicate with those microservices. Some components, such as banking as a service, are outside the stablecoin service since they are seen as third-party systems.

Since a stablecoin mainly acts as an infrastructure for the blockchain ecosystem, we expect two main types of users to consume the service. The first type is the common user, who buys and sells their tokens. To allow this access, it is necessary a front-end application operating with a friendly wallet service (such as MetaMask). The second type of user is the business or technical user (intermediary). A business user probably has his platform serving his clients, which integrates with the stablecoin service.

As follows, we present the microservices of the stablecoin service, dive deeper into the smart contract details, and present the centralized bridge solution.

3.1 Stablecoin Service Breakdown

Below we break down the stablecoin service into its microservices to better explain all the functionalities and how they support minting and burning operations.

The *API* gateway exposes all the endpoints of the stablecoin service, such as the endpoints for creating accounts and logging in, the endpoints for minting, burning, and bridging the stablecoins, and the endpoints to retrieve the historic information of operations. The *API* is RESTful, and it is the entry point for all the operations that a user can execute. It is recommended that the *API* be public unless the system has some specific restrictions on access.

The *Banking Events Webhook* is responsible for listening to the events that come from the banking service. To avoid having to poll the banking service for information, it is common for the banking services to allow their clients to register their own server in which they can listen for real-time events. There are various tasks executed by the *Banking Events Webhook* for the stablecoin service. To describe some of the tasks, we

use the Pix (a real-time electronic funds transfer system in Brazil). In September 2023, it reached 41% of transactions carried out surpassing credit cards, debit cards, and cash (EBC Agencia Gov, 2023). For instance, a Pix transfer made by the user means that the user has paid the order and the system must mint the tokens from the blockchain to the user's wallet address. A Pix transfer event made by the system means that the system has burned the tokens and the money has been transferred successfully to the user's banking account. Regardless of the event, the final action usually is either minting or burning tokens.

The *Smart Contract Interface* is the touchpoint with the blockchain. It is responsible for sending the transactions to the blockchain and interacting with the stablecoin token smart contract. The *Smart Contract Interface* consumes a message queue in charge of holding the blockchain interactions, so any other microservice can enqueue an operation that will eventually be handled by this microservice. As important as sending the transactions to the blockchain is verifying if it executed successfully, if it failed, or if it is hanging due to an insufficient amount of gas. This is made by tracking each transaction and checking its status in the blockchain. When the transaction is mined, the system must check if it was a success.

The *The Data Persistence* is a persistent storage that allows all transactions to be recorded in a way to ensure the transaction's durability. This safeguards the stablecoin issuer against any claims of not delivering the tokens when minting, or not transferring the money when burning. The events that must be stored are all the blockchain operations and their results, events from the banking service (e.g. Pix transfer), the entities taking part in the transactions (e.g. the users), and requests for buying and selling the stablecoin.

The *The Communication Infrastructure* permits the internal services communicate by two means: synchronous and asynchronous. The *gRPC*, a Remote Procedure Call (RPC) framework implemented by Google, is used for synchronous communication. Its main use is querying the *Smart Contract Interface* (for example, to check a user's balance of a certain token) or querying the *Proof of Reserves* for the most recent data it has available. The *Messaging Queues*, on the other hand, are used for asynchronous communication. They are especially important for providing a way to interact with the *Smart Contract Interface*, which in turn interacts with the blockchain asynchronously, increasing system's overall performance, reliability, and decoupling.

The *Operation Events Webhook* is useful only to the users who consume the API through their tech-

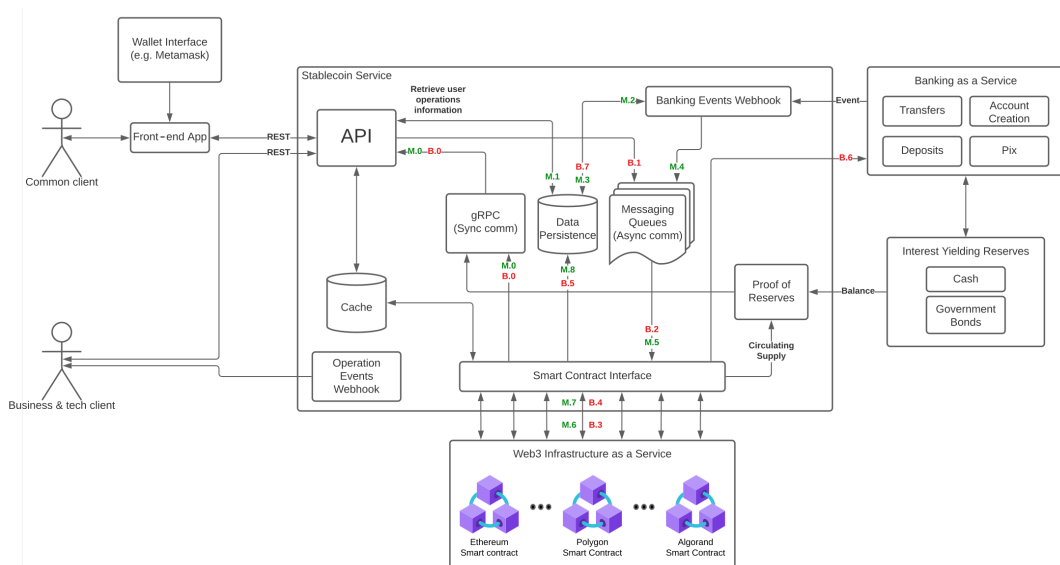


Figure 1: General architecture for developing stablecoin solutions considering interoperability mechanism.

nical implementation. Its responsibility is to act the same way as the banking service acts, sending real-time updates of events related to the user’s account. Similarly, the system sends real-time updates of events related to the minting and the burning operations to any user that registers a server on which they are listening.

The *Proof of Reserves* is responsible for providing transparency regarding the reserves backing the stablecoin. The task it must accomplish is very simple: just expose the total circulating supply of the token, and the current money reserve in the bank, and show that the reserve is greater than the circulating supply. By itself, it is not enough. In general, it is a common practice to have a third-party company audit the reserves since someone might fake the numbers.

The *Cache* is a common way to reduce the load on the server and the database processing, being present in many APIs in general, and the stablecoin service is no exception. Some functions that are very usual to call in the blockchain are the ones related to getting the current gas price to pay for the transactions. This is a value that does not change so quickly, so it makes sense to cache it for multiple transactions instead of re-evaluating it before each transaction.

When a user buys or sells stablecoins, the flow that is executed inside the stablecoin service is indicated in Figure 1 by the green M’s followed by a number and the red B’s followed by a number. The M stands for Mint, and the B stands for Burn, whereas the numbers indicate the order of the operations. Here we explain each one of the steps for executing the mint operation, when stablecoins are bought. The process begins with input validation (M.0) to ensure the accu-

racy and integrity of the data. Subsequently, an entry is created in the database (M.1), indicating the system’s anticipation of a deposit from the user. Upon payment of the invoice by the client, verified through a deposit notification from the Banking Service, the system verifies the received information (M.2) before updating the deposit status to ‘paid’ and storing the payment details (M.3). Following this, a mint operation is initiated and queued (M.4), and upon receipt, the operation is executed (M.5). The system proceeds to create, sign, and broadcast a mint transaction to the blockchain (M.6) to interact with the *Stablecoin Smart Contract*, awaiting confirmation of the transaction’s success (M.7). Finally, the mint result is stored in the database (M.8) for future reference and auditing purposes.

3.2 The Stablecoin Smart Contract

This is the smart contract that represents the stablecoin token in the blockchain. It must be secure, transparent, compliant, and, at the same time, flexible. The main features of the stablecoin smart contract are presented in Figure 2, mainly concerned with the ERC-20 standard implementation. However, other functions are also relevant and shall be incorporated in a real scenario.

To make the smart contract an actual fungible token, we need to implement the ERC-20 standard and that is what the imported libraries do. Regarding the implemented functions, the *mint* function allows the creation of more tokens when anyone buys the token. The *burnFromWithPermit* is the function responsible for burning tokens, which is made when some-

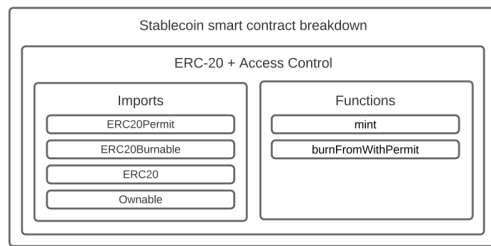


Figure 2: Key features of the stablecoin smart contract.

one wants to sell the tokens.

Due to the way that ERC-20 tokens are implemented, if another address (being a smart contract or another actual wallet) wants to update the balance of tokens, effectively subtracting any amount from it, it must first send an approval transaction to the blockchain, specifying which address can do so, and at most how many tokens it can update. Such a feature is troublesome because sending this approval transaction to the blockchain costs gas, which means that the user must hold a certain amount of the blockchain's native tokens, but we, as the stablecoin issuer, want to minimize any barriers for our end user to buy and sell our tokens.

To remove this barrier when the system has to burn someone's tokens, the system makes use of the ERC-2612 standard (Lundfall, 2020). The main idea behind it is to implement a way to delegate the approval transaction to someone else. This is made by generating an off-chain (outside the blockchain) signature which can be verified on-chain (inside the blockchain). The function that validates this signature and executes the approval is called *permit*. The *burnFromWithPermit* function is simply a concatenation of a *permit* operation followed by a burn operation.

The implementation of the stablecoin smart contract shall consider additional functions related to security, compliance, and upgradeability.

The security functions handle the access control of the stablecoin smart contract. Only the issuer should be able to mint and burn tokens, whereas any other addresses should only be able to transfer tokens they already have. So the mint and burn functions should be protected by some logic that checks who is interacting with the smart contract. We can go further and better separate different concerns into different roles, such as an *Owner* role (which grants the ability to upgrade the smart contract), an *Operator* role (which grants the ability to mint and burn tokens), a *Pauser* role (which grants the ability to pause and unpauses the smart contract), and a *Compliance* role (which grants the ability to blacklist and whitelist addresses).

The compliance functions are related to the necessity of eventually complying with regulations since

the stablecoin is fiat-backed. Some functions are of interest. The *pause* function completely stops the smart contract, disabling any token transfer between any addresses, and the *unpause* function returns it to its operating state. The *blockAddress* and *unblockAddress* are the blacklist and whitelist functions, effectively working as the pause and unpause operation but aimed at a specific address. Finally, the *isWhiteListed* is a simple function that returns whether an address is blacklisted or not.

The upgradeability functions refer to upgrading the smart contract. When smart contracts are deployed behind a proxy, so they can be upgraded, they lose the constructor function (which is a function called as soon as the smart contract is created and deployed). This is where the *initialize* function comes into play. It is a special function that can be called only once after linking the proxy to the execution smart contract, effectively acting as the constructor. The *_authorizeUpgrade* is a function called before upgrading the smart contract and should have any logic regarding the authorization of this upgrade, such as checking if the function called has the upgrader role, for example.

3.3 The Centralized Bridge Design

Since implementing a decentralized bridge is much more complicated and the API for minting and burning the stablecoins would already be considered a centralized way to buy the tokens, it makes sense to implement a centralized design for the bridge.

The bridge, from the API point of view, is just another operation, but it is fundamental to cover it apart because it is the feature that enables cross-chain interoperability and allows the users to move their tokens from one blockchain to another without first returning to the asset backing the stablecoin, in this case, the Brazilian Real.

Figure 3 shows, in very basic terms, how the bridge operation is processed. A user creates a bridge requisition in the API, and it is executed by first burning the user's tokens in the input blockchain, and minting the same amount in the output blockchain. It is a two step process, but nothing more than a concatenation of two already implemented operations.

Since the mint operation in the output blockchain must only be made once the burn operation is confirmed in the input blockchain, it is important to highlight, again, the necessity of correctly handling reorganizations in the input chain. If the API does not properly wait for the finality of the mint operation, by waiting for enough blocks to be mined after, a reorganization can happen, and the user might have his

tokens returned to his wallet in the input chain while they were also minted in the output chain.

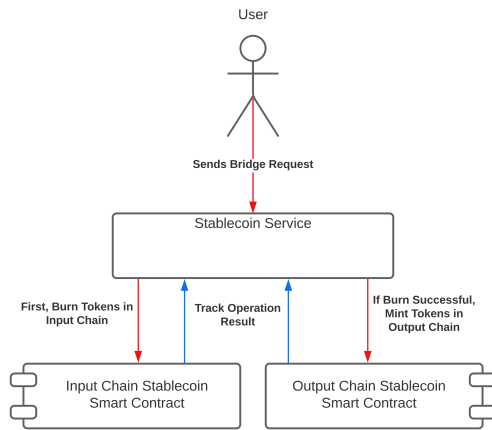


Figure 3: Centralized bridge operation.

4 A PROOF OF CONCEPT

For a proof of concept, a simpler version of the stablecoin service and the smart contract was implemented. The idea was to keep only the essential functionalities related to the minting and burning of the stablecoin. The supported blockchains were also restricted to only Ethereum and Polygon to simulate bridge operations. The simplified version of the stablecoin service is based on the architecture shown in Figure 1, but without the complementary microservices as *Operation Events Webhook*, *Proof of Reserves* and *Cache*.

Interactions with the banking service were simulated in the proof of concept. The simulated banking service can receive a request for a bank transfer, return a result, and send events regarding Pix deposits. This way, both the burning and minting of the tokens can be properly implemented. We also assume that the bank operations never fail to further simplify the implementation. The smart contract was kept with the essential functions as presented in Figure 2, in a way to hold the basic functionalities of minting, burning, and access control, without worrying about compliance, security, and upgradeability.

The repository with all the code supporting the implementation of this project is available in GitHub (Bastos, 2023). The deployment of the system is easy and pretty much plug-and-play due to the containerization technology (using Docker technology) and can be customized to some extent by changing the configurations. A user-friendly interface is available as a front-end application (written with Javascript and CSS). The API itself is also fully documented, using the OpenAPI 3.0 specification, and the stable-

coin smart contract is simple and ready to be deployed in any EVM-compatible blockchain. The *API*, *Bank Webhook* and *Smart Contract Interface* are written in Go language. The *Messaging Queues* are implemented with RabbitMQ (VMware, 2007), whereas the *Data Persistence* uses PostgreSQL. To exemplify the use of the developed application, we show the mint, burn, and bridge operations below.

With the user logged in and his MetaMask wallet connected, he is ready to buy some amount of stablecoins. This can be made in the *Deposit* tab, where the user can input the amount in Brazilian centavos (basically, $amount \times 100$). The wallet address and the blockchain are automatically fetched from MetaMask, and he can just click the generate invoice button, which registers the purchase order in the Stablecoin Service backend, and shows a QR code that would contain the bank account information for the user to pay the order. This process is demonstrated in Figure 4.

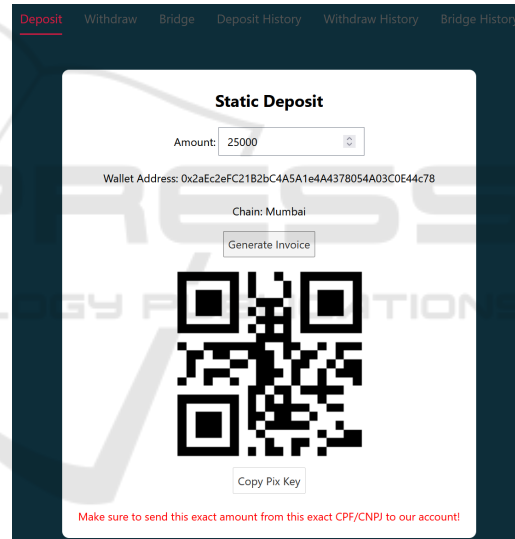


Figure 4: Buying stablecoins.

Due to the simulation of the user’s payment in the backend, after a couple of seconds, the invoice is considered to be paid and the minting process starts. In no more than a couple of minutes, the tokens are deposited in the user’s wallet, and he is ready to use them for anything he wants. In the *Deposit History* tab, he can track the status of his transaction, including the status of the operation in the blockchain. Figure 5 shows the just executed purchase operation. By clicking the link in the *Blockchain Hash* column, the user can see the actual blockchain transaction that was responsible for minting his tokens.

With the user logged in and his MetaMask wallet connected, he is also able to sell some amount of

STATUS	AMOUNT	TAX ID	WALLET ADDRESS	CREATED AT	CHAIN	BLOCKCHAIN HASH
PAID	250	66967926004	0x2aEc2eFC21B2bC4A5A1e4A4378054A03C0E44c78	2023-11-05T20:00:09.368274Z	Polygon	Click to check

Figure 5: History of purchase operations.

stablecoins. This can be made in the *Withdraw* tab, where the user can input the amount he wants to sell, in Brazilian Centavos, and his Pix Key, which encodes his bank account information. The wallet address and the blockchain are automatically fetched from MetaMask. So he can just click the withdraw button, which starts the burning operation and, if successfully executed, the bank transaction, right after. Figure 6 presents the withdraw interface. After pressing the withdraw button, MetaMask prompts the user to sign the permit which is necessary for the API to burn the user’s tokens. Only after accepting and signing the message, the burn operation effectively starts. Likewise, the *Withdraw History* tab contains the tabulated information of past sale transactions.

Figure 6: Selling stablecoins.

The last operation of the proof of concept is the bridge operation. This operation involves two blockchain transactions: burn in one blockchain and mint in the other. It can be started in the *Bridge* tab, and the history of past bridge operations can be verified in the *Bridge History* tab. Figure 7 presents the bridge interface. An amount of stablecoins to bridge is requested from the user, as well as the permit to burn his tokens in the input chain. Since the API currently supports only two chains, the front end automatically selects the input and output blockchain according to the current blockchain selected in MetaMask.

Figure 7: Bridging stablecoins.

5 CONCLUSIONS

We proposed an architecture for the backend of a stablecoin service with cross-chain interoperability. We presented the needed microservices and how they deal with the client’s requests to orchestrate the steps of the burn, mint, and bridge operations. We also detailed how a stablecoin is created in blockchain by using a smart contract based on the ERC-20 standard. Regarding the proof of concept, the proposed architecture was successfully implemented. The solution made use of EVM-compatible blockchains, but there are other types of blockchains with growing ecosystems that could offer even cheaper and faster transactions, being ideal for the development of stablecoins.

In addition to the proposed centralized bridge solution, a promising avenue for future work is to allow a trustless bridge solution. As the blockchain and decentralized finance ecosystems continue to grow, the need for secure and seamless inter-connectivity between various blockchain networks becomes increasingly critical. Ensuring transparent and tamper-proof transactions between these networks, without relying on centralized intermediaries, holds great potential for enabling a more robust and decentralized digital economy.

By simulating the banking service in the proof of concept, we postponed an important part of the implementation which is correctly integration with a real payment system, understanding the format of its re-

sponses and its limitations. The possibility of errors was not considered, but a real application would have to be robust to errors in operations. For future work, we suggest studying the challenges and risks of integrating other payment methods, such as debit and credit cards. Pix is a system currently exclusive to Brazil, and international cards are used in the entire world, so this would be a great way to increase the user base.

Our proposal considered a fiat-backed stablecoin, but it could be extended to a stablecoin backed by any asset acting as the collateral for the stablecoin reserves. Studying and developing stablecoins with a different pegging mechanism, such as crypto-backed stablecoins, is another topic to be researched. Algorithmic stablecoins, in particular, still have proven to be quite challenging to implement with a trusted pegging algorithm.

Another topic of interest is to reason about the real usage of the stablecoin. Although everything was developed in a test environment, to avoid any real costs of using the blockchain, a real stablecoin ready to be used in production would need many mechanisms to incentive users to buy stablecoins. These mechanisms could be anything, from offering forex exchange capabilities with other stablecoins to any activity that uses real money, such as sports betting.

REFERENCES

- Antonopoulos, A. M. (2017). *Mastering Bitcoin: Unlocking Digital Cryptocurrencies*. O'Reilly Media.
- Bains, P., Ismail, A., Melo, F., and Sugimoto, N. (2022). *Regulating the crypto ecosystem: the case of stablecoins and arrangements*. International Monetary Fund.
- Banco Central do Brasil (2023). BCB selected 14 institutions to collaborate with the development of the piloto RD. Central Bank of Brazil. <https://www.bcb.gov.br/en/pressdetail/2481/nota> Accessed on 2024-01-02.
- Bastos, E. (2023). Project repository. <https://github.com/EricBastos/ProjetoTG> Accessed on 2024-01-02.
- Baughman, G., Carapella, F., Gerszten, J., and Mills, D. (2022). The stable in stablecoins. FEDS Notes. <https://doi.org/10.17016/2380-7172.3224> Accessed on 2024-01-02.
- EBC Agencia Gov (2023). Pix reaches impressive numbers in just three years. <https://agenciagov.ebc.com.br/noticias/202311/pix-chega-a-numeros-expressivos-em- apenas-tres-anos> Accessed on 2024-01-02 *In Portuguese*.
- Infante, S., Kim, K., Orlik, A., Silva, A. F., and Tetlow, R. J. (2022). The macroeconomic implications of CBDC: A review of the literature. Finance and Economics Discussion Series 2022-076, Board of Governors of the Federal Reserve System. <https://www.federalreserve.gov/econres/feds/the-macroeconomic-implications-of-cbdc-a-review-of-the-literature.htm> Accessed on 2024-01-02.
- Lundfall, M. (2020). Erc-2612: Permit extension for EIP-20 signed approvals. <https://eips.ethereum.org/EIPS/eip-2612> Accessed on 2024-01-02.
- Mell, P. and Yaga, D. (2022). Understanding stablecoin technology and related security considerations. Technical report, National Institute of Standards and Technology. <https://nvlpubs.nist.gov/nistpubs/ir/2023/NIST.IR.8408.pdf> Accessed on 2024-01-02.
- Nageswaran, S., Knottenbelt, W., and Leung, K. (2019). Digipound: A proof-of-concept stablecoin audited in real time.
- Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system. *Decentralized business review*, page 21260. <https://bitcoin.org/bitcoin.pdf> Accessed on 2024-01-02.
- Phillips, K. (2020). How stable are stablecoins and what factors affect volatility? Lukka. <https://lukka.tech/how-stable-are-stablecoins-and-what-factors-affect-volatility/> Accessed on 2024-01-02.
- Pillai, B., Biswas, K., and Muthukkumarasamy, V. (2020). Cross-chain interoperability among blockchain-based systems using transactions. *The Knowledge Engineering Review*, 35:e23.
- Stripe, I. (2010). Stripe. <https://stripe.com/> Accessed on 2024-01-02 *In Portuguese*.
- VMware (2007). Rabbitmq. <https://www.rabbitmq.com/> Accessed on 2024-01-02.
- Xie, T., Zhang, J., Cheng, Z., Zhang, F., Zhang, Y., Jia, Y., Boneh, D., and Song, D. (2022). zk-Bridge: Trustless cross-chain bridges made practical. <https://arxiv.org/pdf/2210.00264.pdf> Accessed on 2024-01-02.
- Yaga, D., Mell, P., Roby, N., and Scarfone, K. (2018). Blockchain technology overview NISTIR 8202. <https://nvlpubs.nist.gov/nistpubs/ir/2018/NIST.IR.8202.pdf> Accessed on 2024-01-02.